## Cardiac Monitoring Device

Katie Carroll & Deepti Gopinath

## Introduction and Background

*Anatomy and Physiology of the Heart*

The heart is made up of four chambers: left atrium, right atrium, left ventricle, and right ventricle. Oxygenated blood from the lungs travels through the pulmonary veins into the left atrium, left ventricle, and finally the aorta to be pumped through the whole body. The deoxygenated blood then returns to the heart through the superior and inferior vena cava, the right atrium, right ventricle and finally pulmonary artery to be pumped back to the lungs so it can be oxygenated.

When the cardiac muscles in the ventricle contract it’s called systole. This causes the blood present in the ventricles to be pushed out of the heart (into the aorta and pulmonary artery). When these muscles relax, it’s called diastole. This allows the blood present in the atria to move into the now relaxed ventricles.

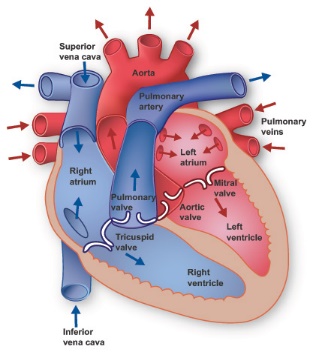
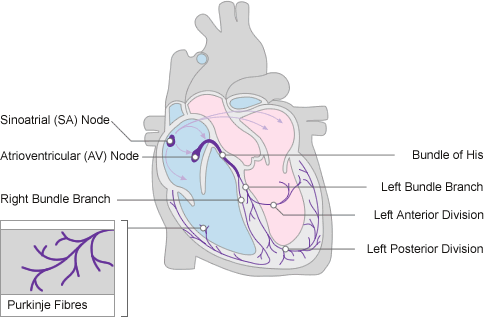


Figure : Diagrammatic representation of conduction system (left) and anatomy (right) of the Human Heart []

*Electrocardiography (ECG)*

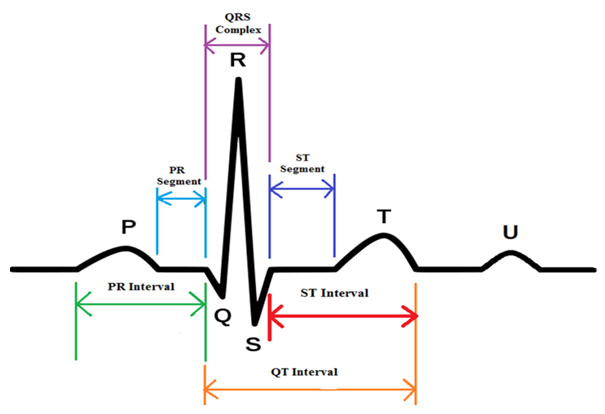
Electrocardiography is the process by which the electrical activity of the heart is recorded using electrodes placed on the arms, legs, and chest. [1] The electrical activity recorded corresponds to the heart depolarizing and repolarizing during the beating of the heart.

Figure 2 depicts a typical ECG waveform. The P wave represents atrial depolarization, the QRS complex represents ventricular depolarization, the T wave represents ventricular repolarization [3], and the U wave may represent delayed repolarization of the Purkinje fibers, but its cause is unclear [4]. The U wave, however, is not always observed in patients and is usually disregarded [5].

Figure : ECG waveform [2]

For the proposed cardiac monitoring device, our group aims to detect the peak of the QRS complex to measure both instantaneous and average heart rate. A resting human heart rate should fall in between 48-100 beats per minute. In the case where the heart beat is beating too slow (~40 BPM), or bradycardic, the device has a warning light that flashes at 2 Hz to alert the doctor. If the heartbeat is too fast (~180 BPM), or tachycardic, the warning light flashes at 10 Hz. The device is functional: programmed with a pause button, a reset button to clear heart rate values and a dial to manipulate the amplitude of the heart rate signal.

## Materials & Methods

*Hardware- Analog Signal*

The hardware aimed to provide the Arduino with clean analog and digital signals representing the heartbeat of a 3-lead ECG signal. Circuit design took into consideration high frequency noises over 100 Hz, low frequency noise under 0.5 Hz and 60 Hz noise. The hardware of the analog signal followed this logic:

Figure shows block diagram of analog signal hardware

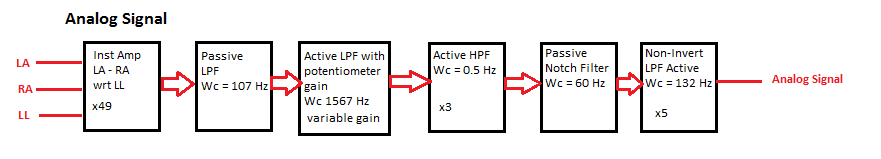
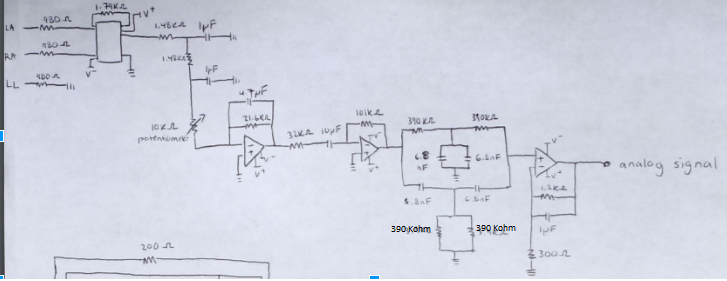
After thorough testing, the following circuit schematic was developed to be the analog input into the Arduino microcontroller:

Figure : Circuit schematic of analog signal hardware

*Hardware- Digital Signal*

In the next stage, the analog signal was transformed into a digital signal. This required a peak and hold circuit, a comparator, and a voltage clipper shown in Figure 6. The peak and hold caused the signal to level off below a certain gate voltage input into the transistor part of the circuit. This threshold voltage was set to 1.5 V in order to remove all waves except for the QRS complex. The output of the circuit was the level at 1.5 V and peaked at the QRS complex. To get a signal that was either +5 V or -5V (with no in between values) a comparator was implemented. The comparator had a reference voltage of 2 V, so the signal railed positive above 2 V and negative under 2 V. The range of this signal was, however, still too high to input into the Arduino. To combat this, a diode voltage clipper was used to bring the voltage down into the spectrum of 0 to 3.3 V. The logic can be summarized in the Figure 5 block diagram:

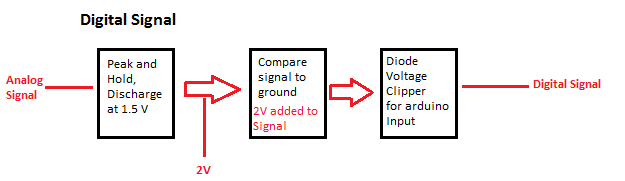


Figure : digital signal hardware block diagram

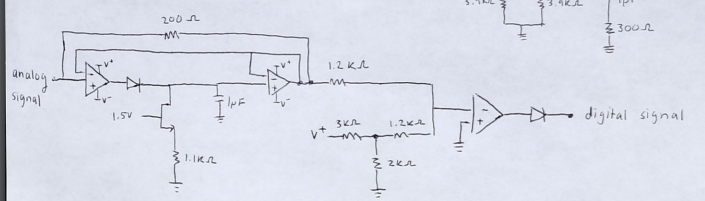
With further testing the following schematic in Figure 6 was utilized to create the digital input signal for the Arduino:

Figure : digital signal hardware circuit schematic

*Hardware- Lights and Buttons*

In addition to the analog and digital signals, the Arduino received inputs from two buttons (pause and reset) and outputted logic to two LEDs. The following circuit schematics in Figure 7 were utilized for the buttons and LEDs:

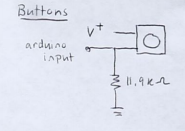
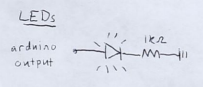


Figure : button circuit schematic (left) and LED circuit schematic (right)

A complete inventory of all hardware is shown below:

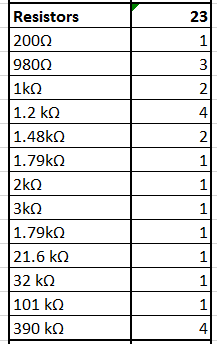
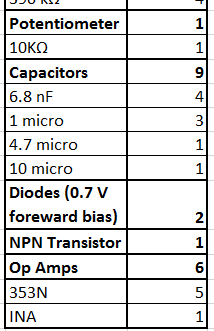


Figure : Inventory of cardiac monitoring device hardware

*Arduino Code*

The Arduino has many inputs and outputs that must be considered in the development of the code. The code needs to receive information from the digital signal, analog signal, pause button and reset button. The Arduino also outputs a warning light for bradycardia/tachycardia, a light that beats with the heart and an instantaneous/average heart rate read on the Serial Monitor. The inputs and outputs are shown below in figure 9:

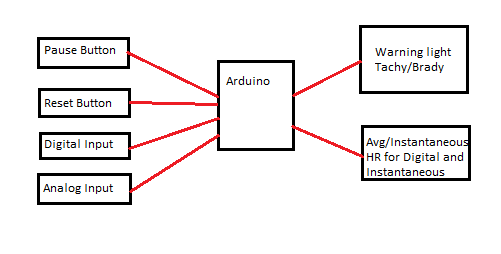


Figure : inputs (right) and outputs (left) that Arduino code must take into consideration

*Digital Signal*

The digital signal was transformed into heart rate measurements with the following block diagram logic:

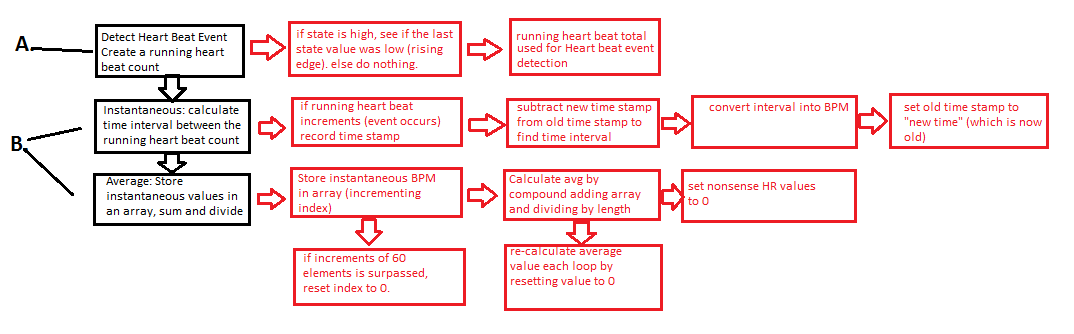


Figure : block diagram logic for digital signal code, in black are the three main steps: detecting heart event, instantaneous heart rate calculation and calculation of average heart rate. Red blocks signify sub-steps of black boxes.

The function *beatDetectDig* detects if a heartbeat event occurs and outputs a running total of the number of heartbeats. The function *beatDig* takes that heartbeat count and associates a time stamp with each event. The function takes the differences of the time steps, or interval, and outputs an instantaneous heart rate. Further, it stores the instantaneous heart beat values in an array. Once 60 samples have been entered into the array, the average converges to the correct value. The functions works as a moving average function, once 60 samples have been recorded, the index resets to zero and writes over old array elements. The signal sets any egregious heart rate values (>250) equal to zero, as to minimize error for the average heart beat array.

Analog Code:

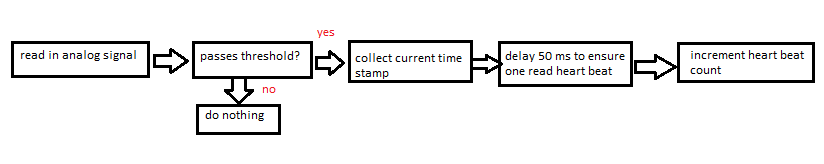
The analog portion of the code uses the same structure for calculating the instantaneous and average heart rate values, but uses a different method to detect the heartbeat. The function *AnaDetect* detects if a heartbeat event occurs and likewise outputs a running total of the number of heartbeats. The function utilizes a threshold that is a value larger magnitude than the P and T waves, but smaller than the QRS complex. After the signal surpasses the threshold, a delay is used to ensure that the detection function does not double count a heartbeat. This delay value was determined through experimentation. The general flow of the analog heart beat detection is shown below in Figure 11:

Figure 11: Block diagram of analog signal code, heart beat event detection function (AnaDetect)

*Output Light Signal*

It was important to implement code that minimized the use of the *delay()* function because the output of the device was time dependent. For the bradycardia/tachycardia alarms, we utilized *millis()* and time stamps to create an interval that caused a light to blink at a given Hz value. If the heart rate fell between 40 and 180 BPM, the light would be turned off. We also implemented an LED that beats with the heartbeat, as read by the digital signal. View code in Appendix.

*Input Button Signal*

The code for the buttons utilized the interrupt functions so that at any point in the code, the heart rate monitor could be paused or the average/instantaneous heart rate values could be reset. The pause button interrupt function was activated if the state of the pause button switched from low to high. At this point, it would flip the state of the *pauseFlag*. If *pauseFlag* was at a LOW state, the heart rate calculation loop would run; if the state was HIGH, the code would run in an empty loop. Likewise, the reset button interrupt function was activated by a rising edge and cleared all elements of the average heart rate arrays when activated.

## Results

Figure 12 shows the digital and analog signal of our device concurrently. The digital signal stays at HIGH until the QRS peak is detected and flips to the low rail.

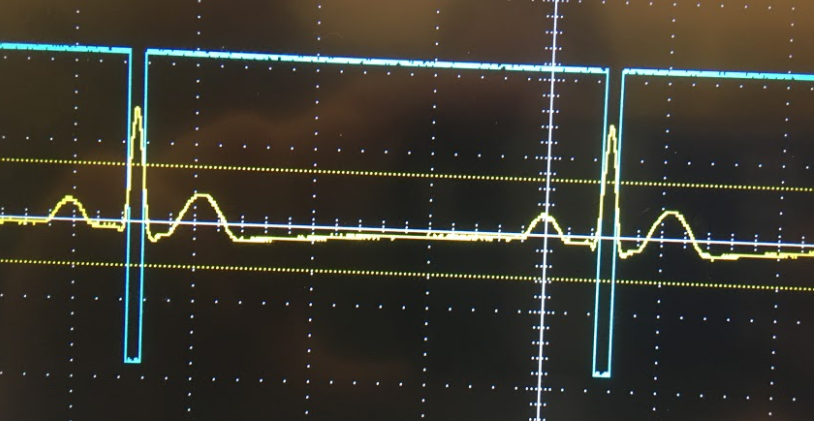
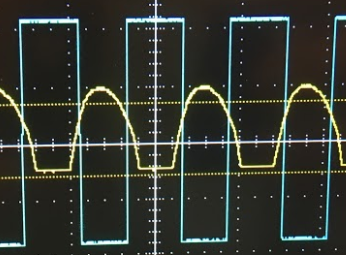
The heart rate signal accurately identified heart beats between 40 and 180 beats per minute. Both the digital and analog instantaneous heart rates were equivalent in the serial monitor. It was relatively accurate, when the ECG simulator was at a 60 BPM our device output 59 BPM. When the heartbeat value was switched to a float, it read 59.65 BPM which for our purposes is close enough to 60. The average heart beat array for both the digital and analog signals converged when enough data points were sampled.

Figure : digital and analog signals shown on the oscilloscope

If the heartbeat was raised to above 200 BPM, the analog signal began to resemble a rectified sinusoid as shown in Figure 13 and the average heart rate and instantaneous heart rate could not be calculated with our device.

Regardless, the device performed to the design specifications and is considered a success.

Figure : inaccurate analog signal at 200 BPM heart rate

## Discussion

*Overall Approach*

The overall approach for designing the circuit/software was to first obtain a clean analog signal. From this point, the group shifted their focus—focusing on Arduino and digital signal hardware separately. It was fairly straightforward to implement the Arduino code with the analog and digital signal.

*Hardware Design Insights*

A non-inverting low pass op amp was utilized in the last stage of the analog signal hardware (Figure 4). This is due to the relatively high impedance of the previous notch filter stage. If the op amp were to be inverting, the impedance was not high enough to move the signal along to the next stage and the signal vanished. By using a non-inverting op amp with an impedance significantly higher than an inverting op amp, the analog signal could be appropriately amplified and filtered after the notch filter.

*Design Specification Alterations*

The code calculates an average heart rate based on number of samples, not time period as specified. Further development is necessary to implement a time based average heart rate calculation.

It will be necessary to investigate the exact cause of the incorrect analog heart signal at 200 BPM. It is possible that the various filtering stages may have impacted this signal. Reevaluation of the cut-off frequencies and amplifications would prove beneficial in the further improvement of our device.

*Digital vs. Analog Signal*

The instantaneous and average heart rates between the digital and analog signals were the same values, and verify that our code is working as it should. It seems that a digital signal would allow for a more accurate and repeatable signal, as it is less variable than the analog signal. The digital signal requires less code, and less estimating with thresholds. The digital portion of the hardware follows the same logic as the analog peak detection code without worrying about delays. The digital signal is either high or low, easily detectable by the Arduino Microcontroller.

## Conclusion

The team successfully designed a cardiac monitor that utilized a 3-lead ECG signal for both digital and analog design specifications. Through implementation of both software and hardware, the group amplified and minimized noise to accurately estimate both instantaneous and average heart rate between 40 and 200 beats per minute. Although both analog and digital signals can be used to monitor cardiac events, the group feels that digital signals may create more robust and repeatable results.

## References (Can you actually cite these)

[]<http://www.nottingham.ac.uk/nursing/practice/resources/cardiology/function/anatomy.php>

[]<http://www.texasheart.org/HIC/Anatomy/anatomy2.cfm>

[]<http://www.nottingham.ac.uk/nursing/practice/resources/cardiology/function/conduction.php>

[]<http://www.wkcardiology.com/Patient-Information/Physiology/>

[1] <https://www.nhlbi.nih.gov/health-topics/electrocardiogram>

[2]<https://www.researchgate.net/figure/287200946_fig3_Fig-14-Schematic-representation-of-normal-ECG-waveform>

[3]<https://www.ncbi.nlm.nih.gov/books/NBK2214/>

[4]<https://lifeinthefastlane.com/ecg-library/basics/u-wave/>

[5]<https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1513620/>

[6]

## Appendix- Arduino Code

/\* EKG CODE

\*BME 790

\*Katie Carroll & Deepti Gopinath

\* Code takes in 3 lead- EKG signal and outputs both instantaneous and average HR for digital and analog signal

\* LED flashes in response to Bradycardia and TachyCardia. LED input is based on Digital Signal.

\*/

// Pin Numbers

int digitalPIN = 12;

int pausePIN =2;

int resetPIN = 3;

int LEDPIN = 8;

int analogPIN = A0;

int hrLightPin = 4;

// LED Parameters

int tach\_f = 100; // 10 Hz

int brad\_f = 500; // 2 Hz

int ledState = LOW; // set light LED

unsigned long previousMillis = 0; // last time value

int LEDstateHR = LOW;

// Interrupt Function Variables

boolean pauseFlag = LOW; // set pause flag to low, runs loop

// Digital Signal Parameters

int state; // current state

int oldBeat; // last beat count

int beatD = 0; // heart beat counter used to determine heart rate

int last = 0; // to be compared to current state

float told; // last time step

float tint; // time step interval

unsigned long t = 0; // current time

int hrD; // output, instantaneous heart rate from digital signal

int avgD; // output, average heart rate from digital signal

// Dummy variables

int i = 0;

int x = 0;

int p;

// Heart rate array parameters

int lengthArray = 60; // input length of indicies for heart rate average

int avgDig[60];

// Analog Signal Parameters

int sigA;

int beatA; // counts number of heart beats for analog signal

int sig2A;

int sig1A;

unsigned long t1= 0;

unsigned long t0;

int oldCount;

int hrA;

unsigned long tA;

float tintA;

float toldA;

int avgAna [60]; // average analog HR array

int avgA; // average analog HR output

///////////////////////////////// SETUP ///////////////////////////////////////////////

void setup() {

// put your setup code here, to run once:

Serial.begin(9600);

pinMode(LEDPIN,OUTPUT);

pinMode(digitalPIN, INPUT);

pinMode(analogPIN, INPUT);

pinMode(hrLightPin,OUTPUT);

attachInterrupt(digitalPinToInterrupt(pausePIN),pauseF,RISING); // Pauses Function

attachInterrupt(digitalPinToInterrupt(resetPIN),reset,RISING); // Resets heart Rate information

}

///////////////////////////////// LOOP //////////////////////////////////////////////////

void loop() {

if(pauseFlag == LOW){

oldBeat = beatD; // used to trip bitDig flag

beatDetectDig(); // used to detect if digital HR event occurs

HRlight(); // light blinks with digital heart Rate

beatDig(); // outputs both average and instantaneous HR for digital signal

tachy(); // LED if HR is Tachycardic

brady(); // LED if HR is Bradycardic

oldCount = beatA; // flag used to trip anaBeat

anaDetect(); // used to detect if analog HR event occurs

anaBeat(); // outputs both average and instantaneous HR for analog signal

if (hrD> 40 && hrD<180){

digitalWrite(LEDPIN, LOW); // Turns light off if the HR is in between HR standard

}

}

// if flag HIGH, nothing occurs in loop "pause"

}

//////////////////////////////// INTERRUPT FUNCTIONS //////////////////////////////////////////////////////////

//set to 0 code runs, if interrupt fxn activate flag is 1 and code is paused

void pauseF (){

pauseFlag = !pauseFlag;

digitalWrite(LEDPIN,LOW);

}

// clears all heart rate values (Avg/Instant for Analog/Digital)

void reset () {

for (int j =0; j< lengthArray; j++){

avgDig[j] = 0; // reset values to zero in average digital array

hrD = 0; // reset instantaneous digital HR to 0

avgAna[j] = 0; //reset values to zero in average array

hrA = 0; // reset instantaneous analog HR to 0

}

}

////////////////////////////// NORMAL FUNCTIONS ///////////////////////////////////////////////////////////////

//Light Functions: Flash lights Tachy at 10 Hz, Brady at 2 Hz

void tachy (){

unsigned long currentMillis = millis();

if(hrD >= 180){

if (currentMillis - previousMillis >= tach\_f){

previousMillis = currentMillis; // runs at 10 Hz

if (ledState == LOW){

ledState = HIGH; // if light is off, turn it on (blinking)

} else {

ledState = LOW; // if light is on turn it off

}

digitalWrite(LEDPIN, ledState); // apply state changes

}

}

}

void brady (){

unsigned long currentMillis = millis();

if(hrD <= 40){

if (currentMillis - previousMillis >= brad\_f){

previousMillis = currentMillis; // runs at 2 hz

if (ledState == LOW){

ledState = HIGH; // if light is off, turn it on (blinking)

} else {

ledState = LOW; // if light is on turn it off

}

digitalWrite(LEDPIN, ledState); // apply state changes

}

}

}

void HRlight() {

if (oldBeat-beatD != 0){

LEDstateHR = HIGH;

digitalWrite(hrLightPin, LEDstateHR);

delay(20);

LEDstateHR = LOW;

digitalWrite(hrLightPin,LEDstateHR);

}

}

// Digital Functions

void beatDetectDig(){

state = digitalRead(digitalPIN); // read state of digital pin

if (state != last){

if(state == HIGH){

beatD++; // if state changes and the state is high, count heart beat, (rising)

} else {

}

delay(15); // debouncing techinque to avoid double heart beat counts

}

last = state; // reset last state flag to compare for next loop

}

void beatDig(){

if (oldBeat - beatD != 0) { // detect event (it changed aka not 0)

Serial.print(hrD); // print instantaneous digital HR

t = millis(); // current time stamp

tint = t-told; // interval calculation

tint = tint/1000.000; // in seconds

hrD = (60/tint); //heart beat in minutes

told = t; // reset flag

Serial.print(" "); // print space

if (hrD> 250){

hrD = 0; // get rid of any HR that is greater than a phsyiological bound

}

avgDig[i] = hrD; // assign heart rate values to avg array

i++; // increment index

if (i % lengthArray == 0){ // reset index, once 60 beats is obtained, to form running average

i = 0; // reset index

}

avgD = 0; // clears average

for (int j =0; j< lengthArray; j++){

avgD += avgDig[j]; // sum up elements of array

}

avgD = avgD/lengthArray; // find average

Serial.print(avgD);

Serial.print(" ");

}

}

void anaBeat(){

if (oldCount - beatA != 0) { // detect analog event

Serial.print(hrA); // serial print instantaneous analog HR

tA = millis(); // current time stamp

tintA = tA-toldA; // analog time interval

tintA = tintA/1000.000; // in seconds

hrA = (60/tintA); //interval in minutes

toldA = tA; // reset flag

if (hrA> 250){

hrA = 0; // get rid of large HR outside of phsyiological bounds

}

Serial.print(" ");

avgAna[p] = hrA; // assign heart rate values to avg array

p++; // increment index

if (p % lengthArray == 0){ // reset index, once 60 beats is obtained, to form running average

p = 0; // reset index

}

avgA = 0; // clears average

for (int j =0; j< lengthArray; j++){

avgA += avgAna[j]; // sum up elements of array

}

avgA = avgA/lengthArray; // find average

Serial.println(avgA);

}

}

void anaDetect(){

sigA = analogRead(analogPIN); // read signal from analog pin

if (sigA> 100){ //make sure its the Big peak, delay to ensure we are only detecting ONE heart beat

t0 = millis(); // current time stamp

if (t0 - t1 >= 50){ // delay used to only detect one Heart beat, time value delay guess and checked

t1 = t0; // reset flag

beatA++; // incremental beats

}

}

}