Final Project: Robust Cardiac Monitor

Jocelyn Corey and Sarah Jacobs Allen Yin and Taylor Konrath

Abstract— **Our final project implemented a robust cardiac monitor that measured an ECG signal from a TechPatient Cardio Simulator. The cardiac monitor converted the incoming ECG signal into a dynamic analog and digital signal to be processed by an Arduino/Genuino 101. The cardiac monitor displayed instantaneous and one minute average heart rates calculated from both the analog and digital forms of the ECG signal. The cardiac monitor provided 'pause' functionality to pause heart rate measurement as well as 'reset' functionality to adjust measurement to a new patient. We also included LED indicator lights to alert operators to measured heart rate deviations and ideal heart rate ranges.**

I. INTRODUCTION

In this project, an electrocardiogram was taken from three separate leads on the skin on the right arm, left arm, and left leg to get a picture of the electrical activity of the heart. The signal was then filtered and put through a peak and hold circuit with dynamic gain and diode clippers in order to produce an analog signal that utilized the analog input range of the arduino. Similarly, the peak and hold signal was passed through a comparator to produce a digital signal of the heartbeat. The heart rate data, in the form of a digital signal and an analog ECG waveform were then interpreted by an Arduino, which then displayed instantaneous and one minute averages of the heart rates for the digital and analog signals. Additionally, if the heart rate is more than 40 beats per minute (bpm) or less than 180 bpm, an LED light up. If the analog instantaneous rate differs more than 10% from the digital instantaneous rate, a second LED will blink at a 2 Hz rate. The peak and hold circuit additionally has a reset feature that allows the circuit to calibrate thresholds used to determine the peaks for the peak and hold by discharging the capacitor with a MOSFET. Electrocardiograms have immense clinical usage, as they are able to show the electrical activity of specific regions in the heart.

Electrical signaling from the sinoatrial node causes action potentials to propagate through the heart and contract. A standard ECG for a healthy human will have P, T, and U waves along with the QRS complex, each of which shows a particular part of the heart polarizing or depolarizing. Specifically, the P wave shows atrial depolarization and the PR time interval corresponds to a time lag from the beginning of atrial depolarization to the beginning of ventricular depolarization. The QRS complex shows ventricular depolarization, and the T wave shows ventricular repolarization. The QT interval shows the time between ventricular depolarization and repolarization, which is inversely related to heart rate. Atrial repolarization happens between the P wave and QRS complex but is so small in magnitude that it does not show up on the ECG, and the U wave shows the repolarization of Purkinje fibers.

While electrocardiograms themselves are incredibly useful because the variations of the signal can be directly linked to the part of the heart that may be working incorrectly, heart rate data is a valuable way to give a clinician immediate feedback as to the condition of the patient. High or low heart rates can indicate significant health issues as well as give information on how the patient handles a clinical intervention.

II. BACKGROUND

The heart rate monitor utilized an Arduino microcontroller, circuit components, and a TechPatient Cardio simulator to generate the ECG signal at voltages of 0.5 to 4 mV. The circuit amplified the signal using an LED and photoresistor in order to dynamically adjust the amplitude of the analog signal. A peak and hold signal with 75% gain was inputted into a comparator in order to produce a digital signal that was triggered with the QRS complex of the ECG signal.

Additional functionality for the heart rate monitor included a pause button which paused heart rate measurement and displayed the one minute averages for the digital and analog signals. The monitor additionally had two LED outputs, one of which would light up when the heart rate was within a range of 40 to 180 beats per minute, and the other of which would blink at a 1 Hz frequency if the analog and digital heart rates differed by more than 10%.

II. MATERIALS AND METHODS

A. Hardware

The ECG signal was initially processed by a series of stages in hardware to obtain usable signals for the Arduino to calculate heart rate. The hardware followed the steps provided in the block diagram seen in Fig. 4. The initial processing steps followed the ECG circuit that was used in "Experiment 4: Biopotential Amplifier/ECG" as outlined for Duke BME 354. The only difference was that in the last stage of the circuit, the 10 KΩ resistor was replaced with a 100 KΩ resistor in order to decrease the gain in that stage as can be seen in the final circuit schematic in Fig. 5. All Op-Amps in the circuit were powered at ± 12V.

The next stage divided our signal into analog and digital parts. Part of the signal was diverted to a peak and hold circuit with a 2N7000 transistor that allowed us to selectively discharge the peak and hold in order to calibrate our cardiac monitor to new patients. A resistor was attached to the gate of the transistor in order to stabilize the gate. The output of the peak and hold was then sent through an inverting LF353 Op-Amp with ¾ gain. This reduced our signal to roughly ¾ of its maximum peak. Due to some current leakage, the peak and hold did not perfectly hold the max peak, but it formed a good enough approximation for the digital signal to still be obtained. The reduced amplitude signal was sent through another inverting Op-Amp with no gain in order to invert the signal once more back to the original orientation. This signal was then sent through an LM311 which compared the ¾ of the maximum peak to the incoming signal from the Experiment 4 ECG filtering and gain stage as described above. This can also be observed in the block diagram and circuit schematic in Fig. () and (). The comparator went HIGH whenever the incoming signal reached above ¾ of the maximum peak. This corresponded to it going high whenever the QRS complex was reached. It also meant that it only read HIGH once per heartbeat. This HIGH/LOW signal was sent into an Arduino digital pin.

The analog signal meanwhile passed through a non-inverting LF353 Op-Amp with variable gain. A photoresistor was connected to provide negative feedback in the non-inverting Op-Amp. The resistance of the photoresistor was modulated by an LED that was attached to the output of the peak and hold stage. As a result, larger maximum peaks from larger amplitude signals caused the LED to glow more brightly. This reduced the resistance of the photoresistor more than for small signals. By placing the photoresistor in the negative feedback stage, the gain decreased as the input voltage increased. Because the photoresistor was not linear, not all voltages were scaled to the same output voltage by the non-inverting Op-Amp, but lower voltage signals had a larger gain applied to them than higher voltages. The final output signal from the analog stages was then passed through a diode to remove negative portions of the signal, buffered to prevent current from being diverted by the Arduino, and passed through a diode to remove any more negative portions once more before being sent into an Arduino analog pin.

Three digital output pins from the Arduino were also used. They provided either a 3.3V or 0V signal. Two pins passed through a resistor to an LED as seen in Fig. 5. These LEDs were illuminated under different conditions. One blinked at 2Hz if the analog and digital heart rates differed by over 10%. The other LED illuminated in the heart rate was between 40 and 180 bpm. The final output pin controlled the gate voltage on the transistor in the peak and hold in order to discharge the capacitor and obtain a new peak for a new user.

B. *Software*

The software developed for the Arduino for the cardiac monitor can be found in Appendix C. The software was used on an Arduino/Genuino 101. The code had four main calculations and then a series of control signals to display cardiac information.

The first calculation was the instantaneous digital heart rate. The instantaneous digital heart rate was calculated using an interrupt scheme. Whenever a HIGH signal was sent to the digital input pin from the circuit, an interrupt service routine was executed. In our interrupt service routine, a boolean was toggled and a timer was started. In order to calculate the heart rate, in the main loop, if the boolean was true, a variable, *t1end*, recorded the time. If the boolean was false, a variable, *t2end*, recorded the time. Then, the time between each heartbeat was measured as the difference between *t1end* and *t2end*. Using the five most recent times measured between heartbeats, an exponential weighted average was applied. If five beats had not occurred, the previous beats were set to 0. The result of the exponentially weighted average was stored as the instantaneous digital heart rate and printed on the LCD screen in the main loop. The exponential average was calculated as

$$Instant\ HR = \sum_{n=1}^{5} \frac{60000}{t2(n)-t1(n)} exp(\frac{(n-5)}{2})$$

The time between heartbeats was also stored in an array of 201 times which updated for every heartbeat. In order to find the one minute average heart rate for the digital signal, a for loop constantly added these times up from the most recent to the least recent stored time. For each time added, a local variable called *count* incremented by one. If the times added up to more than 60,000 ms, *count* stopped incrementing and the value of count was stored as the number of heartbeats that occurred during the past minute. This was returned to the main loop and printed on the LCD screen as the one minute digital average heart rate.

The instantaneous analog signal could not be calculated through interrupts as interrupts only work on the Arduino for digital signals. Instead, we created a software peak and hold for the analog signal. For the first five seconds of operation, the analog signal calibrates in the setup of the Arduino. The Arduino simply reads the analog voltages input to the pin and stores the voltage if the input it reads is higher than previous maximum level that it stored. This value was our analog peak. Once the calibration was done, in the main loop, a function called instantA() was called. This set a threshold for our analog signal at approximately ⅘ the maximum analog value. This let us read the QRS complex but not be triggered by the T wave. Then we kept reading the analog pin. Whenever the incoming value was larger than the threshold, we toggled a boolean and a timer was started. The rest of the instantaneous heart rate logic followed the same method as used for the digital instantaneous logic. We also used a delay of 57 ms immediately after our analog signal passed the threshold. We found that this gave us sufficient time to not accidentally read the same QRS complex twice.

The time in between analog heartbeat signals was also stored in a vector of 201 analog times. In order to calculate the one minute average analog heart rate, these were added in a for loop and a counter was incremented in the same method as mentioned above for the one minute average digital heart rate.

The instantaneous and one minute averages were stored in global variables for both the analog and digital signal. In the main loop of the Arduino code, we checked if the analog average was between 40 and 180. If it was, we sent a HIGH signal from our pin, *rangePin,* to illuminate one LED. We also used a method called blink() which checked if the analog and digital instantaneous heart rates were more than 10% apart. If they were, this method turned on and off an LED at a rate of 2 Hz by sending a HIGH or LOW signal every 500 ms. Finally, we read the buttons on every loop. If the down button was pressed, we detached interrupts and reset all variables to 0 as well as sent out a HIGH signal for 1 second from a digital pin to the gate of the transistor in the peak and hold. This discharged the capacitor and reset the peak and hold. Then interrupts were attached again and the loop continued. If the up button was pressed, all heart rate measurements were stopped by toggling a boolean to false. If the boolean was false, none of the modular heart rate measurement functions were executed. The LCD screen would only display the last recorded minute averages. Once pressed again, the boolean would toggle to true and measurement would resume.

The materials used to create our cardiac monitor can be found in Table 1.

Table 1: Materials Used in Cardiac Monitor

Resistors	1 10 Ω 1 51 Ω 1 100 Ω 1 1 KΩ 1 4.7 KΩ 1 7.5 KΩ 9 10 KΩ 1 15 KΩ 1 20 KΩ 1 100 KΩ 1 150 KΩ 1 330 KΩ 2 1 MΩ 2 3.3 MΩ 1 photoresistor
Capacitors	1 1μF 1 10 nF 1 1.5 nF 1 0.1μF
Switches	1 switch 1 ISO-switch leads by Ohmic Instruments
Transistors	1 2N7000 transistor
Diodes	3 1N914 Diode 3 LED
MCU	1 Arduino/Genuino 101
Op-Amps	1 INA126 8 LF353 1 LM311

III. RESULTS

The heart rate monitor adequately determined the instantaneous and one minute average heart rate for heart rates between 40 and 200 beats per minute when the ECG signal ranged between 1 and 4 mV. The following figures display the filtered and amplified signals that were input into the microcontroller. The blue signal is the analog input and the yellow signal is the digital signal.

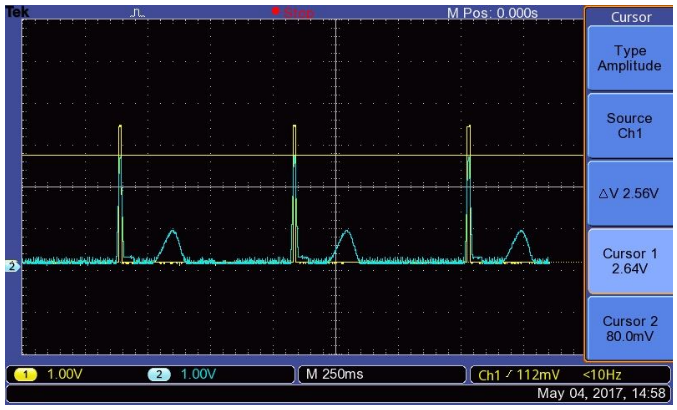


Fig. 1: Digital and Analog Signal at 4 mV and 86 bpm

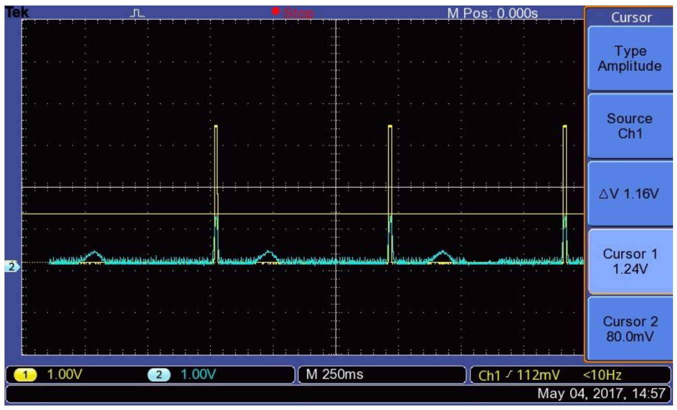


Fig. 2: Digital and Analog Signal at 1 mV and 86 bpm

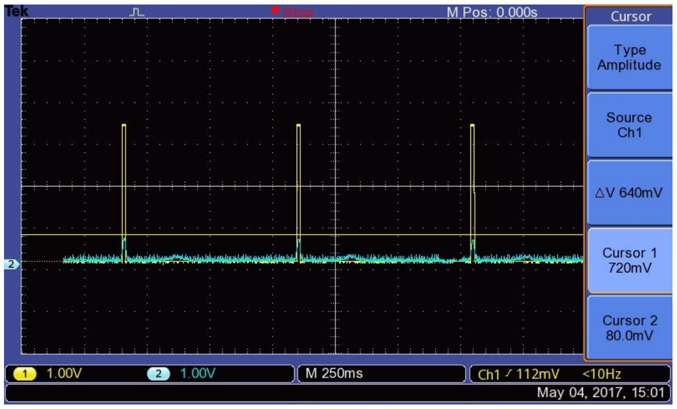


Fig. 3: Digital and Analog Signals at 0.5 mV and 86 bpm

From these figures, it is also evident that the dynamic gain was not as effective as desired since lower amplitude analog signals were not able to map to the full 3.3 V range of the arduino.

The heart rate monitor, when it was able to obtain a signal, was able to accurately determine the heart rate, which is evident in the data collected in Tables 2 and 3. It reliably was able to handle signals between 1 and 4 mV and heart rates between 40 and 200 beats per minute. With low voltages, the heart rate monitor would be unable to trigger a digital signal. Additionally, the software was unable to handle minute averages above 200 beats per minute due to the configuration of the code.

Dramatically decreasing the speed would occasionally cause the digital signal to no longer register the beats at all. However, this was only at beats lower than 40 beats per minute and out of the official range of the device. At high heart rates, the analog and digital minute average would sometimes miss beats, leading to lower than expected minute averages. These were occasionally visible in the instantaneous rates for both analog and digital as well. The increased potential for missing peaks also lead to 'false positive' flashes of the blinking LED, as the digital and analog signals would be different values for instantaneous heart rate, even though the values should be theoretically the same.

Changing the magnitude of the signal without recalibration would also lead to error. Recalibration was always necessary when changing the magnitude of the signal.

At mid range values, like 86 in Table 2, the signal was reliable for instantaneous rates and minute averages, for digital and analog values. They were also reliable for a wide range of signal magnitudes.

However, some interesting insights were gained in critically examining the data in the range of 40 to 180 beats per minute for both analog and digital minute averages.The standard deviations in error of the heart rate measurements are reported below in in Table 3.

	Digital Minute Std Error (bpm)	Analog Minute Std Error (bpm)
Overall	1.1	1.7
Low Rate (40-80)	0.4	0.2
Mid Rate (90-130)	2.5	2.8
High Rate (140-180)	2.1	3.7

The analog signal got progressively worse with higher heart rates while the digital signal performed similarly erroneous for mid and high heart rates. The analog signal was very limited by the frequency of the analog signaling in missing peaks, which got progressively worse as the frequency of the signals increased. The analog signal also only provided underestimates for heart rate as it missed beats, creating an underestimate bias, especially when examining high heart rates.

The digital signal performed more poorly at mid range heart rates and similarly at high rates. When using the function generator with 20% duty cycles and fast rates, the digital signal was far more accurate, indicating to us that the issue for the software was the interrupts catching all of the digital signals given the short duration for mid and high heart rates. There was again an underestimate bias, as any major deviation in heart rate (> 2 bpm) was an underestimate. The overestimates likely came from accidentally double counting a single heartbeat due to the interrupt scheme going low due to noise and then having the secondary rising edge for the same beat be counted as a separate beats. The majority of the low rates again came from the interrupt missing the short in duration signals.

IV. DISCUSSION

We approached this project in two main stages. First, we developed the hardware. We created a block diagram and a circuit schematic to help us analyze how we could get our desired signals. We decided that the final desired signals from the output of the hardware circuit would be a digital signal that varied between 3.3V when a QRS complex peak was detected, and 0V in all other situations. Our desired analog signal was simply an ECG signal scaled to be between 0V and 3.3V in order to not burn out the Arduino pins and get high resolution. In the end, we did not manage to use the full range of resolution for our analog signal as our dynamic gain was not linear. This also meant that we sacrificed all of the negative parts of the ECG signal instead of shifting it up.

In order to accomplish these two desired signals, we used a series of filters and op-amps to give the desired gain and noise reductions. We used a peak and hold and comparator to convert the analog ECG to a digital signal and a diode to truncate the analog negative parts. We used the peak and hold for dynamic digital gain. This was more successful than our dynamic analog gain although it still had a number of issues. The peak and hold capacitor had a slight leakage due to current bleeding through the transistor and our use of a stabilizing 10 KΩ resistor on our transistor gate in parallel with the 10 KΩ resistor attached to the drain of our transistor. This meant that in between beats, our maximum peak for the peak and hold decayed slightly, so we were not truly comparing our analog signal to ¾ of the max peak. Given more time, we would try other transistors to try to find a transistor that more easily turns off and on and would not require a stabilizing resistor on the gate.

Our analog gain did have some dynamic characteristics but was not dynamic enough to be able to scale all input signals to a 0 to 3.3V range as noted in the Results. Instead, we could use a 1mV-4mV input signal. The 4mV signal required us to increase the current limit of the DC voltage supply as our circuit was not stable enough to handle 4mV at the normal current limit. It also required a longer discharge of the capacitor on the peak and hold during reset. Ideally, we would add more stabilizing capacitors on our Op-Amps and change some of the chips to try to help stabilize the 4mV input more. We could not get our circuit to produce a strong enough signal from the 0.5mV input to be detected by the Arduino and accurately read even with dynamic gain. Our dynamic gain was in a non-inverting Op-Amp. The photoresistor formed the negative feedback and then we used a 1 MΩ resistor between ground and the inverting input. Therefore, our dynamic gain was $G=1+R_{photoresistor}/1\ M\Omega$. For the 0.5mV signal, the analog output was only scaled to about 0.5V. In order to improve our dynamic analog gain, we would try to include a DC offset so that we could use smaller resistors. We were not able to implement this due to the lack of additional power sources. We also wanted to use the power supplies to create diode clips so that we could clip signals that exceeded 3.3V to protect our Arduino. With the supplies we had, we could have tried to implement voltage division from the 12V and -12V supplies that powered our Op-Amps in order to get the desired voltage offset. We were also interested in investigating more linear dynamic analog gain approaches or more ways to control the nonlinearity of the photoresistor.

Next we created the software to match our hardware signals. Our software was more robust. The main issue we encountered in the software was missing beats for analog and accidentally triggering a beat or missing a beat due to noise for digital. We were able to reduce the losses on the digital measurement by throwing out measurements if the time difference was above that of a 35 bpm difference or below that of a 250 bpm difference. Most of the approaches we took to reducing losses of the analog signal did not work, but the analog still measured well because we tended to miss one beat at most and only a few times a minute. This meant our one-minute average was low, but the instantaneous was usually very accurate due to the exponentially weighted average.

With more time, we would try to implement a dynamic comparison in the code for both the digital and analog signals. If the next measured beat was too different from the previous measured rate by a certain amount such as 25% (the amount would be experimentally optimized), the measurement would be discarded or averaged with the previous measurement. This would be difficult to implement though during calibration as often our initial values were either infinity or 0, so we would need to check many edge cases. This also could limit how quickly our device could read heart rate changes. Currently, it can detect them very quickly for the instantaneous rate, so we would not want to sacrifice the speed of adjustment too much to get a more accurate measurement. A robust compromise would have to be experimentally determined.

We could also implement more filtering to our hardware signals before sending them to the Arduino. When testing the software with an ECG and square wave generated by the function generator, beats were almost never missed, and both the analog and digital measurements were highly accurate as the signals from the function generator were very clean. The noise in our actual signal seems to be the main contributing factor that causes us to miss beats.

Another change that could be made in the future would be to use pulse width modulation from one of the digital output pins to provide the threshold comparison value for our comparator instead of the peak and hold. We could use the threshold determined in the analog calibration of our software to be the threshold signal sent out through the digital pin. This would avoid using the entire peak and hold and ¾ gain section of our circuit. This would also provide a more robust comparison for our comparator than our decaying peak and hold signal. Reset would also be more easily implemented by simply re-calibrating the analog signal to find the new max and changing the PWM to match the new max. This would increase the accuracy of our digital minute and instantaneous average by reducing some of the noise.

Finally, we occasionally had to reload our program to our Arduino to clear the memory because recalibrating was no longer enough. With more time and Arduino experience, we would include code to clear variables in memory when we reset the device.

Overall, our instantaneous averages were more robust than our minute averages due to the controls we put in and the exponential averages. The digital instantaneous was more variable because it could vary both above and below the true heart rate, but it would not vary by as much as the analog instantaneous which would only vary below but by as much as 20 bpm for one reading. The minute averages for both were still fairly accurate for heart rates between 40-130 bpm, but often were off by up to 15 bpm for extremely high or low heart rates.

In summary, a second generation design would improve upon the existing dynamic gain, include additional power supplies to implement diode clippers, use a microcontroller that was able to sample the analog signal at a higher rate, implement dynamic comparison within the code, use pulse width modulation for the digital signal, and implement a more robust memory storage system for the Arduino.

A second generation design for clinical use would also isolate the circuit from the user controls using some sort of casing to protect the circuit. Audio alarms could also be implemented for high and low heart rates in addition to the LEDs using piezoelectric speakers or a different auditory output. With the auditory alarms would come a button to push to mute said alarms.

The second generation design would also include an input panel to allow the clinician to set their own parameters for the alarm functionality, as differently aged patients have different ranges for acceptable heart rates.

V. CONCLUSIONS

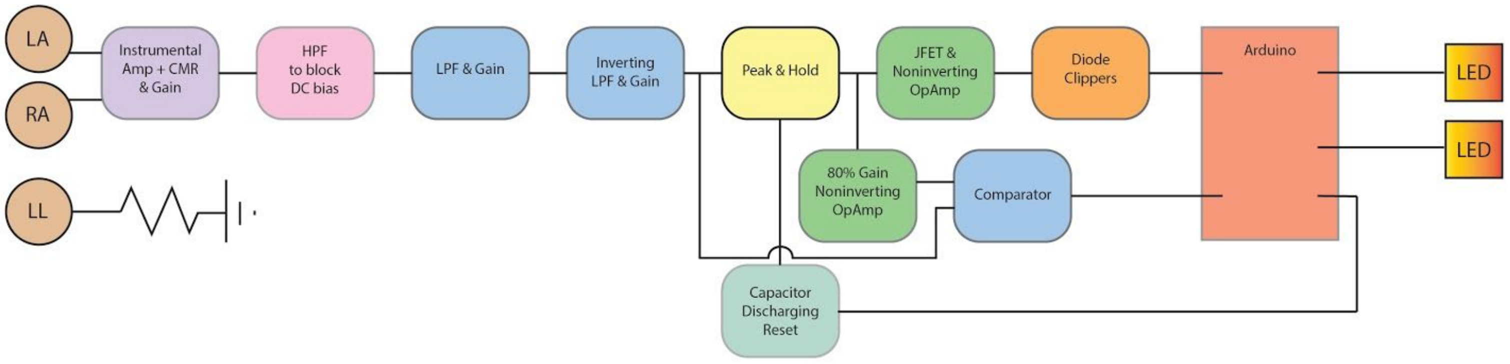
Overall, we successfully approached the design and implementation of the robust cardiac monitor as desired. We implemented hardware to create our desired digital and analog signals by combining simple processing stages. We wrote a modular code to calculate instantaneous and one minute average heart rate for both our digital and analog signals, and we addressed multiple edge cases to make our code more robust such as not recording digital signal reads that were not in the realistic heart rate range. We included reset and pause capabilities for our device, and provided multiple modes to display our information by including all four measurements on our LCD screen and by including LEDs to indicate the range of our heart rate and differences between the analog and digital signals. Although our analog signal would occasionally miss a beat and our digital signal would sometimes trigger for noise, our measurements overall were fairly accurate. If we were to implement improved designs of our device, we would change both the hardware and software to make cleaner signals and to be more robust to bad signal detections.

VI. APPENDIX

A. Equations

$$Instant\ HR = \sum_{n=1}^{5} \frac{60000}{t2(n)-t1(n)} exp(\frac{(n-5)}{2}) \qquad (1)$$

B. Figures and Tables



LA
RA
LL
Instrumental Amp + CMR & Gain
HPF to block DC bias
LPF & Gain
Inverting LPF & Gain
Peak & Hold
JFET & Noninverting OpAmp
Diode Clippers
Arduino
LED
LED
80% Gain Noninverting OpAmp
Comparator
Capacitor Discharging Reset

Figure 4: Hardware Block Diagram

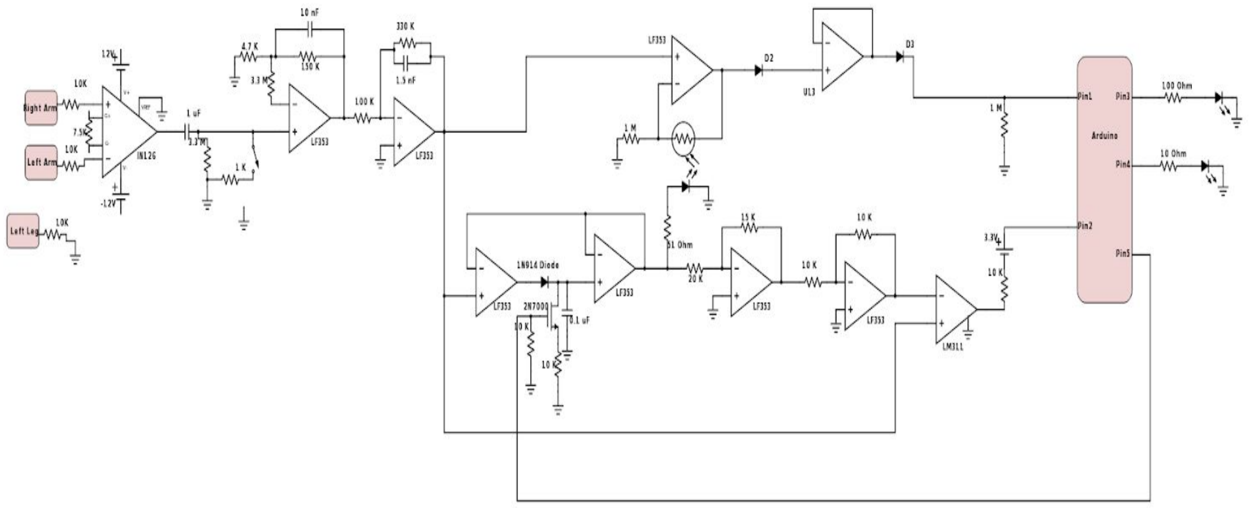


Figure 5: Circuit Schematic for Cardiac Monitor Hardware

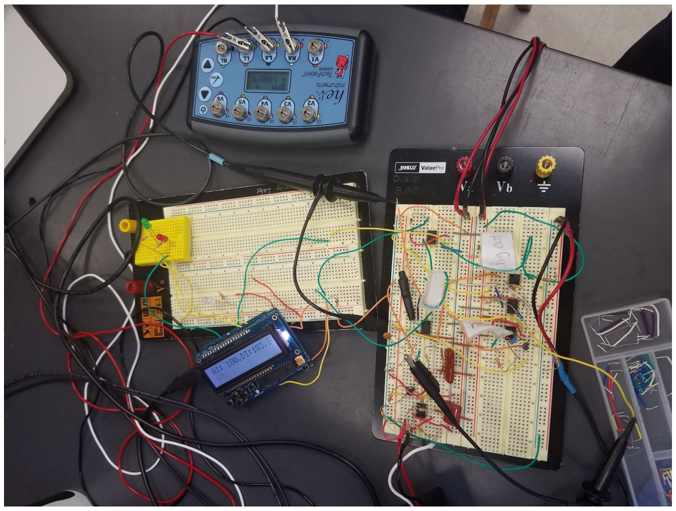


Figure 6: Picture of Operating Robust Cardiac Monitor

Table 2: Heart Rate Monitor with Varying Voltages

Voltage (mV)	Input (bpm)	Analog Inst.	Analog Minute	Digital Inst.	Digital Minute
4	86	86.2	86	86.16	86
2	86	86.2	86	86.35	86
1	86	86.6	86	86.63	86
0.5	86	86.2	86	87.11	86

Table 3: Heart Rate Monitor with Varying Rates at 2 mV

BPM	Analog Inst.	Analog Minute	Digital Inst.	Digital Minute
40	39.9	40	39.98	39
50	50	50	49	49
60	60	59	59	59
70	70.1	70	70.09	70
80	80	80	79	79
90	90.1	89	90.09	80
100	100	89	100.1	100
110	107	102	111.8	109
120	119	119	120.9	121
130	130	120	130.4	123
140	140	140	140.1	141
150	149	141	149.9	145
160	160	156	160.4	161
170	170	159	170.4	161
180	180	169	180.7	182

C. *Arduino Code*

```
// include all the libraries!
#include <Wire.h>
#include <math.h>
#include <Adafruit_RGBLCDShield.h>
#include <utility/Adafruit_MCP23017.h>
Adafruit_RGBLCDShield lcd =
Adafruit_RGBLCDShield();
#define WHITE 0x7

// declare pins
int digHR = 8; // digital input pin
int analogHR = A2; // analog input pin
int rangePin = 4; // range LED output
int blinkPin = 6; // blink LED output
int resetPin = 2; // reset output

//assign integer values
int digMin;
int maxPeak;
int track;
int anlgMin;

// assign floats (for math functions)
float digAvg;
float anlgAvg;
```

```
// declare boolean variables
boolean beat;
boolean change;
boolean pause;
boolean diff;
boolean aBeat;
boolean aChange;

//assign timers (longs)
unsigned long tstart;
unsigned long timer;
unsigned long timer2;
unsigned long timer3;
unsigned long atStart;


// assign time values (longs)
long t1end;
long t2end;
long tdiff;
long tdiffOld;
long at1end;
long at2end;
long atdiff;

// assign time vector to collect minute averages
long times[201];  // arduino is c based which
autoinitializes undeclared values to 0
long atimes[201];


void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600); // allow serial monitor time to
initialize
  lcd.clear(); //clear LCD

  // assign input or output for pins
  pinMode(digHR, INPUT);
  pinMode(analogHR, INPUT);
  pinMode(resetPin, OUTPUT);
  pinMode(blinkPin, OUTPUT);
  pinMode(rangePin, OUTPUT);

  // initialize booleans
  beat = false;
  aBeat = false;
  pause = true;
  diff = false;

  // run analog calibration code
  analogCal();

  // put in interrupts for digital signal
  attachInterrupt(digitalPinToInterrupt(digHR),
instantD, RISING);

  //LCD screen code for calibrating screen
  lcd.begin(16, 2);
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Calibrating");
  lcd.setCursor(0, 1);
  delay(1000);
  lcd.clear();
}

void loop() {
  uint8_t buttons = lcd.readButtons(); //initialize
buttons

  // pause function! *when pause is true - it's not
paused*
  if (pause) {
    if (beat == true) {
      t1end = (long) tstart; //sets start of heartbeat
    }
    if (beat == false) {
      t2end = (long) tstart; //sets end of heartbeat
    }
    tdiff = (long)t2end - t1end; // finds duration of 1
beat
    tdiff = abs(tdiff);
    timeUpdate(tdiff); // tdiff put into time vector
    digAvg = average(times); // This should take the
instantaneous digital HR
    digMin = minAvg(times); // Finds minute average
of digital
```

```
    instantA(); // runs instantaneous analog code

    if (millis() >= timer3 + 3000) { //every 3 seconds
update screen
      timer3 = millis();
      Print(); // go to print function! ~so modular~
    }
  }
  if (pause == false) { //if it is paused
    lcd.setCursor(0, 0);
    lcd.print("Dig:");
    lcd.setCursor(5, 0);
    lcd.print(digMin); //display dig minute
    lcd.setCursor(9, 0);
    lcd.print("Ana:");
    lcd.setCursor(13, 0);
    lcd.print(anlgMin); //display ana minute
    lcd.setCursor(0, 1);
    lcd.print("Paused");
  }
  if (buttons & BUTTON_UP) { //if up button pushed
    delay(500); // keeps one push of button from
toggling multiple times
    pause = !pause; // flip pause boolean (pause and
unpause)
    lcd.clear();
  }
  if (buttons & BUTTON_DOWN) { // this is the reset
code
    detachInterrupt(digitalPinToInterrupt(digHR)); //
detach interrupt
    for (int i = 0; i < 201; i++) { //resets all values to 0
      times[i] = 0;
      atimes[i] = 0;
    }
    digitalWrite(resetPin, HIGH); //outputs voltage to
MOSFET
    delay(1000);
    digitalWrite(resetPin, LOW); // stops outputting
voltage to MOSFET
    digitalWrite(blinkPin, LOW); // don't blink
    digitalWrite(rangePin, LOW); // reset range pin
    beat = false;// reset to initialized variables
    aBeat = false;
    pause = true;
    attachInterrupt(digitalPinToInterrupt(digHR),
instantD, RISING); // reset interrups
    lcd.begin(16, 2); //initialize LCD
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("Calibrating");
    lcd.setCursor(0, 1);
    analogCal(); // calibrate the analog signal
    delay(1000);
    lcd.clear();
  }

  if (anlgAvg > 40.0 && anlgAvg < 180.0) { //if in
range of 40-180, pin should be high
    digitalWrite(rangePin, HIGH);
  }

  if (anlgAvg > 180.0 || anlgAvg < 40.0) {
    digitalWrite(rangePin, LOW);
  }

  if (millis() >= timer2 + 250) { //every 250
milliseconds, check parameters for blinking
    timer2 = millis();
    Blink();
  }
}

void instantD() { //interrupt function -> flips boolean,
starts timer
  beat = !beat;
  tstart = millis();
  change = true;
}

void timeUpdate(long tdiff) { //updates times array
  if (change == true) {
    for (int i = 0; i < 201; i++) {
      times[i] = times[i + 1]; // this should shift the
whole array to the left by 1 for each time update
    }
    if (tdiff > 2000 || tdiff < 250) {
      times[200] = times[199];
    }
    else {
      times[200] = tdiff; // add in the newest value
    }
    change = false;
  }
}

float average(long times[]) { //average times
  float sum = 0; //clear the sum at the beginning of
each call to the function
  long ans;
  for (long i = 0; i < 5; i++) {
    ans = times[i + 196];
    sum = (60000 / (float)ans) * exp(((float)i - 4) / 2)
+ sum; // exponential average of the 5 most recent
values
  }
  sum = sum / 2.333;
  if (sum != INFINITY && sum > 250) { // if sum is
within bounds, sum is equal to digital avg
    sum = digAvg;
  }
  if (anlgAvg > sum && anlgAvg != INFINITY) {
    sum = digAvg;
//If analog average is greater than calculated
average, return previous digital average to avoid
too low reads from missed interrupts
  }
  return sum;
}

int minAvg(long times[]) { //find minute average
  long sum = 0; // clear sum every time
  int count = 0;
  for (long i = 200; i >= 0; i--) {
    sum = times[i] + sum;
    if (sum < 60000) { //if sum of times is less than
one minute, keep counting beats
      count++;
    }
  }
  return count; // return number of beats
}

void analogCal() { // analog calibration!
  while (timer < 5000) {
    timer = millis(); //set timer
    track = analogRead(analogHR); //analog read
vals

    if (track > maxPeak) { //find max val for analog
signal
      maxPeak = track;
    }
  }
}

void instantA() { //instantaneous analog values
  track = analogRead(analogHR); //reads signal
  int threshold = ceil(0.8 * maxPeak); //sets
threshold
  if (track >= threshold) {
    aBeat = !aBeat; // flip a boolean
    atStart = millis(); //set a time
    aChange = true;
    if (aBeat == true) {
      at1end = (long) atStart; //sets start of beat 1
    }
    if (aBeat == false) {
      at2end = (long) atStart; //sets start of beat 2
    }
    atdiff = at2end - at1end; //finds time between
beats (length of 1 beat)
    atdiff = abs(atdiff);
    analogUpdate(atdiff); //adds it to array
    anlgAvg = anlgAverage(atimes); //find
instantaneous
    anlgMin = aminAvg(atimes); // find minute
    delay(57);// delays long enough that 1 heartbeat
doesn't read as multiple heartbeats
  }
}

void analogUpdate(long atdiff) { //adds time of
heart beat to array

  if (aChange == true) { //if a beat has occurred
```

```
    for (int i = 0; i < 201; i++) {
      atimes[i] = atimes[i + 1]; // this should shift the
whole array to the left by 1 for each time update
    }
    atimes[200] = atdiff; // this should add in the
newest value
    aChange = false; //ssets aChange back to false
(will flip when next beat occurs)
  }
}

float anlgAverage(long atimes[]) { //find
instantaneous analog HR
  float sum = 0; //clear the sum at the beginning of
each call to the function
  long ans;
  for (long i = 0; i < 5; i++) {
    ans = atimes[i + 196];
    sum = (60000 / (float)ans) * exp(((float)i - 4) / 2)
+ sum; /// exponential average of the 5 most recent
values
  }
  sum = sum / 2.333; //divides by exponential
'weights'

  return sum;
}

int aminAvg(long atimes[]) { //find minute average
analog HR
  long sum = 0; // clear sum every time
  int count = 0;
  for (long i = 200; i >= 0; i--) {
    sum = atimes[i] + sum;
    if (sum < 60000) { //if sum of times for hearbeat
is less than a minute, add another heartbeat
      count++;
    }
  }
  return count;
}

void Print() { //print all the values!
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("AI:");
  lcd.setCursor(4, 0);
  lcd.print(anlgAvg);
  lcd.setCursor(8, 0);
  lcd.print("DI:");
  lcd.setCursor(11, 0);
  lcd.print(digAvg);
  lcd.setCursor(0, 1);
  lcd.print("AM:");
  lcd.setCursor(4, 1);
  lcd.print(anlgMin);
  lcd.setCursor(8, 1);
  lcd.print("DM:");
  lcd.setCursor(11, 1);
  lcd.print(digMin);
}

void Blink() {
  float test1;
  float test2;
  //check if > 10% diff between dig and ana
  test1 = abs((digAvg - anlgAvg) / digAvg * 100);
  test2 = abs((digAvg - anlgAvg) / anlgAvg * 100);
  if (test1 > 10 || test2 > 10) { //if there is a >10%
    diff = !diff;
    // flip boolean -> running this function every 250
ms will lead to blinking as it alternates high and low
    if (diff == true) {
      digitalWrite(blinkPin, HIGH);
    }
    if (diff == false) {
      digitalWrite(blinkPin, LOW);
    }
  }
  else { // <10% difference
    digitalWrite(blinkPin, LOW); //pin should not blink
  }

}
```

D. *User Manual*

1) Attach ISO-Switch leads to Cardiac Simulator or electrodes on a patient.
2) Press the 'Down' button to calibrate the device. A message "Calibrating…" will appear.
3) After approximately 5 seconds, the screen will display "AI" in the top left corner with the Analog Instantaneous heart rate in beats per minute after it. It will display "DI" in the top right corner with the Digital Instantaneous heart rate in beats per minute next to it. It will display "AM" in the bottom left corner with the Analog Minute Average heart rate next to it in beats per minute. In the bottom right corner, it will display "DM" with the Digital Minute Average heart rate next to it in beats per minute.

If you wish to pause measurement:
1) Press the 'Up' button. This will stop all measurements and display the Analog Minute Average in the left corner and Digital Minute Average in the right corner.
2) To resume measurement, press the 'Up' button once more.

The device will illuminate a red LED if the heart rate is between 40 bpm and 180 bpm. If the analog and digital instantaneous heart rate differ by more than 10%, a green LED will blink at 2 Hz until they are in range once more.

If you wish to switch to another user:
1) Detach leads from current user.
2) Attach to next user or change settings on simulator.
3) Press the 'Down' button and hold for about 1 second.
4) The device will now calibrate to the new user.

E. Troubleshooting Guide

If it's giving inf for digital instantaneous or analog instantaneous, your best best is pressing the calibration button. If calibration doesn't work, try re-uploading code. If that still doesn't work, you may need to look at the magnitude of the signal being inputted and adjust the gain by changing out a resistor in the initial ECG circuit, likely the gain resistor in the INA126.

Common Issues:

Full Memory-

If the Analog measurements fail to update or will not change from 'inf' after calibration, the Arduino memory is full. Simply disconnect the Arduino from power and then reconnect it. You can also resend the code to the Arduino device to clear memory as well.

Calibration too Short-

If the DC power supply drops from the desired voltages (± 12V, +3.3V) when a new user is connected, the input voltage of their ECG signal is too high. By pressing and holding the calibration longer in order to discharge the capacitor longer, the power supply will return to normal. If it still does not, you may need to increase the current limit on the power supply.

REFERENCES

[1] Duke BME 354 Lab Protocol: "ECG I, Biopotential Amplifier", Spring, 2017.