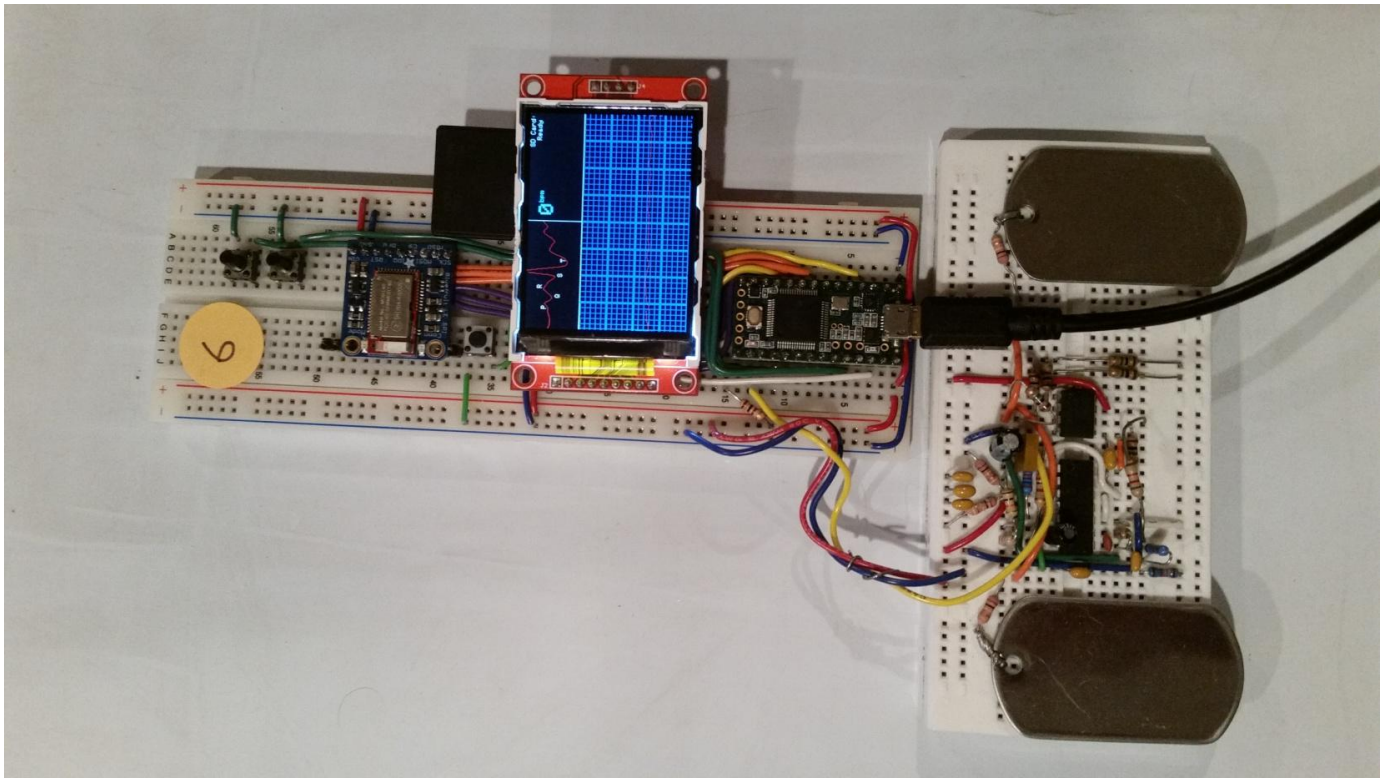
Heart Rate Monitor

Dec 2015

Alex Vassallo



1 Abstract

This project was created as a final project for CSE 466 Embedded System class at the University of Washington. The goal of the assignment was to detect the QRS complex of a heartbeat, display it to the screen, and write it to the SD card. The class was provided a Teensy 3.1 micro-controller and a schematic for a front end circuit used to capture and amplify the differential voltage across a pair of electrode leads. When the circuit was originally built and tested it performed very poorly due to excessive noise. Prior experience with signal conditioning led me to find circuit changes which allowed superior performance. The software design included additional digital filtering, heartbeat detectors, options menu, and a calibrated display of the data including calculated heartbeat characteristics.

2 Hardware

2.1 Provided Schematic

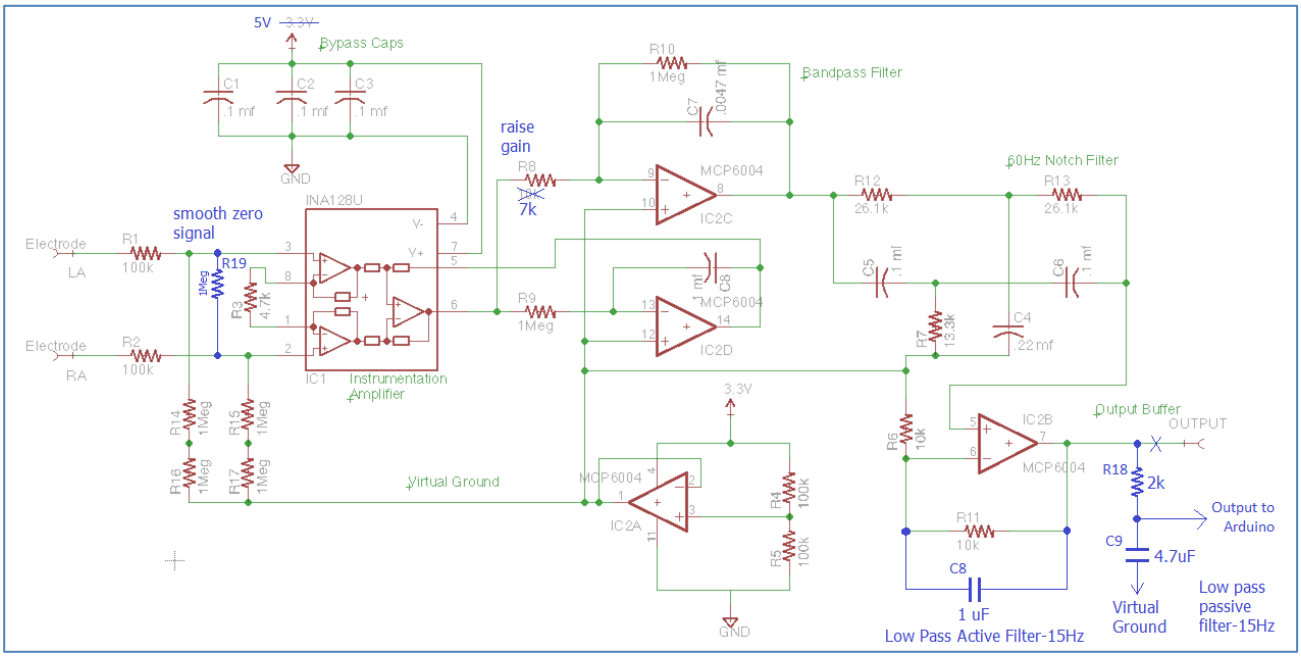


Figure 1: Schematic

Modification 1:

The amount of noise around 60Hz was extremely high. For this project, we only wanted signals up to about 10Hz. To help with this, I added two low pass filters shown in Blue within Figure 1. An active low pass filter was created by adding C8 to the final op-amp feedback path. This can be done since it outputs directly into a high impedance input on the micro-controller. The 2nd low pass filter was created by adding R18 and C9 to the previous output. Using the equation 1/(2*Pi*R*C), I was able to calculate the proper R and C values to achieve a -3db roll off frequency around 15Hz to attenuate the signal.

Modification 2:

When the electrodes are not being contacted by anything, the amount of ambient noise across them is enough to trigger the rhythm detector. To resolve this undesired effect, a 1 Mega-ohm resistor was placed across the inputs of the instrumentation amplifier. However, this reduced the amplitude of the signal, so the value of R8 was lowered to restore the previous gain level.

Modification 3:

The IC chips used were the BB INA128U Low-Power Instrumentation Amplifier and the Microchip MCP6004 Quad Low-Power Op-amp. Upon review of their datasheets, it was discovered that those parts performed much better at 5V instead of 3.3V, so the source voltage was moved to the 5V USB supply.

2.2 System Schematic

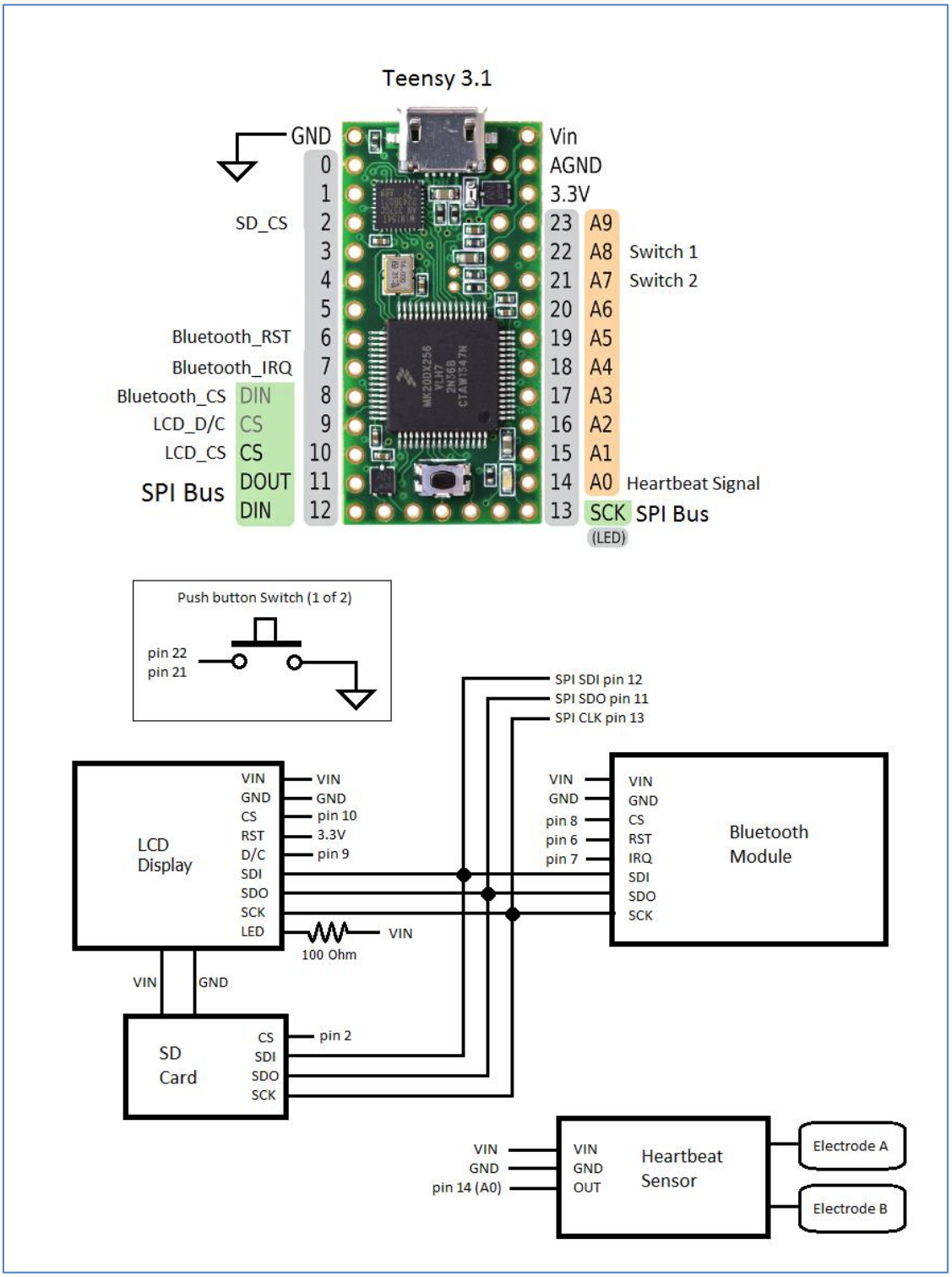


Figure 2: Schematic

3 Software

3.1 Data Flow

In my design, I used the onboard Programmable Delay Block (PDB) as a timer source for capturing data. The PDB triggered the Analog to Digital Converter (ADC) to capture a data point, and upon ADC conversion completion, the ADC triggered the Direct Memory Access (DMA) to copy that 16 bit value directly into our array. The DMA was setup to trigger a software interrupt when 200 points of data has been captured, which is when we rotate our array indexes to perform different steps of operations. This allows each main loop in software sufficient time to perform calculations or interact with peripherals. As Figure 3 indicates, the DMA target is pointed to overwrite the oldest block of data, the Digital Filter is pointed at the sample that DMA has just completed filling, and the Rhythm Detector is pointed at the block of data that the filter has just processed. The blank block is necessary to preserve data that may be needed when capturing a complete heartbeat rhythm into an array.

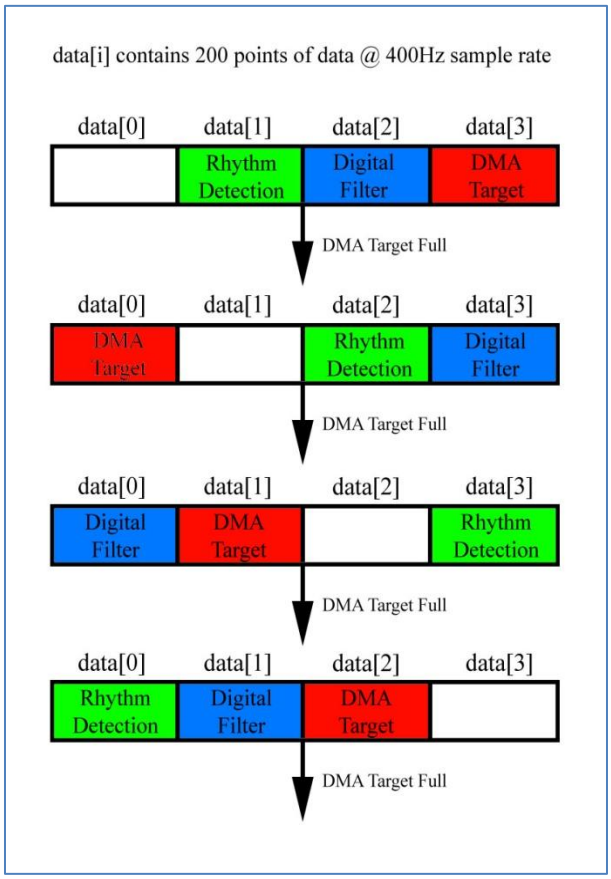


Figure 3: Data Flow

If an R peak is detected within a sample block, the arrays are scanned backwards 119 points of data, and forward 200 points of data to capture the entire rhythm into a known format, The rhythm is copied into a new array of length 320, with the R peak aligned at position 120, where it can be stacked with other confirmed heartbeat rhythms for further analysis over time.

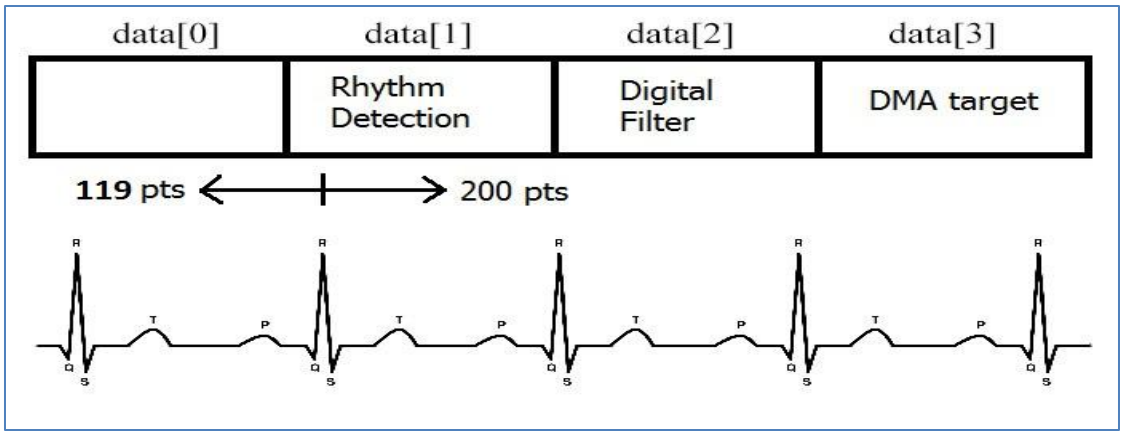


Figure 4: Rhythm Collection

3.2 Digital Filter

The digital filter is a simple design which uses the average slope to determine which direction the line is heading. By excluding the point that we are attempting to modify, any large noise that the single point produces is not included in the calculation. This results in a very smooth line that we can use for finding the derivative of the data plot.

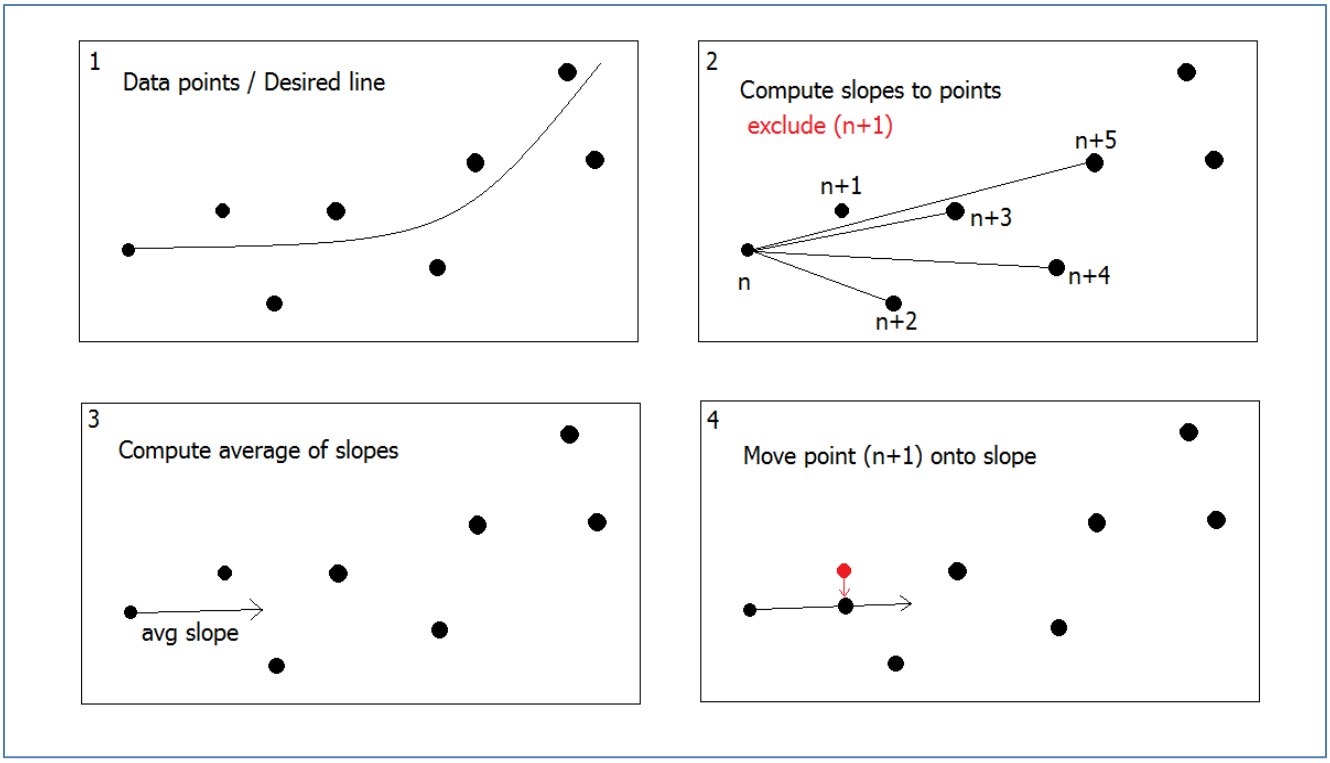


Figure 5: Digital Filter

3.3 Rhythm Detection:

During the filter stage, the calculated slopes are stored as an additional element in the array. A bitmap is also created where a 0 is stored normally, but a 1 is stored if the slope transitions from positive to negative, or negative to positive. This captures all of our points of interest in the potential heartbeat rhythm, and no calculations are required from this point forward. When attempting to pass a segment of data through our heartbeat detection, we can quickly scan our bitmap to find the associated values of data.

To locate a heartbeat, we scan our bitmap for the first value of 1, and assign it as point T in our rhythm. We then go backwards through the data samples, assigning P,Q,R, and S to the previous points of interest by using the associated bitmaps. If the group of points does not meet the requirements, then we slide all of our points over to the next point of interest and check again. This is very simple to do since we can assign P=Q, Q=R, R=S, S=T, and T= the next point. In parallel with this operation, we are also calculating the area under the curve between each point in our derivative data. Since our data has a finite amount of points, this is accomplished by simply summing up the derivate values as we progress across, and carrying through those values when we assign P=Q, Q=R, etc.

For the detection algorithm, I used a simple matching algorithm. Using the derivate data we check for the following:

1. *The area under the curve between points Q to R is above the limit.*

2. *The negative area under the curve between points R to S is above the limit.*

3. *The value of Point Q is roughly the average value of the data.*

4. *The distance between points Q and S is within the human limits.*

If so, we capture it as a rhythm and use point R as our zero point. Then to locate points P and T, we first check the bitmap for those points, and if they are not a great enough distance from the QRS complex, we move along the bitmap until we find a point that is within the bounds expected of the human heart.

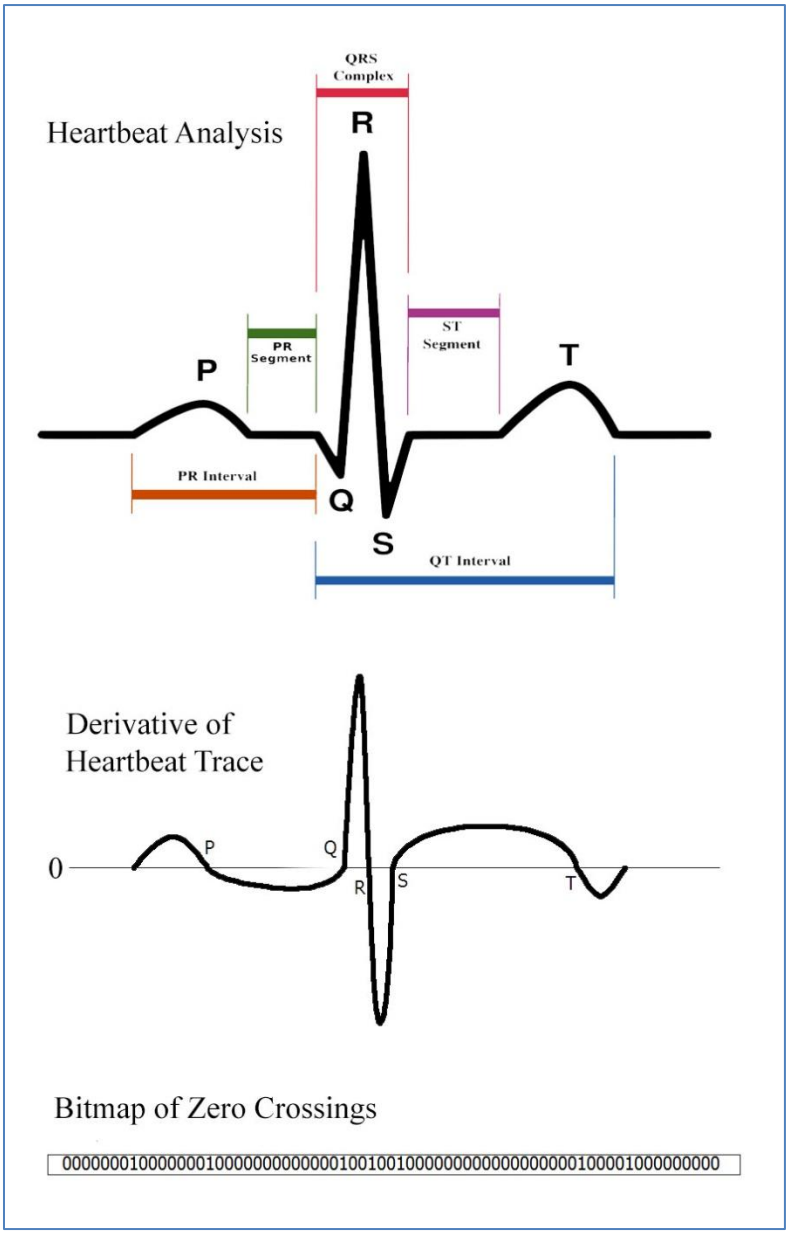
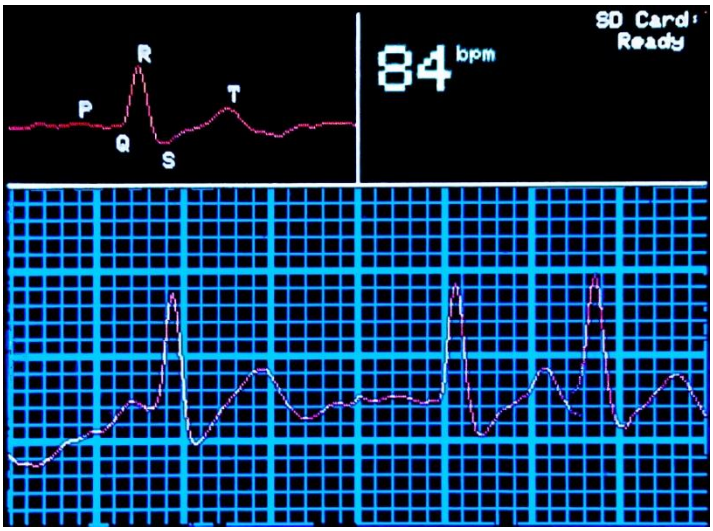
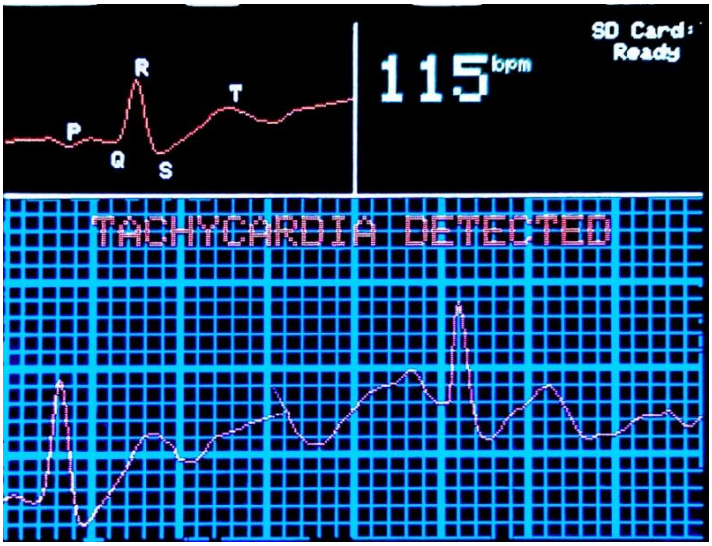


Figure 6: Rhythm Detection

3.4 Graphical User Interface

The Graphical User Interface displays a heartbeat trace grid, captured rhythm area, status information, and an optional menu. The heartbeat trace has been calibrated to the specification of 1 vertical line per 1 mV and 1 horizontal line per 40 ms, which is maintained as it automatically zooms in and out. The bpm is calculated from averaging only concurrent rhythms. Any detected heart conditions are displayed inside of the trace grid.





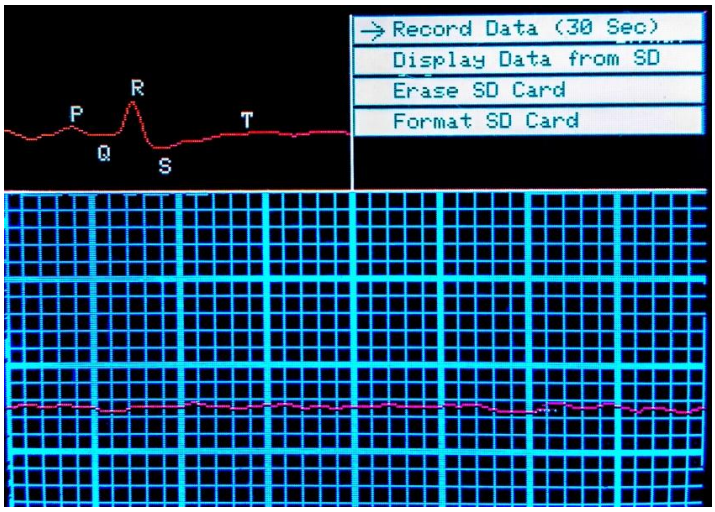
Menu

Record Data (30 Sec):

Selecting "Record Data" will attempt to record 30 seconds of heartbeat data onto the SD card. It will only record while a valid rhythm is detected. Selecting "Stop Recording" from the menu before 30 seconds will close the recording session and save the data collected.

Display Data from SD:

When selecting "Display Data" from the menu, the software will open each file on the SD card sequentially and display a full screen of data at a time. Pressing the "select" button will cycle through the next screen of data, until all files have been read. Selecting "Live Data" from the menu will switch out of this mode and back to displaying live data.



Erase SD card: This feature will erase all files on the SD card.

Format SD card: This feature will format the installed SD Card with a FAT32 format.

3.5 Software Outline

Main Loop()

- Draw Graphics *allows live trace graphics to update between stages*
- Switch on (stage++)
  - ❖ case 0: filterData() Run the digital filter on the captured data
  - ❖ case 1: scanForQRS() Scan for QRS complex in the previously filtered data
  - ❖ case 2: drawRhythm Draw the captured rhythm to the screen
  - ❖ case 3: checkButtons() Check button flags and adjust menu navigation
  - ❖ case 4: drawMenu Draw the current menu to the screen
  - ❖ case 5: writeToSD() Write data to SD card if selected
  - ❖ case 6: sendBlueTooth() Send data to bluetooth module for broadcast.

filterData(n)

For each point in the selected data (offset by n)...

- Find the value of the point
- Find the memory address of the next point of data
- Find the memory address to write slope value
- Find the memory address to write bitmap value
- Calculate slope to n points of data, and get average
- Set bitmap value to 1 if slope has changed polarity
- Use slope to overwrite next point of data
- Write slope and bitmap value to memory addresses

scanForQRS()

For each marked value in the bitmap data...

- P=Q, Q=R, R=S, S=T for value, integral area, and width
- T = new point associated with bitmap
- check P, Q, R, S, T against rules, if pass then captureRhythm(data position of R)
- build rhythm object from formatted heartbeat data
- calculate average rhythm data
- check rhythm object against known heart conditions and display any found.

CheckButtons()

- If button 1 is pressed, rotate the menu selection
- If button 2 is pressed, switch on menu selection
  - ❖ case -1: No menu open, if 'Display Data' flag set then [readData() + drawData()]
  - ❖ case 0: Record Data, toggle write flag, either 'start new file' or 'close current file'
  - ❖ case 1: Display Data, clear screen, toggle flag
    If flag is true then [openFile() + readData() + drawData()]
  - ❖ case 2: Erase SD Card, eraseCard()
  - ❖ case 3: Format SD Card, formatCard()

Interrupt Routine for Buttons (1 of 2)

- De-bounce input
- Set button flags
- Force redraw of menu