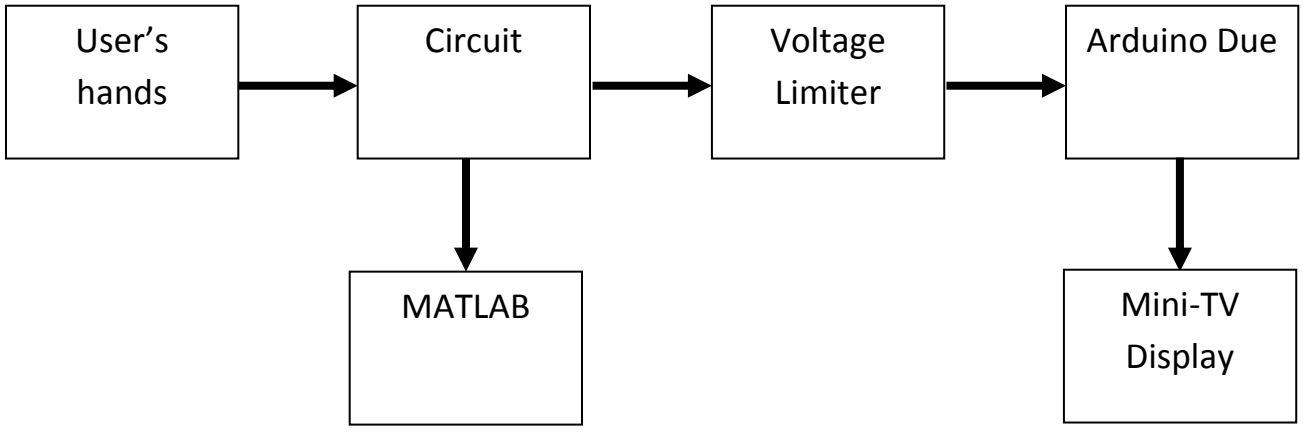
Hands On ECG

Sean Hubber and Crystal Lu

The device. The black box contains the circuit and microcontroller, the mini tv is set on top, the bars on the sides are for holding it and reading hand voltage, and the power switch is on the front along with the output jack.

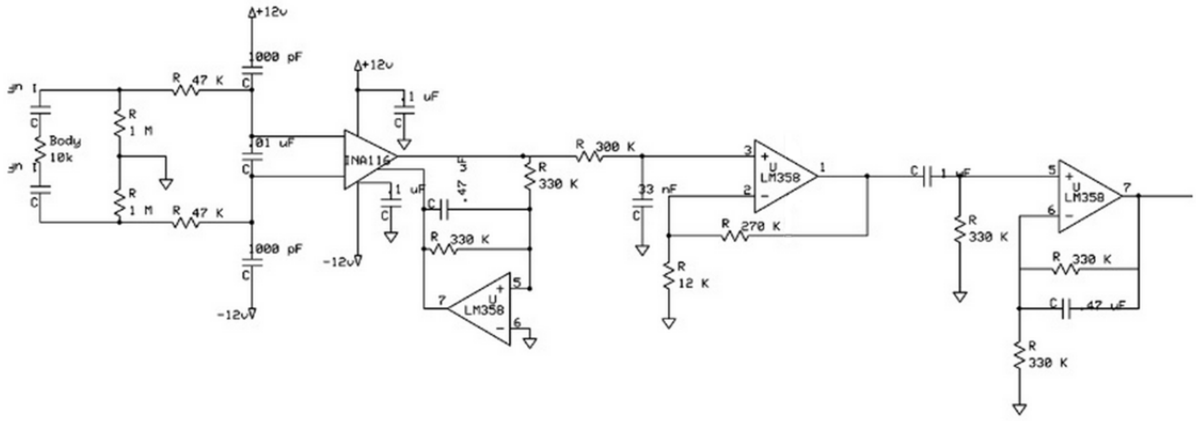
In Cornell's ECE 5030 Bioinstrumentation course, we designed and built an ECG system that extracted the electrical output of the heart. This hardware in combination with a Matlab function could be used to accurately measure a person's heart rate. From this starting point, we developed a battery operated, packaged product using an Arduino Due that provides users with a safe and easy way to view the waveform of their own heart on a mini-TV screen. This paper will detail the underlying circuitry, the Matlab function we used, and the use of the Arduino and mini-TV to display the waveform. The following block diagram illustrates the main components of the overall system.



General block diagram of the system.

Circuit

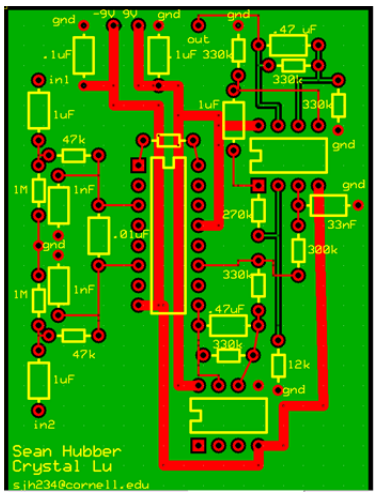
The main circuit for this system is separated into several stages in order to ensure that retrieving the signal would be safe for the user and that sufficient amplification could be made to pull a readable ECG signal. The first stage is the conditioning stage, which ensures user safety through DC-isolation by connecting the dry electrode signals directly to capacitors and resistors initially. The capacitors help with DC-isolation and provide a DC offset correction while the resistors limit the current passing through. This input conditioning stage is followed by amplification and filtering that yield an output with a high signal-to-noise ratio. After the circuit block, the signal is used by both the MATLAB and voltage limiter blocks.



The schematic above is of the entire circuit block. It contains the conditioning stage, differential amplifier, and filtering stages

Directly after DC isolation, the signal is sent into a differential amplifier, the INA116, and with an RG value of 1 kOhm, an initial gain of 51 is obtained. The INA116 has a low bias current, which allows for the high impedance signal source. The differential amplifier also utilizes a feedback loop which prevents it from saturating.

Following the differentiation stage, the signal is passed through multiple filters and receives additional amplification. The first is a low pass filter with a cutoff frequency of approximately16 Hz, this filter is primarily used to eliminate 60 Hz noise. The second filter is a high pass filter with a cutoff frequency of approximately .5 Hz. This filter is primarily used to eliminate DC offset. The total amplification at this stage is 10, and since the noise is now significantly reduced and the signal to noise ratio is large, this amplification produces a very strong and clear signal. With these stages done, the signal is now strong enough to be digitally analyzed. This concludes the circuit block, and now the signal travels to both the MATLAB and voltage limiter blocks.



The entirety of the circuit block (conditioning stage, differential amplifier, and filters) was built on a board designed in PCB Express.

MATLAB

In order to use MATLAB on the signal produced by the circuit you must first connect the device to the computer. The packaged product contains a phone jack that is connected to the output of the circuit block. We used this jack to feed the signal to the computer using a NIDAQ. Once the device is connected the MATLAB program can be used to analyze the signal.

The program starts by running a few initialization steps before entering the data reading and display section. These initialization steps include initializing variables used in the program, defining the input device, setting up the GUI, initializing the plots, and initializing data collection. Most of the variables are set to values that give the best results, while others are just initialized to 0 or empty vectors. It is important to note that the bpmstaticprev (1/sec between the previous two beats) and bpmstring (location in time of all previous beats) variables are initialized to 80. This tells the program that initially heart rates near 80 are reasonable. If the subject has a heart rate much less than or much greater than 80 the program will re-adjust accordingly.

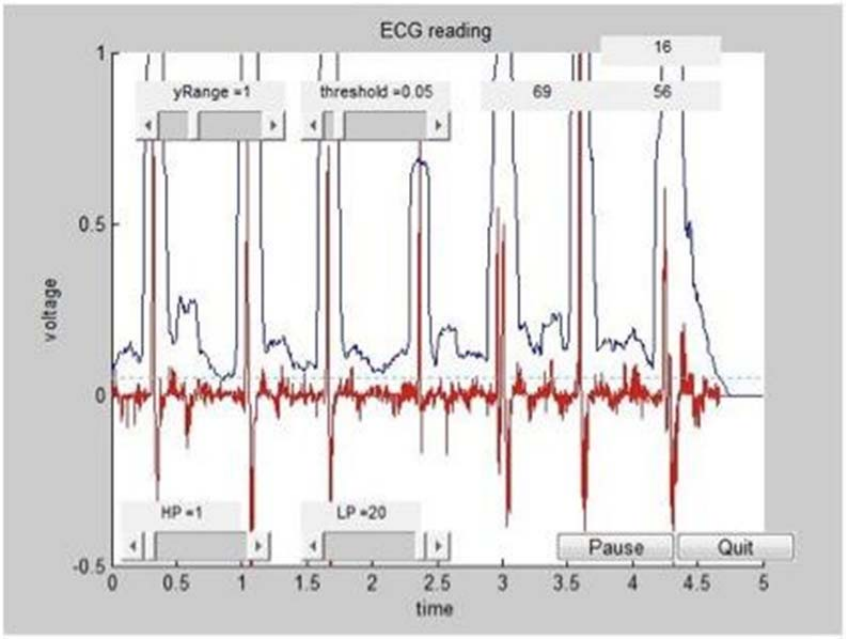
Once the GUI is initialized, the plots of the ECG waveform, Pan-Tompkins plot, and threshold line are initialized. The Pan-Tompkins plot is a processed version of the original signal that is much easier to threshold. In the last step before data collection and the display begin, the program begins continuously sampling the input device. The program initially pauses for one second so that enough data can be collected to build the first plot. After this the program begins collecting and displaying data and results. Until the user presses the Quit button, the program continuously loops through a while loop collecting and displaying data. There are several important steps in the loop. The first step is sampling the data from the input device; once this sample is captured it is placed at the end of the vector of values to be displayed. This vector is then shifted to the left, dropping the number of values equal to the size of the sample that were at the front. The vectorized signal then passes through a low and a high pass filter; the cutoff frequencies of these filters are settable in the GUI. The filtering process produces transients, and these transients are removed so they do not interfere with beats per minute (BPM)

calculations. Finally, the vector is multiplied by 10 times the absolute value of itself, a manual amplification. This amplification makes the signal more visible on the plot, allowing for easier threshold adjustments.

In order to determine the location in time of the beats, the program takes the processed signal and runs the Pan-Tompkins algorithm on it. This requires two simple steps. The first is to square the signal so it consists of only positive values and the second is to convolve it with a rectangle. The result of this process is a much smoother waveform that can be used to determine the locations of each of the heart beats. The user sets the threshold to find the rising edge of each beat. It is important that this threshold be high enough so that only the rising edges of real beats will cross it, but low enough so that “weaker” beats will still reach it. Once the locations of the beats are determined we run them through our heart rate calculating algorithm to determine the heart rate in BPM.

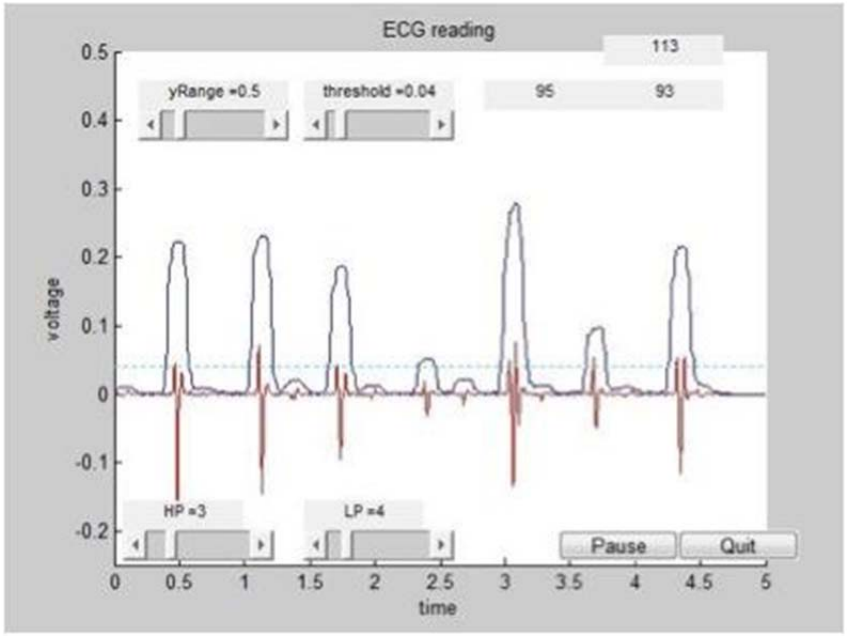
The heart rate calculating algorithm was by far the most difficult piece of the program to write. The algorithm is run only once every .5 seconds, or every fifth time through the while loop. The first step is to compute bpmstatic from the period of the two most recent beats. Once this is determined it is compared to the most recent valid heart rate calculation (stored as bpmstring(end)). If it is near enough to this value (heart rates do not change rapidly) then it is considered valid and is added to the bpmstring vector. The bpmstring vector contains all “valid” heart rate readings. If the new bpmstatic value is not close enough to the previous value then the last value in bpmstring is taken as the heart rate and re-added to the bpmstring vector. Finally, if the bpm reading is outside the range [30,180], it is tossed. Once a “valid” heart rate has been determined and added to the bpmstring vector, the outputted value for heart rate is calculated by averaging the last twenty values in bpmstring. The number of values used can be increased to give a more stable reading or decreased to give a reading more responsive to rapid heart rate changes. Finally, the heart rate reading is displayed on the GUI.

Once the heart rate has been determined, the program plots the updated versions of the ECG waveform, the Pan-Tompkins waveform, and the threshold waveform. The user can suppress this update by pressing the pause button and can resume real time display by pressing it again. The program then updates the current time (rounded to the nearest second) during which the program has been running.



The figure above shows the MATLAB display with no filter adjustment. The red is the filtered waveform signal, the blue is the Pan-Tompkins waveform, and the dotted teal line is the threshold.

The filter adjustments built into the GUI allow the user to customize the filter cutoffs themselves. After adjusting the filters, the signal from the same person as above now looks like this:



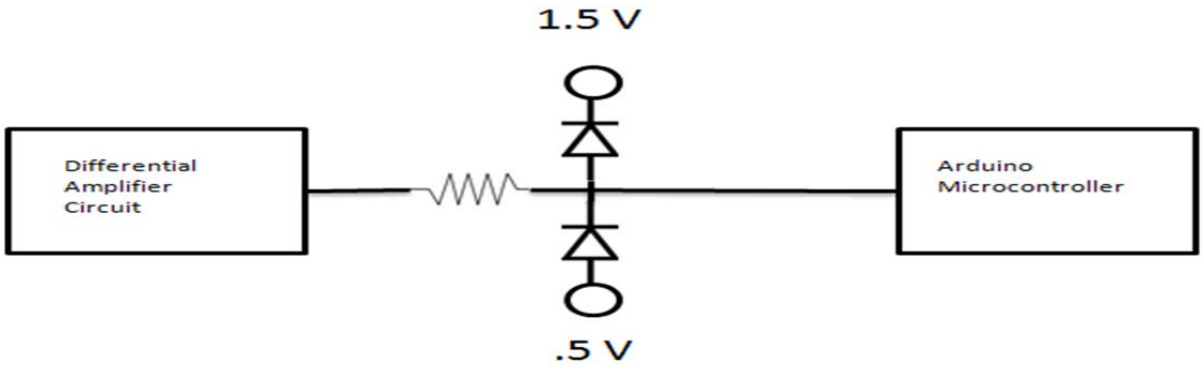
After adjusting the filter cutoff frequencies the signal is much cleaner.

Finally, index is increased by the size of the sample so that timekeeping is consistent. There is a pause so that each loop takes exactly 100 ms (this value is determined by the variable realtime, which also determines the size of the sample). The program will continue

looping through this process until the user presses the Quit button, at which time the program exits the loop, closes all figures, and deletes the input device.

Voltage Limiter

The output of the circuit block can have values anywhere between -9V and 9V. The Arduino microcontroller only accepts a limited range of inputs. After rigorous testing, we determined that the Arduino would crash when given an input voltage outside the range of .5-1.5 volts. The voltage limiter uses two diodes to limit the voltage, one leading to a 1.5V node, the other leading from a .5V node:



Voltage limiter schematic.

This prevents any voltages above 1.5V or below .5V from reaching the Arduino. The resistor in the circuit limits the current that is seen by the Arduino. Following the block diagram, the output of the voltage limiter block becomes an input to the Aruino Due block.

Arduino

The Arduino Due was used to convert the heartbeat waveform to an NTSC signal that could be used by a mini-TV. The Arduino continuously sampled the input provided by the voltage limiter at 240 samples per second. Similar to MATLAB, the vectorized signal was shifted left to make room at the end for the most recent sample. This allowed for a continuous real time display of the incoming signal. Each frame outputted to the mini-TV contains two waveforms. One has a screen width of 1s and the other has a screen width of 5s. This allows the user to see both a standard version (5s) and a more zoomed in version (1s). Each frame also contains an integer representing the elapsed time of the program. This code was produced by Bruce Land.

Mini-TV Display

Finally we've reached the end of the block diagram. Here, the mini-TV receives a signal from the Arduino that tells it what to display. In accordance with the due VGA library read me, the Arduino was connected to the NTSC TV like this:

```
Due pin 36 -> 3k3  resistor -> Video In
Due pin 37 -> 1k6  resistor -> Video In
Due pin 38 -> 820R resistor -> Video In
Due pin 39 -> 390R resistor -> Video In
Due pin 40 -> 200R resistor -> Video In
Due pin 41 -> 100R resistor -> Video In
Due pin GND -> Video GND
```

The resistors are connected to Video In via a 100 uF capacitor that provides AC coupling.

Results

The final product of this project was a mobile display of one's heartbeat. Packaged into a box, there is a switch on the bottom to turn the system on and off. The TV is planted on the top of the box while the handlebars are on either side for the user to easily switch the device on and then place his or her hands over the bars to see a heart reading on the screen.

We were unable to accurately compute a person's heart rate on the Arduino Due due to the limited voltage range on the input and substantial baseline wander in most peoples' signals. To make up for this, the final product contains an output port that is connected to the output of the differential amplification circuit, before it is voltage limited. This port can be used to sample the signal into MATLAB, where a program we wrote can very accurately compute BPM and the user can further view and manipulate the signal.

The device in action. The top line (green) has a one second time base while the bottom line (yellow) has a five second time base.