Respiratory Rate as a Clinical Indicator (December 2015)

Zachary D. Weeden

***Abstract*— In this project, the detection of respiratory issues was the goal. The system was designed to monitor breathing rate and to sound an alarm if the breathing rate detected was abnormal. Values from the *Archives of Disease in Childhood 1993*, were taken and used to determine the boundaries of what was deemed as a normal breathing rate. In this project, FIR filters, low pass filters and high pass filters were made use of and allowed for proper filtering of which breathing rates fell in and out of range. Later discussed will be the actual implementation of the design. Matlab software was also used in the design phase and provided a visually pleasing way of determining the actual boundaries of each filter. With the Arduino Uno platform, both simulated and sampled data was tested. With these results and the use of excel, many measurements were plotted and analyzed to determine whether the system was functioning correctly or not.**

***Index Terms*—** *Digital Filters, Low-Pass Filters, High-Pass Filters, FIR Filters, Matlab, Excel Breathing Rate*

I. Introduction

Before delving into the intricacies of the implementation it should be noted that the LM61 temperature sensor was the device used to carry out all measurements and sampling, more particularly, the SOT23. Because the casing of the temperature sensor acted as an insulator, proper cool off was obstructed by the retention of heat in the material. The SOT23 surface mount was used instead and properly wired. Further tuning of the system will be discussed later on. The system sampled readings from the temperature sensor, removed noise to “extract” a more precise, fine reading of the signal and make a decision as to whether the breathing rate fell into the acceptable breathing rate zone. Although the actual breathing rate wasn't sampled, the temperature readings from the sensor could allow the inference of the actual breathing pattern of the patient. In the design, several filters were used to provide a 'window' for normal breathing rates. Anything outside of this window was deemed abnormal and sounded an alarm, whether the breathing rate was too slow or too high. The specifics of the decision making will also be discussed at a later time. The concept of dithering the raw signal was introduced and even further improved with averaged dithering. This reduced the quantization noise and allowed for a finer reading of the signal and provided a proper extraction of the data presented by the sampling. As referenced before, normal breathing rates were determined to fall within the range of 12 to 39 breathes per minute. Breathing rates below 12 breaths per minute, bradypnea, sounded a lower in tone and lower frequency alarm whereas, 40 breaths per minute, tachypnea, a key indicator of pneumonia, would set off a higher pitch and higher frequency alarm.

II. Filtering, Signal Conditioning, and Design

Throughout the course of the project, filtering played a major role in the success of the system. Filtering made the window of acceptable breathing rates possible and what also allowed the determination of whether an alarm as needed to be sounded. The filter system was comprised of low pass and high pass filters, both of which were designed to pass frequencies that fell outside of the normal breathing rate zone. These filters established the lower and upper bounds of the band-pass filters. The actual designing of these filters was completed in Matlab. The Checvycheb function was used along with the zero pole diagram and bode plots for each of the filters. The low pass filter was a 6 pole filter with a cut frequency of 11 breaths per minute and the high pass filter was a 14 pole filter with 40 breaths per minute as the cut frequency. The actual design phase was expedited with the modification of variables in Matlab. This allowed for quick redesign after testing with any produced filter. Coefficients for the designed filters were generated and could be easily implemented in filter function in the Arduino IDE. In Matlab. Bode plots provided visual proof for a working filter.

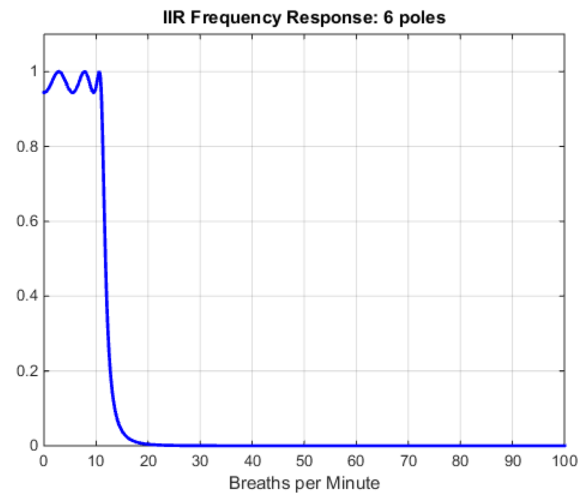

Figure 1: Frequency Response of Low Pass filter– This bode plot is the frequency response of the designed low pass filter. It shows the corner frequency around 11bpm and the ripple in the pass band. The sharp cut is an

indication that the filter had the resolution to differentiate between consecutive breaths per minute

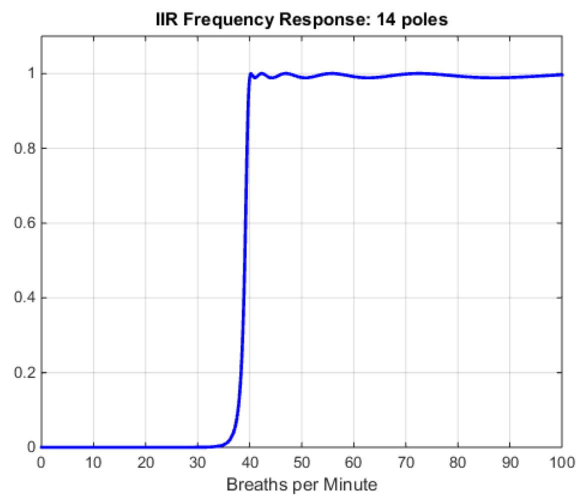

Figure 2: Frequency Response of High Pass filter– This bode plot is the frequency response of the designed high pass filter. It shows the corner frequency around 39bpm and slight ripple in the pass band. This is due to the number of poles. The sharp cut is an indication that the filter had the resolution to differentiate between consecutive breaths per minute. This, paired with the design from Figure 1, passed frequencies between, 0-11, and 40 and above.

These two filters were then put the test to see if they properly filtered the unwanted frequencies. Simulated data was produced in a sweep-like fashion where signal was partitioned into quintiles to fully test the response of the system. The first 20% of the sample were used to test stability in that the bpm simulated fell well within the range of normal breathing rates. From 20-39% of the total sample, bpm was set to 5 to test the low pass filter and the low breathing rate alarm/flag, the successive sweep set the bpm back to the normal breathing rate. For the 4th and 5th quintile, the breathing rate was set to 60 to test the upper end of the spectrum in the high pass filter and the alarm, and then set back to a normal breathing rate for the last 20% of the samples.

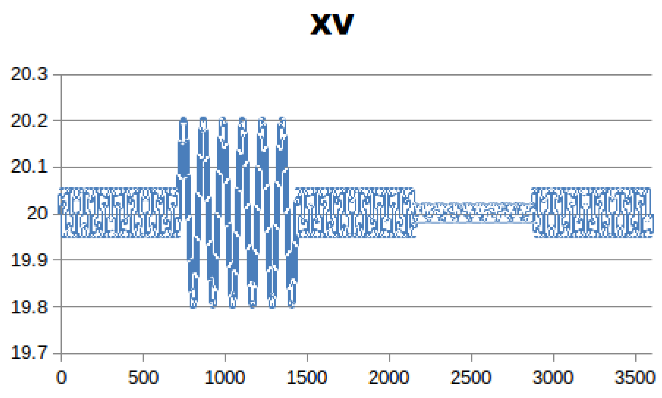

Figure 3: Simulated Sample (3600 samples) – this figure shows the simulated sample with quintile sweeps. The first, third and last fifth of the data can be seen as the same sinusoid as these acted as the default value and was the value that fell inside the range of acceptable breathing rates frequencies.

With a visual example of the simulated data, filter outputs could be seen and verified that they occur at their proper frequencies.

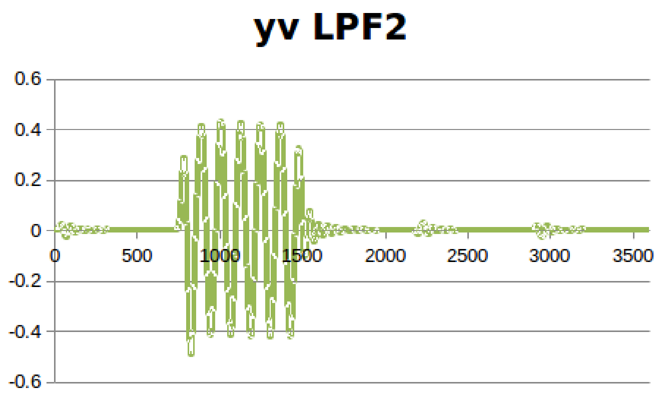

Figure 4: Low Pass Filter Output (3600 samples) – this figure shows the low pass filter implemented and its effects on the simulated data. The filters output can be seen acting on the second quintile of the signal which, incidentally, is the sweep of the sample where the signal was set to trigger the low pass alarm.

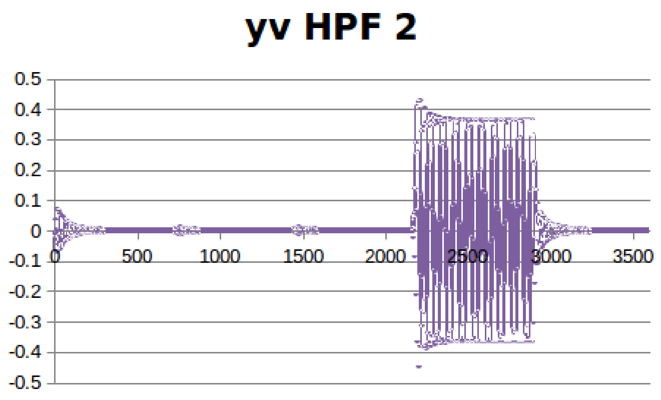

Figure 5: High Pass Filter Output (3600 samples) – this figure shows the high pass filter implemented and its effects on the simulated data. The filters output can be seen acting on the fourth quintile of the signal which, incidentally, is the sweep of the sample where the signal was set to trigger the high pass alarm. Noted in this output is the spike in energy around the initialization. This was minor and proved to be negligible throughout testing.

To further test the properties of the temperature sensor, a 256 sample test was completed to test the cool off and response. With this test, raw values as well as averaging and dithered averaging was graphed to show how beneficial dithering was in implementation of the system. In this test, the raw data as well as the average of the signal and dithered average were displayed. The standard deviation for these three methods were each calculated. The method that yielded the best results was then used.

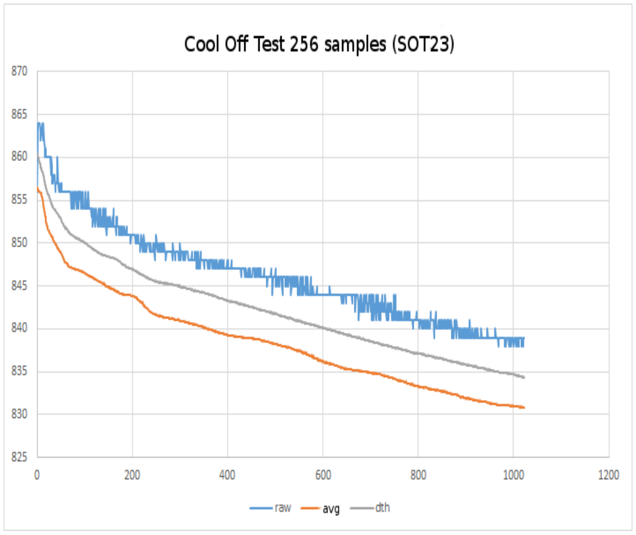

Figure 6: Cool off test (zoom in) – this figure shows the cooling and minor heat retention in the temperature sensor. 256 sub samples were used in the 1024 sample set. In blue can be seen the raw data output by the sensor. The orange corresponds to averaging the raw data and the grey line represents the data after it has been dithered and averaged. The y axis is expressed in millivolts. The test involved heating up the sensor by applying warmth and then allowing the sensor to cool off in isolation and collect data. This is why the peak is at the initiation of the test and then falls into its decay thereafter. This allowed an estimation of the tau value of the sensor and in turn, how quickly the sensor could respond to changes. Plateaus and hiccups were of the greatest concern in the slope as this was an indication of a responsive/faulty sensor. Standard deviations were found in the magnitude of 4 thousandths for dithered average and similar but higher values were found for the averaged signal.

With this cool off test, the tau value was calculated to be roughly 6.82 seconds. For the actual project, this means that within 6.82 seconds of detecting an out of bound frequency occurrence, an alarm would sound. This also further proved the usefulness of dithering. The dithering averaged of the signal in Figure 6 can be seen in grey. The slope is gradual and smooth indicating a good response from the thermal sensor.

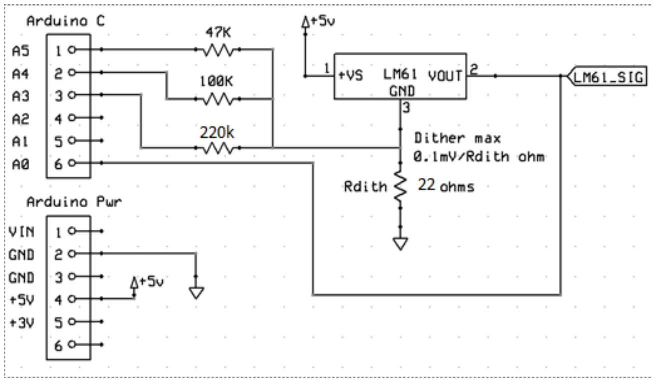

Figure 7: Dithering Circuit – this figure shows components and their values involved in the dithering of the circuit. Dithering resistor (22 ohms) attached to ground was tested and modified until desired results were found. Tested values ranged from 10 to 33 ohms. 22 ohms provided the smoothest response in the cool off test from figure 6.

III. Decision Making & Alarm Implementation

With the components of the system functioning, the transitioning from simple boolean flags for the alarms to auditory tones was next in task.

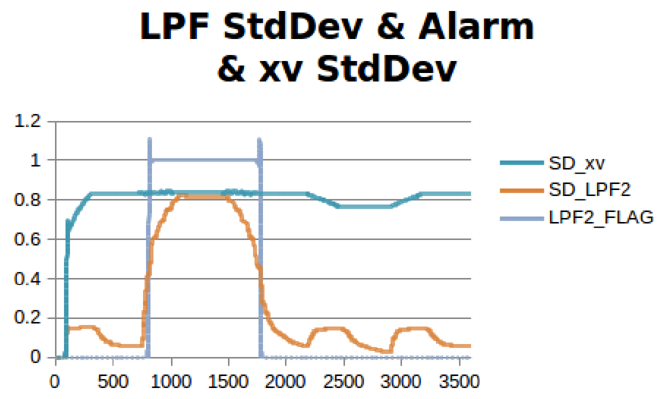

Figure 8: Functioning Low Pass Filter flag – this figure shows a trial of the simulated data shown in Figure 3 where the standard deviation of the simulated data is shown along with the standard deviation of the low pass filter and the change in the boolean flag of the low pass filter alarm. The boolean value would toggle to true if and only if the normalized standard deviation was greater than .5.

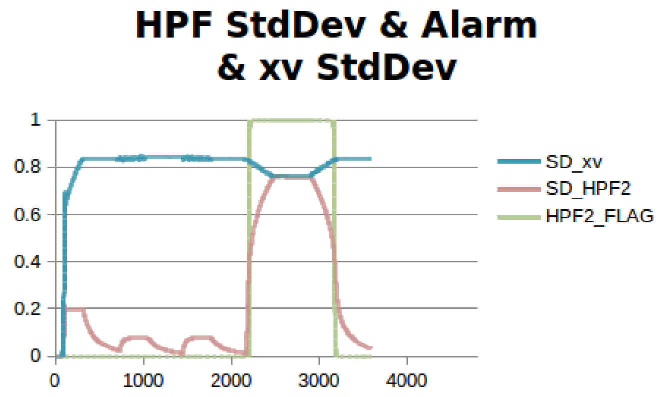

Figure 9: Functioning High Pass Filter flag – this figure shows a trial of the simulated data shown in Figure 3 where the standard deviation of the simulated data is shown along with the standard deviation of the high pass filter and the change in the boolean flag of the high pass filter alarm. The boolean value would toggle to true if and only if the normalized standard deviation was greater than .5.

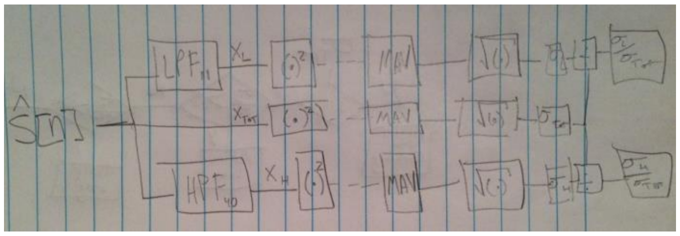

Figure 10: Normalized Standard Deviation Derivation– this flowchart shows how the input signal was parsed and passed through to arrive at values

for low and high normalized standard deviations. This is also known as the coefficient of variance.

With a method on how to go about calculating the actual values of relative error for the high pass and low pass filters, the system then had to determine whether it was acceptable to call an alarm or not. Taking into account the normalized standard deviation, this was simple as anything greater than ½ would prove to fall without of range.

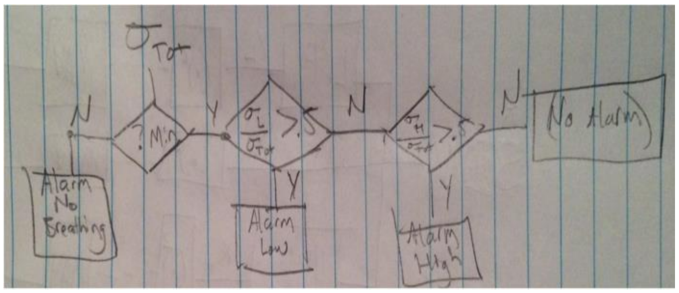

Figure 11: Decision Making Pathway– this flowchart shows how with just the standard deviation of the signal and the normalized standard deviation of the two filters, a decision could be correctly made by the system.

Figures 8 and 9 proved that the boolean flags were correctly triggering and that the design from Figures 10 and 11 were implemented properly but there was another obstacle that had to be surmounted. Because the sampling of the temperature sensor makes use of timer1 in the microprocessor, further use of the tone library for alarms would throw off the sampling rate of our measurements. Because of this, external libraries and outside resources had to be used. ToneAC was stumbled upon and implemented in the design. Fortunately the Arduino environment permits the use of external libraries that are in .zip format. Because toneAC makes use of PWM and the concept of push/pull on two pins, the timer1 register is left along and the sampling remains true.

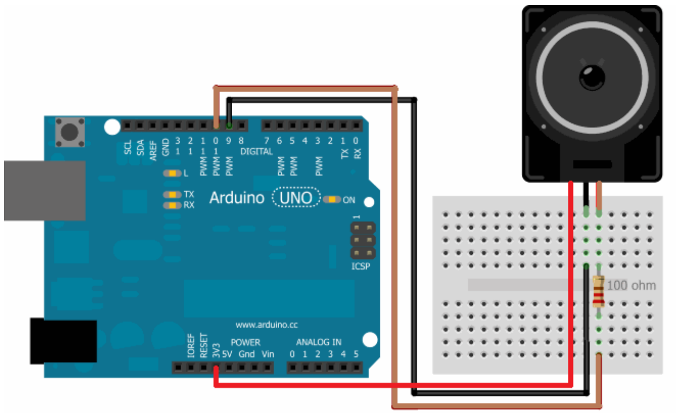

Figure 12: toneAC Schematic– this image shows the schematic of how the toneAC library was used. Ground was tied to pin 9 and the audio In of the speaker was tied to pin 10. Because the temperature sensor was powered by 5 volts, the speaker made use of the 3.3 volts supply that is also on board the Arduino.

The toneAC function allowed to create tones with various volumes and frequencies. This allowed for the two alarms to be distinct from each other.

IV. Results

Tests with the simulated data reliably yielded successful results. Tests on 'Gracie' and measured breathing were successful as well but false positives were encountered in that for a normal breathing rate, an alarm would sound. This was later found to only occur very close the edges of the filters. Outside sources for the temperature sensor specification sheet and toneAC library cam be round in the section below. Overall, the system proved to function properly and the goal was accomplished.

V. Reference

Specification sheet for LM61:
http://www.ti.com.cn/cn/lit/ds/symlink/lm61.pdf

Documentaion for toneAC library:
https://bitbucket.org/teckel12/arduino-toneac/wiki/Home