Temperature Based Respiratory Rate Detection System

D. Kuehnert

***Abstract*— The combination of an Arduino Microcontroller, a simple LM61 temperature sensor, and a piezo buzzer can provide an accurate, efficient, and cost effective method for detecting breathing rates of a patient. A sensing system consisting of a LM61 sensor and a dithering circuit are feed into the Arduino's analog to digital converter. From there, the signal is processed through a series of digital filters implemented on the microcontroller. Statistical analysis of the final filtered signal is used in order to determine the patient's current respiratory rate. Since the system is designed in order to alert medical staff of a deviation in normal breathing patterns, the piezo buzzer outputs three different tones. One for a low breathing rate, one for a high breathing rate, and a tone for a disconnected system.**

I. Introduction

RESPIRATORY analysis is often used for diagnosis purposes in a variety of cases, not just respiratory complications. This includes cardiac, metabolic, neurological, and psychological conditions as well as many more. Though respiratory assessment can consist of a multitude of tests, that all look at different key features of the patients lung functionality, the most significant is the breathing rate of a patient [1]. The standard method for breath rate determination is to count the number of breathes over a minute period. A few issues arise due to this. The first is that some patients may alter the way they are breathing when medical staff are performing the tests, skewing data accuracies [1]. The second is the rate these tests are performed. Breath rate is only taken every so often as it takes time out of a nurse's schedule. The resulting ambiguities in data will typically cause medical providers to search for other verifications methods in order to make a diagnostic [2]. With the advancement in technology, cheap, and accurate breathing detection systems can be made in order to solve these problems. As a system can monitor a patient during all hours of the day, creating a more sound, and validated data set for medical staff.

Though having a second method for diagnosis validation is always a good idea, the convenience of breath rate analysis plays a crucial role in pediatrics. As these measurements can be typically taken in a noninvasive manor. The resulting is an easier time not only for the patient, but also the medical staff trying to determine a diagnosis. Furthermore, when a case comes into play such as pediatric pneumonia, the breathing rate of a patient is determined to be a better overall predictor. Some other predictive characteristics have a higher accuracy, however, only a small range of patients will develop these characteristics [3].

The following paper introduces a design concept in order to monitor breathing rate of pediatric patients. The only stipulation is that it has to be adapted to a fixed configuration system. A low cost ATMEGA328p microcontroller, with only a 10 bit analog to digital converter and 2 kilobytes of RAM, is allowed for processing purposes. The sensing mechanism is constrained to a LM61 sensor with no additional analog components. This method will allow for a cheap, fast, reliable, and accurate data processing.

The overarching goal is to be able to notify medical staff of any important changes in a child's (age 11 months to 5 years) respiration rate. Anything above forty breathes per minute would require staffs attention, as the child may have pneumonia. If the breathing rate is below twelve breathes per minute, staff are alerted with a different tone. As this is considered a low rate, and can be an indication of some abnormal condition. Finally staff need to know if the system is functioning properly, thus a third tone is needed. Staff should not be alerted at all if the patients breathing rates are normal, and the system is functioning properly. This all needs to be done real time, and alert staff within two minutes, in order to assist the patient.

II. System Description

The system created for breathing rate detection can be broken down into four parts in order to reduce complexity. The sampling process, signal preprocessing, filter bank processing, and finally speaker state processing. A very simple high level over view of the system is shown below in ***fig. 1***. This diagram is essentially the fixed configuration stipulation mentioned prior.

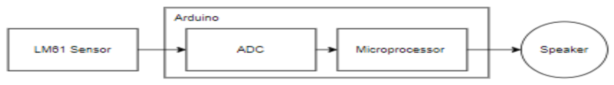

Fig. 1 High level over view of the breathing rate detection system

A. Sampling Process

The relationship between the LM61's readings and temperature is expressed by the transfer function

$$V_0 = 10\text{mV/°C} * \text{T°C} + 600 \text{ mV} \quad (1)$$

The Arduino's 5v ADC has 10 bits allotted to it allowing for code values from 0 to 1023 to be produced. Therefore it can be found, using the transfer function in equation 1, that the Arduino can sense 0.488 °C per code value. The LM61 is being used to detect variation in temperature, in order to correlate to breathing rate, which typically will only vary less than a degree. Therefore, a method of increasing the accuracy of sampling is needed. The solution is to attach a dithering circuit to the sensor, as well as using a technique of oversampling in averaging.

The sampling rate was set to once every ten milliseconds or 600 breathes per minute. Within this time frame, 160 subsamples are taken, and averaged together at the time of the sample rate. The oversampling and averaging allows for a smoother and more accurate transition between code values, however, just this technique is not enough. By adding dither (noise) to the incoming signal, we force the incoming signal to cross more quantization thresholds at lower rates of voltage change. The result is a smoother input signal with a greater signal to noise ratio than having no dither, seemingly counterintuitive.

In order to better understand the reasoning behind dithering and oversampling ***fig. 2***, and ***fig. 3*** have been included. A 'pinch' test was performed and measured via the Arduino with no modifications (***fig. 2***). In the figure, the sensor appears to have an exponential decay, however we have a 'stair case' drop with chatter, due to the signal being between quantization thresholds. Another test was completed to determine the effects of just over sampling and averaging. No figure was included for this, however statistical analysis was completed to determine the signal to noise ratio displayed later on in ***table 1.*** Finally a test of oversample/averaging plus the addition of dither noise was done. A windowed segment, with drift estimate of this result is shown in ***fig. 3***. This result was analyzed as well to determine if the resulting increased the signal to noise ratio of the LM61 sensor.

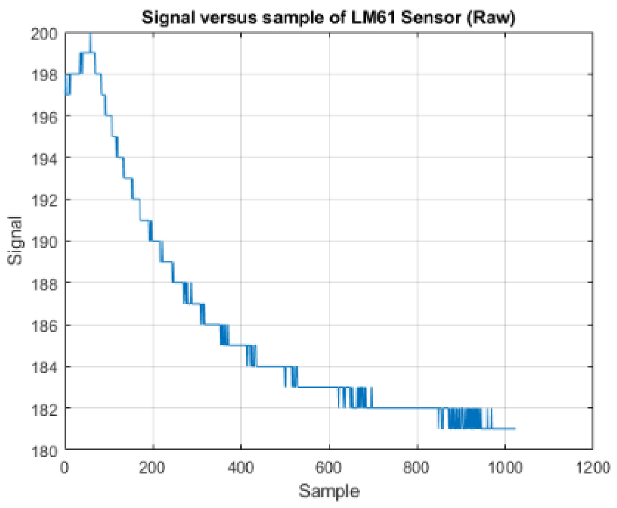

Fig. 2 Raw reading of the LM61 sensor by Arduino using a pinch test

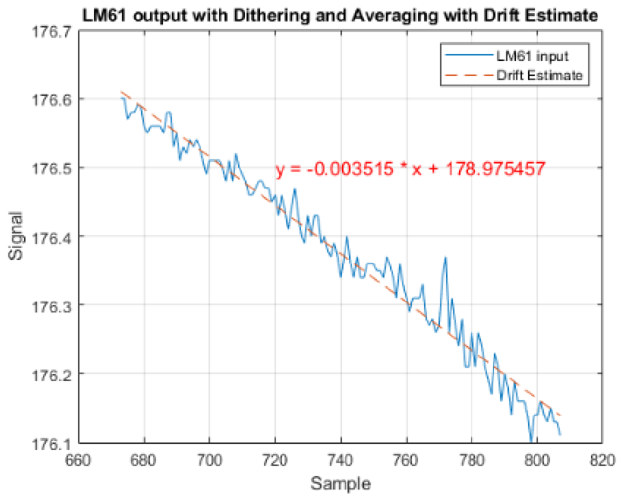

Fig. 3 Windowed segment of LM61 using a oversample/averaging as well as the addition of dithering

After the signal has been sampled, it is still in units of code values. Using the transfer function shown in equation 1, the signal is converted from code value to volts, and then from volts to Celsius degrees. The full sampling process is shown below in ***fig. 4*** with a description of the signal units being passed shown below each arrow.

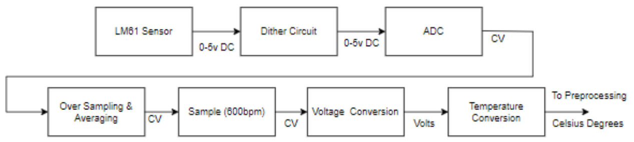

Fig. 4 Full sampling process block diagram. The output leads into the preprocessing portion of the code. Note the values below the arrows represent each unit the signal is currently being represented by after each stage

B. Signal Preprocessing

Due to the design of the LM61 sensor, a leaky integrator effect occurs while trying to take measurements. As a result the running average of the filter slowly increases over time, and a corresponding drift can be seen formulating on the output. Additionally, in order to processes the signal and determine breathes per minute, the signal needs to be centered on the x axis, that is, it needs to have 0 DC gain. Currently, even if the LM61 sensor produced no drift, a gain of around 30 Celsius degrees is seen, as that is the average temperature of human breath.

A simple solution to this problem is the inclusion of an equalizer in the signals path. An equalizer acts as a derivative, which ultimately cancels out the effects of the drift. The derivative is effectively removing DC gain from every sample, keeping the output drifting away. A plot of the frequency response of the equalizer is shown in ***fig. 5***. In the plot you can see maximum attenuation at 0 breathes per minute (also known as the DC voltage), with the rest of the range minimally attenuated. One thing to note is the ripple in the pass band of

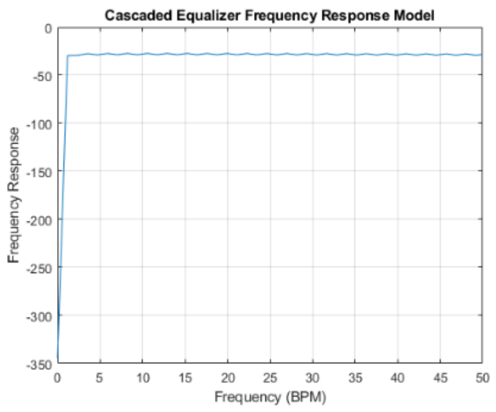

Fig. 5 The frequency response for the equalizer used in the breathing rate detection system

the equalizer. Due to this ripple, we are effectively adding noise to our signal which is no good (summarized in ***table 1***). The signal needs to be as clean as possible when going into the filter banks for statistical analysis. To smooth out the equalized output, a moving average filter is used.

The equalizer uses fixed point scaling, as well as the filter banks, all in order to increase processing speeds of the signal. As long data type math is a lot faster than floating point data type math. To keep this efficiency continuous throughout the system a finite impulse response sinc filter with fixed point representation is used.

Using a scalar of 2^{18}, a high accuracy smoothing filter can be created, with minimal noise do to quantization of the signal, meaning faster processing than floating point representation, with almost identical results. This scalar is used for the kernel representation for fixed point. The signal representation in fixed point is multiplied by one thousand, as this provided the best accuracy with minimal chance of roll over.

An additional benefit to the smoothing filter created is we can set a maximum breathing rate that is allowed into our system. The filter banks will be detecting a range of 0 – 70 breathes per minutes, so anything higher needs to be filtered out before entering. Since the windowed sinc filter is a low pass filter, just the corner frequency needs to be adjusted, and a full preprocessing system is complete. For this specific application, a corner frequency of 50 breathes per minute was chosen. This seems low, and normally would be attenuating values of 50 BPM by half, however the amplification of the equalizer causes a shift of the actual corner frequency of the preprocessing filter to be closer to 60 BPM. The magnitude of the frequency response of the full preprocessing system is shown below in ***fig. 6***.

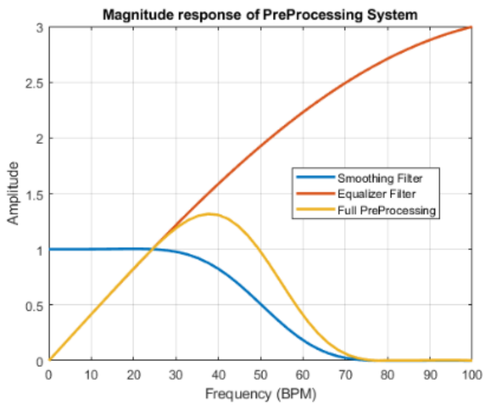

Fig. 6 Frequency response of the entire signal preprocessing filter for breathing rate detection system

Implementation of these two filters are displayed in a block diagram below (***fig. 7***). The filters are placed in series with one another, such that the signal from the sampling process enters the equalizer. The output of the equalizer is then fed into the smoothing filter. A breakdown of the signal to noise ratio after the equalizer and smoothing filter are displayed in ***table 1***.

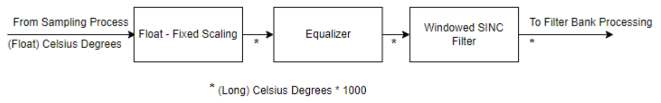

Fig. 7 Full signal preprocessing block diagram. The input comes from the output of the signal sampling block diagram, and the output leads in the filter banks.

C. *Filter Bank Processing*

At this point of the system, a fully smoothed out, equalized signal is in the Arduino. This signal represents a variation in temperature of a user breathing, and ultimately shows the frequency of the current breathing. In order to simply decipher this data three filter banks are designed, with a partitioning filter, and a statistical analysis attach to each bank. To get a better idea of the system before a deeper dive, a diagram has been included in ***fig. 8***.

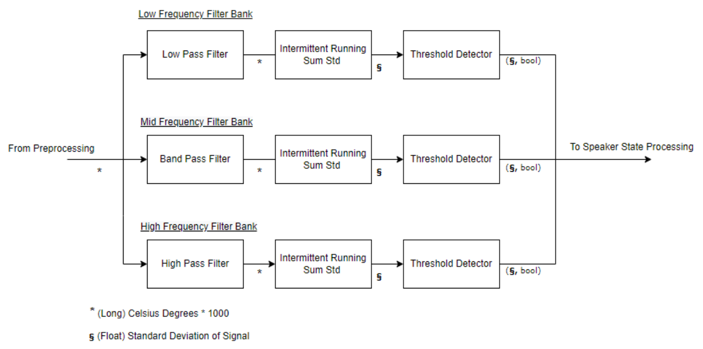
Low Frequency Filter Bank
Low Pass Filter
Intermittent Running Sum Std
Threshold Detector
Mid Frequency Filter Bank
Band Pass Filter
High Frequency Filter Bank
High Pass Filter
From Preprocessing
To Speaker State Processing
(§, bool)
* (Long) Celsius Degrees * 1000
§ (Float) Standard Deviation of Signal

Fig. 8 displays the breakdown of the filter bank processing for the breathing rate detection system

Each filter bank is in charge of a range of frequencies. The low frequency filter bank searches for any breathing that is between zero and eleven breathes per minute. The middle frequency bank searches for any breathing that is between twelve and thirty-nine breathes per minute, and the high frequency bank looks for breathing that is between forty and seventy breathes per minute.

These filter banks are placed in faux parallel with one another. That is, no threading is done, so they don't actually run concurrently. This allows each individual filter bank to see the same input signal.

The resulting output signal of the filter banks partitioning filter can be statistically analyzed to determine if the current input signal contains any sinusoidal of the goal frequency. Any signal that does not have a BPM that falls under the pass band is attenuated to a point that is very minimal when compared to the pass band sections. If an intermittent running sum to the output, it can be seen that a very small standard deviation for any signal that doesn't have a frequency inside the stop band range, and a very large frequency otherwise. A resulting threshold level can be determined from this that would be capable of triggering high if a valid signal is being passed through the filter at any given time.

A few things need to be looked at in order to determine the filters used for each individual filter bank. First, is the system has to sample every 10ms, as such, we need to have this processes completed before the next sample comes in. Having to process three filter banks in the method described above, these filters need to be as efficient as possible. The second thing to account for is the roll off of each filter. The sharper the roll off, the better the statistical calculations are in the end. The two options for the partitioning filters then are either a Finite Impulse Response filter, or an Infinite Impulse Response Filter. Both running in fixed point notation. Either one in theory would work, however to increase the roll off of an FIR filter, an increase in kernel size is needed. This ultimately will delay the final output, and additionally further lengthen calculation times. The other option is high order IIR filters. These can be implemented recursively on the Arduino, with much faster processing time than FIR filters. The downside of IIR filters though is they all have ripple in the pass band, and as a result, will destroy our signal structure. The filter banks are the last stop for the signal though, so keeping a large signal to noise ratio post filter bank is not necessary.

After deciding on Chebyshev in order to maximize our roll off, an iterative design approach took place to determine the rest of the IIR filter characteristics. Increasing order of the filter, as well as the amount of ripple allowed both impact the speed of the roll off. However too much ripple, or too high of an order and the filter becomes unstable. The following ***table 2*** are the determined filter characteristics for the breathing rate system

Filter Bank	Cut Off	Order	Ripple
Low	11 BPM	9	3dB
Mid	12-39 BPM	5	3dB
High	40 BPM	9	3dB

Table 2 Filter characteristics for partitioning filter in each filter bank channel

The impulse response and frequency response of each individual partitioning filter are displayed in the ***fig. 9 – 14***.

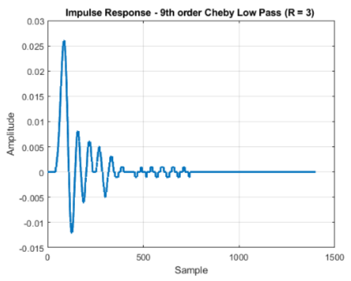

Fig. 9 Impulse response of low frequency filter bank

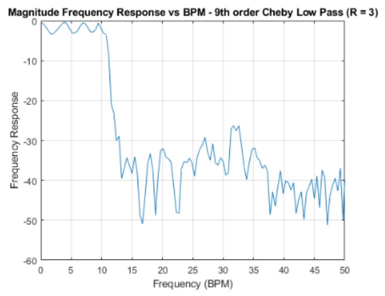

Fig. 10 Frequency response of low frequency filter bank

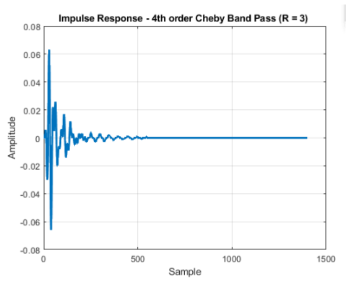

Fig. 11 Impulse response of mid frequency filter bank

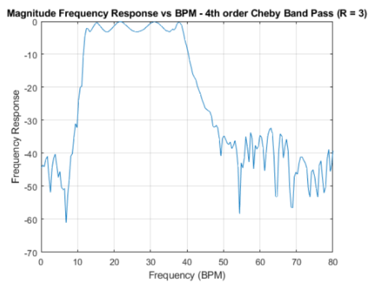

Fig. 12 Frequency response of mid frequency filter bank

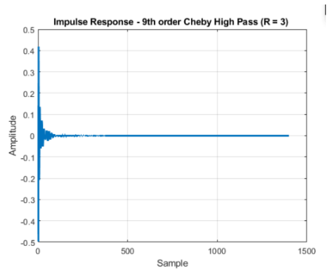

Fig. 13 Impulse response of high frequency filter bank

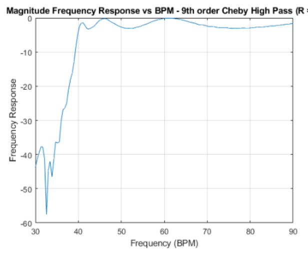

Fig. 14 Frequency response of high filter bank

A specific threshold level must be chosen for each filter bank. The chosen thresholds must be high enough to trigger for a sinusoidal of minimum expected amplitude inside its goal detection range. A quick check of sample test data pumped through our preprocessing system, showed result in a wave form that has a maximum amplitude of about one degree Celsius, and a minimum amplitude of 0.1 degrees Celsius. Using a test sinusoidal with varying amplitudes (0.1, 0.3, 0.9), and constant frequency. Each filter bank was individual tested around at crucial points (frequencies along intersections), and the intermittent running standard deviation was calculated. The results were used to determine the threshold levels and are as follows:

Filter Bank	Threshold
Low	0.024
Mid	0.04
High	0.027

Table 3 The threshold levels for each individual filter bank

To better understand how the next portion of the breathing rate detection system works, ***figures 15 – 22***, are included. These figures are the 'crucial points' used to determine the threshold levels.

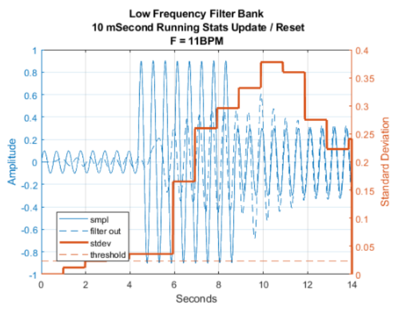

Fig. 15 Displays a valid signal being passed through the low frequency filter bank

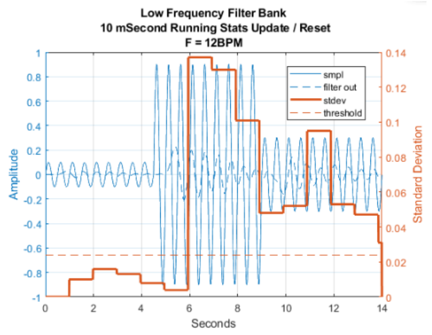

Fig. 16 Displays an invalid signal being passed through the low frequency filter bank

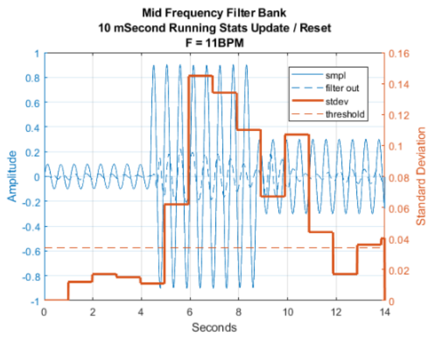

Fig. 17 Displays a invalid signal being passed through the mid frequency filter bank, testing lower bounds

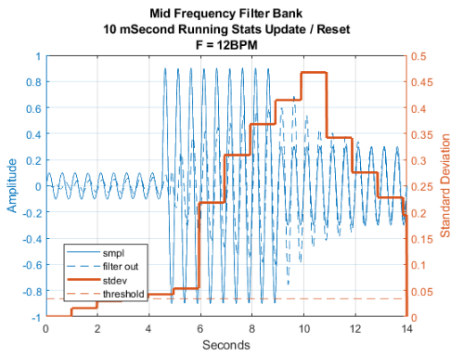

Fig. 18 Displays a valid signal being passed through the mid frequency filter bank, testing the lower bounds

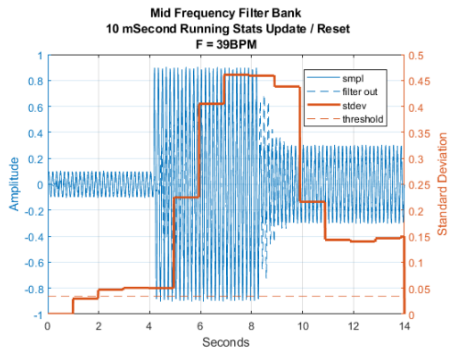

Fig. 19 Displays a valid signal being passed through the mid frequency filter bank, testing the upper bounds

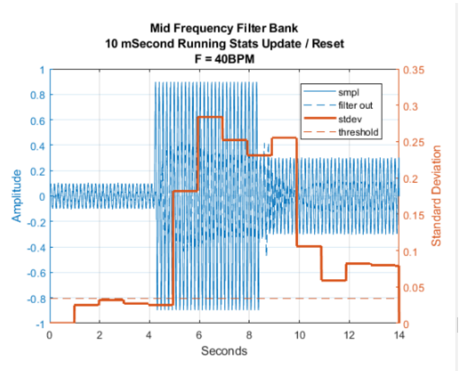

Fig. 20 Displays an invalid signal being passed through the mid frequency filter bank

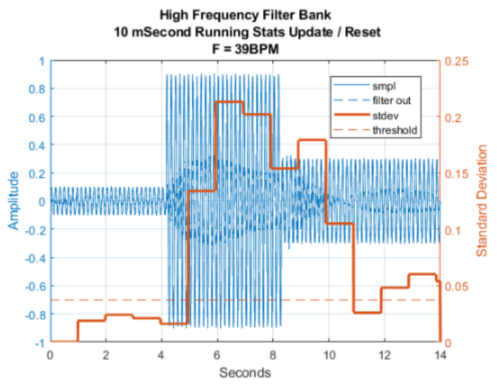

Fig. 21 Displays an invalid signal being passed through the high frequency filter bank

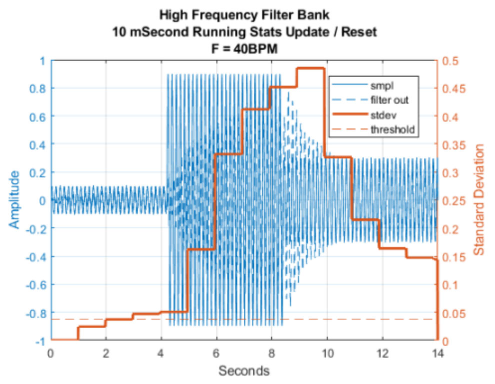

Fig. 22 Displays an valid signal being passed through the high frequency filter bank

D. *Speaker State Processing*

The goal of the threshold detector, is to detect if the minimum amplitude frequency signal is currently inside of the filter bank, nothing more. This however creates some complications. Since the statistical calculations are produced by determining how much the current signal deviates away from zero, a large magnitude sinusoidal will create issues. For example, our 12 BPM signal in the low frequency bank (***fig. 16***) will be triggered, even though it is not in the desired frequency. However the same 12 BPM signal in the mid frequency bank (***fig. 18***) will triggered. If we were to look at their individual intermittent running standard deviation after triggering, it can be seen that in ***fig. 16***, the max standard deviation is just under 0.14 (due to the amount of attenuation placed on it via low frequency banks partitioning filter). Whereas a comparison to the same signal in the mid frequency bank after triggering, the value jumps up to around 0.4 (as the signal is allowed to pass through in the mid frequency bank here). With that said, a since the mid frequency filter bank has a higher standard deviation, the signal must be within the twelve to thirty-nine breathes per minute rate, which is indeed the case.

With that said, the system cannot simply determine what range the signal lies in merely on what filter banks are triggered, as multiple filter banks may be triggered at once. So, a final function is formed in the arudino C code to determine what frequency banks are triggered, and if multiple, find the filter bank where the highest standard deviation occurred. This function then outputs an integer representing one of four states. Zero means the system is disconnected, or a user is too far away from sensor. A one means that the patients breathing is currently in the low range (zero to twelve breathes per minute). A value of two means the patient is breathing in a normal range (twelve to thirty-nine breathes per minute), and the final value, three, means the patients breathing rate is high, a sign of pneumonia (greater than forty breathes per minute).

It was discovered while testing, that sometimes while swapping from various ranges, the wrong state is triggered for a second. This is due to the statistical measurement playing catch up. Ultimately the proper detection is achieved, however, accidently signaling the wrong level is considered a false positive. To get rid of this issue, a delay is emplaced on the speaker state. In order for the speaker to change states, the new state must be held for two seconds. If it isn't held for two seconds, then the old state remains constant. This ultimately delays are alarm by a few seconds, however, the system can still alert medical staff within a few seconds of breathing change (well under the two minute requirement).

Whatever state is determined by the previously defined speaker state 'filter', will be the determining factor on what tone is produced on the piezo buzzer. This step is as simple as passing in the desired frequency (depending on the state) to the Arduino Tone library. This produces a PWM voltage in order to drive the speaker at the desired frequency. The individual states and the piezo frequencies broadcasted are summed up in ***table 4***, with the remainder of the block diagram structure shown in ***fig. 23***.

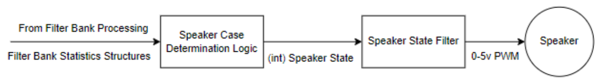
From Filter Bank Processing
Filter Bank Statistics Structures
Speaker Case Determination Logic
(int) Speaker State
Speaker State Filter
0-5v PWM
Speaker

Fig. 23 Block Diagram of the speaker state determination, the final portion of the breathing rate detection system

Breathing Range	Frequency Output	State
Low	400 Hz	1
Normal	No Tone	2
High	1000 Hz (pulsing at one second intervals)	3
Disconnected	200 Hz	0

Table 4 Summary of the speaker state, breathing range, and resulting frequency output

III. Results and Analysis

Testing of the breathing rate detection system consisted of three parts. Preprocessing filter testing, filter bank testing, and then full system testing. Multiple point of test data were used in order to make sure a full range of cases were covered. These included real life data sets, recorded via the LM61 sensor, test vector sinusoidal with ranges of amplitudes and frequencies, as well as various pieces of synthetic data formulated from the transfer function in equation 2.

$$T = T_0 + T_{drift} * n + T_p sin(2\pi \times n/(\frac{samples}{cycle})) + T_{noise} \quad (2)$$

A. Preprocessing

The first half of the breathing detection system, a large emphasis is placed on trying to improve the signal to noise ratio. ***Table 1*** are the calculated noise values throughout the preprocessing phase.

	Standard Deviation	Signal to Noise Ratio
Sensor Noise	0.13415	
Quantization Noise	0.29	
Total Noise After Sampling (no dither)	0.31952	Impaired
Total Noise After Sampling (dither)	0.024333	Improved
Noise due to equalization (estimation)	0.079421	
Total Noise After Equalization	0.2946	Impaired
Noise due to windowed sinc filter (*10^7)	9.2183	
Total Noise After Smoothing	0.20297	Improved

Table 1 Summary of the important noise values throughout the breath rate detection system

As was noted in Section II, by introducing dithering and oversampling to our sampling process, the noise in the system went from 0.31 all the way down to 0.025. As a result we saw an improvement in the signal to noise ratio. It was then discussed that equalization would cause an increase in noise (caused by the ripple in its passband). In the above table, you can see we jumped from seeing noise of 0.025 all the way up to 0.29 which is no good at all. To improve this, we added a windowed sinc filter for smoothing. This resulted in an improvement in the signal to noise ratio. It should be mentioned that even though the windowed sinc filter is in fixed point, the noise caused by this is fractional, and when added in quadrature, is almost completely whipped out. After this point the noise on our signal is no longer an issue, as after it passes through the filter banks, the signal is no longer being used.

Three sets of breathing data of various breath rates given for testing were then passed through the preprocessing stage of the Arduino. The results were then validated against a MATLAB equivalent equalizer and low pass sinc filter. ***Fig.24-26***, display the input signal, the signal equalized by the Arudino, and the signal equalized and smoothed by the Arduino respectively. Every succession in the process you can see the quality of the signal change as described in ***table 1***.

While this processes occurred a timing analysis was completed. Computation time for preprocessing averaged at about 7.8 microseconds to complete. If we had not used fixed point notation for the smoothing kernel/signal processing, it would be expected for the system to take closer to 30 microseconds. No matter the case, this still leaves plenty of time for filter bank processing between each sample.

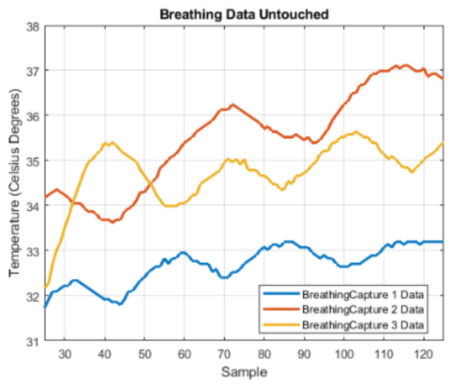

Fig. 24 Three separate breathing data samples passed into the Arduino for testing purposes

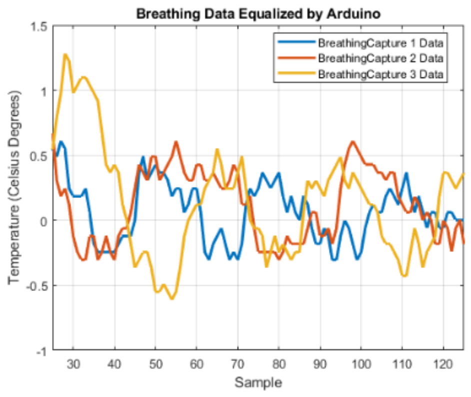

Fig. 25 Data from ***fig. 24*** equalized by the Arduino preprocessing system

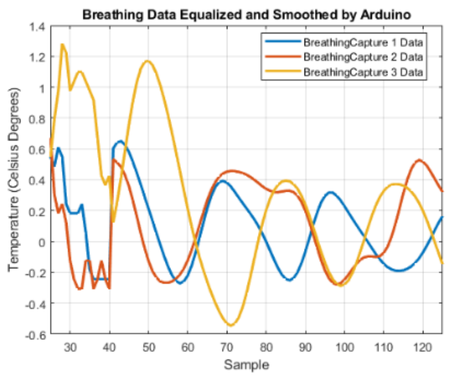

Fig. 26 Data from ***fig. 24*** equalized and smoothed by the Arduino preprocessing system

B. *Filter Bank Testing*

After the completion of the individual filter banks, and a determination of individual threshold levels, the filter banks were brought together a series of test vector sinusoidal were generated via the Arduino, and passed through just the filter banks. Frequencies that would be considered problem zones, ones right on filter bank boundaries, were used for verification, as if these are accurate, an assumption can be made that frequencies in-between should be accurate as well. Below, ***fig. 27-30*** are the resulting input test vector and speaker state determination for the problem zones. Where ***fig. 31*** shows a test sinusoidal of varying frequency and varying amplitudes.

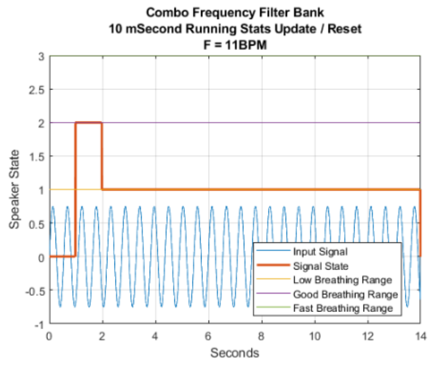

Fig. 27 Shows the test results of filter combination for a 11 BPM signal

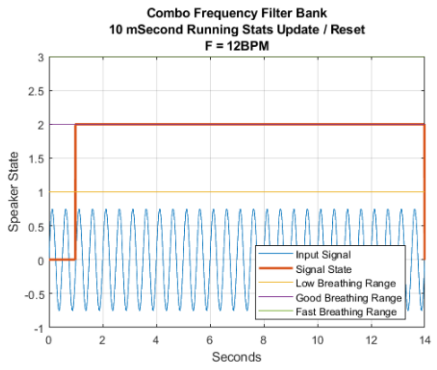

Fig. 28 Shows the test results of filter combination for a 12 BPM signal

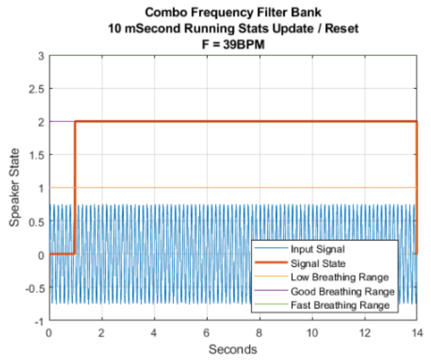

Fig. 29 Shows the test results of filter combination for a 39 BPM signal

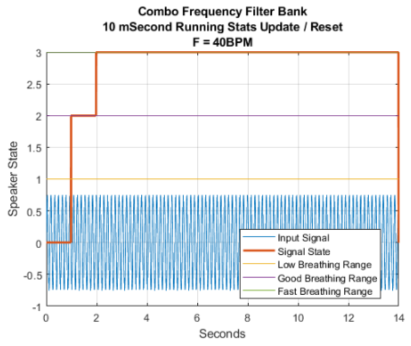

Fig. 30 Shows the test results of filter combination for a 40 BPM signal

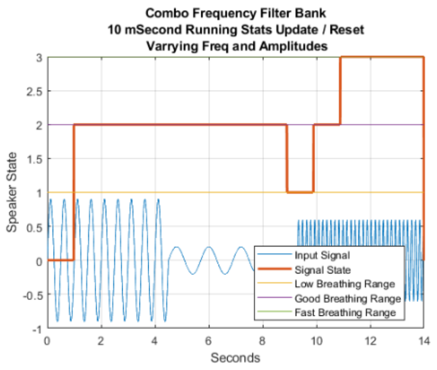

Fig. 31 Shows the test results of filter combination for an varying frequency and varying amplitude sinusoidal

The resulting outputs performed as expected except for ***fig.27*** and ***fig. 30***. They both demonstrated a false negative for a second by output the wrong value. A dig into the individual intermittent standard deviation at the time of those false negatives show that sometimes the statistical analysis takes a second or two to build up. As such we saw a trigger of the normal breathing range, but ultimately dropped/ or ascended back to its proper breathing range. This was when the aforementioned speaker state filter was designed and added into the system.

Something to note here, is in ***fig. 31***, we see all three states trigger as expected, however it takes a little bit for the normal to low breathing switch to occur. This is again another effect of statistical analysis and having to wait for the build up to trigger a change. However a change does eventually occur, and is an accurate one, so the system cannot be faulted on this.

C. *Full System Testing*

Verification of each individual component as the system was built allowed for fast development time. By the time the system was fully combined, only a few adjustments needed to be made in order for full functionality. Since everything up until this point works, several sets of data were sent over to the Arduino to be processed. The Arduino returned data between every major block in the system. This allowed for cross verification between old tests to verify portions of the full system integration such as the equalizer and the smoothing filter. One issue however still remains is determining accuracy of the system for breathing data given for testing. The breathing capture data (charted on ***fig. 29***), were brought over to the frequency domain in chunks of ten samples. The maximum spectrum produced in this ten sample chunk was then determined. This allows for a crude representation of the frequency spectrum to be placed in the time domain. Though crude, it proved to be a great method for verifying unknown breath rates. ***Figures 32-34***, display the discussed breathing capture data, represented with BPM vs time. The resulting output of the breathing detection system is shown in ***figures 35-37***.

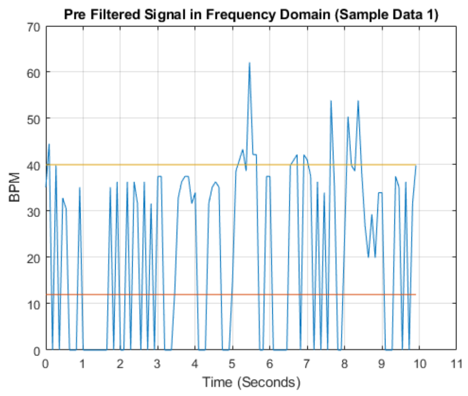

Fig. 32 Shows max BPM every ten samples for Breathing Capture 1 data

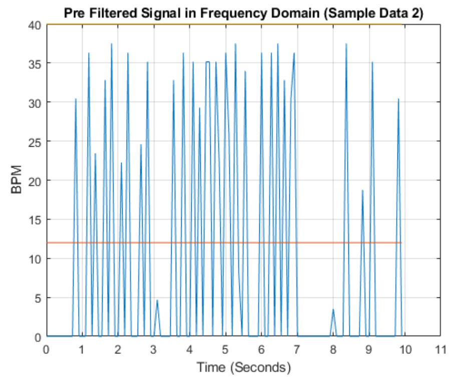

Fig. 33 Shows max BPM every ten samples for Breathing Capture 2 data

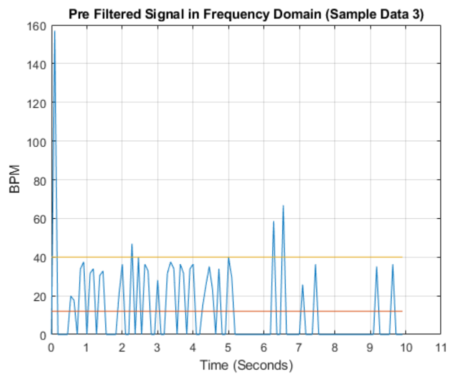

Fig. 34 Shows max BPM every ten samples for Breathing Capture 3 data

Evaluation of the graphs shows expected results. For breathing capture one data we see a large spike in the breathing rate. This is accurately reflected around the six second output of ***fig. 35***. The system then seems to drop down to the disconnected state, even though a normal breathing range is detected. A dig into the sample at that range shows the maximum amplitude to be incredible small. As such the sensor is too far away from the user, so the system should say that it is disconnected, and this is a valid output.

Another concept verified here is our preprocessing smoothing filter. In ***fig. 34*** there is a large spike in breath rate, approximately 155 breathes per minute. However, our high frequency bank was never effected by this, due to the design of the preprocessing filter.

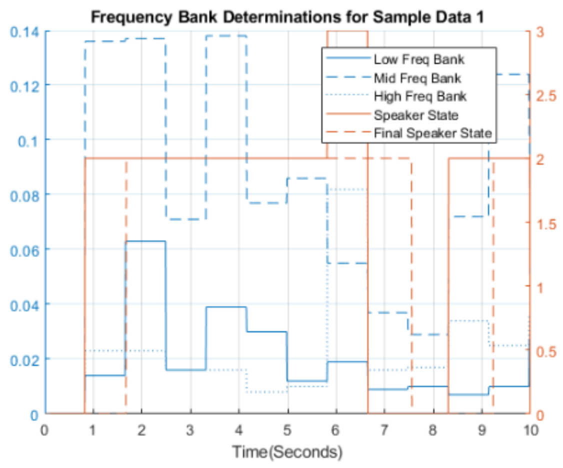

Fig. 35 Resulting system output based on breathing capture 1 data

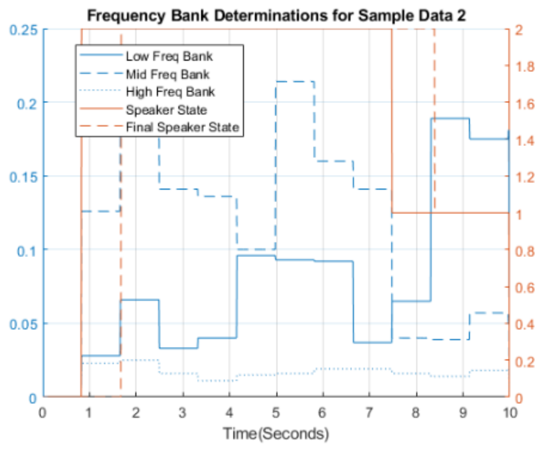

Fig. 36 Resulting system output based on breathing capture 2 data

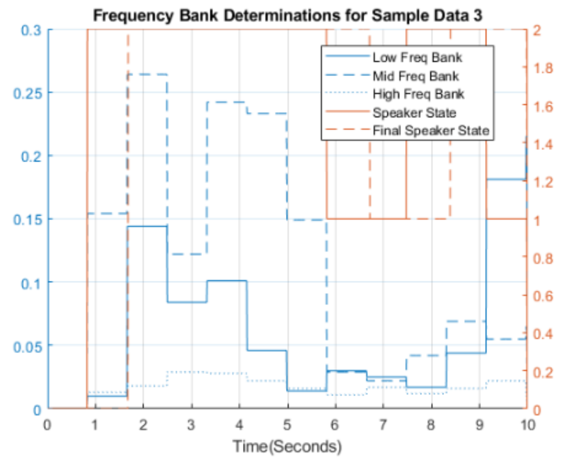

Fig. 37 Resulting system output based on breathing capture 3 data

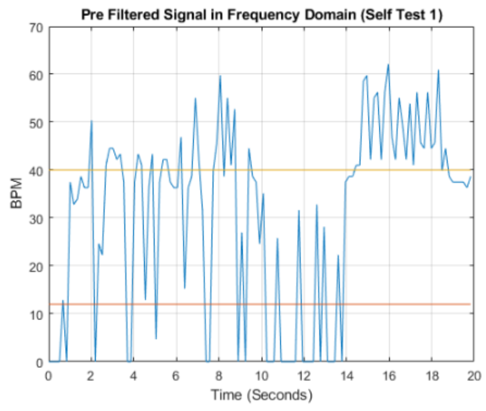

Fig. 38 The first of three self-testing done for the breathing detection system. This is the frequency analysis of input signal displayed in BPM vs Time

Moving on, the system was tested as a whole. That is data that is collected from the LM61 sensor from one of the group members, and passed through the entirety of the system (unlike the previous data which only passed through the preprocessing and filter bank processing). Three sets of data are displayed below ***fig. 38-45***. Each set contains a equalized and smoothed input read from the LM61 sensor, with a BPM vs time analysis, as well as the output of the breathing description.

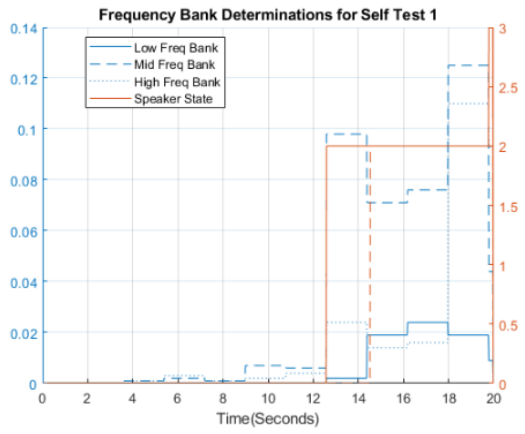

Fig. 39 The first of three self-testing done for the breathing detection system. This is the frequency bank analysis, along with the speaker state, and the filtered speaker state (‘filtered speaker state is the orange dashed lines’)

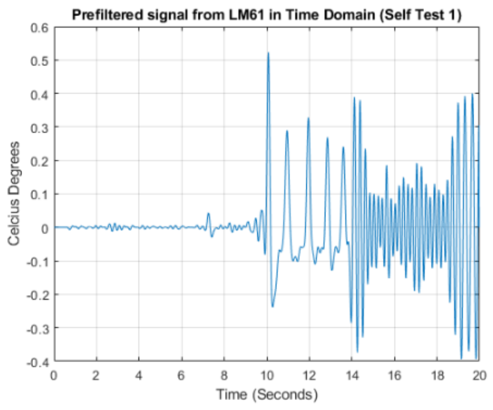

Fig. 37 The first of three self-testing done for the breathing detection system. This is the smoothed and equalized signal read from the LM61 sensor.

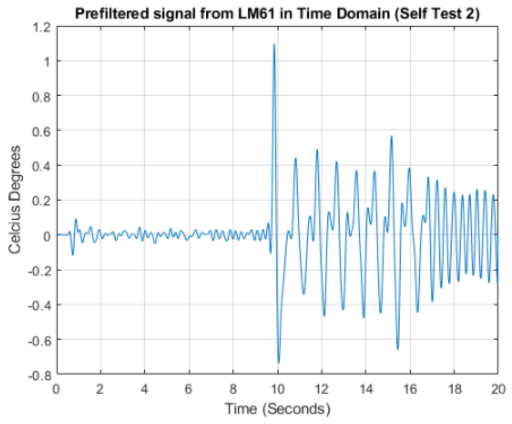

Fig. 40 The second of three self-testing done for the breathing detection system. This is the smoothed and equalized signal read from the LM61 sensor.

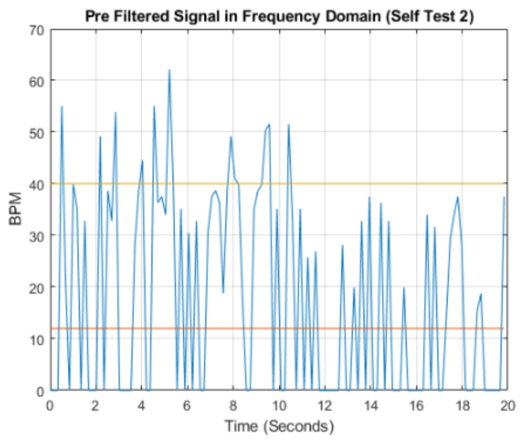

Fig. 41 The second of three self-testing done for the breathing detection system. This is the frequency analysis of input signal displayed in BPM vs Time

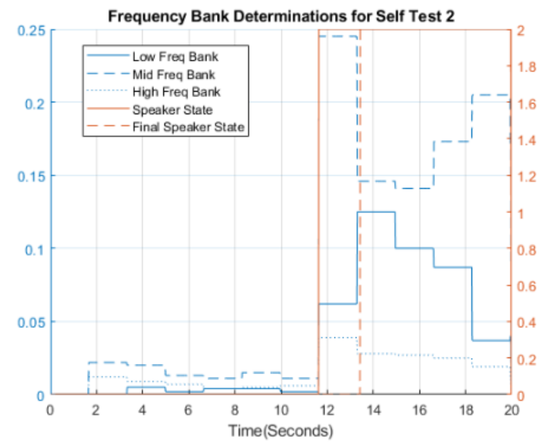

Fig. 42 The second of three self-testing done for the breathing detection system. This is the frequency bank analysis, along with the speaker state, and the filtered speaker state

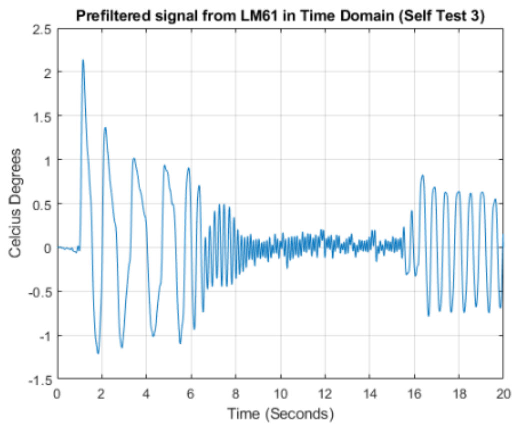

Fig. 43 The final of three self-testing done for the breathing detection system. This is the smoothed and equalized signal read from the LM61 sensor.

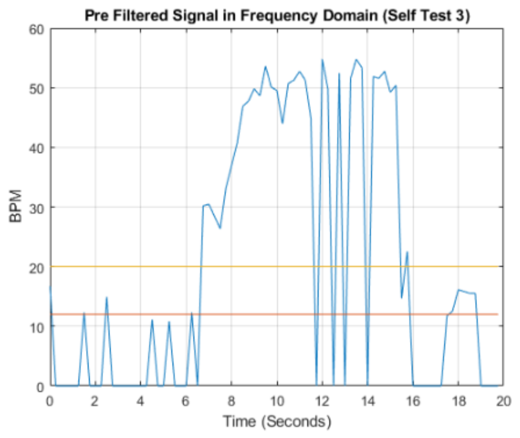

Fig. 44 The final of three self-testing done for the breathing detection system. This is the frequency analysis of input signal displayed in BPM vs Time

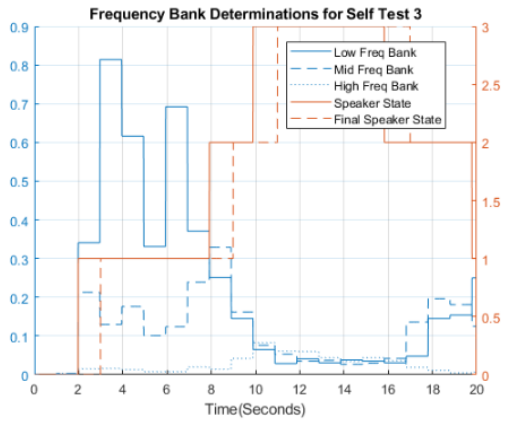

Fig. 45 The final of three self-testing done for the breathing detection system. This is the frequency bank analysis, along with the speaker state, and the filtered speaker state

The objective of these self-tests were used to verify two things. First, the accuracy of our breathing detection system. The second, was our analysis of the frequency domain in order to cross verify our output. The three self-tests samples were recorded with breathes matching a clock constraint. As a result, semi accurate low, normal and breathing ranges could be inputted into the system. The resulting frequency analysis produced results as expected. For self-test one and two (***fig. 37–42***) it can be see that the first half, the sensor was placed far away from the users mouth. As a result, the frequency analysis shows large spikes above in the fast breathing range (due to the noise). However, the noise isn't large enough to cause the high frequency filter bank to trigger.

Looking into the output of each self test, the resulting filtered or final speaker state of the system can be seen to accurately portray our goal breathing ranges, as well as cross verifies well with the frequency analysis. The first tests goal was to go from a normal breathing rate two a high breathing rate. Note, the speaker state triggers high at the end, but the final speaker state did not. This is from running out of time, however, it can be seen the statistics building up and triggering the proper threshold. Therefore, if the testing was given a few more seconds, we would see it hold the fast breathing state in the final output. For the second test just the normal breathing range. With the final testing low, high, and then down to normal ranges.

Several more tests were conducted to verify the results. Except instead of using a stopwatch to verify breathing ranges, the frequency analysis method proposed throughout this paper was used for verification. A series of five more tests running for about three minutes were done, all portraying accurate results with no false positive/negatives.

Additionally, throughout the full system testing, execution times for each sample were taken. These were averaged and outputted at the time of completion to get an idea of computational time required for each sample cycle. The results showed an astounding 45.47 microseconds. If we remove the time found to take for preprocessing, it can be determined that the filter banks take approximately 37.67 microseconds. Since we are sampling at a 10ms rate, with a 160 subsamples per sample. This means our subsamples are triggering every 62.5 microseconds. Thus, our processing will not run into any interference with the interrupt triggering for sampling, even with our filter banks running in a faux parallel manor.

IV. Conclusion

Given the challenge of creating a rate of respiration sensing system using a prefixed system resulted in a few design challenges. The first was finding a way to increase the signal to noise ratio of the sensor. The LM61 sensor temperature sensor required for this project produces a leaky integrator effect, causing drift. Additionally to determine breathing rate of a patient, the sensor needs to detect a variation of temperature of less than 1 Celsius degree. Without any modification, the conversion and quantization of this signal results in detecting changes of 0.48 Celsius degrees. This will not give the required resolution, and as such an implementation of oversampling and dithering were added to the system. This allowed for increase resolution on the signal, as well as a decrease in the overall noise on the input.

The next challenge was removing the drift from the signal, the solution for this was to place an equalizer after the sampling process. The design of the equalizer acts as a derivative to cancel out the leaky integrator of the LM61. The downside to this inclusion is the noise added to our signal, due to the ripple in the pass band of the equalizer. A solution for this was adding a low pass windowed sinc filter in series with the equalizer. The result is a nice smoothed, equalized wave form, with minimal attenuation, to be passed into the filter banks.

The final challenge was the actual breath rate determination of the signal. Accomplishing this was a simple as creating three filter banks. One for each goal range being sought after. A method of calculating the intermittent running standard deviation was applied in order to statistically analyze each individual filter bank. The idea of the filter bank is to trigger when the minimum magnitude signal, of a valid frequency is seen in the filter bank. Thus a threshold level was determined for each bank. Whenever a threshold level was broken, the current standard deviations of each bank were compared in order to determine where the true breathing range actually lied.

Once the system knew what range the breathing range was, the rest involved simply tying a determined state to a specific frequency to be played on the piezo buzzer. During testing, it was discovered that statistical analysis in this method takes some time to 'build' up, and as such the wrong breathing range may be triggered for a second. This was quickly fixed by adding a required two second hold. That is the speaker state is not updated, unless the breathing is in that range for at least the allotted amount of time. This effectively filtered the speaker state, and resulted in a clean system that produced no false positives.

Though this system is defined for breathing ranges of children, it can be modified to suit the needs of teenagers and adults by simply adjusting the corner frequencies of each filter banks partitioning filters. Additionally, more filter banks could be added to create a larger spectrum of breathing ranges (instead of three), so that integration into data collection can be completed. This would allow medical staff to have a clean, accurate, and fruitful source of information over a large spectrum of patients. Meaning that breath rate analysis can once again, be a more trusted source of diagnosis of patients.

Despite limiting restrictions on hardware, and microcontroller performances. It was found that using a combination of several digital signal processing techniques, that a cheap, and cost effective, breathing rate detection system could be created. The system can accurately, and quickly, detect low breathing (less than eleven breathes per minute), normal breathing (between twelve and thirty-nine breathes per minute), and fast breathing (greater than forty breathes per minute), as well as if the sensor is disconnected/too far away from the patient.

References

[1] Kennedy, Siobhan. "Detecting Changes in the Respiratory Status of Ward Patients." *Nursing Standard*, vol. 21, no. 49, 2007, pp. 42–46., https://doi.org/10.7748/ns2007.08.21.49.42.c4604.

[2] KRIEGER, B., et al. "Continuous Noninvasive Monitoring of Respiratory Rate in Critically Ill Patients." *Survey of Anesthesiology*, XXXI, no. 4, 1987, p. 221., https://doi.org/10.1097/00132586-198708000-00029.

[3] Dai, Y, et al. "Respiratory Rate and Signs in Roentgenographically Confirmed Pneumonia among Children in China." *The Pediatric Infectious Disease Journal*, U.S. National Library of Medicine, Jan. 1995, https://pubmed.ncbi.nlm.nih.gov/7715990/.