DSP Breath Rate Monitor

Daniel E, Brittney M, Alex R, and Kayleen W, *Member, IEEE*

Abstract—**Using an Arduino Microcontroller and LM61 Temperature sensor, analog temperature measurements were taken and analyzed as a method of breathing rate detection. Dither averaging, Oversample averaging and an Equalizer optimize data acquisition for subsequent use as a warning system for dangerous respiratory conditions in children that can signify need for immediate medical attention.**

***Index Terms*— Digital Signal Processing, Dither Averaging, Equalizer, Pediatric Pneumonia, Temperature Sensor**

I. Introduction

THE project uses the Arduino and the LM61 temperature sensor, build a breathing rate monitor to warn of a potential acute respiratory infection, otherwise known as pneumonia, in a child age 11 months to 5 years.

Creating a cheap device that is able to detect when a child is breathing too fast or too slow is important because acute respiratory illness is the root cause of over four million child deaths annually. Change in respiratory rate can signify difficulty breathing and can be a symptom of acute respiratory illness that requires immediate attention. Additionally, quantifiably measured breathing rate can decrease risk of incorrect qualitative diagnosis. Shallow rapid breathing is a good indicator of Stage 1 Pneumonia and can increase chances of early detection and early treatment. A cheap and accessible device eliminates unnecessary and expensive additional tests for patients not yet diagnosed, and is more widely available on a global scale.

The specifications given is the breathing rate monitor must detect if the breathing rate is greater than 40 breaths per minutes (bpm) with both visible and audible alarm. When the breathing rate is between 12-39 bpm, this is considering normal, there will be no alarm. There is a possibility of the sensor indicating it is disconnected if the breathing rate is below normal, between 0 to 10 bpm, if this happens a warning sound should occur, this sound needs to be distinctly different from the high breathing rate warning. The sensor should operate continuously and reliably, the warning should also be sounded within 2 minutes of the breathing rate entering into either above or below breathing rate. The monitor should also not produce either false positive or false negatives.

The problem requires manipulation of a fixed system configuration into a high sensitivity sensor. The Arduinos 8-bit microprocessor and on board 10-bit Analog to Digital Converter provide a low cost processing system with 2K RAM memory. The LM61 sensor is a low cost small package ideal for a sensor used on the face, and requires little other analog circuitry. The transfer function (1) of the LM61 sensor describes linear relationship between the voltage output of the sensor and the change in temperature detected.

$$V_{LM61} = 600mV + 10\frac{mV}{℃} * T℃ \tag{1}$$

The signal contained enough noise to effectively self-dither, so no further increase of signal resolution and noise reduction was necessary. The self-dither was measured prior to ADC sampling, so there is no concern of ability to enable effective oversample averaging. The circuit shown in Figure 1 below is the schematic used.

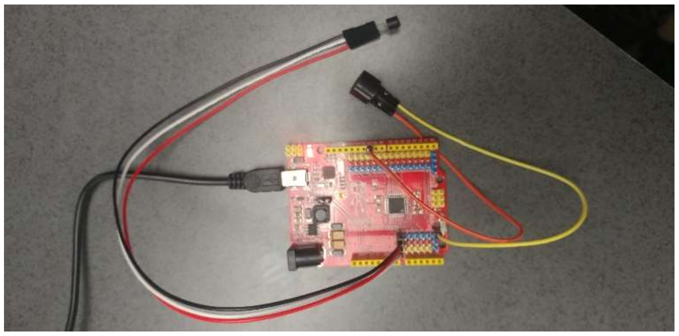


Fig 1. Image of circuit in Fig 1 built on Arduino. Pin 10 is the speaker, Pin 8 is for the LED and A0 is the sensor.

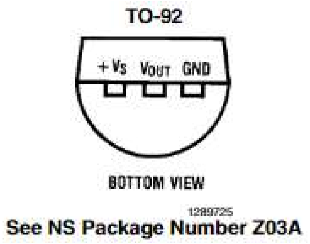
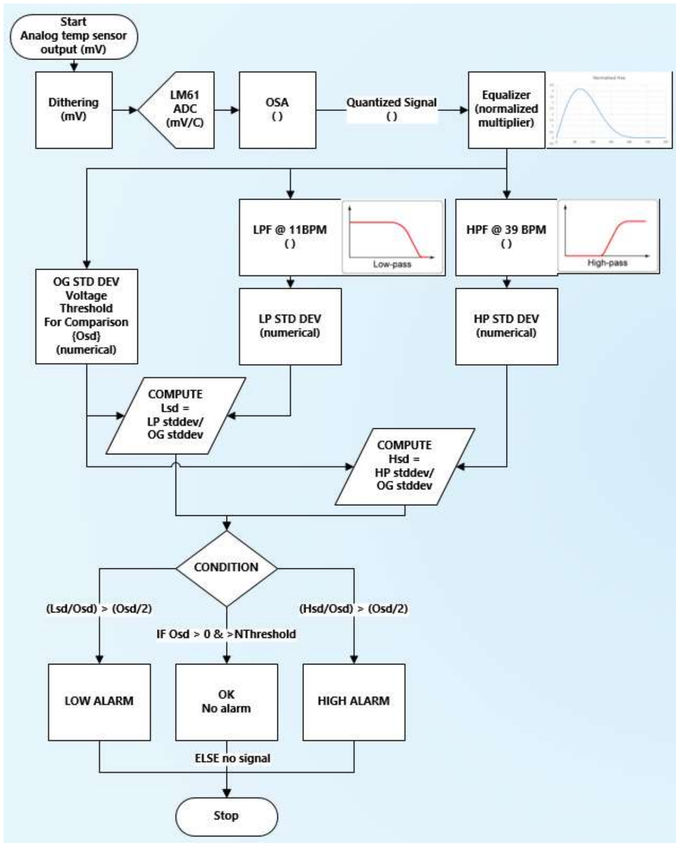


Fig. 2. Drawing of LM61 sensor topography and I/O pins. See reference at the bottom.

Following digital processing, a small speaker serves as an alarm when dangerous respiratory conditions are detected.



II. Methods

A. Data collection

Breathing on the LM61 sensor while connected to the circuit, the Arduino begins taking 3000 samples at a sampling rate of 100ms.

B. Signal Conditioning

The LM61 was placed in a bag to minimize drift but there still was some error, the variance of this error, which is close to the total measurement noise, it was .31 using equation 1.

$$\sigma_{tot} = \sqrt{\sigma^2{}_{sensor} + \sigma^2{}_{quant}} \tag{2}$$

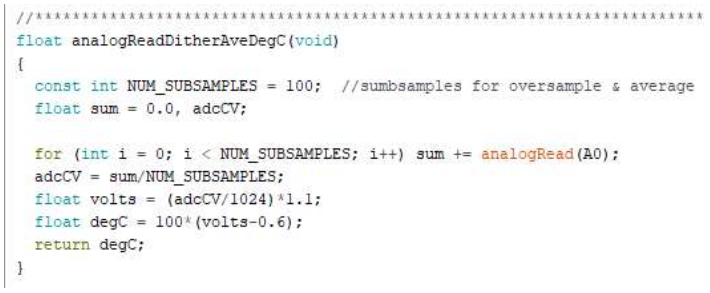
Adding sufficiently large “dither” noise to the sensor signal prior to ADC sampling. Dithering allows the total noise to be larger than the sum of the sensor and the quantization noise to suppress it. Oversampling gives more data points to average which yields a more accurate representation of what the signal is. Dithering increased the SNR by ensuring the signal would pass the threshold to be detected by the next increment of the microcontroller’s ADC. The sensor is required to detect a temperature difference as little as 1 degree Celsius. The LM61’s output voltage changes by 10mV per degree Celsius difference. This means the signal the sensor outputs is 10mV peak to peak. Quantization noise is calculated using equation 3.

$$\sigma_{QN} = 0.29 * \frac{Full\ Scale\ Voltage\ Range}{2^{Nbits}} \tag{3}$$

The full scale voltage range in this case is 1.1V and N equals 10 because it uses the 10 bit DAC. Using these values yields a quantization noise of .3115mV. The signal to noise ratio can be calculated by using equation 4.

$$SNR = 20\log_{10}(\frac{signal}{\sigma_{QN}}) \tag{4}$$

Using Equation 4 yields an SNR of 70.9586 as the best possible case, using a maximum signal of 1.1V and the $\sigma_{QN} = 0.29 * \frac{Full\ Scale\ Voltage\ Range}{2^{Nbits}}$ of 0.3115mV.

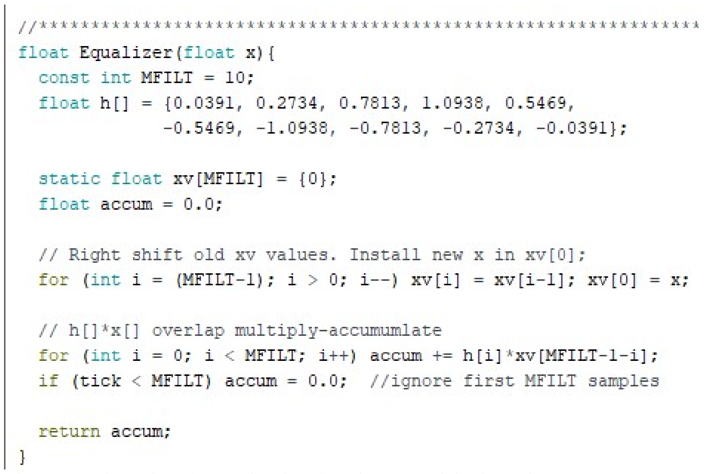


```
//***************************************************************************
float analogReadDitherAveDegC(void)
{
  const int NUM_SUBSAMPLES = 100;  //sumbsamples for oversample & average
  float sum = 0.0, adcCV;

  for (int i = 0; i < NUM_SUBSAMPLES; i++) sum += analogRead(A0);
  adcCV = sum/NUM_SUBSAMPLES;
  float volts = (adcCV/1024)*1.1;
  float degC = 100*(volts-0.6);
  return degC;
}
```

Fig 3. Dither Averaging and Oversample Averaging function.

There is an internal analog reference which is built in to the Arduino and set to 1.1v. Having this internal reference takes out the need for an external hardware dithering circuit, this keep the design small and easier to maintain. In Fig. 4, this function takes in subsamples and sums them, divides the result which is equal to the averaged input from the sensor. This is then converted into voltage, then degrees Celsius which is outputted.



```
//***********************************************************************
float Equalizer(float x){
  const int MFILT = 10;
  float h[] = {0.0391, 0.2734, 0.7813, 1.0938, 0.5469,
              -0.5469, -1.0938, -0.7813, -0.2734, -0.0391};

  static float xv[MFILT] = {0};
  float accum = 0.0;

  // Right shift old xv values. Install new x in xv[0];
  for (int i = (MFILT-1); i > 0; i--) xv[i] = xv[i-1]; xv[0] = x;

  // h[]*x[] overlap multiply-accumumlate
  for (int i = 0; i < MFILT; i++) accum += h[i]*xv[MFILT-1-i];
  if (tick < MFILT) accum = 0.0;  //ignore first MFILT samples

  return accum;
}
```

Fig 4. Equalizer function code, showing the pass of the kernel.

The equalizer function is intended to increase the gain of higher frequency signals. This functionality is implemented via an FIR filter with a parabolic kernel and a low pass filter.

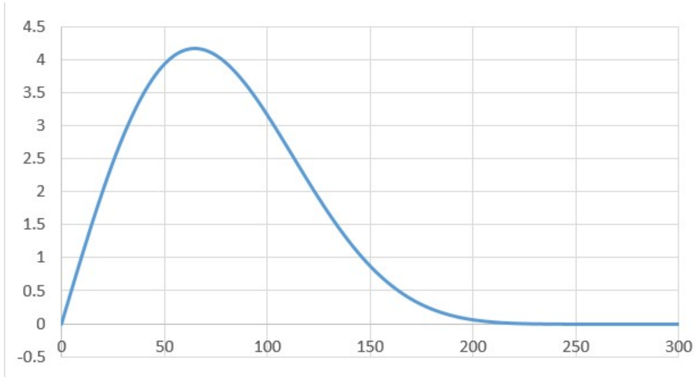


Fig 5. Normalized output of Equalizer shown as numerical gain versus BPM.

Due to the parabolic nature of the kernel, very high frequencies that are impossible breathing rates are suppressed — the corresponding noise being suppressed as well. In this way the equalizer functions similarly to a band pass filter, the band pass being the range of BPM used for the whole system.

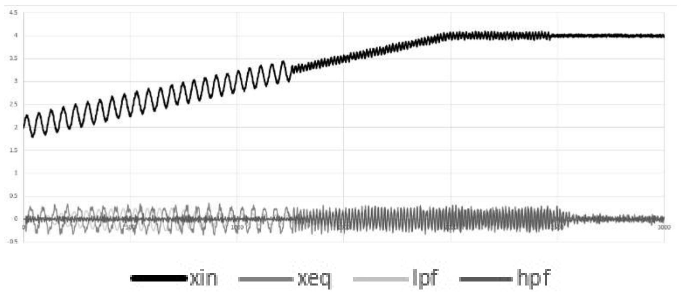


Fig 6. Equalizer plot from breathing sample showing the input showing amplitude versus sample number.

NEED TO FIX VERTICAL AXIS FOR FIG 6 AND 7 AND DESCRIBE WHAT IT IS

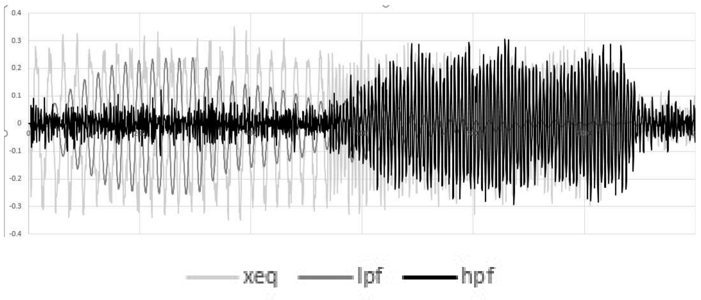


Fig 7. The expanded graph of Fig 7 showing only xeq, lpf and hpf, as signal amplitude versus BPM.

As can be seen in fig 6, the equizer successfully removes drift from the desired signal, and in the expanded image in fig 7, the gain of the signal increases in a ramp proportional to the increase in BPM. At the expense of some signal attenuation at low frequencies (less than10 BPM) the signal is increased in a linear ramp by a gain of 2 at 20BPM, 3 at 30BPM and the signal is multiplied by a gain of 4 at 40BPM. The ramp can be seen in fig 6. While the gain increase at higher frequencies is integral to successful detection of high breathing rates, it comes at a measurable cost.

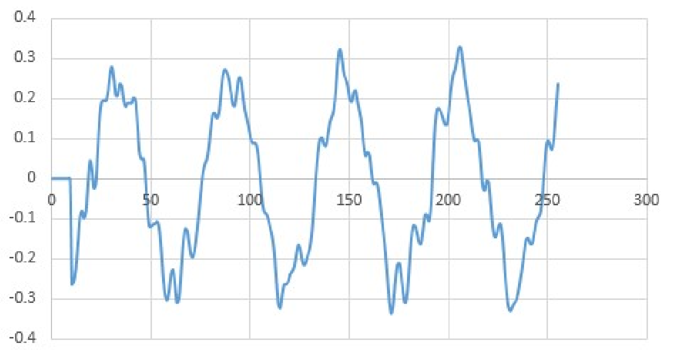


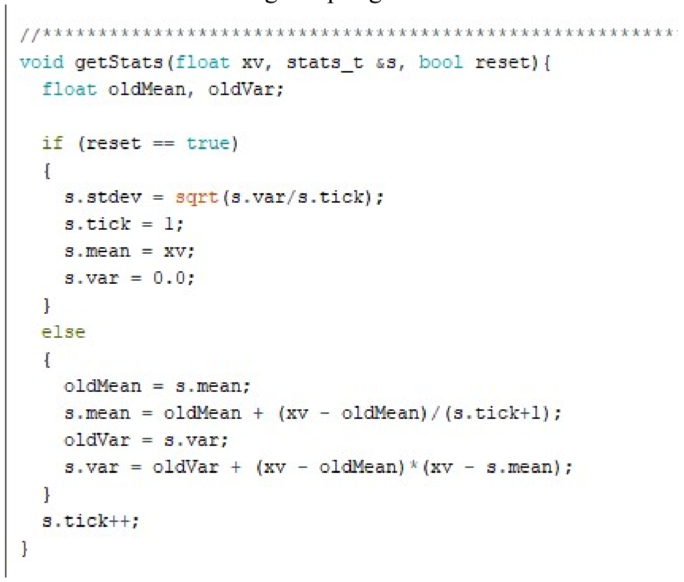
Fig 8. An excel graph of the equalizer output processing real breathing signal measured amplitude in mV versus sample number.

NEED TO DESCRIBE SAMPLE RATE (SAMPLE INTERVAL)

As can be seen in fig 8, noise and harmonics on the signal are also multiplied by the gain ramp, and at a BPM of 20 or higher, they are at least doubled.

C. Filtering

Below is the code for calculating running mean and running standard deviation during sampling.


```
//**********************************************************
void getStats(float xv, stats_t &s, bool reset){
  float oldMean, oldVar;

  if (reset == true)
  {
    s.stdev = sqrt(s.var/s.tick);
    s.tick = 1;
    s.mean = xv;
    s.var = 0.0;
  }
  else
  {
    oldMean = s.mean;
    s.mean = oldMean + (xv - oldMean)/(s.tick+1);
    oldVar = s.var;
    s.var = oldVar + (xv - oldMean)*(xv - s.mean);
  }
  s.tick++;
}
```
Fig 9. Code for calculating the running mean and standard deviation. It utilizes the formula for calculating variance, shown below in Figure 8.

$$\sigma^2 = \frac{1}{N-1}\left[sum\ of\ squares - \frac{sum^2}{N}\right] \qquad (2)$$

Both the standard deviation and mean are used for statistical analysis and viewing the changes in data over time. As seen in the following figures, sudden changes in amplitude or frequency will affect one or both of these methods of data analysis. However, each of the methods has its own advantages and disadvantages.

In these cases, the standard deviation statistic is more responsive to temperature deviations than the mean is because standard deviation is calculated using the power instead of amplitude of the signal, which is done by squaring the difference between the sample value and the mean. This allows standard deviation to be more sensitive to changes. It is harder to see where amplitude changes occur in the sample set when viewing the running mean as the mean value almost immediately decreases. However, when viewing the running standard deviation of the amplitude burst sample set, it is clear when the amplitude change occurs and when it stops. The mean is the better method to detect the change in frequency since frequency doesn't affect the amplitude, which represents the temperature. However, since the goal is to detect changes in temperature and not the rate at which they occur, standard deviation is clearly the better method of data analysis.

The lab had us design a resource-efficient IIR filters with good performance, but are subject to implementation based stability limitations. The use of Matlab along with Arduino was used, the purpose of Matlab was to create the direct calculation from the transfer function numerator and denominator, it also created the cascade of second order sections, and all of these were placed into the Arduino code. The low pass filter shown in Fig. 11 was a 9th order, and the high pass filter shown in Fig. 14 was a 15th order. These were derived via Matlab, taking the direct constants for the cascade of second order systems and putting them into Arduino.

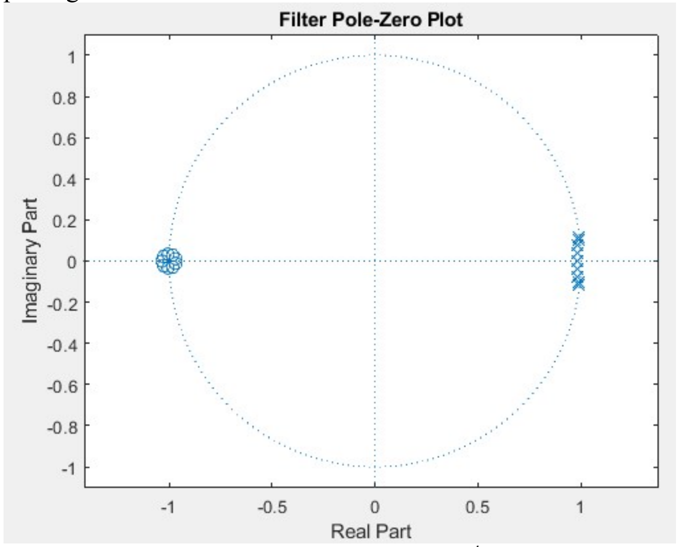


Fig. 10. Pole-Zero Plot of Low Pass Chebyshev, 9th order to demonstrate stablility with a cutoff frequency of 11 BPM.

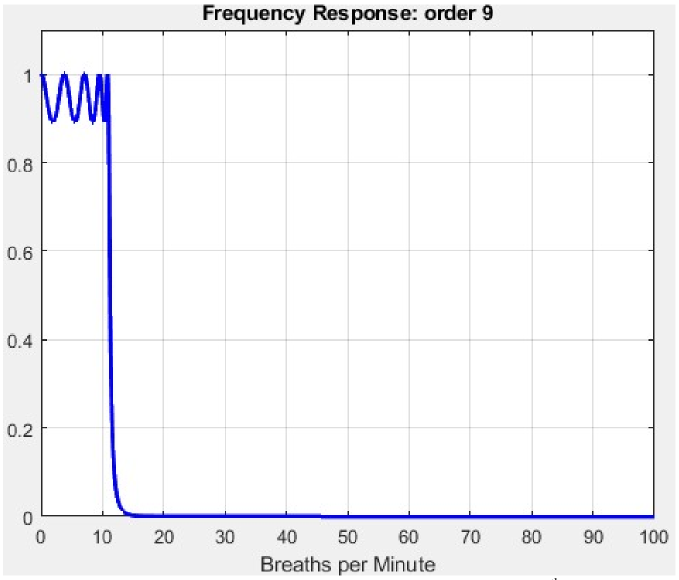


Fig. 12. Frequency Response Plot of Low Pass Chebyshev, 9th order with a cutoff frequency of 11 BPM.

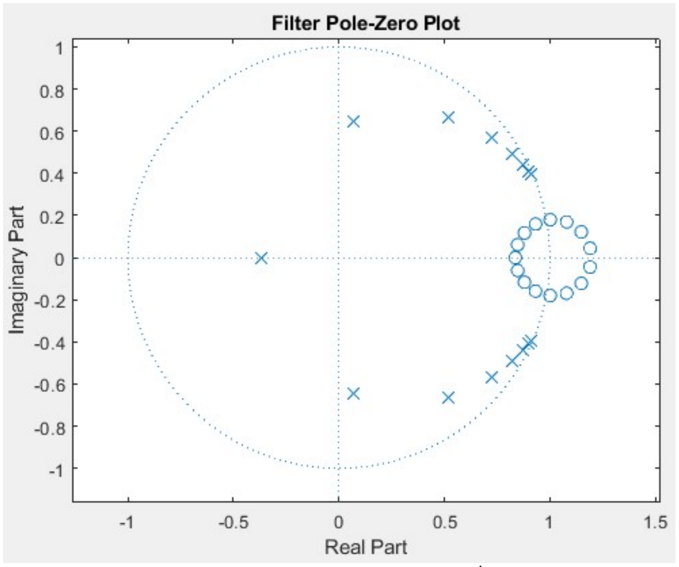


Fig. 13. Pole-Zero Plot of High Pass Chebyshev, 15th order to demonstrate stablility with a cutoff frequency of 39 BPM.

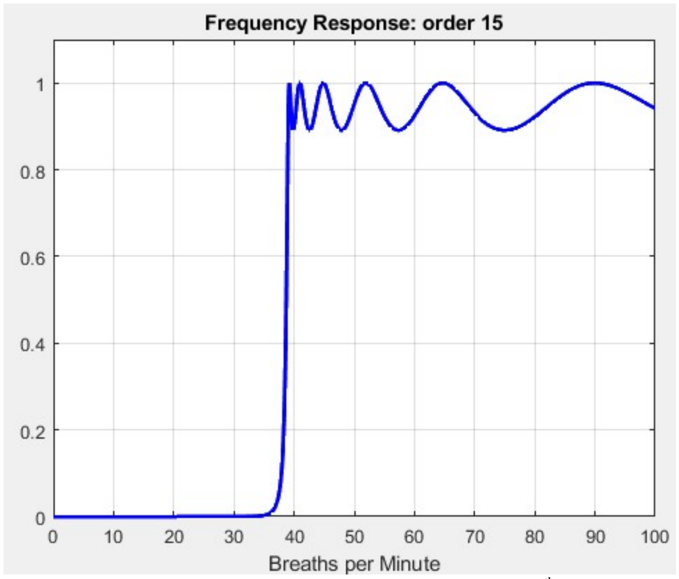
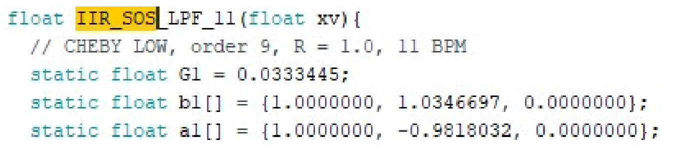
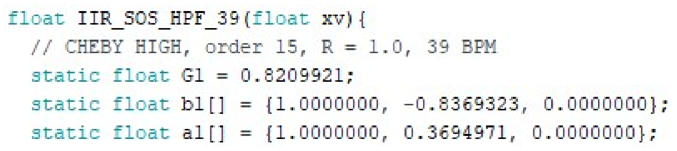


Fig. 15. Frequency Response Plot of High Pass Chebyshev, 15th order with a cutoff frequency of 39 BPM.



```
float IIR_SOS_LPF_11(float xv){
  // CHEBY LOW, order 9, R = 1.0, 11 BPM
  static float G1 = 0.0333445;
  static float b1[] = {1.0000000, 1.0346697, 0.0000000};
  static float a1[] = {1.0000000, -0.9818032, 0.0000000};
```

Fig. 16. Low Pass Filter Chebyshev, cutoff frequency at 11BPM and to the 9th order. This is made up of cascaded second order system from MATLAB.

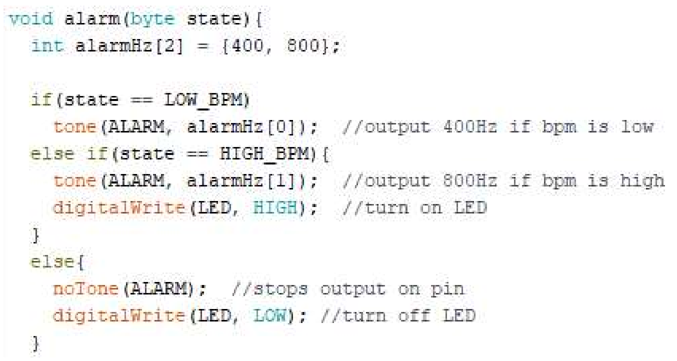


```
float IIR_SOS_HPF_39(float xv){
  // CHEBY HIGH, order 15, R = 1.0, 39 BPM
  static float G1 = 0.8209921;
  static float b1[] = {1.0000000, -0.8369323, 0.0000000};
  static float a1[] = {1.0000000, 0.3694971, 0.0000000};
```

Fig. 17. High Pass Filter with a cutoff frequency of 39 BPM and to the 15th order. This is made up of cascaded second order system from MATLAB.

A Chebyshev filter was chosen over a Butterworth because the pass band ripple doesn't matter in this application and the transition bandwidth is sharper. Choosing the highest possible order for both the high pass and low pass allowed for a sharper cut off frequency. The highest order for the high pass was the 15th order while still being stable as shown in Fig. 12. The highest order for the low pass filter is the 9th order and is stable as shown in 9.

D. *Alarm*



```
void alarm(byte state){
  int alarmHz[2] = {400, 800};

  if(state == LOW_BPM)
    tone(ALARM, alarmHz[0]);  //output 400Hz if bpm is low
  else if(state == HIGH_BPM){
    tone(ALARM, alarmHz[1]);  //output 800Hz if bpm is high
    digitalWrite(LED, HIGH);  //turn on LED
  }
  else{
    noTone(ALARM);  //stops output on pin
    digitalWrite(LED, LOW); //turn off LED
  }
```

Fig. 18. Code for the alarm and when it goes into each mode.

This is a simple state machine to determine when the alarm is turned on and at what tone. The project requires an auidable and visible warning for the high bpm, when it goes into the high bpm state, the tone is 800hz and an LED is turned on. At low bpm, the requirement is a distinctly different audiable noise so in Fig. 15, the tone is set to 400hz which is clearly different from 800hz. If it is in neither state, it naturally falls into no tone and no LED since there is no need for a warning.

III. Results & Analysis

The system was tested using a simulated breathing test vector prior to using it with actual breathing. The test vector had both low and high frequencies in it located near the cutoff frequencies of the low and high pass filters that were implemented in the system. This was done with the purpose of testing how well the filters differentiate between different frequency bands. Fig 19 demonstrates the detection of low and high frequency breathing in the test vector. When the standard deviation of the output of either the low pass or high pass filters exceeded half of the standard deviation of the equalized input, the corresponding alarm would be raised, along with the LED if the breathing frequency was too high.

Following the successful test of the system using the test vector, actual breathing tests were carried out, with low (Fig. 20), good (Fig. 23) and high (Fig. 26) breathing rates. Figures 22, 25 and 28 demonstrate the effectiveness of the system at determining what frequencies were present the input breathing signal. For the low frequency breathing test, the standard deviation of the low pass filter was above the threshold for detection, as was the standard deviation of the high pass filter for the high frequency breathing test. For normal breathing, the filters did not pass through sufficient amounts of high or low frequencies, so that neither of the filters standard deviations passed the threshold for either alarm to be raised.

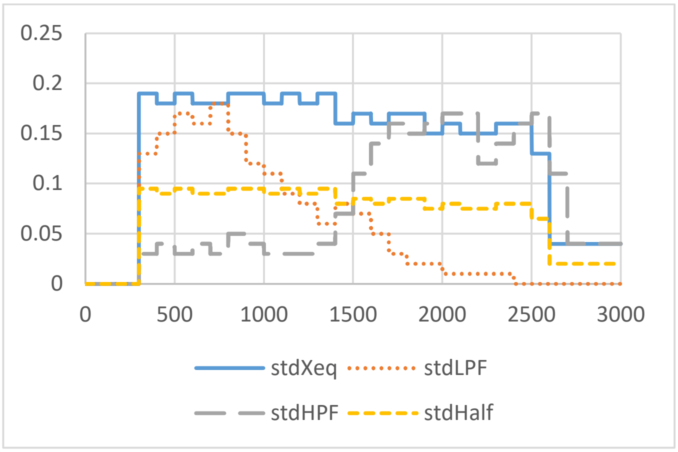


Fig. 19. This the standad deviation of the equalizer, low pass and high pass outputs. Stdhalf is .5 the value of the stdXeq and is used as a thereshold to set off the alarm if either filter standard deviation goes above it. The x axis is samples and the y axis is amplitude.

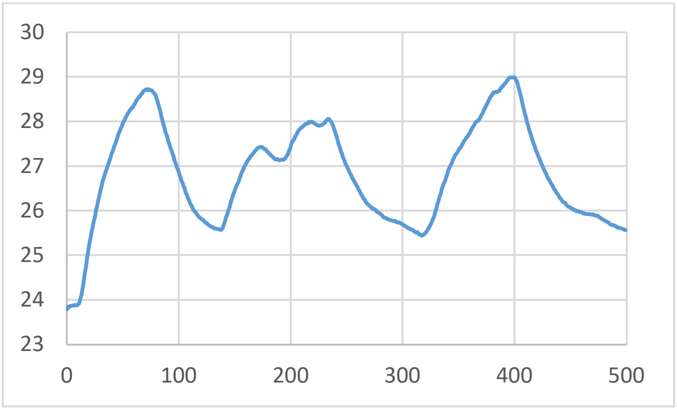


Fig. 20. This is the input signal for actual Low Frequency Breathing. The x axis is samples numbers and the y axis is degrees C.

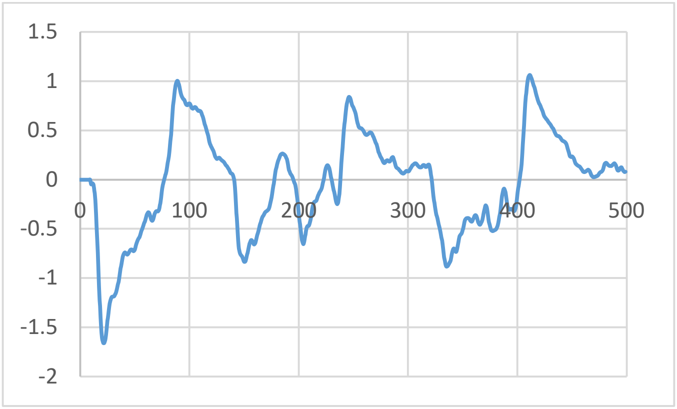


Fig. 21. This is the input signal for Low Frequency Breathing put through the equalizer. The x axis is samples and the y is amplitude.

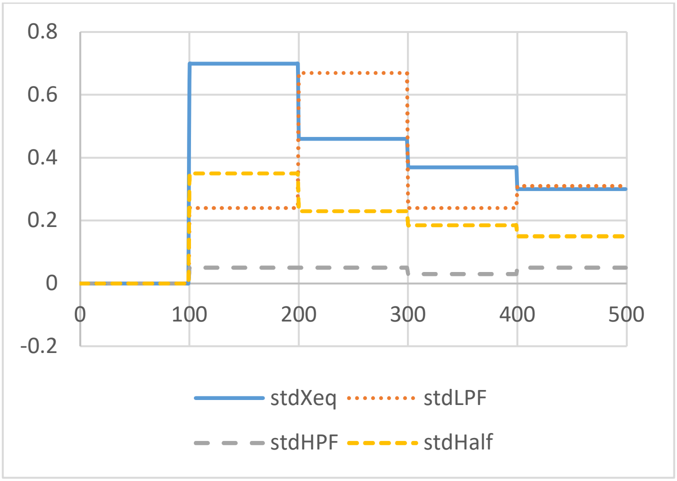


Fig. 22. This the standard deviation of the actual Low Frequency Breathing. The x axis is samples and the y axis is amplitude.

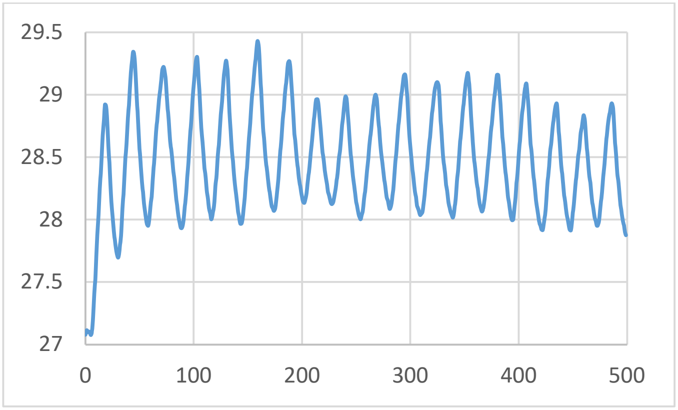


Fig. 23. This is the input signal of the acutal good breathing, the x axis is samples and the y axis is amplitude.

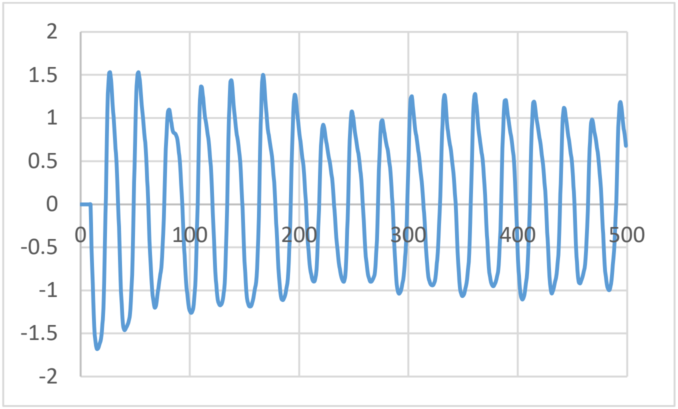


Fig. 24. This is the input signal of the actual good breathing after the equalizer, the x axis is amples and the y axis is amplitude.

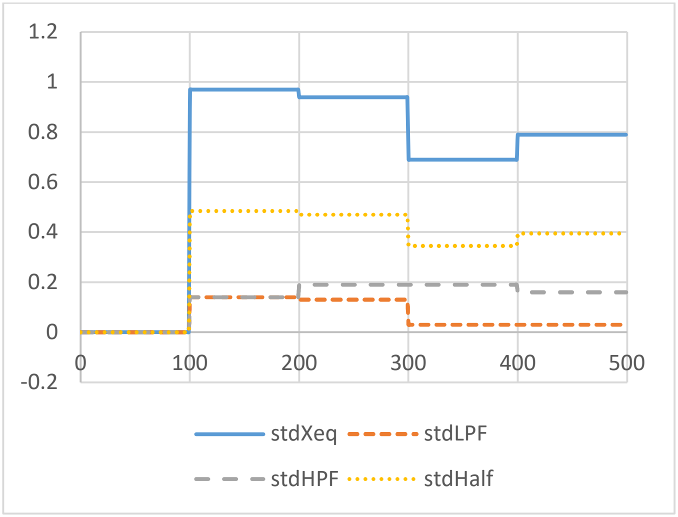


Fig. 25. This the standard deviation of the acutal good breathing, the x axis is samples and the y axis is amplitude.

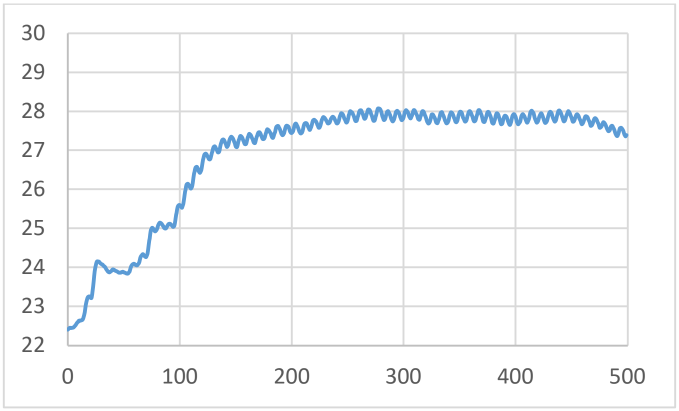


Fig. 26. This is the input signal of the acutal High Frequency Breathing, the x axis is samples and the y axis is amplitude.

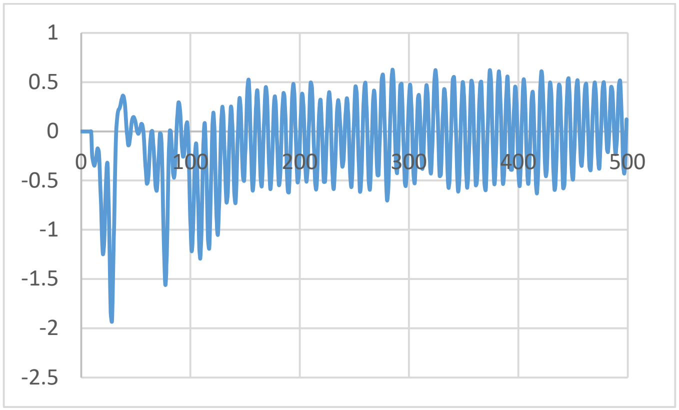


Fig. 27. This is the input signal of the acutal High Frequency Breathing after the equalizer, the x axis is samples and the y axis is amplitude.

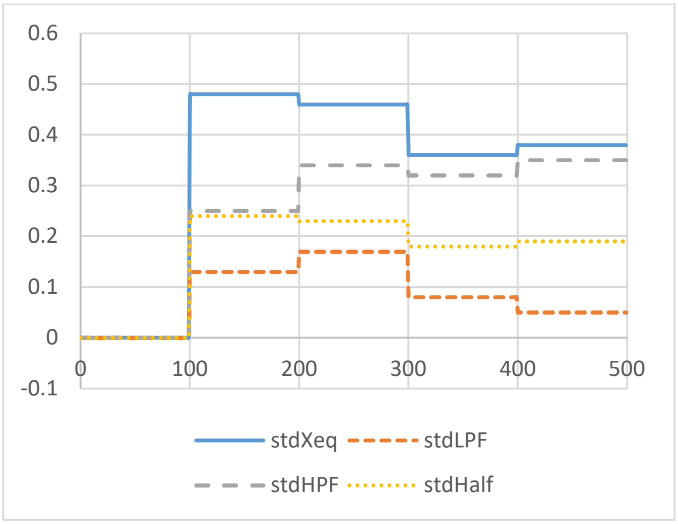


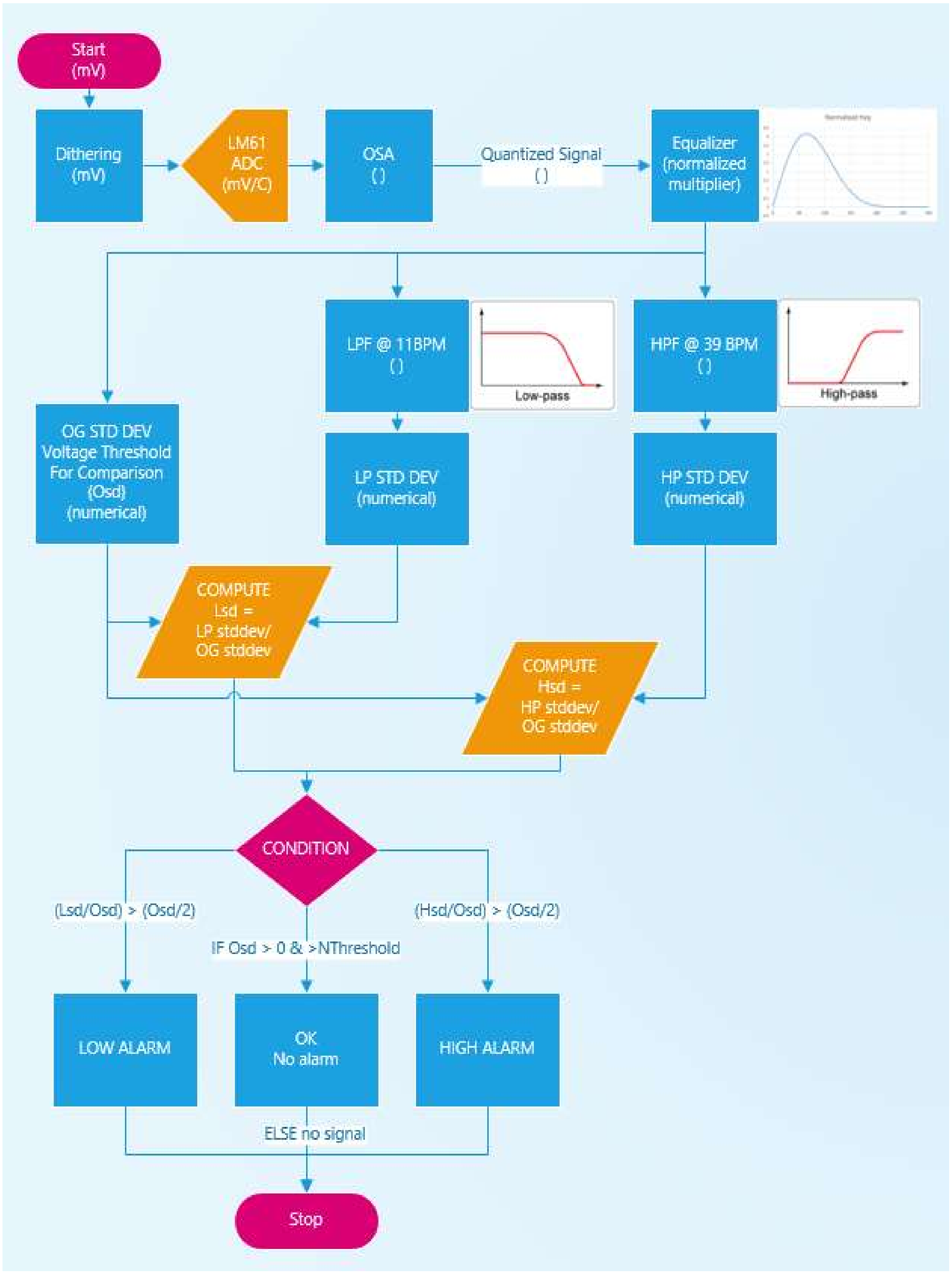
Fig. 28. This the standard deviation of the actual High Frequency Breathing, the x axis is samples and the y axis is amplitude.

IV. References

Texas Instruments, “LM61 Temperature Sensor Datasheet .” Dallas, TX.

Appendix

Appendixes, if needed, appear before the acknowledgment.


Start (mV)
Dithering (mV)
LM61 ADC (mV/C)
OSA ( )
Quantized Signal ( )
Equalizer (normalized multiplier)
Normalized freq
LPF @ 11BPM ( )
Low-pass
HPF @ 39 BPM ( )
High-pass
OG STD DEV Voltage Threshold For Comparison (Osd) (numerical)
LP STD DEV (numerical)
HP STD DEV (numerical)
COMPUTE Lsd = LP stddev/ OG stddev
COMPUTE Hsd = HP stddev/ OG stddev
CONDITION
(Lsd/Osd) > (Osd/2)
IF Osd > 0 & >NThreshold
(Hsd/Osd) > (Osd/2)
LOW ALARM
OK No alarm
HIGH ALARM
ELSE no signal
Stop