1

Final Project: Breathing Rate Detection System

Marco Smith, Christian Aguilar, Eloy Ramirez, Sushil Bhattari, *IEEE*

***Abstract*— An Arduino Uno with a ATMEL MEGA 328P microcontroller was used to create a breathing rate detection system used to detect pneumonia in infants and young children. A LM61 temperature sensor was used to capture the breathing rate in real time. This signal was then filtered and the active filter was determined by comparing the standard deviations of the signals, with the highest energy indicating what was active. Based on the filter that was active, an alarm would output a tone or not play anything at all. There were three specific alarm tones. One for a low breathing rate, one for a high breathing rate and lastly one for a non-functional system.**

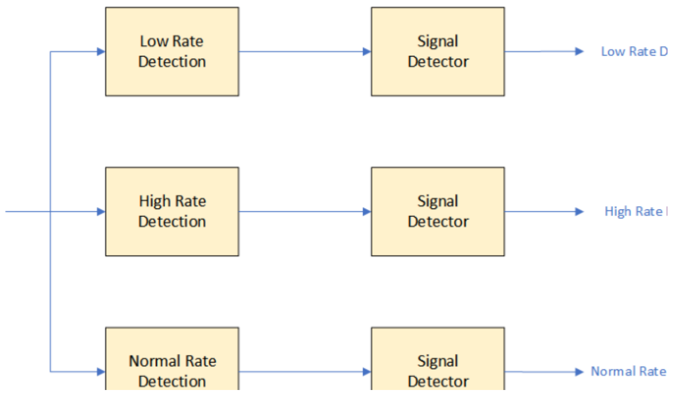
I. Introduction

The breathing rate detection system was produced to detect potential respiratory infection in young children. The specifications for the system were to have it detect breathing rates below 12 beats per miniature and to play one warning tone and to detect a breathing rate above 40 beats per minute and to play a different warning tone. There is also a third tone that will play when the system is not functioning as intended. The system was built on 5 different systems within it. The systems include data acquisition, signal pre-processing, breathing rate detection system, warning logic and user signaling. These systems were constructed from a number of subsystems. In order to finish this system an understanding from all previous labs was demonstrated as this system encompasses all subjects and concepts learned until this point.

II. Analysis

A. Breathing Rate Detection System

In the development of a breathing rate detection system, there are procedures and objectives that students are to follow. Students are provided with an outlined set of functions in a high-level block diagram system. In this project, students are to further develop the breathing rate detection subsystem which includes a series of various types of filters, lowpass, highpass, and bandpass combined to detect different rates of breathing. These filters were used for energy detection and filter banks as it played a crucial role for the system. Filters were all Chebyshev filters which offered fast roll off and disregarded the ripples for band pass. These filters were crucial to minimize the overlapping in the filters.



First part of the procedure was to create a low range breathing rate detector. This is required to detect energy below a rate of 12 breaths per minute. In the system, this low pass filter will be used to pass signals below 12 breaths per minute and attenuates signals with higher rate. Students designed an IIR low pass filter with fifth order with a corner frequency of 12 breaths per minute. This low pass filter would detect the low breathing rate which indicates an abnormal condition or issue with the sensor. Students were to plot an impulse and frequency response graph using the MATLAB tools, the C-Code generated within and Arduino. Figure 1 shown below represents the impulse response.

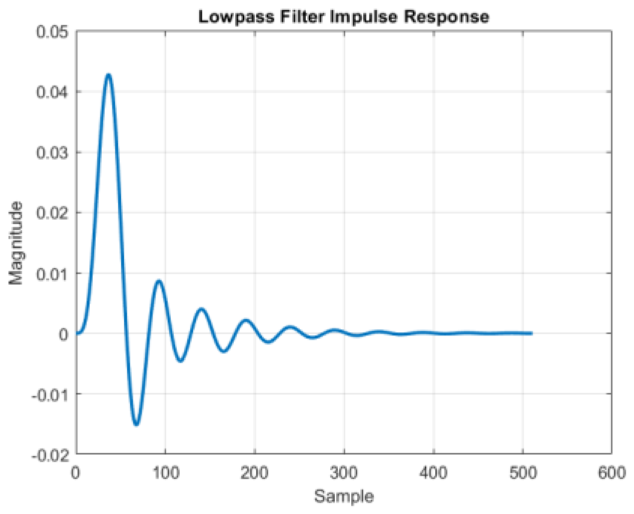


Figure 1: 5th Order Low Pass Filter Impulse Response Filter Bank

1

Using the data collected from Arduino the frequency response was created for the low pass filter. It is represented by Figure 2 which is shown below.

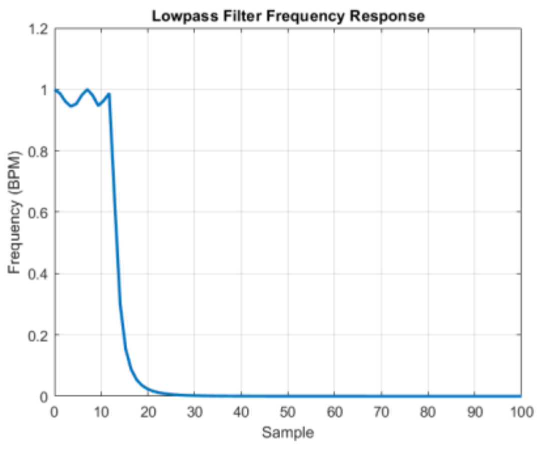


Figure 2: 5th Order Low pass Filter Frequency Response Filter Bank

For the last part of the Low Pass Filter, students were to capture the input and output of the test vector which was used to verify that the filter is working properly in the time domain. After students adjusted the Arduino code to verify the filter and captured the data. Using that data, a low pass filter test response was created. Figure 3 shows that the filter is working properly as everything below 12 BPM is passed while everything above attenuates signals.

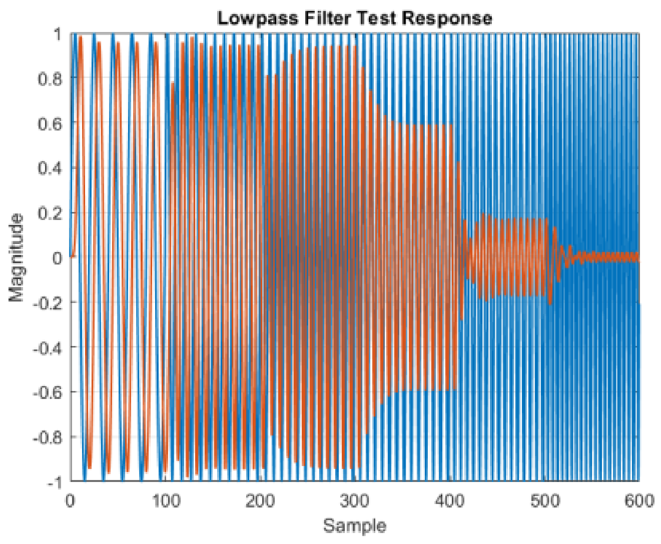


Figure 3: 5th Order Low Pass Filter Test Response

The second phase of the project, students were to create a mid-range breathing rate detector. Low pass filter collected the data which were below 12 BPM and the mid-range breathing rate detector is required to detect signals above a rate of 12 BPM but below 40 BPM. To do this operation, the band pass filter is used. Signals with energy outside of this range will be attenuated. Students were to design an IIR bandpass filter using the fifth order which has a lower corner frequency of 12 BPM and an upper corner frequency of 40 BPM. The transition bandwidth of each corner needed to be less than about 5 BPM. For this portion, students used the Chebyshev filter to provide fast filter roll-off and used a ripple of 0.5dB. This bandpass filter was designed to detect the normal range of breathing rate.

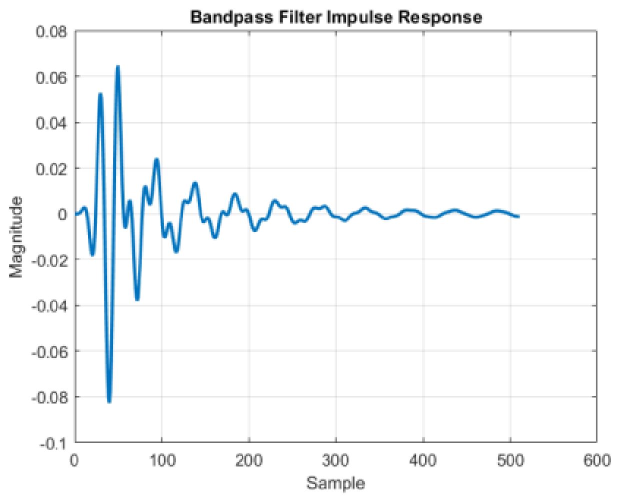


Figure 4: 5th Order Bandpass Filter Impulse Response Filter Bank

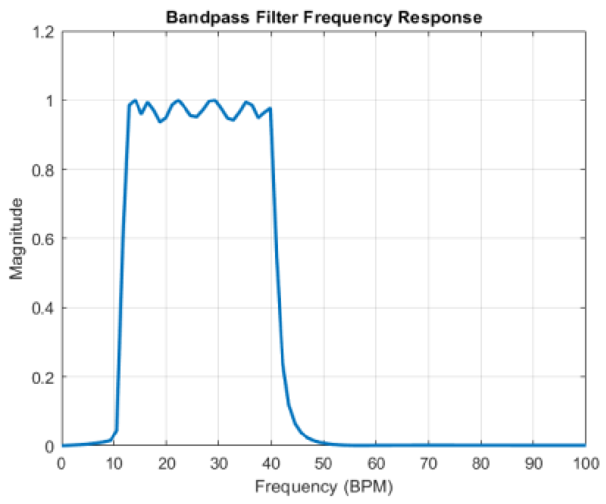


Figure 5: 5th Order Bandpass Filter Frequency Response Filter Bank

After students plotted the impulse and frequency response, students tested those responses to make sure the output was doing the work correctly. With the adjusted Arduino coding and MATLAB commands, students were able to properly plot the test response. Students checked to make sure that the Filter was working correctly and the test response showed that it was working correctly as shown in Figure 6 below.

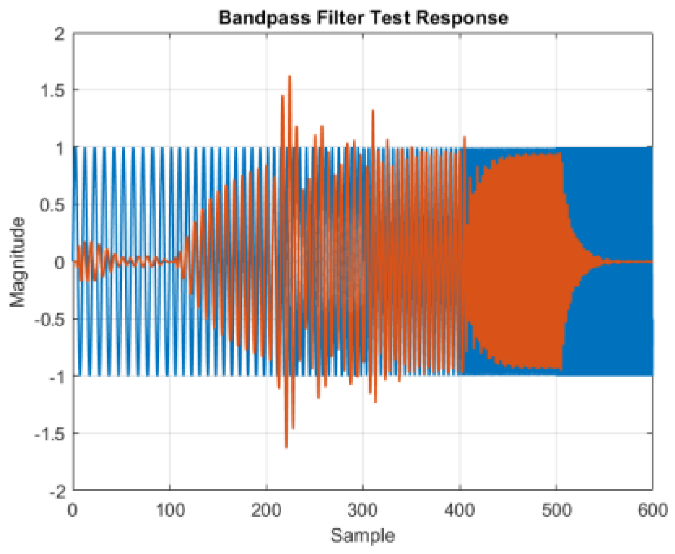


Figure 6: 5th Order Bandpass Filter Test Response

The last filter that was being used to detect the pass signals with rates above 40 breaths per minute was a high pass filter. This filter was designed to have a cut off frequency of 40 BPM, and with the 0.5db ripple. Students adjusted the code to generate an impulse and frequency response for the high pass filter. It's shown below as Figure 7 and Figure 8. This filter detected the signals that were above 40 BPM which indicated that the patient had pneumonia.

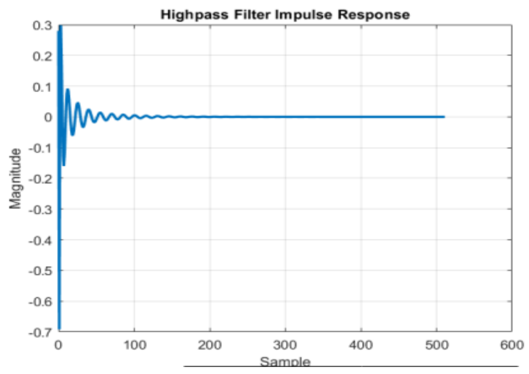


Figure 7: 5th Order High Pass Filter Impulse Response Filter Bank

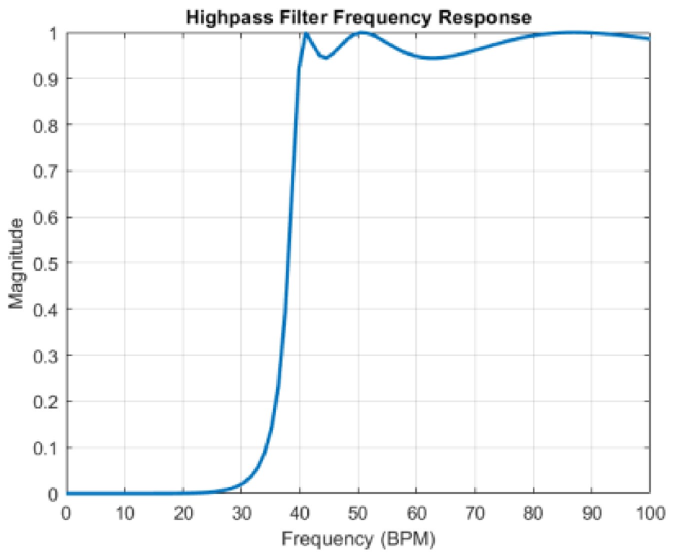


Figure 8: 5th Order Highpass Filter Frequency Response Filter Bank

Students performed a test to make sure that the filters were properly functioning and were successful in doing so. Test response captured the input and the output of the test vector generator to verify the filter is working correctly in the time domain. As shown in Figure 5 below.

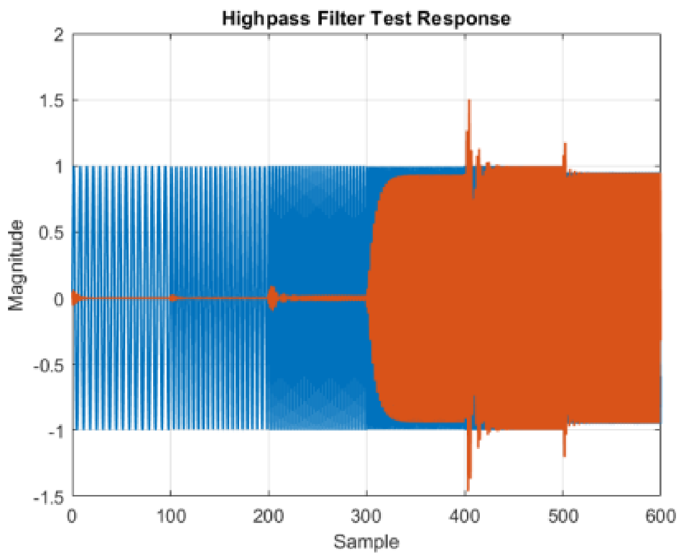


Figure 9: 5th Order High Pass Filter Test Response

B. Standard Deviation

Using running statistics, the standard deviation of each filter was used to determine which filter was active at the time. This is used later on to trigger the warning logic of the system. The standard deviation of the output for each filter was calculated and determined the energy in each filter. The standard deviation for every filter was calculated every 10 samples. Figures 10, 11, and 12 show the standard deviations of the lowpass, bandpass, and highpass filters respectively. The filters in parallel had an average execution time of 1739.12 microseconds.

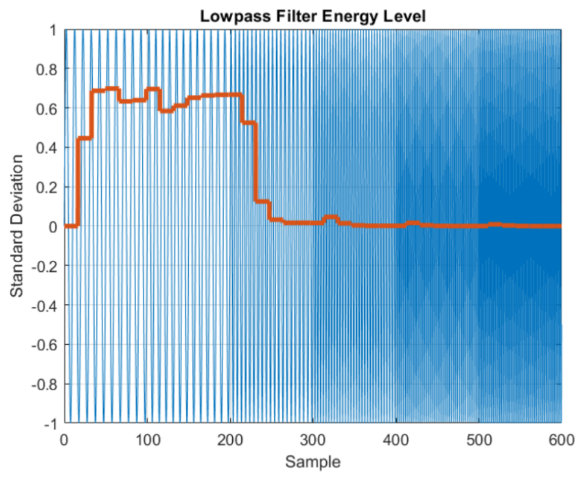


Figure 10: Lowpass filter standard deviation output

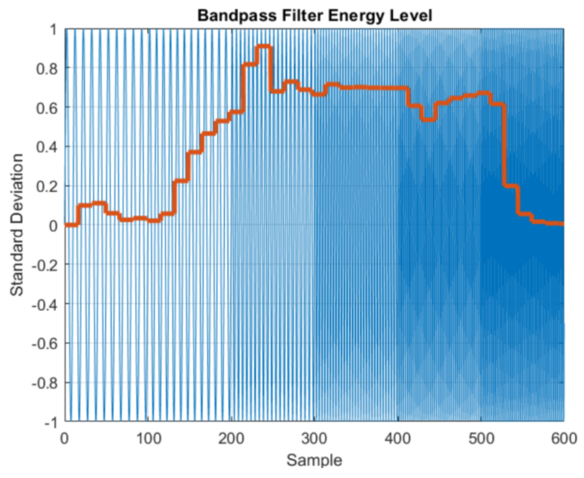


Figure 11: Bandpass filter standard deviation output

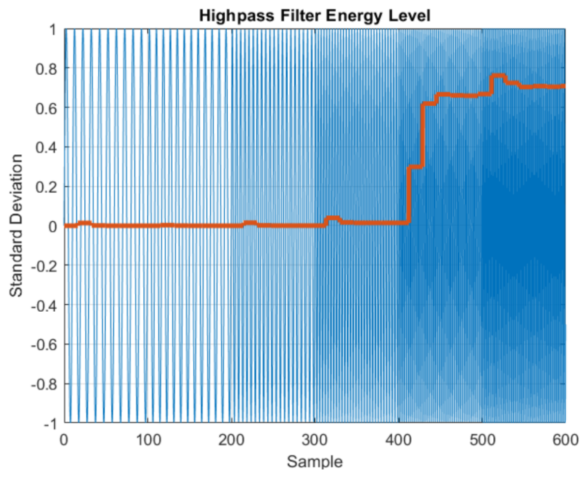


Figure 12: Highpass filter standard deviation output

C. Equalizer

Typically, the LM61 temperature sensor holds identical properties to that of a leaky integrator. This means that at low frequencies, there is more likely to exist some noise or unwanted variation in the signal. To address this issue, an equalizer component was designed to mitigate the drifts and variations that come in from the temperature sensor. This in turn would create a more stable input signal that will be seen by the rest of the monitoring system. To craft this equalizer, a generic Finite Impulse Response (FIR) function was devised in ArduinoC. This method would take in the long datatype coming from the sensor as well as the current sample count of the system and combine them with a pre-established 4-tap FIR filter (as seen in figure 0.0).

D. Low Pass Filter Output Vs Equalizer Output

The low pass filter was used to minimize the possibility if high frequency noise enters the detection system. This noise can be introduced by the USB port and or the temperature sensor itself. To implement this crucial noise filter, a cutoff frequency of 70 BPM was used as the breathing rate. In order to attenuate the incoming noise, a FIR lowpass filter was used that was 61 samples long. When needed, the filter parameters could be adjusted within the lowpass function and C code to obtain the desired response.

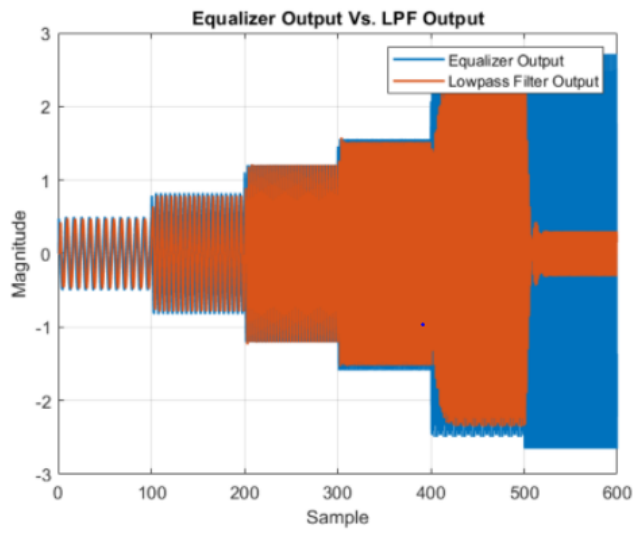


Figure 13: Equalizer Output vs LPF Output

E. Warning Logic

Warning logic was integrated to be able to differentiate the different states of each filter. The state being driven by the outputs of the standard deviation calculations for the lowpass, bandpass, and highpass filters. The tones were being produced from a small Piezo speaker. Starting at `alarmCode = 0`, the system will produce a low tone signifying that there is no input above the 0.7 threshold. The next state, the lowpass, produces a slightly higher pitched tone signifying a low breathing rate. The midpass state has no tone signifying a regular breathing rate. Then the highpass has a high pitched beeping tone to signify a high breathing rate.

F. Noise Level Calibration

Noises are produced and present is every portion of the room. For example, the vents on the ceiling would impact the data from the sensor which would create problems in the system. These external noises can tamper the LM61 temperature sensor, which is the reason behind the noise level being calibrated before taking the breathing rate data. To calibrate the external noise the input value, the low pass filter, equalizer and the three filter banks with standard deviation calculations were integrated.

During the process, the equalizer and FIR lowpass filter were used to reduce high frequency noise and other external noises by altering the frequency response of the input signal. After that, students had to find a way to detect how the system was working and whether it was working properly. Students used the outputs of the standard deviation calculation of all the outputs to detect if the system was working correctly. The standard deviation calculations were read after 100 ticks of the counter. After running the data and making sure that the threshold of all three outputs were correct, the system was actively reading data from the sensor. Students used breathing methods to properly make sure that the system was working correctly. If the values didn't exceed the noise threshold, it would appear that the system was not functioning as it supposed to or the patient was not breathing or the sensor was not working properly. Students performed numerous tests to make sure that system was working properly and successfully collected that data.

G. System Breakdown / Signal Processing

To improve the accuracy of the sensor, a Digital-to-Analog Converter (DAC) circuit was introduced into our sensor design to act as an initial buffer to enhance the quality of the input signal before it is even processed by the equalizer. The DAC circuit improves the signal's quality by inserting an insubstantial amount of noise to eliminate inconsistencies and errors that may have occurred during the quantization process.

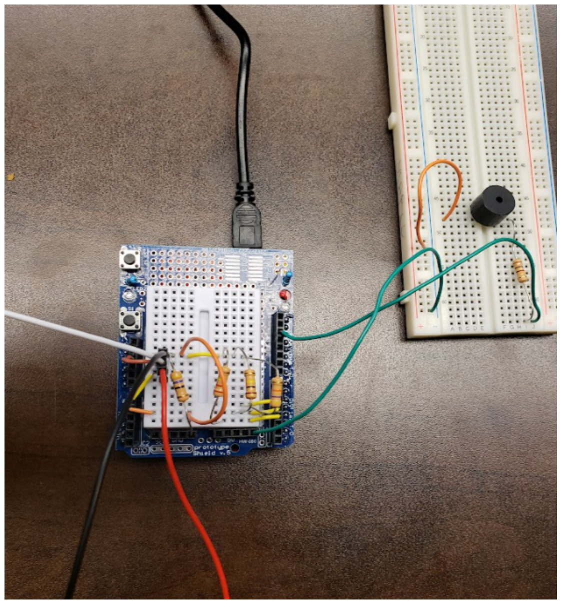


Figure 14: Hardware build

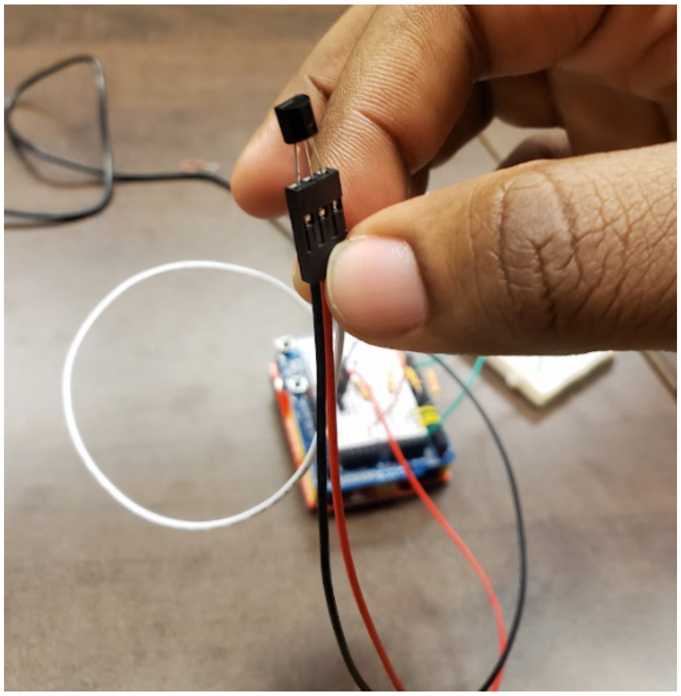
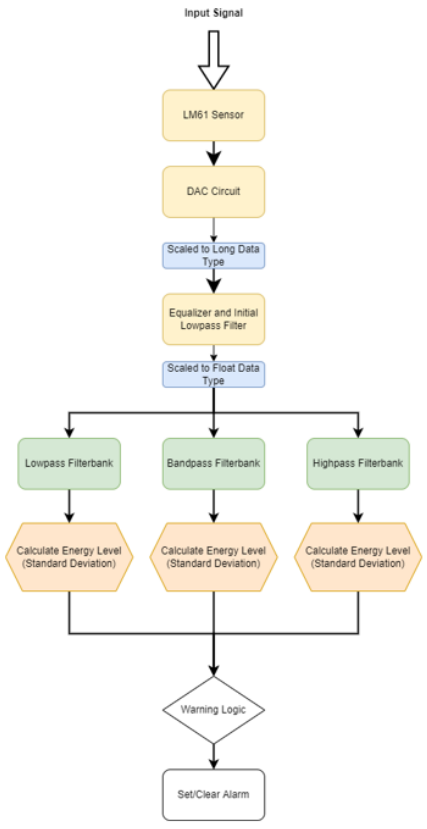


Figure 15: Hardware build


Input Signal
LM61 Sensor
DAC Circuit
Scaled to Long Data Type
Equalizer and Initial Lowpass Filter
Scaled to Float Data Type
Lowpass Filterbank
Bandpass Filterbank
Highpass Filterbank
Calculate Energy Level (Standard Deviation)
Calculate Energy Level (Standard Deviation)
Calculate Energy Level (Standard Deviation)
Warning Logic
Set/Clear Alarm
Figure 16: Block Diagram of the project

Once the initial input signal has been oversampled, dithered , and averaged it is then scaled into a long datatype to be compatible with the equalizer filter method that was setup in the Arudiuno code. Once it has been equalized and passed through an initial low-pass filter (70 BPM cutoff), the signal is then re-scaled back into a float datatype to be compatible with the rest of the system's logic. The signal is then run in parallel across the three filter banks which calculate their respective filter's energy level. These energy levels are then analyzed by our warning logic code which moderates the state of the system. Once the correct state has been determined, the system then fires off an alarm function that triggers the corresponding audio output through the attached Piezo speaker according to what state the system is in. This process in its entirety has been measured to take approximately 1739.12 microseconds to complete when run with a max sample size of 3600.

III. Conclusion

The breathing rate detection system functionality was a result of a completely functional subsystem as well as a balance of accuracy to memory usage of certain aspects of the system. The most accurate system was designed at the affordability of the memory restrictions of the ATMEL MEGA 328P microcontroller. The first portion of the procedure was data acquisition, focused on taking the input from the LM61 temperature sensor and making that a usable signal. A dither circuit was built on the Arduino prototyping shield with a variety of resistors running on it (200k ohms, 100k ohms, 50kohms). These are also in parallel with a 47 ohm resistor in series to ground. Across all tests, the dither and oversampling, filtering, and alarm logic worked as expected to make a well processed signal and expected system output.

The preprocess accomplished this by using a low pass filter with a cutoff frequency of 70 bpm to eliminate any high frequency noise that may enter the system. This was then passed on to an equalizer before entering the filter banks. The equalizer allowed the system to focus mainly on the breathing rate. When implementing the equalizer, low memory performance was experienced when below available memory dipped below 300 bytes. The filter banks consisted of three filters. First, a low pass with a cutoff frequency of 12 bpm to detect a low breathing rate was used. Next was a bandpass filter with a cutoff frequency of 12 bpm to 40 bpm, this filter was used to detect the normal breathing rate. Lastly, there was a high pass filter with a cutoff frequency of 40 bpm. This was used in order to detect high breathing rates often associated with pneumonia.

In order to detect the active filters some statistics were used. The standard deviation of the filters were compared and the one with the highest standard deviation was determined to be the active filter. After the active filter was found, logic would be used to figure out if it was one that required an alarm to trigger the low & high pass filter. Using the warning logic, four different states were identified using the standard deviation of each filter. The four states each have a designated sound that would alert the user of the current state of the system. When all the filters are in parallele the average execution time is 1739.12uSec.

Although these low-cost components are no substitute for the high-end medical sensors, the breathing rate monitor using the LM61 temperature sensor can in fact accurately detect small variations in an input signal that can warn a user of potential danger by accurately processing changes in frequency and magnitude, while combating noise across the system. With children from the ages 11months to 5 years having the particular risk of acute respiratory infection, or pneumonia, utilizing signal processing techniques, allow for engineers to use low-cost components to create tools that can accurately detect the warning signs of acute respiratory infection.