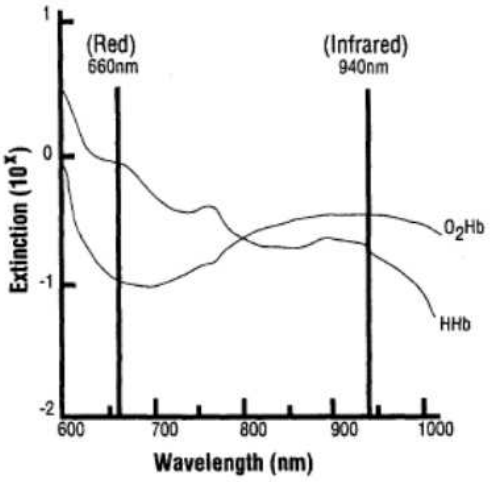
Wireless Pulse Oximeter Project Report
Project ID# micro13JM299

Introduction

My wife recently gave birth to triplets. They were ten weeks premature and were in the Neonatal Intensive Care Unit for a month. While there, they were connected to monitoring equipment to monitor their heart rate, respiratory rate, temperature, and at times blood oxygen levels. At times their heart would stop briefly which would trigger an alarm, and a nurse would come to the rescue. This is a common occurrence among premature babies. The babies were not released from the NICU until they went at least seven days without one of these events. But how would I know if this happened while they were at home? What I needed was a heart rate monitor. I searched the internet for a heart rate monitor that was affordable for the average consumer. The few I did find were expensive and could not be used unless the baby was supervised by an adult. This was because of the risk of strangulation from the wires connecting the baby to the monitor. There needed to be a wireless transmitter on the baby that would not interfere with the baby's movements, such as on his or her foot. Such a device could also be used to alert a parent of other life threatening conditions such as Sudden Infant Death Syndrome (SIDS). That is what prompted me to build a wireless pulse oximeter.

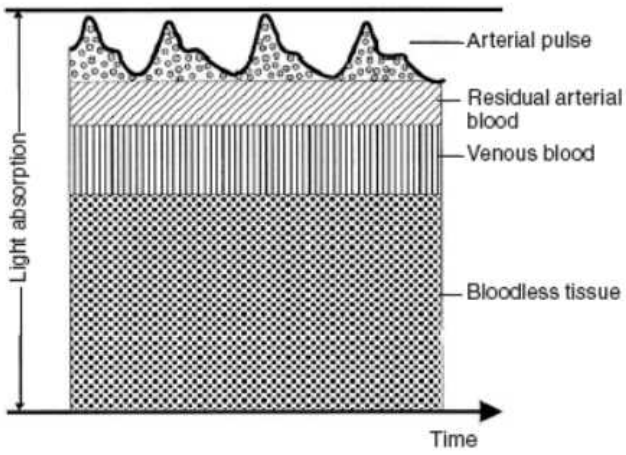
Theory of Operation

My first challenge was to make a wired pulse oximeter. After much research, I came across a project report by the Worchester Polytechnic Institute. Their project was much more in-depth than mine, but I learned that oxygenated hemoglobin absorbs light differently than un-oxygenated hemoglobin. At 660nm un-oxygenated hemoglobin absorbs about ten times as much light as oxygenated hemoglobin. This can be seen in the following graph.



O_2Hb is oxygenated hemoglobin and HHb is un-oxygenated hemoglobin, also called reduced hemoglobin.

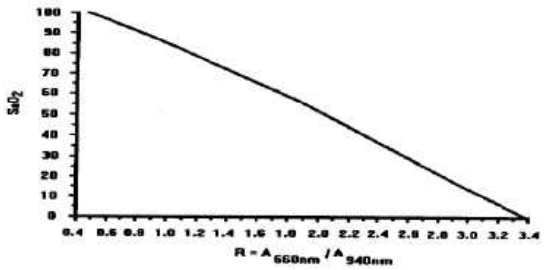
Although all tissue in the body absorbs light, the light absorbed by only the blood can be calculated by observing the change in light level as the heart pumps the blood through the arteries and veins. This is demonstrated in the following figure.



The area of minimum light absorption to maximum light absorption can be attributed entirely to blood. This area is approximately 1% of the total light absorption and will be referred to as the AC portion of the signal, although, in my design, this is not actually AC. By differentiating this signal, an intermediate value called the Normalized R ratio can be calculated. The DC portion of the signal in the following equation is the level of minimum light absorption.

$$R = \frac{\left(\frac{ACrms660nm}{DC660nm}\right)}{\left(\frac{ACrms940nm}{DC940nm}\right)}$$

This Normalized R ratio represents the ratio of oxygenated hemoglobin to un-oxygenated hemoglobin. The normalized R ratio can then be calculated based on empirical data. Empirical oxygen saturation curve is depicted below.



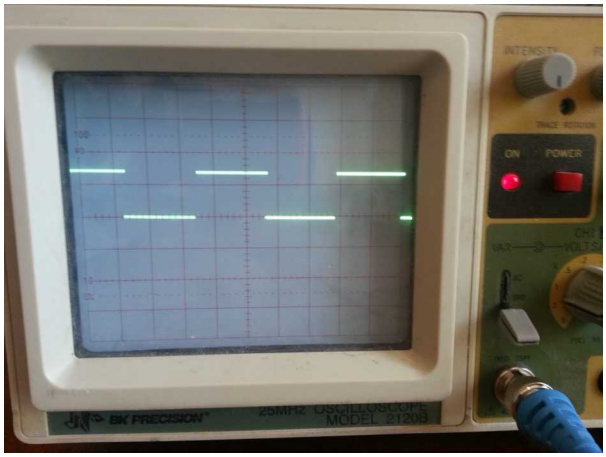
Since this graph is very close to linear, the following equation could be used:

$$SpO2 = 110 - 25R$$

This equation may have to be adjusted to accurately calculate SpO2 using different hardware.

Project Design

Using this theory of operation I started building a pulse oximeter. The design uses an Arduino microcontroller with an LCD display. The sensor uses a red LED that emits light in the range of 620-625 nm and an infrared LED that emits light in the 940-950 nm range. These LEDS are pulsed alternately and their light intensity is measured using a TSL-230RD Programmable Light-to-Frequency converter. The TSL-230RD outputs a square wave signal of 50% duty cycle. The frequency of this signal is proportional to the intensity of the light it receives. The TSL-230RD output can be seen in the picture.

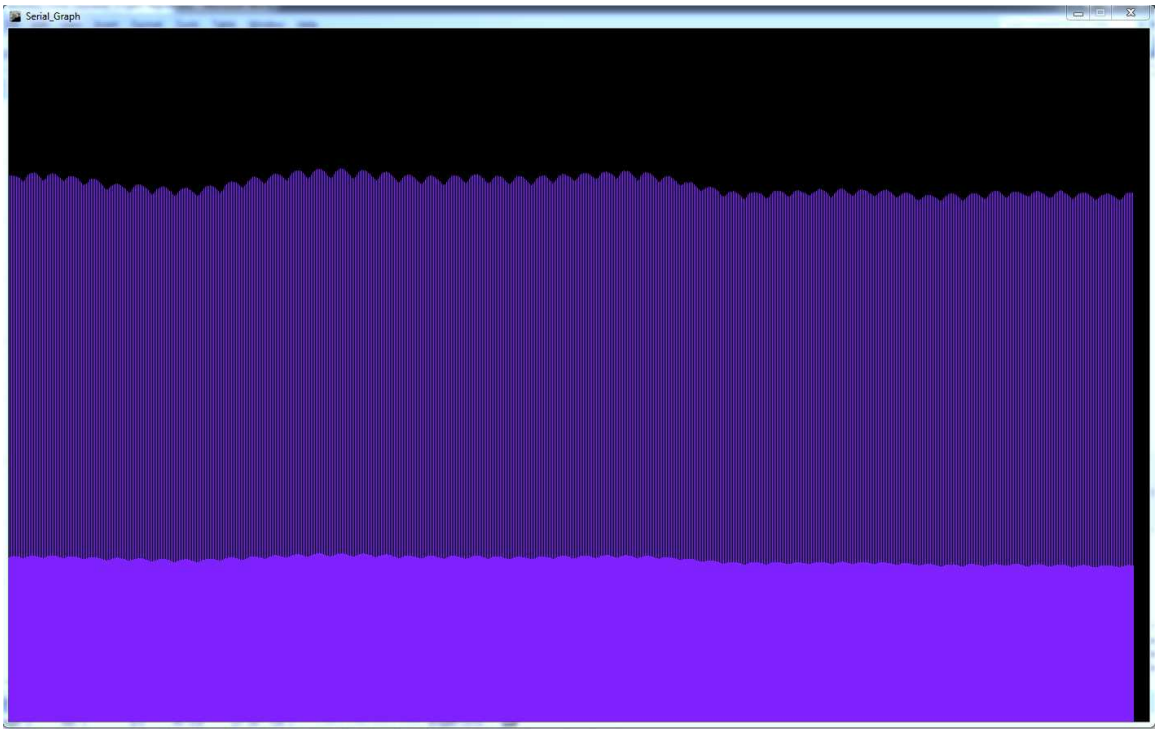


TSL-230RD Output

The Arduino software measures the pulsewidth of the square wave signal. Since it is 50% duty cycle, this is exactly half of a full cycle. This width is stored and compared to the next pulse width. If the pulse widths are getting larger then the light intensity is decreasing. The largest value is found and stored as maximum pulse width. The minimum value is also stored in a similar fashion. The time between minimum pulse widths is the time between heartbeats.

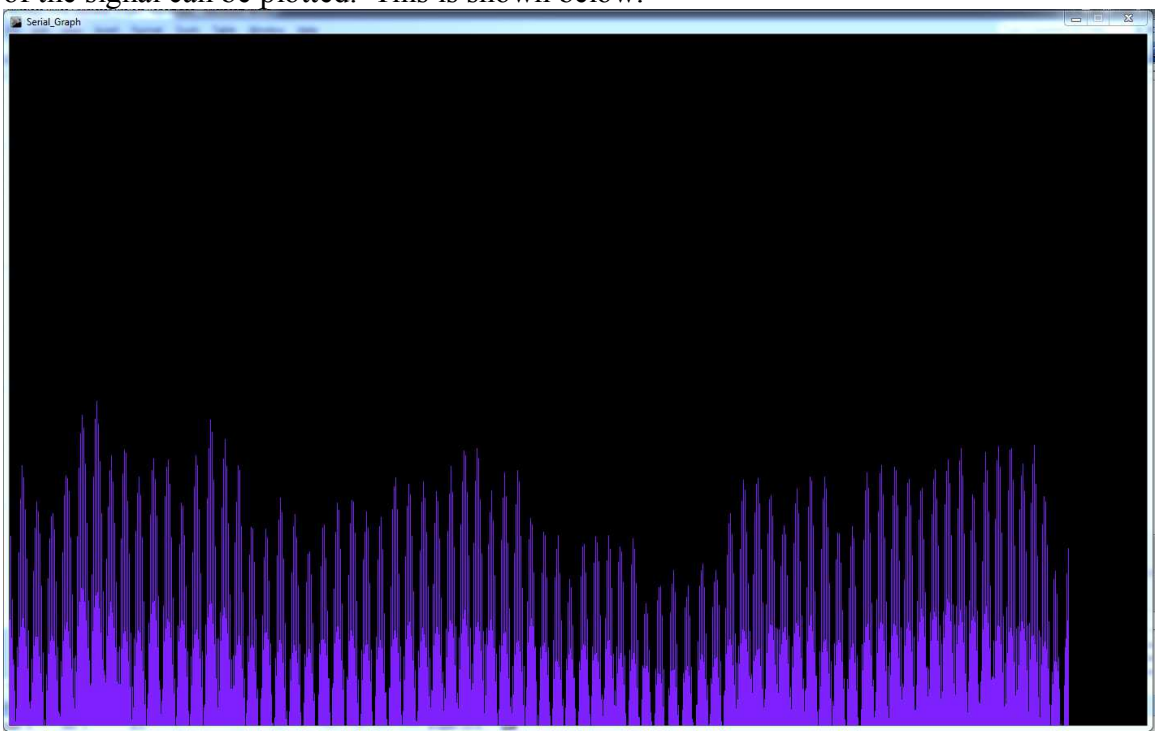
Arduino microcontroller and LCD display

Although this worked to find a pulse rate, the slightest movement caused a lot of noise. A lot of effort was spent trying to eliminate this noise. One method is to average a number of samples. This worked well. I also incorporated an exponential smoothing equation which worked even better. Exponential smoothing eliminated most of the noise and allowed quicker initial pulse rates and SpO2 measurements. A graph of the pulsewidths coming into the Arduino is shown below. This graph plots one pulsewidth of the IR LED, then it plots one pulsewidth of the red LED. The x-axis is time, and the y-axis is pulsewidth. The red LED is the lower plot and the IR LED is the upper plot.



The pulse rate is determined by measuring the time between each minimum pulsewidth. The last five pulse rates are averaged to reduce rapid fluctuations. There is a pulse rate associated with the red LED, and one is associated with the IR LED. The average of these two pulse rates are displayed on the LCD display.

To determine the SpO2, the RMS value of the AC portion of each LED must be calculated. By subtracting the minimum pulse width from the total pulse width, the AC portion of the signal can be plotted. This is shown below.



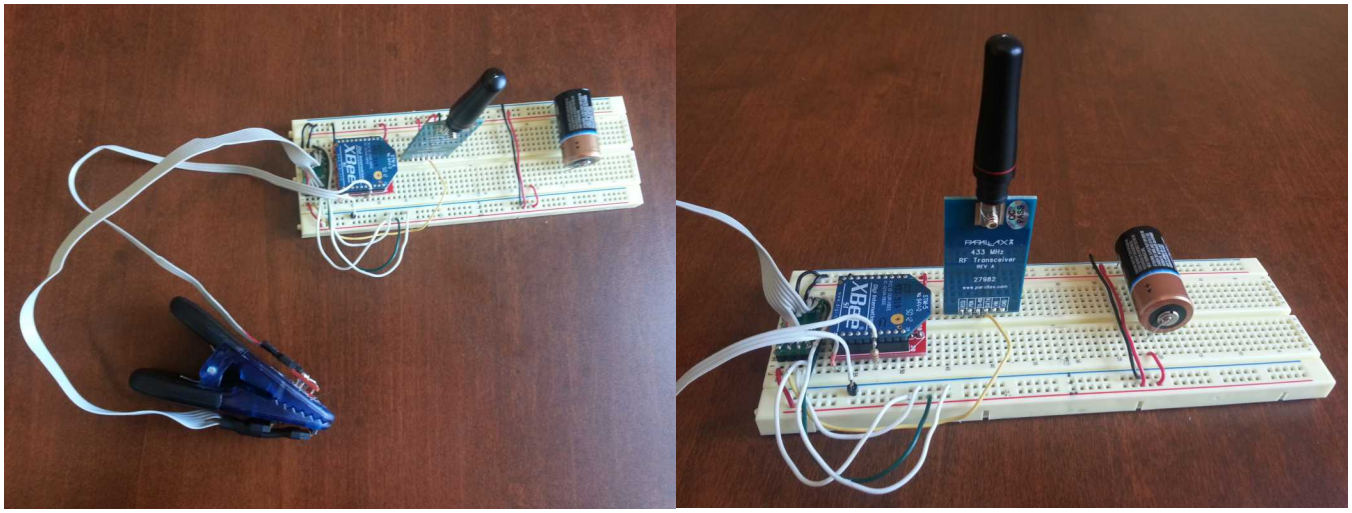
Since this isn't actually AC, the RMS value of each signal is the same as the average value. Each pulsewidth is summed, and when the minimum pulsewidth is reached they are averaged. It can be seen in the graphs that this value fluctuates quite a bit. This is due to any movement, such as breathing, or even unwanted light entering the sensor. To reduce these fluctuations, I take an average of two hundred complete cycles instead of one. Averaging these RMS values gives a more stable SpO2 value. The last five SpO2 results are again averaged in the same manner as the pulse rate before displaying the results on the LCD display.

After getting the pulse oximeter to work reliably, I tried it on other people, but on some people it didn't work well. I had to change the sensitivity values of the light sensor to accommodate different people. I found that there were only four reasonable settings that worked. The software waits for five seconds, and if no heart beat is detected, it changes to another sensitivity setting. It continues this cycle until a heart beat is received. This worked well but requires four more wires to the light sensor. For the wireless version, the light sensor is hardwired to setting 0, but this feature could easily be implemented using four channels on an XBee module.

Wireless Finger Sensor

Once I had a wired system working, I needed to transmit these signals wirelessly. To transmit the control signals to the remote sensor, I used a pair of XBee modules. These modules were able to send the LED switching signals, but they were not fast enough to detect the slight changes in the square wave signal coming from the light sensor, so I used a 433MHz transmitter from Parallax (P/N 27982) to send the output signal to the base unit. With more time I may be able to find a small transceiver that can do both. The remote unit is powered by a 3 volt lithium camera battery. The remote system draws a maximum of 75mA from this battery.

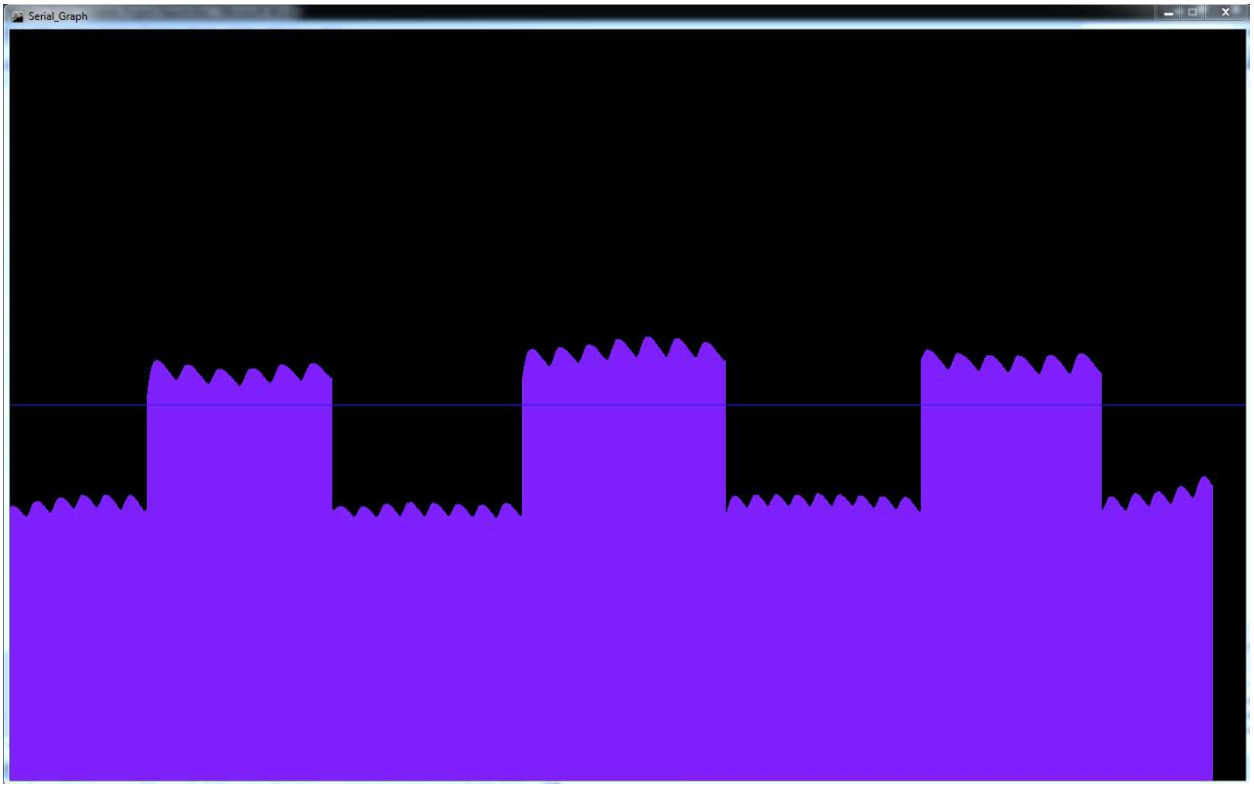
Remote wireless finger sensor



The base unit wireless module was designed to plug directly into the finger sensor wiring. This allows the finger sensor to be moved between wired mode or wireless mode.

Wireless base unit

The wired system measured the width of one pulse with the IR LED on, then measured one pulse with the red LED on. This can be seen in the previous graph of pulsewidths. This became a problem with the wireless system. The wireless transceivers could not send the control signals and receive the response fast enough for the Arduino to accurately measure the correct pulse-widths. This was not a problem with the wired system. I had to change the programming to take 200 measurements with the IR LED on then take 200 measurements with the red LED on. This can be seen in the following graph.



This change actually improved the accuracy of the RMS calculation used in the SpO2 calculation.

Conclusion

The project appears to work, but further testing needs to be done. I need to be able to lower my oxygen level significantly to establish my own empirical data. I am using a store bought pulse oximeter to calibrate my project. When I hyperventilate I can make the store bought unit climb to ninety-nine percent SpO2. I then hold my breath as long as I can, and I can make it get as low as eighty-four percent. This same procedure on my project yielded similar results. When I hyperventilated my project went to ninety-nine percent and when I held my breath, it dropped as low as seventy percent. This is obviously wrong, but it is responding. I adjusted the linear equation to reflect a more accurate SpO2 value. My project now follows the store bought unit, but I can only test it in that small range.

The pulse rate is very accurate and follows the store bought unit very closely.

There is room for plenty of improvements if I had more time. A smaller transceiver module would be a great improvement. I think I could put the whole system on a small Bluetooth HC-05 module, and it wouldn't be much bigger than the battery pack. This could be strapped to a baby's foot, and the results would be sent to the base unit via Bluetooth serial data.

Some programming improvements could be to monitor the status of the remote unit and display an error message if needed. Another improvement would be to display a message when the finger is removed from the sensor instead of displaying the last known results.