ABSTRACT

The following report details the design of a health monitoring system that would transmit real-time information when a radio link was available, and store the information locally when the link was unavailable. This project involved the implementation of three distinct pieces into a single system. The first phase of this project utilized a microprocessor and a heart rate sensor to read the pulse of a subject. The algorithm within the microprocessor either sent the information to be transmitted or saved the information in an SRAM depending on the received signal strength of a signal coming from a computer to the microprocessor. The next phase of this system transmits the information from the microprocessor collecting the data to the computer logging the data, XBEE modules were used for this transmission. Finally the computer read the information from the XBEE and converted the binary transmission into a matrix of the integer values which these binary characters represent. These integers were plotted on a graph which represented the pulse of the subject.

INTRODUCTION

Sensors are widely used in many capacities to attain precise values and even attain values over a continuum of time. Some of the most sensor intensive spaces are hospitals. Some hospital rooms contain a conglomeration of wires and tubes which can make movement around a patient nearly impossible. At the same time the mobility of a doctor such as a surgeon may mean the difference between life and death in some cases. By developing sensors which communicate with their base stations wirelessly the number of wires and tubes in these busy hospital rooms can be lessened and doctors can concentrate more on the patient instead of how they are going to avoid ensnaring themselves.

The specific goal of this project is to provide a monitoring system for patients to move around freely, and data transmission can be delayed as necessary. While storage space is limited in this design, it serves as a proof of concept for systems which could store data for days or weeks at a time, depending on size and power requirements. The data could then be managed at a central location and analyzed for specific patterns and traits. The enhanced mobility would make advanced technology practically useful for many patients who want to carry on with regular activities, or for busy doctors who simply do not have the time to monitor all patients simultaneously.

A photoplethysmograph was used to capture the pulse information. This type of sensor uses light to make a measurement of an organ. In this case, the sensor had a light that shined onto the surface of the subject’s skin. The light was then reflected and scattered, and picked up at a detector cell also on the sensor. The waveform of received light intensity matches the pressure waveform of the subject’s pulse. Further processing could be done on the pulse, but for this simple project it was decided to be unnecessary.

The next component of the system was the memory. Different types of memory are available, and the three major options available are Flash, EEPROM, and SRAM. Flash requires being written in larger blocks, and is better for larger amounts of data than are transferred per sample in this case. Flash was not considered as an option in this design. EEPROM stands for Electronically Erasable Programmable Read-Only Memory. EEPROM is nonvolatile, meaning the data saved is retained without power being applied to the chip. However, at 3ms per write, it has slower write times than SRAM, and it also can only be written a finite amount of times before degrading. SRAM is volatile, meaning it loses data when power is lost. However, it has much faster write times, and can be written an unlimited number of times.

METHODOLOGY

A. Overall System

The design as a whole is shown in Figure 1. The pulse information is read by the pulse sensor, which outputs an analog voltage read by the Arduino. The Arduino automatically saves the data to the SRAM immediately. When the Arduino detects a viable signal strength, it turns on the XBee module, fetches the earliest saved data from the SRAM, and begins to transmit in a FIFO order. The receiving XBee module is connected to a computer via an RS-232 cable. MATLAB code grabs the transmitted data, parses it, then saves and graphs the result.
B. Data Acquisition

Data acquisition was made with a Pulse Sensor Plus photoplethysmograph designed for the Arduino. It has three connections – power, ground, and an analog output. The analog output was fed directly into an analog port on the Arduino configured as an input, where the analogRead() method was used to capture the data.

Samples were captured at 500Hz, or every 2ms, based on the program suggested for use by the manufacturer of the Pulse Sensor Plus. According to the Nyquist theorem, this allows accurate capture of signals up to 250Hz, corresponding to a heart rate of 15000bpm. Although this rate will not be achieved without the aid of Mountain Dew, the higher frequency component is important in capturing some of the details of the nonsinusoidal waveform. Timing was accomplished by configuring the Timer 2 module to generate interrupts every 2ms. The data was then read in the interrupt service routine.

C. Data Storage

The design approach was optimized for poor radio connectivity by always saving data to the SRAM before transmitting. If the link strength was good enough to transmit, the main body of code would fetch data from the SRAM via the SPI module, and then send it to be transmitted via the XBEE module. Conceptually, the data existed on the SRAM in a sliding window. Two variables were used to track the first and last valid address of stored data. If the two values were the same, no data was stored, and nothing needed to be transmitted. If the two were different, all values in between needed to be transmitted. The SRAM was configured to wrap around from the highest address (0x1FFFF) to the lowest address (0x00000), forming a continual loop for sequentially written data bytes. The value from the sensor was stored as an integer type, which is two bytes long. This meant that addresses were always incremented by two, instead of one each time. In this scheme, there is room for 65535 integer variables, which allows 131.07 seconds of untransmitted data to be stored.

D. Wireless Transmission

XBEE Pro modules were utilized with the 802.15.4 wireless standard to transmit the data from the Arduino microcontroller to the Computer which was used to manipulate the data into a useful form. The
The link budget shown in Table 1 describes the propagation losses of the signal.

<table>
<thead>
<tr>
<th>Category</th>
<th>Value</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transmitter Power</td>
<td>+18dBm</td>
<td>[1]</td>
</tr>
<tr>
<td>Antenna Gain</td>
<td>-10 dB</td>
<td>[2]</td>
</tr>
<tr>
<td>Transmitter Losses</td>
<td>0 dB</td>
<td>No figure was found</td>
</tr>
<tr>
<td>Net Transmitted Power</td>
<td>8 dB</td>
<td></td>
</tr>
<tr>
<td>Losses</td>
<td>61 dB</td>
<td>Experimental Value</td>
</tr>
<tr>
<td>Receiver Antenna Gain</td>
<td>-10 dB</td>
<td></td>
</tr>
<tr>
<td>Receiver Losses</td>
<td>0 dB</td>
<td></td>
</tr>
<tr>
<td>Net Receiver Gain</td>
<td>-10 dB</td>
<td></td>
</tr>
<tr>
<td>Net Received Power</td>
<td>-63dBm</td>
<td></td>
</tr>
<tr>
<td>System Performance</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Minimum SNR</td>
<td>28.9 dB</td>
<td>NF = 3.5 dB</td>
</tr>
<tr>
<td>Minimum Detectable Signal</td>
<td>-91.1 dBm</td>
<td></td>
</tr>
</tbody>
</table>

With a maximum loss of 69.1 dBm and using the Friis free space equation, shown in Equation 1, with antenna and receiver gains of unity and a carrier frequency of 2.4 GHz the calculated maximum distance that this signal can travel is 225.8m according to the Friis Free Space Equation which corresponds to the mile of advertised propagation for the XBEE module. Using the log-normal propagation model for a cluttered office space, the predicted range is 10-33.5m, depending on how many TPS reports are in the office. The reference power at 1 meter was experimentally determined to be 0dBm

\[ d = \sqrt{\frac{P_t \times g_t \times g_r \times (f/c)^2}{16 \times \pi^2 P_r}} \]

Equation 1: Friis Free-Space Equation

\[ P_L(d) = P_L(d_0) + 10n \times \log_{10}\left(\frac{d}{d_0}\right) \]

Equation 2. Log Normal Propagation Loss

This signal strength was read by the Arduino from frequently sent out a signal from the computer used to accumulate the data and capturing this signal in the mobile XBEE module. The XBEE module gives a Pulse Width Modulated (PWM) output which corresponds to the received signal strength of the last packet. The larger the PWM pulse, the smaller the received power of the last packet was. To control the times at which the Arduino was sending data the amplitude of this PWM signal was checked and if it was below a certain value then data was transmitted from the Arduino. If the PWM pulse was above a certain value then the data was saved instead of being sent out of the XBEE. The code used to control the output of the XBEE from the Arduino is shown in Figure 1.

**Figure 4. Determining XBee RSS on Arduino**

E. Data Processing

Once the data was captured by the XBEE it was read from the communication port which the receiving XBEE shield was utilizing and into MATLAB as a string of characters. This string was sent into the function strsplit.m. This function transforms a binary array of character variables, received from the XBee, into a matrix of integer variables.

This array is passed back to MATLAB and is added to the array of all of the data points that were previously logged. If this matrix is longer than 100 data points then the most recent 100 points are plotted. If the matrix is shorter than 100 data points then all of the data points are plotted. An updated plot is generated at each iteration of the loop which collects data and plots data. A diagram of the algorithm which the computer utilizes to save and plot the data is shown in Figure 5.
RESULTS

The system is capable of measuring and transmitting data over the link, and displaying a graph of the result, as shown in Figure 6.

![Graph of captured heart rate data](image)

**Figure 6. Captured Heart Rate Data**

When configuring the XBEE modules a master-slave configuration was originally attempted. This configuration associated the XBEE modules correctly but did not have successful two-way communication. After some research, a peer-to-peer network was set up and the destination address of one XBEE was set equal to the serial number of the other XBEE and vise-versa. This made two-way communication possible and simplified the configuration parameters in question by making a successful connection completely dependent on the baud rate and the firmware version instead of also making communication dependent on the six other parameters which are used to configure master-slave relationships.

When building the project, there were issues with setting the sampling rate and the UART speed, especially when debugging. When the baud rate of the UART, used both for debugging or sending data to the XBee, was too low, the program would crash after saving a small number of samples. When more lines of debugging text were included, the number of completed samples dropped, and when less text was included, the number of samples rose. When the baud rate was raised or the text was shortened sufficiently, the Arduino would capture and save data continuously without issue.

CONCLUSIONS

This project was used with a specific sensor, but the application of a data backup mechanism is applicable to not only other health sensors, but other fields altogether. While design tradeoffs were made to simplify the project as more of a proof-of-concept, the entire system could be made more robust by swapping out parts for higher performing alternatives. A critical aspect for real-time data applications is the throughput of the wireless link. For example, data rates above 9600bps on the XBEE are comparatively less reliable, but a sampling rate of 500Hz creates data at 8000bps. This leaves little to no room to increase transmission rate to catch up on backlogged information. At some point, users will have to decide if they want higher resolution or higher reliability. In this case, some bit errors are acceptable, as the data is redundant enough that small errors do not impact the recreated graph very much.

REFERENCES

http://www.maximintegrated.com/app-notes/index.mvp/id/2875