Wireless Oximeter

by

Andrew Young

School of Information Technology and Electrical Engineering,

University of Queensland

Submitted for the degree of

Bachelor of Engineering (Honours)

in the division of Computer Systems

October 2003

October 29, 2003

The Head of School
School of ITEE
University of Queensland
St Lucia, Qld 4072

Dear Professor Kaplan,

In accordance with the requirements of the degree of Bachelor of Engineering (Honours) in the division of Computer Systems Engineering, I present the following thesis entitled “Wireless Oximeter”. This work was performed under the supervision of Dr Adam Postula.

I declare that the work submitted in this thesis is my own, except as acknowledged in the text and footnotes, and has not been previously submitted for a degree at the University of Queensland or any other institution.

Yours sincerely,

____

Andrew Young

Acknowledgements

I would like to thank the following people who helped in the completion of this thesis:

Dr Adam Postula, for supervising me throughout the year.

Keith Bell, for his assistance with PCB design.

Matt D'Souza, for his advice on and help with working with wireless devices.

My friends, who provided the occasional needed relief and distraction from work.

My family, for supporting and encouraging me throughout the year.

Abstract

The medical field has been quick to embrace wireless technology. The advantages in being able to monitor a number of patients from a distance are obvious, and lead to improved efficiency in a busy hospital.

To this end, what was proposed was a device which monitors people's blood oxygen levels remotely - in other words, a wireless oximeter - in the form of a small finger clip. In a room, several of these would be worn by patients, and a nurse or other person would be able to view each patient's data via a receiver connected to a laptop.

Unfortunately, the product did not meet some of these specifications, but the basic functionality - getting the sensor data from the oximeter to the PC - was achieved.

Table of Contents

Acknowledgements ....**iii**

Abstract ....**iv**

List of Figures ....**vii**

List of Tables ....**viii**

Chapter 1: Introduction **1**

1.1 Project Overview 1

1.2 Project Objectives 1

Chapter 2: Background Theory **3**

2.1 Pulse Oximetry 3

2.1.1 Limitations of an oximeter 4

2.2 Wireless communication 4

Chapter 3: Design and Planning **6**

3.1 Oximeter Design 6

3.1.1 Sensor 6

3.1.2 Oximeter Controller 7

3.1.3 Power Supply 7

3.1.4 Wireless Transceiver 8

3.2 Receiver Design 8

3.2.1 Receiver Controller 9

3.2.2 PC communication 9

3.3 Wireless Protocol 9

3.3.1 Oximeter Protocol 9

3.3.2 Receiver Protocol 11
3.3.3 Message Formats 12
3.4 PC Display 13

Chapter 4: System Implementation 14

4.1 Oximeter Implementation 14
4.1.1 Sensor Circuit 14
4.1.2 Transceiver 15
4.1.3 Oximeter Firmware 16
4.2 Receiver Implementation 18
4.2.1 USB Communication 18
4.2.2 Receiver Firmware 19
4.3 PC Display 19

Chapter 5: Evaluation and Conclusion 21

5.1 Review 21
5.2 Further Improvements 21
5.3 Conclusion 22

References 23

Appendices 24
Appendix A: PCB Design 24
A.1 Oximeter PCBs 24
A.2 Receiver PCB 25
Appendix B: Source Code 26
B.1 Oximeter code 26
B.2 Receiver code 31
B.3 Display code 35

List of Figures

Figure 1.1: Block diagram of overall system 2
Figure 2.1: Light absorption characteristics of the two types of hemoglobin 3
Figure 2.2: Illustration of the principle of a pulse oximeter 3
Figure 3.1: Block diagram of the planned oximeter 6
Figure 3.2: Photoplethysmograph waveform 7
Figure 3.3: Block diagram of the planned receiver 8
Figure 3.4: Oximeter side of wireless protocol 10
Figure 3.5: Receiver side of wireless protocol 11
Figure 4.1: Sensor circuit for the oximeter 14
Figure 4.2: RFW102-M module 15
Figure 4.3: Pin layout of PIC16F876 microcontroller 17
Figure 4.4: The DLP-245M USB to Parallel module 18
Figure 4.5: Screenshot of the PC display program 19

List of Tables

Table 1: RFW102-M interface 16
Table 2: DLP-245M module interface pins 18

Chapter 1: Introduction

1.1 Project Overview

The medical field has been quick to embrace wireless technology. The advantages in being able to monitor a number of patients from a distance are obvious, and lead to improved efficiency in a busy hospital.

One such application which would benefit from a wireless implementation is an oximeter. As it stands now, an oximeter's medical applications include monitoring patients during anaesthesia, intensive care, or those with conditions such as asthma. They are also used by EMTs with people complaining of respiratory problems.

There are also various uses for oximeters beyond the medical. They can warn a pilot of how close he is to hypoxia (deficiency in the amount of oxygen reaching body tissues), for example. Similarly, this applies to any other situation where people are working at high altitudes. Within these applications, it is easy to see the convenience of wireless technology.

1.2 Project Objectives

To this end, what was proposed was a device which monitors people's blood oxygen levels remotely - in other words, a wireless oximeter - in the form of a small finger clip. In a room, several of these would be worn by patients, and a nurse would receive the data with a small receiver connected to a laptop.

To illustrate the overall design of the system as a central node with branches out to the sensors:

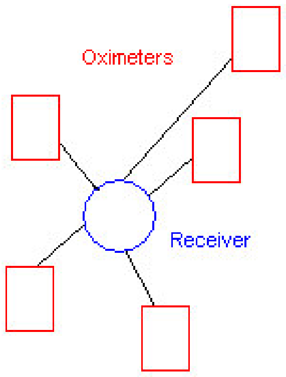


Figure 1.1: Block diagram of overall system.

The aim was for the oximeter to be able to:

- Read the patient's blood oxygen level and wirelessly transmit it to the receiver.
- Have low power consumption, and a functional life of a good number of hours, so as to last through an operation.
- Have a range of several metres, or enough to cover a medium-sized room.
- Be small and lightweight, such that it is not a major annoyance when clipped on.

And similarly, that the receiver would:

- Wirelessly receive data from an oximeter device, and pass it on to a laptop.
- Be able to distinguish between multiple oximeters.
- Be relatively small so that it is easily connected to a laptop, with the space restrictions this entails.

Chapter 2: Background Theory

2.1 Pulse Oximetry

The most common non-invasive method used to measure blood oxygen saturation is known as pulse oximetry. This technique is based upon the different red and infrared light absorption characteristics of oxygenated(HbO_2) and deoxygenated(Hb) hemoglobin. The figure below illustrates the differences in absorption of both:

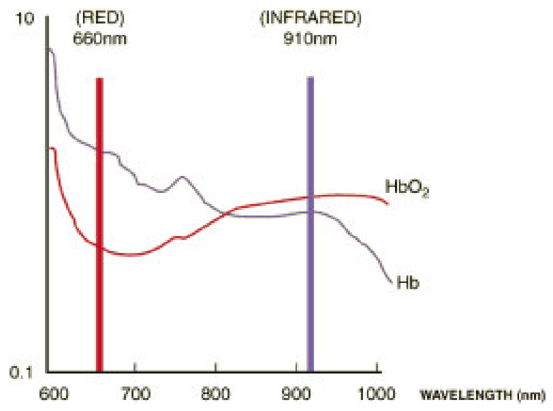


Figure 2.1: Light absorption characteristics of the two types of hemoglobin.

Both red and infrared light are shone through a part of the body that is translucent, and has good blood flow (typically the ear, finger or toe). A photodetector at the opposite end determines the strength of the resulting red and infrared signals.

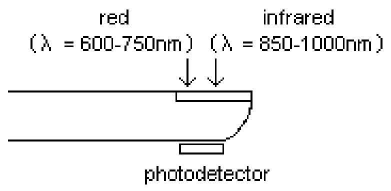


Figure 2.2: Illustration of the principle of a pulse oximeter

The ratio of red to infrared is calculated, and from this the percent of oxygen saturation (or SpO_2 value) can be determined.

Most oximeters are pulse oximeters, which means they can compensate for the fact that the pattern of saturation level at any instant will be a waveform, the peaks occurring at every heartbeat or ‘pulse’.

In general, the normal saturation for a patient at sea level is 95% SpO_2 or above. The level at which a person starts to become noticeably impaired is approximately 90% SpO_2, and a reading close to 80% SpO_2 indicates severe hypoxia.

However, there are limitations to this technique, which can affect the accuracy of the reading.

2.1.1 Limitations of an oximeter

Hypothermia, anemia, shock, and vigorous movement are among conditions under which the readings will unreliable. However, there are also other, more fundamental limitations which cannot be solved by a patient without these kinds of problems.

For example, a pulse oximeter cannot determine between O_2 and CO saturation, since both reflect similar amounts of red light. A patient with, say, a true O_2 saturation of 80% and CO of 15% (possibly found in, say, a heavy smoker) will still have a pulse oximeter reading of 95%.

An oximeter is simply another tool rather than a guaranteed diagnosis, but it is the easiest to use out of all methods of determining blood oxygen saturation.

2.2 Wireless communication

Data transferred through wireless communication requires the signal to be in the form of an analog wave. In addition, it must be modulated to suit the frequency desired. Most medical equipment uses the Industrial, Scientific and Medical (ISM) band, which operates at 2.4GHz.

There are several ways to implement a receiver that can differentiate from multiple users in a system:

Frequency Division Multiple Access (FDMA): Each user is assigned a different frequency within a range, and the receiver separates these into channels. This method allows each user to be transmitting continuously, but the required bandwidth is higher.

Time Division Multiple Access (TDMA): Each user is assigned a time slot over a certain time period in which they are allowed to transmit. The receiver goes through each round-robin style, getting data from each user in turn. This method is the simplest, and likely the easiest to implement, though synchronisation between users needs to be established.

Code Division Multiple Access (CDMA): A unique 'code' or identifier is assigned to each user, and the receiver determines who sent the data by this code. While this may be the most complex method, it enables many users to transmit on the same frequency; the only limit being the maximum data rate of the receiver.

Chapter 3: Design and Planning

This chapter explains the initial design of the system, from the sensor to the display on the PC. However, specific details on the particular components that were decided upon have been left out until the next chapter, as several design changes were made in the actual implementation.

3.1 Oximeter Design

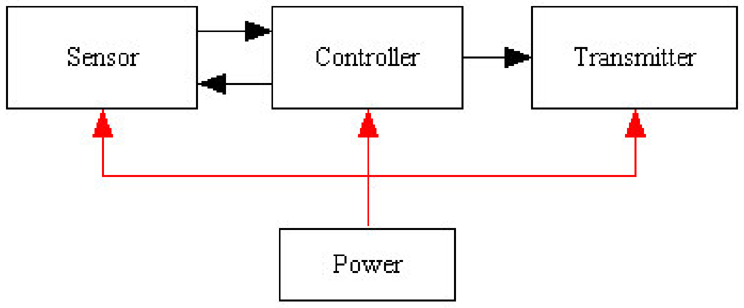


Figure 3.1: Block diagram of the planned oximeter

The figure above shows the bare minimum needed for the oximeter – the sensor itself, a way to perform rudimentary processing on its output, and, of course, the wireless transmitter. Anything more was deemed unnecessary and unreasonable due to the size and weight restrictions.

3.1.1 Sensor

The most simple circuit to measure blood oxygen saturation is the photoplethysmograph, which employs the technique described earlier. It is more or less a simple emitter and detector circuit, with the output from the detector amplified before being read at the output. **Figure 3.2** shows the typical type of output from a photoplethysmograph.

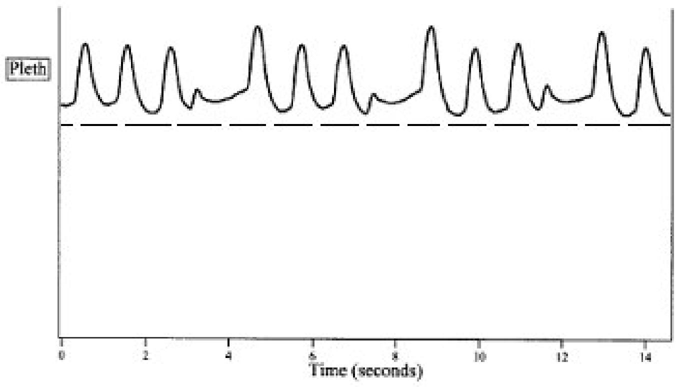


Figure 3.2: Photoplethysmograph waveform

3.1.2 Oximeter controller

The microcontroller for the oximeter needed to perform the following tasks:

- control the sensor LEDs
- convert the sensor output into a digital form
- pass this data onto a suitable form for the transceiver
- protocol handling

Thus an A/D convertor with at least 2 channels (one for each LED), a small number of I/O pins, and possibly a serial interface for the transceiver would be all that was needed. The amount of memory used would be negligible, so it was not a consideration. A microcontroller was chosen over an FPGA, as many have an integrated ADC, saving space, and a reasonable cloak speed, such as 20MHz, would be more than adequate.

3.1.3 Power Supply

Because of the restrictions on size and weight, the power supply was required to be as small and light as possible, not always an easy task. The most suitable was found to be a small lithium cell or cells to total 4-5V, as these retain close to their nominal voltage for much of their functional life, which ensures consistent emissions from the LEDs.

3.1.4 Wireless Transceiver

The two mediums considered for the wireless communication were infrared, and radio. Infrared is suitable for low-power applications, but it requires line-of-sight to operate. While our desired application will generally be contained within a single room, there would be a number of obstructions in the room, and it may be possible for the signal to be lost for a certain period of time. During an operation, for example, this could be dangerous. Thus it was decided to use radio as the communication medium – while functioning at a slightly higher power, it is able to pass through objects, and it is ensured that the signal will able to be received 100% of the time.

The function of the transceiver is obvious, but there were a number of considerations for choosing a wireless chip. Size, bandwidth, power consumption, and ease of interfacing with a microcontroller were each kept in mind while deciding. There was a wide range of small-footprint transceivers to choose from, but in the end those from RFWaves looked most attractive.

3.2 Receiver Design

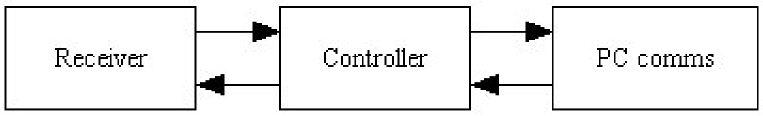


Figure 3.3: Block diagram of the planned receiver

The receiver was needed to both pass on any received transmissions to the laptop, and send any ‘control’ messages from the laptop back to the oximeter. Similar decisions were made regarding the actual chip used, so these will not be covered again here.

3.2.1 Receiver controller

The controller for the receiver had a much simpler job than that for the oximeter - all it would need to do was act as a go-between for the transceiver and PC. The same microcontroller would be used for the receiver, as well, to ease development.

3.2.2 PC communication

The easiest way to communicate with a PC is typically through the serial port. However, to avoid having to supply power to the receiver through a battery, it was decided that USB would be a better choice.

3.3 Wireless Protocol

Rather than using an existing protocol, such as Bluetooth, a simpler wireless protocol was designed to not only allow the readings to be transferred efficiently between oximeter and receiver, but also to enable the receiver to automatically detect any oximeters switched on within range, those which had been turned off or gone out of range.

3.3.1 Oximeter protocol

The aim of the protocol on the oximeter's behalf was to ensure that the receiver was aware of the oximeter's presence, and vice versa. Once this was established, the data would be sent continuously. What follows is a flow chart of this protocol:

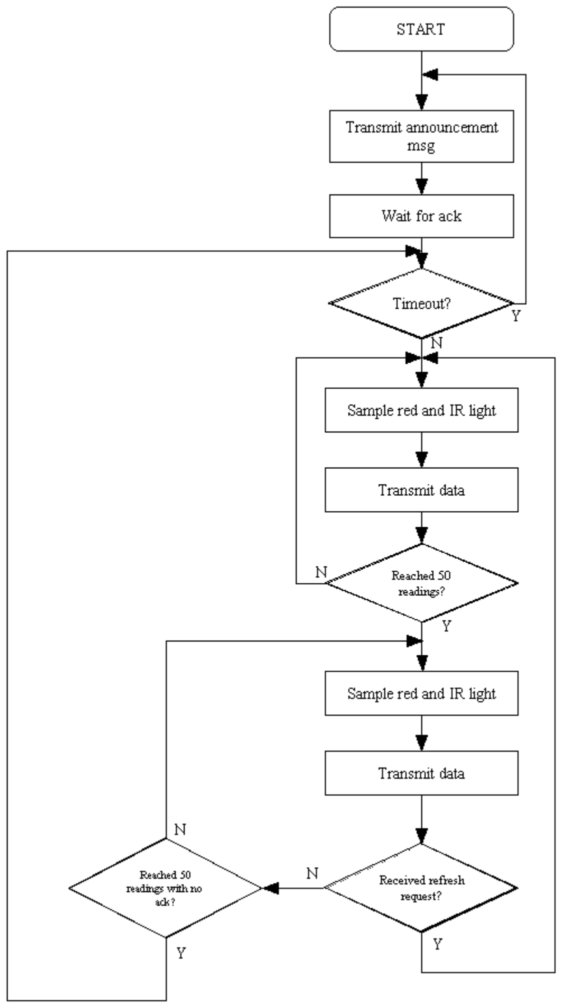


Figure 3.4: Oximeter side of wireless protocol

Upon power on, the oximeter transmits an 8-bit ‘announcement’ message, then waits for a response. If none is forthcoming after 2 seconds, it will try again. The acknowledgement consists of an 8-bit message identifier, as well as a 8-bit ID for the oximeter, which is then used for all subsequent messages.

Sampling of the output from the photodetector then begins, and transmissions occur as a ‘burst’ every 20ms. The actual message contains an 8-bit message identifier, and 8 bits for each LED reading, for a total of 3 bytes.

Once per second, or 50 transmitted readings, the oximeter will begin to watch for a ‘refresh request’ message sent by the receiver, and will respond with an acknowledgement. If this request message is not received after another 50 readings, then it is assumed that the communications have been broken, and will return to the beginning, transmitting the ‘announcement’ message.

3.3.2 Receiver protocol

The protocol on the receiver side is mostly derived from the oximeter’s, taking all messages from the oximeter, and occasionally ‘pinging’ it to see if the connection is still alive. This half of the protocol, shown below, was to be handled by the PC rather than the receiver hardware, for greater flexibility.

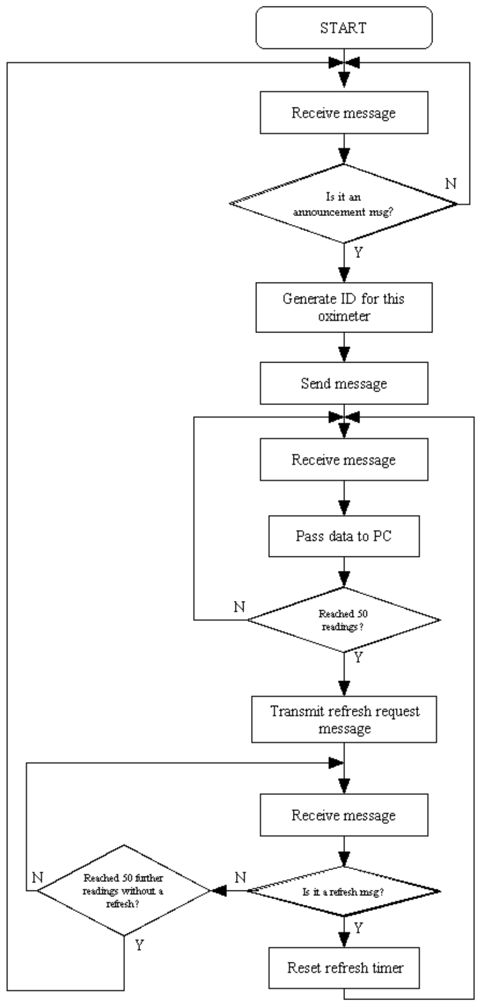


Figure 3.5: Receiver side of wireless protocol

The receiver initially waits for an oximeter to send the ‘announcement’ message. Upon receiving this message, it generates the unique ID for the oximeter, which will tell the receiver which oximeter the data is coming from. From then on, it follows much the same procedure as the oximeter, with the transmissions and receptions reversed.

3.3.3 Message Formats

The actual contents of each message is specified below:

Announcement:

11100111
8

Announcement acknowledgement:

00011000	identifier
8	8

Data:

identifier	Red sample	IR sample
8	8	8

Refresh request:

Identifier	10111101
8	8

Refresh response:

01000010
8

3.4 PC Display

There was to be little to no processing of the data up to this point; it was up to the software on the PC to take care of any calculations and protocol decisions on the receiver side. More specifically, the following:

- convert incoming sensor data from each oximeter into an SpO_2 percentage, using a lookup table
- detect multiple oximeters within range, and display graphs for each
- generate the identifier for each oximeter
- determine when to send the 'refresh request' messages

As processing speed was not much of a concern, the program would be written in Visual Basic, largely for simplicity's sake.

Chapter 4: System Implementation

The basic elements of the initial design were implemented successfully, but due to a number of hardware failures and difficulties in getting replacements, a simplification had to be made to the original design, so that instead of a network of sensors, there would only be a single sensor being displayed on the PC.

Also, it was found to be difficult to synchronise the oximeter and receiver during quick data transfer, so the number of readings per second was reduced in favour of a more reliable wireless channel.

4.1 Oximeter implementation

The oximeter was separated into two PCBs, each intended for a 'layer' of the finger clip. The LEDs were positioned on the top PCB, while the rest of the components were on the bottom PCB, double-sided to save room. The design of both these PCBs can be found in **Appendix A.1**.

4.1.1 Sensor Circuit

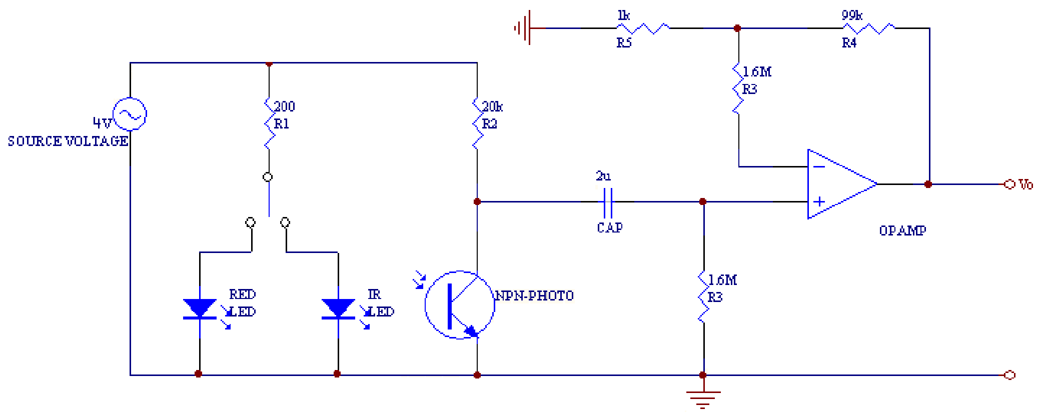


Figure 4.1: Sensor circuit for the oximeter.

The sensor itself consists of the above circuit, a photoplethysmograph. Every time a reading is required, each LED is switched on in turn, and the light on the other side of the finger detected by the phototransistor. As its output will typically be a very slight variation over a high baseline voltage (as seen in **Figure 3.2**), this is eliminated by a high-pass filter, and the result amplified to be read by the A/D convertor. Between readings, each LED is switched off to conserve power.

The QTLP650C-7, a surface mount AlGaAs red LED, which emits at a wavelength of 660nm, was used in conjunction with the HSDL-4400-011, a surface mount IR LED, which emits at a wavelength of 910nm. They have a typical intensity of 20 and 15mcd at 20mA, respectively. A BPW17N phototransistor and TL072 op amp IC were used as well. To power the oximeter, two CR2032DP2 3V lithium coin cells, each with a capacity of 180mAh, were used.

4.1.2 Transceiver

The transceiver chosen was the RFW102-M module with antenna, from RFWaves, a short-range, half duplex wireless radio transceiver, which functions in the ISM (2400-2483.5MHz) band. It supports data transfer up to 1Mbps, and very low power consumption, proportional to the data rate (typically 21mA at 1Mbps, 28μA at 1kbps).

Figure 4.2: RFW102-M module.

Its interface is fairly simple - all that is needed is to set the mode to either transmit or receive, set it to active, and the data either comes in or out via the same pin.

Name	Characteristic
Vcc	A regulated voltage of 2.7-3.6V
GND	Apply the supply ground to this pin
Tx/Rx	Vcc for Tx (transmit mode GND for Rx (receive) mode
ACT	Vcc on working mode GND standby mode
TxD/RxD	In Tx mode: data input pin In Rx mode: data output pin
RSSI	Received Signal Strength Indicator. Indicates the power transmitted in the RFW102 frequency band, allowing determination of whether to transmit (Tx).

Table 1: RFW102-M interface.

However, it differs from many transceivers in that, when a ‘1’ is received, there is only a 700ns ‘pulse’ on the data pin rather than a replication of the transmitted waveform. Also, if transmission side’s data pin is left floating, the output on the receiver will be random noise.

4.1.3 Oximeter firmware

The microcontroller chosen for both the oximeter and the receiver was the PIC16F876, because of its relatively small size and ease of use. It features a 10-bit, 8 channel ADC, low supply voltage, low power consumption, and comes in a 28-pin DIP package.

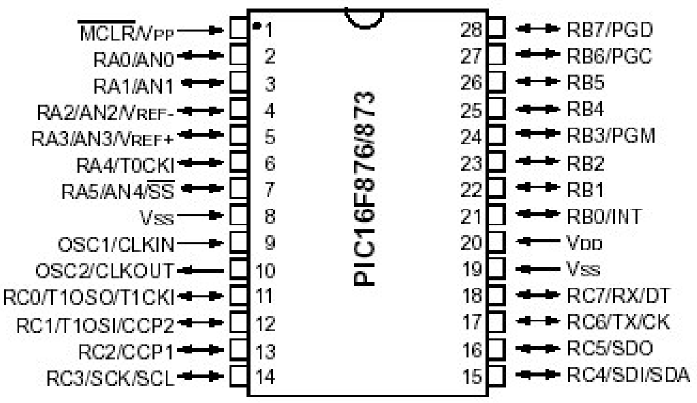


Figure 4.3: Pin layout of PIC16F876 microcontroller.

In order to capture the pulses occuring on the transceiver's data pin, it was tied to the PIC's external interrupt, which triggers whenever it detects one of these pulses - in this way, the device knows when a '1' has been received. A '0' was assumed to be received if 1.5ms had passed without an interrupt occurring. Sending data was a simpler matter, with each bit being shifted left from the register onto the output pin.

After sending the initial announcement message, any incoming data is received in the above fashion. Once 16 bits are accumulated, the first 8 are checked to see if they correspond with the correct 'announcement acknowledgement' message. If so, then the next 8 are used as the oximeter's identifier from now on.

From here, the actual data sampling begins. Each LED was connected to an output pin, and so able to be switched on or off when necessary. The output from the photoplethysmograph was sampled every 40ms, by activating the ADC and reading its output register.

After 50 samples, a 'refresh request' message is watched for during the delay between samples. If it is received within the next 50 samples, then it goes back to sampling the next 50. However, if no 'refresh request' message is detected, then the oximeter goes idle, and starts again from the beginning.

The source code for the oximeter may be found in **Appendix B.1**.

4.2 Receiver implementation

While keeping the receiver small was not crucial, it was managed so that it could fit in the hand. . Refer to **Appendix A.2** for the layout of the receiver PCB.

4.2.1 USB communication

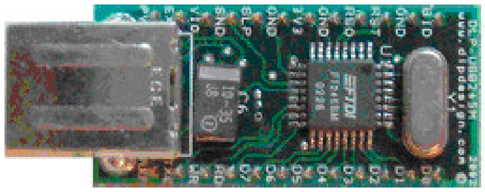


Figure 4.4: The DLP-245M USB to Parallel module.

The DLP-245M USB module from DLP design was largely chosen because of its smaller footprint than similar modules, as many are based upon the same chip in the first place, FTDI's FT245BM USB FIFO. This module is easy to interface with a microcontroller; its method of operation is summarised in **Table 2**.

Pin	Description
10 (VCC-IO)	Power supply from the USB port.
13 (RXF#)	When low, indicates that at least one byte of data is able to be read from the FIFO's 128-byte receive buffer.
14 (TXE#)	When high, indicates that the FIFO's 385-bye transmit buffer is full, and unable to be written to further until it is read.
15 (WR)	Used to write to the USB port. When taken from high to low, the current 8 data lines are written into the transmit buffer.
16 (RD#)	When pulled low, the current byte in the receive buffer is moved to the 8 data lines.
17-24 (D7-D0)	8 bi-directional data bits.

Table 2: DLP-245M module interface pins.

4.2.2 Receiver firmware

The same pin connections were used as in the oximeter, as was the method of transmitting and receiving via the transceiver through shifting and interrupts respectively.

Since the receiver was not in charge of handling any protocol, its firmware consisted only of either reading from the transceiver and writing to the USB port, or vice versa. However, it does pad messages, with zeros, out to 3 bytes for those that are less, to make things easier for the PC program.

The source code for the oximeter may be found in **Appendix B.2**.

4.3 PC Display

The PC was where the other half of the protocol was implemented. Using the Virtual Com Port drivers supplied by FTDI, the USB module is written to and read from exactly like a normal serial port. A screenshot of what is displayed from test data from the oximeter is shown in ***Figure 4.5***.

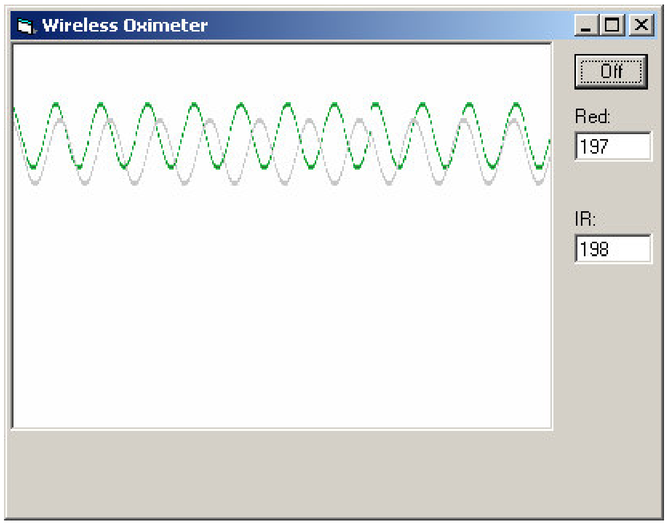


Figure 4.5: Screenshot of the PC display program.

Data is continually read and analysed from the USB port, in ‘packets’ of 3 bytes. From the first bytes, the type of message it is can be determined, and from there the program simply follows the protocol described in Section 3.3. However, due to a lack of appropriate casing on the actual oximeter, the actual calculation of SpO_2 could not be done to reasonable precision, and so it simply displays two separate waveforms.

The graph is drawn by manipulating pixels of a picture, column by column. Whenever a data message is received, the next column is updated. Once the far right side is reached, the readings are displayed from the far left, so that the graph appears to display in ‘sweeps’.

The source code for the display program may be found in **Appendix B.3**.

Chapter 5: Evaluation and Conclusion

5.1 Review

Unfortunately, when this was achieved, the USB module that was being used at the time malfunctioned, and there were no spares. The connectors for the transceivers also failed, and replacements were taking too long to arrive. Because of this, it was decided that the bulk of what was still left to do would not be able to be done, and what was implemented in the end was a one-to-one relationship between oximeter and receiver, rather than a network of sensors.

The other aspect missing from the specification was the actual calculation of SpO_2 to be derived from the data. As the oximeter did not have a suitable case, it was found that ambient light factored too heavily into the data for it to be determined with much accuracy.

5.2 Further Improvements

As the original ‘network’ idea was left out of the actual product, this would be the first improvement to make. A CDMA protocol would be most appropriate, for scalability reasons - each oximeter was, even originally, not going to need to transfer more than a few hundred bytes per second, so with CDMA the bandwidth could be shared. Some sort of rudimentary error-checking could also be implemented.

Other possible improvements would be an alarm of sorts that would sound when the oximeter/receiver connection went down, or even an extension of the ‘network’ system to allow multiple networks to coexist or interact.

5.3 Conclusion

The aim of the project was to create a system whereby a number of patients wearing an oximeter, or a single patient wearing multiple oximeters, would all be able to be monitored by a single person. However, because of several hardware failures and long delays in obtaining replacements, much of the later implementation was not possible.

While it was unfortunate that the whole specification was not met, there was enough of the underlying work completed so that furthering the project can be done with little difficulty.

References

[1] Aston, R. 1990, *Principles of Biomedical Instrumentation and Measurement*, Maxwell MacMillan.

[2] Prahl, S., 1999, "Optical Absorption of Hemoglobin",
Available at: http://omlc.ogi.edu/spectra/hemoglobin/index.html

[3] Ward, T. 2002, "The Photoplethysmograph as an instrument for physiological measurement",
Available at: http://www.eeng.may.ie/~tward/seminar_maynooth/Seminar/

[4] Webster, J.G. 1998, *Medical instrumentation : application and design*, Wiley.

[5] 2003, "BPW17N Datasheet",
Available at: http://www.vishay.com/docs/81516/81516.pdf

[6] 2002, "Pulse Oximetry",
Available at: http://www.oximeter.org/pulseox/principles.htm

[7] 2001, "PIC16F87X datasheet",
Available at:
http://www.microchip.com/download/lit/pline/picmicro/families/16f87x/30292c.pdf

[8] 2003, "QTLP650C-2-3-4-7-B Surface Mount LED Lamp datasheet",
Available at: http://www.fairchildsemi.com/ds/QT/QTLP650C-B.pdf

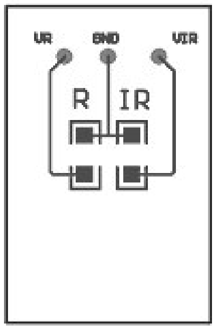
[9] 2002, "RFW102-M ISM Transceiver Module Datasheet",
Available at: http://www.rfwaves.com/GetFile.asp?type=1&path=RFW102-M Module Data Sheet R1_1.pdf

[10] 2003, "USB to FIFO Parallel Interface Module datasheet",
Available at: http://www.dlpdesign.com/usb/dlp-usb245m12.pdf

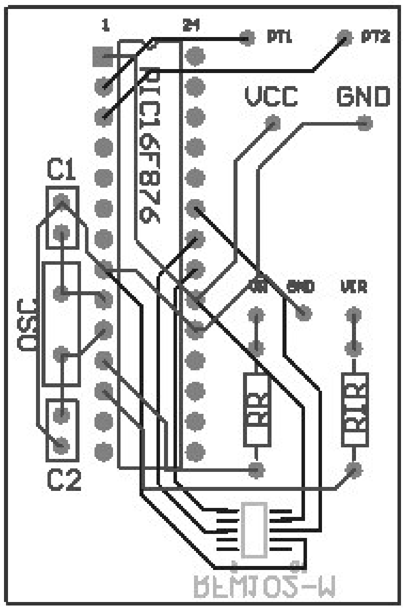
Appendices

Appendix A: PCB Design

A.1 Oximeter PCBs

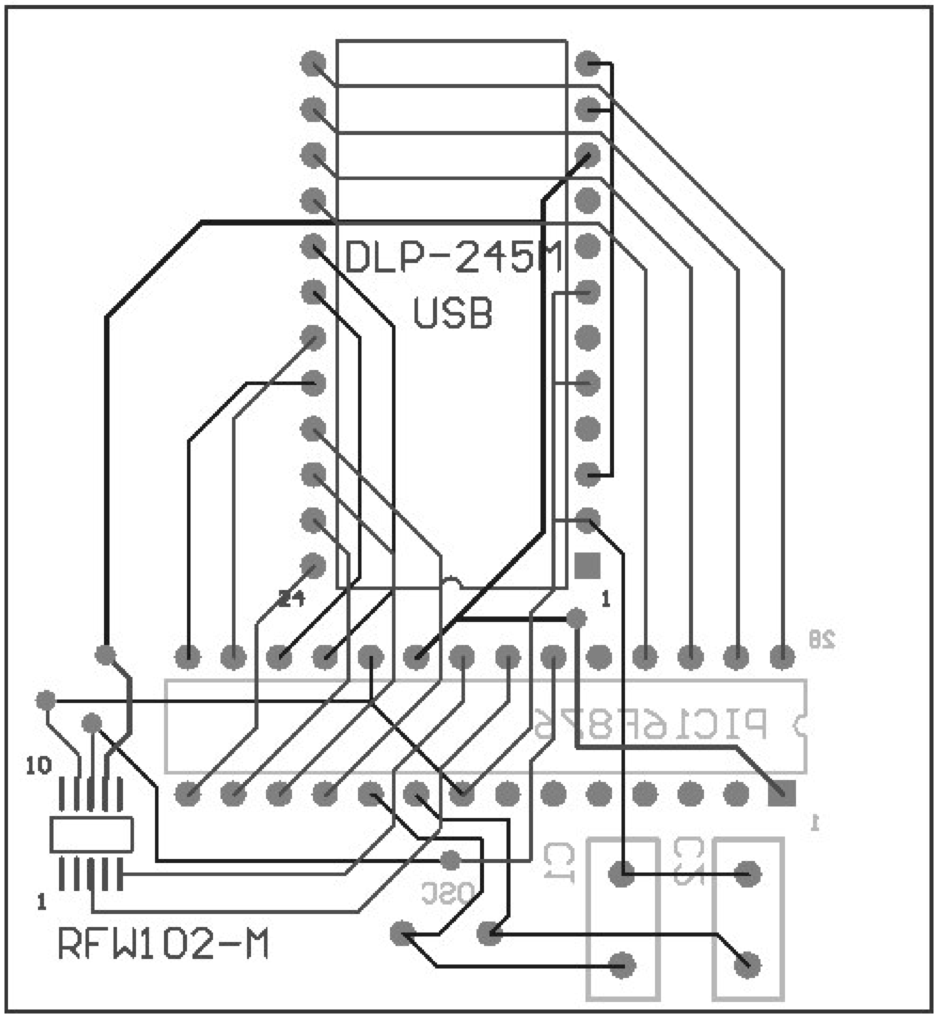


Upper PCB



Lower PCB

A.2 Receiver PCB



Appendix B: Source code

B.1 Oximeter code

```
/*  Thesis: Wireless oximeter - oximeter device code
 *
 *  Description: Code for the oximeter microcontroller; controls the LEDs,
 *               samples the ADC, and passes the result serially to the
 *               transceiver.
 *
 *  Author: Andrew Young
 */

#include "delay.h"
#include "delay.c"
#include <pic.h>

// Fuse config
__CONFIG(0x393A);

#define XTAL_FREQ 20MHZ
#define RF_DATA RB0
#define RF_TXRX RB1
#define RF_ACT RB2

#define LED_RED RC0
#define LED_IR RC1


volatile unsigned int zeroCount = 0;         // Detects if a zero has been received
volatile unsigned char bitsReceived = 0;
volatile unsigned char dataRed = 0x00;               // Red LED sample
volatile unsigned char dataIR = 0x00;        // IR LED sample
volatile unsigned char data = 0x00;                  // Received data, low byte
volatile unsigned char data2 = 0x00;         // Received data, high byte
volatile unsigned char identifier = 0x00;    // The oximeter's unique ID
volatile unsigned int delayCount = 0;

int i;

void sendOne(void)
{
        RF_DATA = 1;
        DelayMs(1);
}

void sendZero(void)
{
        RF_DATA = 0;
        DelayMs(1);
}


void main(void)
{

        // Initialise I/O ports
        TRISA = 0x03;
        TRISB = 0xC0;
        TRISC = 0x00;

        // ADC setup
        ADCON0=0b00000001;
        ADCON1=0b00001110;

        // Interrupt initialisation for RB0 on rising edge
        INTEDG = 1;
        INTF = 0;
        INTE = 1;
        GIE = 1;

        // Start in transmit mode
        RF_TXRX = 1;


        // Send announcement message: 11100111
        sendOne();
        sendOne();
        sendOne();
        sendZero();
        sendZero();
        sendOne();
        sendOne();
        sendOne();

        // Wait for acknowledgement
        while (1) {

                // If we haven't received an interrupt for 1.5ms (7000 counts to be
                // safe), then assume we received a 0
                if (zeroCount > 7000) {

                        // Add zero to received register
                        data = data << 1;
                        data = (data & 0xFE);
                        bitsReceived++;
                }

                if (bitsReceived == 8) {
```

```
                if (data == 0b00011000)
                        identifier = data2;

        }

        if (identifier != 0x00)
                break;
}


// Main procedure
while (1) {

        INTE = 0;

        // Turn transceiver off to save power
        RF_ACT = 0;

        // Turn on red LED
        LED_RED = 1;
        DelayMs(2);

        // Get ADC reading from RA0 and store it
        ADGO = 1;

        while (ADGO == 1) {}
        dataRed = ADRESH;
        LED_RED = 0;

        // Turn on infrared LED
        LED_IR = 1;
        DelayMs(2);

        // Get ADC reading from RA0 and store it
        ADGO = 1;
        while (ADGO == 1) {}
        dataIR = ADRESH;
        LED_IR = 0;

        // Send both these readings through transceiver
        RF_TXRX = 1;
        RF_ACT = 1;


        // 16 bits = 16ms, delay = 24ms, ==> 25 readings per second
        for (i = 0; i < 8; i++) {
                if ( (dataRed & 0x80) == 0x80) {
                        sendOne();
                } else sendZero();

                dataRed = dataRed << 1;
                }

                for (i = 0; i < 8; i++) {
                        if ( (dataIR & 0x80) == 0x80) {
                                sendOne();
                        } else sendZero();

                        dataIR = dataIR << 1;
                }

                // Switch to receive mode
                RF_TXRX = 0;
                INTE = 1;

                while (1) {

                        // Detect zeros
                        if (zeroCount > 7000) {

                                if (bitsReceived > 8) {
                                        // Add zero to received register, high
                                        data2 = data2 << 1;
                                        data2 = (data2 & 0xFE);
                                } else {

                                        // Add zero to received register, low
                                        data = data << 1;
                                        data = (data & 0xFE);
                                }

                                bitsReceived++;
                                zeroCount = 0;
                        }

                        if (bitsReceived == 16) {

                                if ( (data == identifier) && (data2 == 0b10111101) ) {

                                        //  Transmit  refresh  message,  then  go  back  to
sampling
                                        RF_TXRX = 1;
                                        sendZero();
                                        sendOne();
                                        sendZero();
                                        sendZero();
                                        sendZero();
                                        sendZero();
                                        sendOne();
                                        sendZero();
                                        bitsReceived = 0;
                                        delayCount = 10001;
                                }

                        }

                        zeroCount++;
                        delayCount++;

                        // If 20ms have passed, then we need to sample again
                        if (delayCount > 10000) {
                                bitsReceived = 0;
                                break;
                        }

                }

                // Switch back to transmit mode
                RF_TXRX = 1;

        }
}


static void interrupt isr (void) {
(

        // If we're here, then we received a pulse on the INT pin (i.e. a 1)
        if (INTF) {

                // Reset counter for detecting zeroes
                zeroCount = 0;

                if (bitsReceived > 8) {
                        // Add one to high register
                        data2 = data2 << 1;
                        data2 = (data2 | 0x01);
                } else {

                        // Add one to low register
                        data = data << 1;
                        data = (data | 0x01);
                }

                bitsReceived++;

                // Clear INT flag bit
                INTF = 0;
        }

}
```

B.2 Receiver code

```
/*  Thesis: Wireless oximeter - receiver device code
 *
 *  Description: Simple code to control the LEDs, sample the ADC, and pass the
 *               result serially to the transceiver.
 *
 *  Author: Andrew Young
 */

#include "delay.h"
#include "delay.c"
#include <pic.h>

// Fuse config
__CONFIG(0x393A);

#define XTAL_FREQ 20MHZ
#define USBPORT PORTC
#define USB_RXF RB7
#define USB_TXE RB6
#define USB_WR RB5
#define USB_RD RB4

volatile unsigned int zeroCount = 0;         // Detects if a zero has been received
volatile unsigned char bitsReceived = 0;
volatile unsigned char data = 0x00;                  // Low byte of the data
volatile unsigned char data2 = 0x00;         // High byte of the data
volatile unsigned char data3 = 0x00;         // High byte of the data



void main(void)
{
        // Initialise I/O ports
        TRISB = 0xC3;
        TRISC = 0x00;


        // Interrupt initialisation for RB0 on rising edge
        INTEDG = 1;
        INTF = 0;
        INTE = 1;
        GIE = 1;


        // Wait for announcement message
        while (1) {
```

```
        if (zeroCount > 7000) {

                // Add zero to register
                data = data << 1;
                data = (data & 0xFE);
                bitsReceived++;

                zeroCount = 0;
        }

        zeroCount++;

        if (data == 0b11100111)
                break;
}


// Main procedure
while (1) {

        // If we haven't received an interrupt for 1.5ms (7000 counts to be
        // safe), then assume we received a 0
        if (zeroCount > 7000) {

                if (bitsReceived > 16) {
                        data3 = data3 << 1;
                        data3 = (data3 & 0xFE);
                } else if (bitsReceived > 8) {
                        data2 = data2 << 1;
                        data2 = (data2 & 0xFE);
                } else {
                        data = data << 1;
                        data = (data & 0xFE);
                }

                bitsReceived++;
                zeroCount = 0;
        }


        // If we have received 24 bits, pass them onto the USB module
        if (bitsReceived == 24) {

                // Turn off RB0 interrupt
                INTE = 0;

                // Wait until we are able to transmit (/TXE is low)
                while (RB6 = 1) {}

                // Set the USB WR pin high
                USB_WR = 1;
```

```
                        // Write 8 bits to the USB module
                        USBPORT = data;
                        // Set the USB WR pin low
                        USB_WR = 0;

                        while (RB6 = 1) {}

                        // Set the USB WR pin high
                        USB_WR = 1;
                        // Write 8 bits to the USB module
                        USBPORT = data2;
                        // Set the USB WR pin low
                        USB_WR = 0;

                        while (RB6 = 1) {}

                        // Set the USB WR pin high
                        USB_WR = 1;
                        // Write 8 bits to the USB module
                        USBPORT = data3;
                        // Set the USB WR pin low
                        USB_WR = 0;


                        // Turn on RB0 interrupt
                        INTE = 1;

                        bitsReceived = 0;
                        data = 0x00;
                        data2 = 0x00;
                }

                zeroCount++;

        }



}




static void interrupt isr (void) {
(

        // If we're here, then we received a pulse on the INT pin (i.e. a 1)
        if (INTF) {

                // Reset counter for detecting zeroes
                zeroCount = 0;
```

```
			if (bitsReceived > 16) {
				data3 = data3 << 1;
				data3 = (data3 & 0x01);
			} else if (bitsReceived > 8) {
				data2 = data2 << 1;
				data2 = (data2 | 0x01);
			} else {
				data = data << 1;
				data = (data | 0x01);
			}

			bitsReceived++;

			// Clear INT flag bit
			INTF = 0;
		}

	}
```

B.3 Display code

```
' Project: Wireless oximeter
'
' Description: This program reads from the USB port, via the virtual comm port
' drivers, and displays the data as a graph over time.
'
' Author: Andrew Young (based upon code by David Brebner)

Private Type SAFEARRAYBOUND
    cElements As Long
    lLbound As Long
End Type

Private Type SAFEARRAY1D
    cDims As Integer
    fFeatures As Integer
    cbElements As Long
    cLocks As Long
    pvData As Long
    Bounds(0 To 0) As SAFEARRAYBOUND
End Type

Private Type SAFEARRAY2D
    cDims As Integer
    fFeatures As Integer
    cbElements As Long
    cLocks As Long
    pvData As Long
    Bounds(0 To 1) As SAFEARRAYBOUND
End Type

Private Declare Function VarPtrArray Lib "msvbvm50.dll" Alias "VarPtr" (Ptr() As Any)
As Long
Private Declare Sub CopyMemory Lib "kernel32" Alias "RtlMoveMemory" (pDst As Any, pSrc
As Any, ByVal ByteLen As Long)
Private Type BITMAP
    bmType As Long
    bmWidth As Long
    bmHeight As Long
    bmWidthBytes As Long
    bmPlanes As Integer
    bmBitsPixel As Integer
    bmBits As Long
End Type

Private Declare Function GetObjectAPI Lib "gdi32" Alias "GetObjectA" (ByVal hObject As
Long, ByVal nCount As Long, lpObject As Any) As Long
Private Declare Function GetTickCount Lib "kernel32" () As Long
Private refreshCount As Integer
Private c As Integer


Private Sub Form_Load()
' Initial values
c = 0
refreshCount = 100
' Load the source picture
Pictbox.Picture = LoadPicture(App.Path & "\vbsignal.gif")
Pictbox.Visible = False
blankMessage.Visible = True

End Sub

Sub DoSignal()

Dim pict() As Byte
Dim sa As SAFEARRAY2D, bmp As BITMAP
Dim r As Integer, nc As Integer, pos As Integer

GetObjectAPI Pictbox.Picture, Len(bmp), bmp 'dest

If bmp.bmPlanes <> 1 Or bmp.bmBitsPixel <> 8 Then
    MsgBox " 256-color bitmaps only", vbCritical
    Exit Sub
End If

' Use the matrix to point to the pixels in the picture
With sa
    .cbElements = 1
    .cDims = 2
    .Bounds(0).lLbound = 0
    .Bounds(0).cElements = bmp.bmHeight
    .Bounds(1).lLbound = 0
    .Bounds(1).cElements = bmp.bmWidthBytes
    .pvData = bmp.bmBits
End With
CopyMemory ByVal VarPtrArray(pict), VarPtr(sa), 4


    ' White out this column
    For r = 0 To UBound(pict, 2)
        pict(c, r) = 31
    Next

    graph_row% = Val(outputText.Text)
    graph_row2% = Val(outputText2.Text)

    If graph_row% < 0 Then sin_row% = 0
    If graph_row% > 255 Then sin_row% = 255
    If graph_row2% < 0 Then sin_row2% = 0
    If graph_row2% > 255 Then sin_row2% = 255

    ' Draw the position of the current red and IR readings
    pict(c, graph_row%) = 23
    pict(c, graph_row% + 1) = 14
    pict(c, graph_row% + 2) = 23

    pict(c, graph_row2%) = 28
    pict(c, graph_row2% + 1) = 28
    pict(c, graph_row2% + 2) = 28

c = c + 1
If c > UBound(pict, 1) Then
    c = 0
End If


CopyMemory ByVal VarPtrArray(pict), 0&, 4
Pictbox.Refresh

End Sub

Private Sub Command1_Click()

'Turn the display on and off
If Command1.Caption = "On" Then
    Command1.Caption = "Off"
    MSComm1.PortOpen = True
    Pictbox.Visible = True

    blankMessage.Visible = False

    Do


'Receiving code
Dim buffer As String
Dim graphValue As String * 1
Dim graphValue2 As String * 1
Dim bufferLength As Integer
Dim count As Integer

If MSComm1.PortOpen = True Then

    buffer = MSComm1.Input
    bufferLength = Len(buffer)

    graphValue = Mid(buffer, 1, 1)
    graphValue2 = Mid(buffer, 2, 1)
    outputText.Text = Asc(graphValue)
    outputText2.Text = Asc(graphValue2)

If outputText.Text <> 32 Then
        DoSignal
        u% = DoEvents  'process other events
End If

End If


If graphValue = 1 Then
End If

' If we received the announcement message (11100111)
If Mid(buffer, 1, 1) = 231 And Pictbox.Visible = False Then
    Pictbox.Visible = True
    blankMessage.Visible = False
End If


' If we received the refresh acknowledgement message (11100111)
If Mid(buffer, 2, 1) = 66 And Pictbox.Visible = True Then
    refreshCount = 100
End If

' If we received a data message, then display these values
If Mid(buffer, 1, 1) <> 231 And Mid(buffer, 3, 1) <> 0 And Pictbox.Visible = True Then
    outputText.Text = Mid(buffer, 2, 1)
    outputText2.Text = Mid(buffer, 3, 1)
End If



'Sending code

' Send the refresh request message (10111101)
If refreshCount < 50 Then
    send1 = 0
    send2 = 189

    MSComm1.Output = send1
    MSComm1.Output = send2
End If


' If the connection with the oximeter has been broken
If refreshCount = 0 Then
    Pictbox.Visible = False
    blankMessage.Visible = True
```

```
End If


    refreshCount = refreshCount - 1


    Loop Until Command1.Caption = "On"
Else
    Command1.Caption = "On"
    MSComm1.PortOpen = False
    Pictbox.Visible = False

    blankMessage.Visible = True

End If
End Sub
```