

Wipmod: Wireless Patient Monitoring Device

Seth Hara and Anima Singh
E90: Engineering Design
Advisor: Professor Erik Cheever
May 8, 2008

Table of Contents

Table of Figures	4
Abstract	6
1/Introduction	7
1.1/Scope of the project	7
1.2/Organization of the report	8
2/Wireless communication	9
2.1/Overview of Zigbee	9
2.2/Overview of the XBee Module	9
2.3/General specifications	10
2.4/Programming the XBee Module	11
2.5/Testing	12
3/Temperature	12
3.1/Temperature sensor overview	12
3.2/Sensor calibration	13
4/Electrocardiograph	15
4.1/Characteristics of the electrocardiogram	15
4.2/Analysis of timing features of the electrocardiogram	17
5/Electrocardiograph circuit	18
5.1/Single-supply op-amps and virtual ground	19
5.2/Electrodes	20
5.3/Instrumentation amplifier	21
5.4/Filters	23
5.4.1/Band-pass filter	23
5.4.2/2 nd order low-pass filter	25
5.5/Driven-right-leg circuit	26
5.6/Adaptive gain	27
6/Pulse oximeter	28
6.1/Principles of pulse oximetry	28
6.2/Theoretical calibration curve	30
7/Pulse oximeter circuit	30

7.1/Some reverse engineering	30
7.2/Block diagram of pulse oximeter circuit components	32
8/Batteries	33
9/PIC microcontroller	34
9.1/General characteristics	34
9.2/Programming the PIC	35
9.2.1/Digital potentiometer	36
9.2.2/Patient alert system	39
10/Signal processing	41
10.1/Data parsing	41
10.2/Estimation techniques for characteristics of an ECG waveform	42
10.2.1/Heart rate	42
10.2.2/QRS complex width	43
11/Layout and features of the graphical user interface	46
11.1/Brief introduction to Matlab GUI structure	46
11.2/GUI features	46
11.2.1/Main panel	46
11.2.2/Communication Settings panel	47
11.2.3/Alarm Settings panel	48
12/Acknowledgements	49
13/References	50
14/Appendices	51
Appendix A: Wipmod version 1	51
Appendix B: Wipmod version 3	53
Appendix C: PIC code	56
Appendix D: Matlab code for GUI design	61

Table of Figures

Name	Page
Figure 1. General overview of the Wipmod system	8
Figure 2 Block diagram showing the organization of the report	8
Figure 3. Data flow between the base and the remote unit through the XBee modules	9
Figure 4. XBee RS-232 development board with module	10
Figure 5. X-CTU software used to program the XBee module for point-to-point communication	11
Figure 6. Thermistor-based temperature sensor	13
Figure 7. Temperature sensor calibration set-up	13
Figure 8. Voltage divider circuit used for calibration	14
Figure 9. Calibration curve for the tympanic temperature sensor	14
Figure 10. Lead vectors a_1 and a_2 in relation to the cardiac vector M	15
Figure 31. Eindhoven's triangle	16
Figure 42. P wave, QRS complex, and T wave	16
Figure 53. Normal sinus rhythm for leads I, II, and III	17
Figure 64. Tachycardia.	18
Figure 15. Bradycardia.	18
Figure 76. Block diagram of ECG circuitry	19
Figure 17. Multisim schematic of complete ECG circuit	19
Figure 88. Virtual ground circuit	20
Figure 19. 3M Red Dot electrode	21
Figure 20. First stage of instrumentation amplifier	22
Figure 21. Second stage of instrumentation amplifier	23
Figure 22. Band-pass filter	24
Figure 23. Bode plot for the band-pass filter	24
Figure 24. 2 nd order low-pass filter	25
Figure 25. Bode plot for the 2 nd order low-pass	26
Figure 96. Driven-right-leg circuit	26
Figure 27. Extinction coefficients of Hb and HbO ₂	29
Figure 28. AC and DC components of the total absorption of LED lights	29
Figure 29. Pulse oximeter sensor and its components	31
Figure 30. Arrangements of LEDs and photodiode in the pulse oximeter sensor	31
Figure 3110. Block diagram for the pulse oximeter circuit	32
Figure 32. Multisim schematic of the pulse oximeter circuit	32
Figure 33. Output waveform of the pulse oximeter circuit for the red LED	33
Figure 34. Flowchart demonstrating how timer/interrupts were generated to control sampling rate using counters	36
Figure 35. SPI pin connections for the digipot	37
Figure 36. Three snapshots showing the adaptive gain of the ECG signal	39
Figure 37. Flowchart describing the patient alert system	40
Figure 38. The circuit set up for the alert system with the buzzer-snooze pushbutton	41
Figure 39. Voltage threshold filtering for R peak identification	42
Figure 40. R peak search flow chart	43
Figure 41. Q and S point search flow chart	45

Figure 42. Example Q point search	45
Figure 43. Main panel of the GUI	47
Figure 44. Communication Settings panel	48
Figure 45. Alarm Settings panel	49
Figure 46. Complete Multisim schematic of Wipmod v. 1.0	51
Figure 47. Ultiboard layout of Wipmod v. 1.0	52
Figure 48. Complete Multisim schematic of Wipmod v. 2.0	53
Figure 49. Final Ultiboard layout of Wipmod v. 2.0	54
Figure 50. Interior view of the complete remote unit	55
Figure 51. Exterior view of the complete remote unit with all the sensors connected	56

Abstract

The Wipmod (wireless patient monitoring device) system was designed to monitor a patient while offering the convenience of being at home and carrying on with one's day-to-day activities. It acquires, transmits, processes, and displays three physiological signals, using wireless modules to communicate from a remote unit to its base station. The remote unit carried by the patient incorporates sensors to obtain the electrocardiogram, the body temperature, and the blood oxygen saturation concentration and to coordinate their transmission to the base station. The system is intended to be home-based, using a personal computer to run a Matlab-based software for data collection and processing. An integrated patient alert feature alerts the patient if one of the measured physiological parameters is outside of the range set by his or her doctor. The software also has the capability to email the collected data to doctors, allowing greater flexibility and convenience for off-site continuous monitoring.

1/ Introduction

The **Wireless patient monitoring device** (Wipmod) system is designed for continuous monitoring of patients in a home setting which makes it distinct from the current products available commercially. Since some cardiac conditions are activity-induced, continuous monitoring of patients provides critical information that cannot necessarily be obtained while a patient is in a medical facility. In addition, the use of wireless technology provides more freedom and a more comfortable, normal lifestyle while being monitored.

Currently, there are wireless monitors that allow more freedom for a patient both within and outside of a medical environment. Ambulatory electrocardiographs can be taken home and can monitor the ECG of the patient in a few different ways.

Continuous monitors are set up in the doctor's office by a professional and then worn by the patient for 24-72 hours. The recorder accumulates all of the data until the patient returns to the office and the recording tape is read by a computer.

Intermittent recorders require more patient participation as they do not monitor the full ECG at all times. Patients must be aware of their bodies and begin recording data when they detect symptoms of their condition. Loop recorders have electrodes that are always attached to the patient and constantly record heartbeats, but the rhythm of the heart is only recorded when the patient initiates the recorder. Event recorders are not attached to the patient, but must be applied to the chest and initiated to start recording. Event recorders are only used when symptoms of the problem occur.

More recently, hospitals are beginning to implement telemetry systems that wirelessly connect patient monitors to nurse stations, and at times, to the attending doctor themselves. These systems provide more freedom for the patients, relieving the bother of multiple wires tying them to a bed, as well as improving treatment by allowing doctors to be directly contacted in the event of emergencies.

1.1/Scope of the project

The Wipmod system consists of two main components, the remote unit and the base station. The remote unit is a compact portable module that integrates multiple physiological sensors such as electrocardiogram electrodes, a body temperature sensor and a pulse oximeter. All the operations in the remote unit such as data sampling and communicating with the integrated wireless module are controlled by a microcontroller. The base station of the monitoring system includes a wireless-module connected to a computer that receives data from the remote unit and a MATLAB-based graphical user interface (GUI) that processes and displays the physiological data. An RF wireless module called XBee that uses Zigbee protocol, which will be described later, was used to create a wireless link between the remote unit and the base station. In addition to processing and display, the GUI also allows the patient to share the collected data with a medical professional or a doctor through electronic mail. The Wipmod system has an

integrated patient alarm system that activates a buzzer in the remote unit if the base station detects abnormal conditions. A general overview of the monitoring system is depicted in Figure 1.



Figure 1. General overview of the Wipmod system¹

1.2/Organization of the report

The block diagram in Figure 2 shows the how the report is arranged.

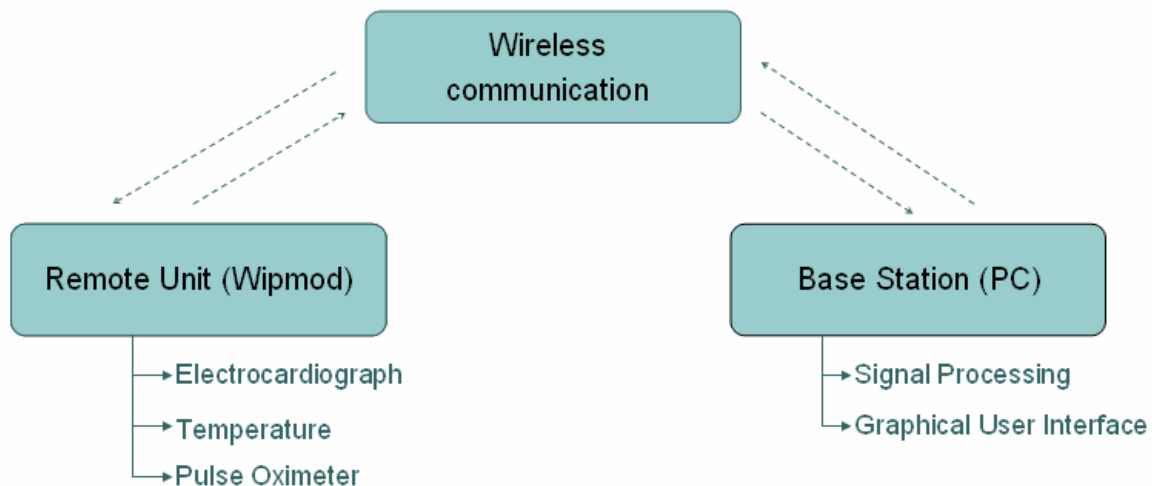


Figure 2. Block diagram showing the organization of the report.

The wireless communication is the integral link between the remote unit and the base station of the monitoring system. Therefore, the specifications and testing of the XBee modules, which are used for wireless transmission of data, are discussed first. This is then

¹ Modified from www.florencecolibrary.org

followed by a general overview and a description of implementation procedures for each of the different components of the remote unit and the base station.

2/Wireless communication

This project revolves around the use of a wireless protocol to allow the patient to move freely within the home while still having important vital signs monitored. To accomplish this task, the IEEE 802.15.4 Zigbee protocol was used. Within the Zigbee protocol, MaxStream produces the XBee module: a single self-contained, programmable wireless transmission module. Two of these are used, one incorporated into the remote unit and one connected to the base station PC, to create the wireless link necessary to allow the freedom of movement desired.

2.1 / Overview of Zigbee

Zigbee is a wireless communication protocol based on the IEEE 802.15.4 standard for wireless networks. Compared to other wireless technologies such as Bluetooth, Zigbee offers a cost-effective and a low power consumption wireless technology with low data transmission rate. However, patient monitoring does not require high speed data transmission and therefore, its low cost and low power consumption makes Zigbee ideal for use in wireless patient monitoring devices.

2.2 /Overview of the Xbee Module

An Xbee module is compatible with any device that has a USART interface such as a microcontroller. For the project, the module in the base unit is connected to a PC through a serial port and the module in the remote unit is connected to the USART pins of the microcontroller (Figure 3).

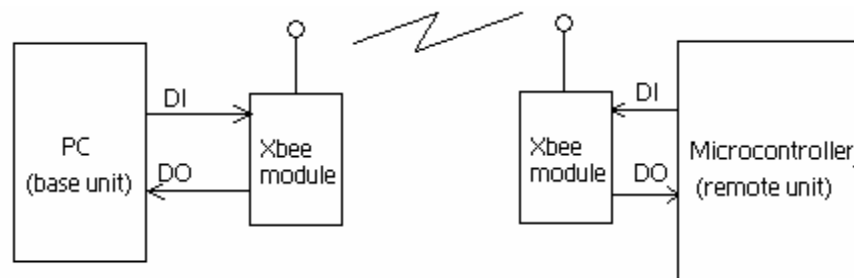


Figure 3. Data flow between the base and the remote unit through the XBee modules. (DI = Data In ; DO= Data Out)

Data received from the DI (Data In) pin is first stored in the DI buffer and then transmitted. The default data flow control is such that the module signals the host device to stop sending information when the DI buffer is almost full. Data loss can be prevented

by ensuring that the data transmission rate is lower than the maximum throughput rate. Data received from transmission first enters the DO (Data Out) buffer which also has a limited amount of space. Therefore, the base unit must read from buffer before it overflows if data flow control is not implemented. The data transmission rate required for the project is approximately 5.0 kbps for which the baud rate was set to 9.6 kbps. This is discussed in detail in Section 2.3. Therefore, active data flow control was not required. The maximum interface data rate of the XBee module is 115.2 kbps.

2.3/General specifications

To limit power consumption and weight of the remote unit, a low-voltage system was desired. The XBee module can be powered between 2.8-3.4 V and draws a typical current of 45 mA when transmitting and 50 mA when receiving data. It is specified to transmit data up to 100 ft. (30 m) indoors and up to 300 ft. (100 m) outdoors. For the purpose of this project, this is an adequate range capability, though MaxStream also produces the XBee Pro module, which would seamlessly replace the XBee module and increase the indoor range up to 300 ft. However, this will increase the power consumption and therefore reduces the battery life.

The serial data baud rate of the module ranges between 1200-115200 bps. The largest concern in choosing the baud rate is ensuring that all ECG data points, which comprises of the majority of the data, can be transmitted. The sampling rate for the ECG waveform used for the project is 620 Hz. Since an 8 bit analog to digital converter was used, each second of data was represented by approximately 5000 bits. By setting the baud rate to 9600 bps, successful transmission of ECG data was accomplished..

The XBee module was obtained as a part of an XBee 802.15.4 Development kit which also included RS323 development board (Figure 4) that allows direct connection to the PC using a serial cable. Along with a RS232 port, the development board has an LED array to indicate signal strength and the direction of communication between the two modules.

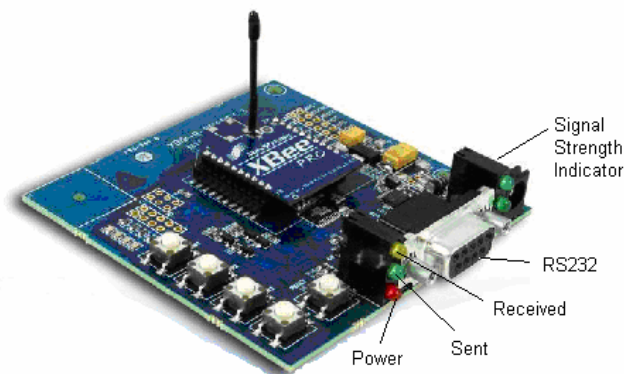


Figure 4. XBee RS-232 development board with module²

² Modified from <http://news.thomasnet.com/images/large/480/480398.jpg>.

2.4/Programming the XBee module

Software called X-CTU is used for programming the XBee module using a development board that connects to a PC through a serial port. Each module contains a unique address and can be given a specific identification name. Once the module is set to command mode using X-CTU software, built-in commands can be used to either read or modify both the destination address and identification name of the chip. Setting the specific addresses and identification information of each chip in a point-to-point communication between two modules prevents cross-talk should other XBee modules be within range.

Through X-CTU, if the serial address of the desired destination is known, it can simply be written to the module currently connected to the computer and saved in its memory. In addition, a module can broadcast a general call to all other modules within range requesting identification name and serial address. In this situation, using a known identification name, the serial address of the desired destination module could be found and then set as the destination address without having direct contact with the module attached to the computer.

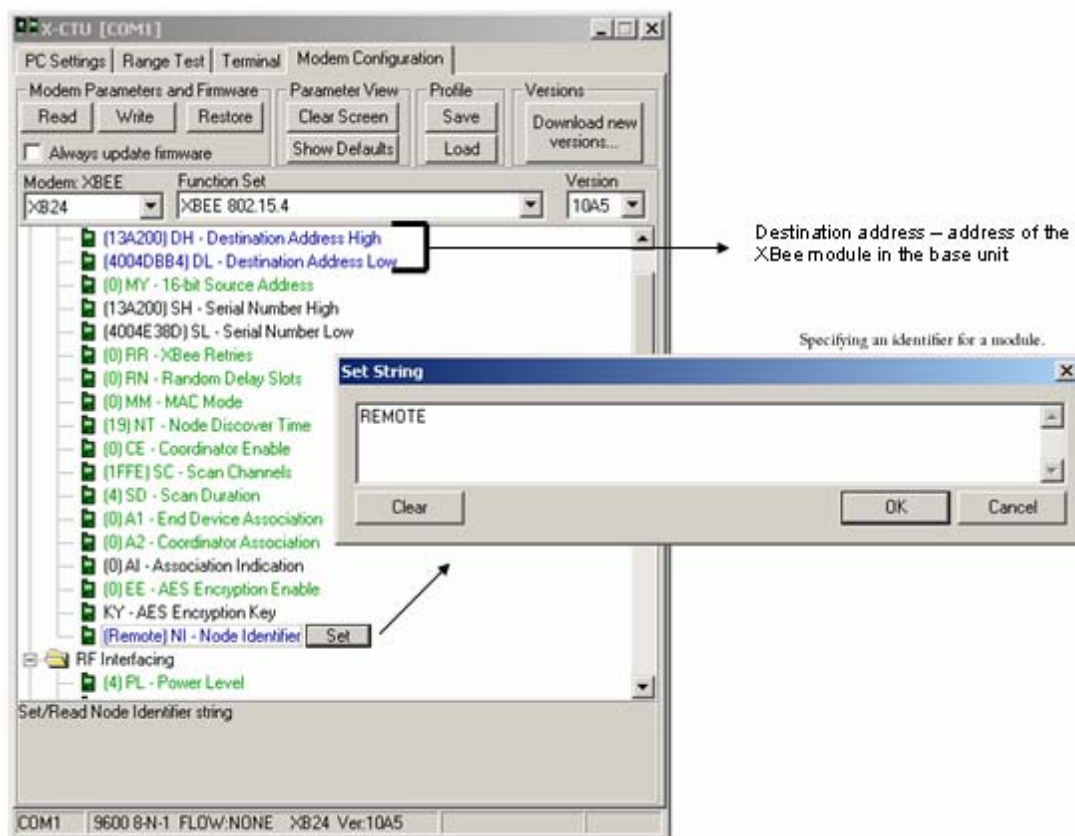


Figure 5. X-CTU software used to program the XBee module for point-to-point communication.

2.5/Testing

Preliminary testing of the modules was conducted using the XBee development boards and HyperTerminal to establish simple two-way communication between two PCs. The next step involved using a PIC microcontroller to transmit to a PC via the XBee modules. Finally, the destination addressing was tested in the presence of other modules and it was confirmed that data was sent solely to the desired destination.

3/Temperature

3.1/Temperature sensor overview

The tympanic temperature sensor used in the project for measuring body temperature is a thermistor-based sensor. Thermistors are semiconductors that are whose resistance changes with its temperature. The thermal coefficient of a thermistor (α) is the change of resistance per unit change in temperature. Thermistors can be classified into two kinds depending on the value of α . Negative Temperature Coefficient (NTC) thermistors have a negative α and its resistance decreases with temperature. On the other hand, the resistance of Positive Temperature Coefficient (PTC) thermistors increases with temperature. The following equation shows the temperature dependency of the thermistor resistance.

$$R_t = R_0 e^{[\beta(T_0 - T)/T_0 T]}$$

where, β = material constant
 T_0 = standard reference temperature, K

The thermal coefficient of a thermistor can be found as follows:

$$\alpha = \frac{1}{R_t} \frac{dR_t}{dT}$$

Although thermal coefficient of a thermistor is a nonlinear function of temperature, a linear approximation can be made for a small range of temperature.

Tympanic temperature sensors are widely used a measure of core body temperature. The tympanic sensor from Novamed (Figure 6) has a comfortable fit that ensures easy placement and minimizes the risk of tympanic membrane perforation.



Figure 6. Thermistor-based temperature sensor

3.2/Sensor calibration



Figure 7. Temperature sensor calibration set-up.

The calibration of the temperature sensor was performed in a controlled temperature bath using a mercury thermometer with a precision of 0.5 degree Celsius (Figure 7). In order to see the effect of temperature on the resistance of the sensor, a voltage divider circuit was used (Figure 8).

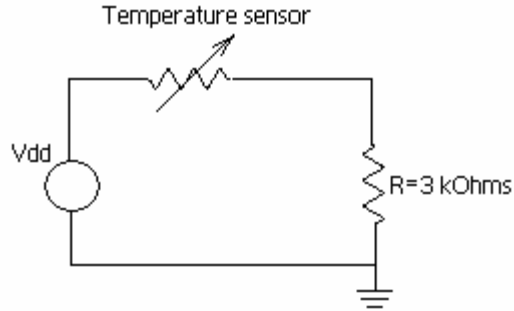


Figure 8. Voltage divider circuit used for calibration.

The voltage across the resistor was recorded using a voltage for different temperature of the water bath. The data collected was then plotted to derive a calibration curve for the sensor. (Figure 9)

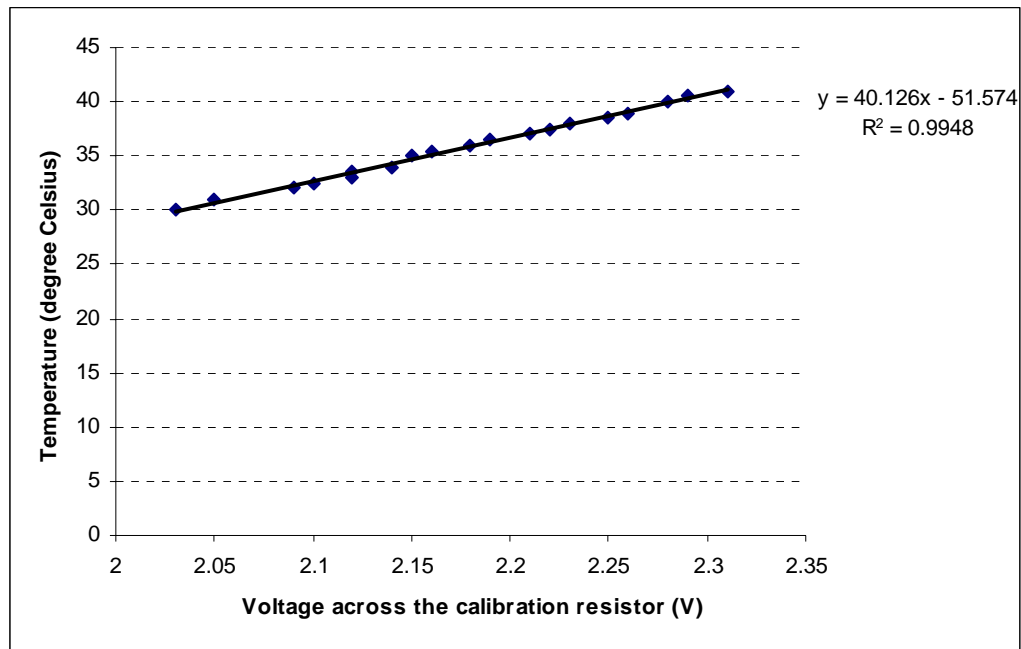


Figure 9. Calibration curve for the tympanic temperature sensor.

It is seen from the data that the voltage across of the calibration resistor increases when the sensor detects a higher temperature. Due to the voltage divider effect, this implies that resistance of the thermistor in the sensor decreases with increasing temperature. Therefore, the calibration data shows that the sensor has a negative temperature coefficient. From the calibration data, it is seen that relationship between the resistance of the sensor and temperature is linear in the temperature range of our interest.

4/Electrocardiograph

The electrocardiograph is an instrument used to produce a graphic representation of the electrical activity of the heart, the electrocardiogram (ECG or EKG), as measured by body-surface electrodes. It is an invaluable tool in the diagnosis of cardiovascular diseases, the monitoring of patients while under anesthesia, and occasionally used for the diagnosis of non-cardiac diseases.

4.1/Characteristics of the electrocardiogram

The activity of the heart can be approximately modeled by an electric dipole, also called the cardiac vector. The magnitude and orientation of the dipole characterize the current state of the heart along the cardiac cycle. To measure these qualities, lead vectors are defined such that, when electrodes are placed along two different vectors that lie in the same plane as the cardiac vector, it can be fully defined. For example, two lead vectors, a_1 and a_2 , can be defined at different angles in the cardiac vector plane (Figure 10). The voltage measured within either lead is the component of the cardiac vector, M , in the direction of that particular lead. In Figure 10, the voltage measured in the a_1 direction is $V_{a1} = |M|\cos \theta$. Since M is orthogonal to lead a_2 , its component V_{a2} is zero. Both leads are necessary to uniquely describe the cardiac vector, but information can still be collected using a single lead.

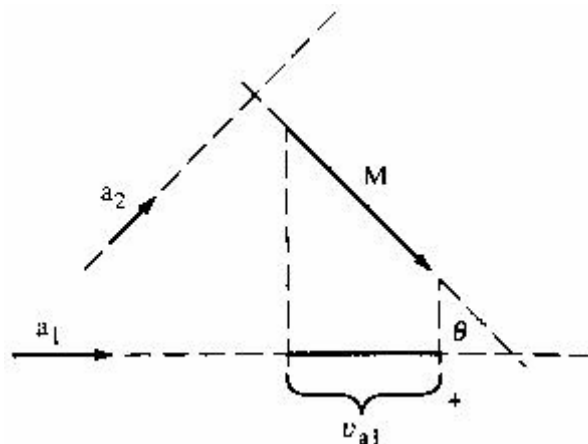


Figure 10. Lead vectors a_1 and a_2 in relation to the cardiac vector M .³

Three basic leads have been defined for the frontal-plane ECG (the plane along the front of the chest). These leads use the convenient markers of the right arm (RA), left arm (LA), and left leg (LL) to define lead vectors Lead I, Lead II, and Lead III that create an equilateral triangle across the center of the chest known as Einthoven's triangle (Figure 11). In this project, the ECG across Lead I will be the main focus.

³ Webster, 236.

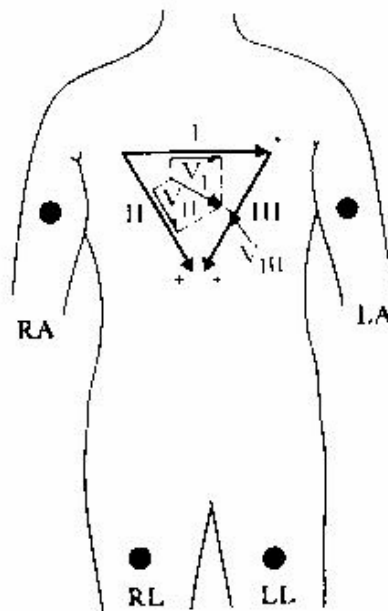


Figure 11. Eindhoven's triangle.⁴

The electrocardiogram is typically composed of three separate components: the P wave, the QRS complex, and the T wave (Figure 12). These components represent the different electrical responses of the heart during the contraction and relaxation phases that compose a single “beat” of the heart.

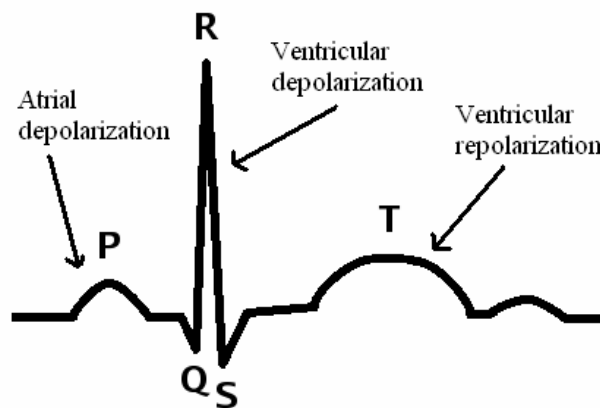


Figure 12. P wave, QRS complex and T wave.⁵

A single heart beat begins when an electrical signal from the S-A node in the atria propagates throughout both atria. This signal depolarizes the atria, therefore causing them to contract and force blood into the ventricles. On the ECG, this is seen as the P wave. The ventricles follow the same fashion when an electrical signal from the A-V node depolarizes both ventricles, causing them to contract. This electrical activity is displayed

⁴ Webster, 237.

⁵ Modified from <http://en.wikibooks.org/wiki/Image:Qrs.png>.

in the ECG as the QRS complex. Finally, the ventricles return to their previous state and prepare for the next contraction, resulting in the repolarization wave, the T wave. Depending on which lead is being examined, the clarity and magnitude of the individual waves varies.

4.2/Analysis of timing features of the electrocardiogram

Time intervals between the different waves of the ECG can help doctors diagnose cardiac abnormalities. Therefore, a few timing features of the Lead I ECG were processed via software to aid in medical diagnosis.

The time between heart beats is one of the most basic qualities that can be extracted from the ECG. Heart malfunction due to an abnormal rhythm of the heart is called an arrhythmia. A fast heart rate, called tachycardia, generally results from an increased body temperature, stimulation of the heart by the sympathetic nerves, or toxic conditions of the heart (Figure 14). The opposite condition, a slow heart rate or bradycardia, may result from problems with the heart's electrical system and natural pacemaker (Figure 15). Extreme cases of bradycardia may mean that the heart is not pumping enough blood to meet the body's needs. To aid in the diagnosis of such conditions, the ECG is processed to extract the average heart rate of the patient and continuously display it at the base station.



Figure 13. Normal sinus rhythm for leads I, II, and III. Note that the peak-peak distance is approximately 3 divisions (0.20 seconds per division).⁶

⁶ Guyton, 118.

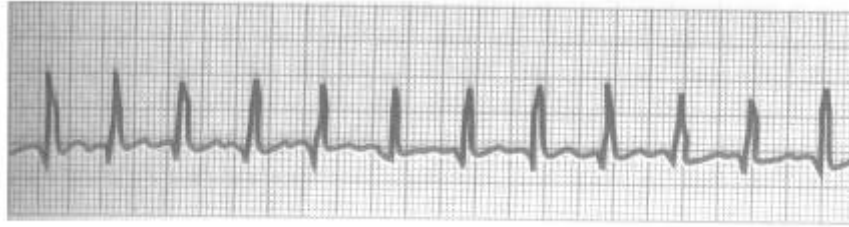


Figure 14. Tachycardia. Note that the peak-peak distance is approximately 2 divisions (0.20 seconds per division).⁷

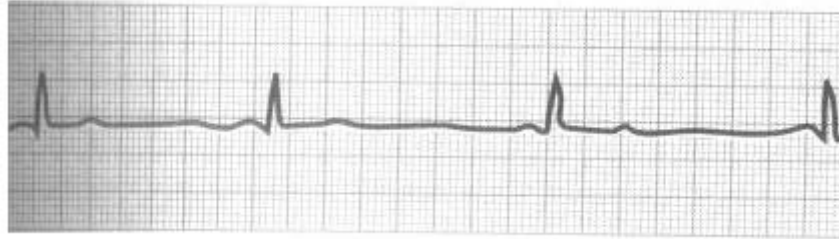


Figure 15. Bradycardia. Note that the peak-peak distance is approximately 7 divisions (0.20 seconds per division).⁸

The duration of the QRS complex is also calculated and displayed at the base station. Since the QRS complex lasts as long as depolarization of the ventricles lasts, an abnormally prolonged or otherwise irregular duration of the complex indicates improper electrical conduction throughout the heart. This can be due to the deconstruction of heart muscle throughout the ventricular system that is then replaced with scar tissue, or the appearance of multiple small local blocks in the Purkinje system, the part of the heart that conducts electrical impulses to the ventricles. A normal QRS complex lasts 0.06 to 0.08 seconds; anything longer than 0.09 seconds is considered abnormal.

5/Electrocardiograph circuit

To acquire the electrocardiogram, multiple stages of analog circuitry were implemented in the remote unit prior to transmitting the signal to the base station. The signal is passed from two sensing electrodes positioned on opposite sides of the chest to a difference amplifier, which then sends the ECG through filters to be sampled by the microcontroller and sent to the base station. To counter the effect of the common-mode, a driven-right-leg circuit provides negative feedback via a third electrode placed on the right leg. A graphical representation of this path can be seen in Figure 16. Each of these blocks will be described in detail individually.

It is important to note that no protective circuitry is included in the design. For the patient, this does not pose any risk from the remote unit; it is run on batteries and the voltage is not sufficient to do any harm in any event. However, should a defibrillator be

⁷ Guyton, 135.

⁸ Guyton, 135.

used on the patient while the Wipmod is still connected, the discharge would destroy the circuitry.

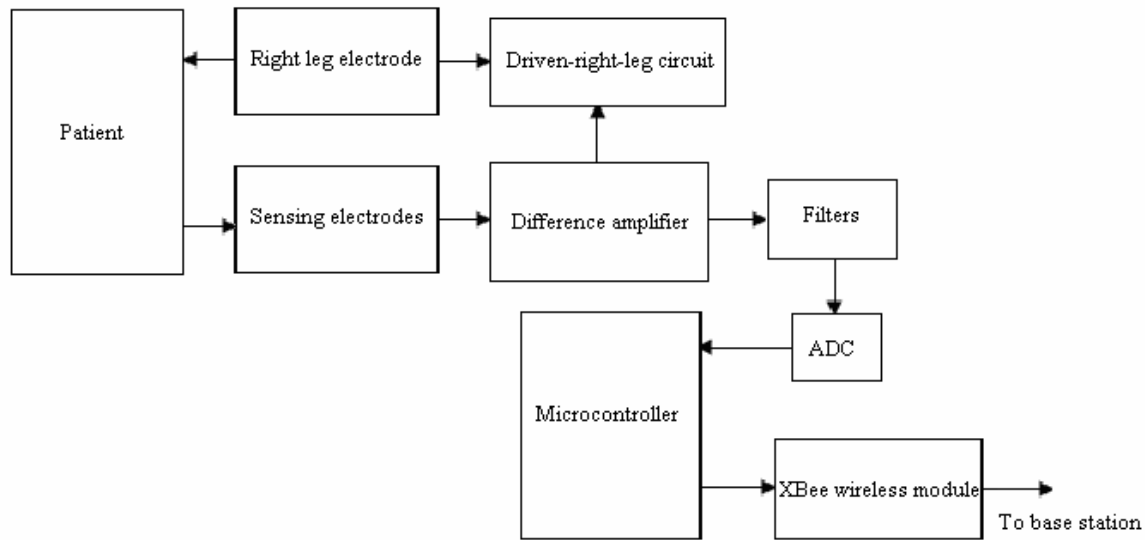


Figure 16. Block diagram of ECG circuitry.

The circuitry used to perform the functions in each block of Figure 16 were designed and simulated in Multisim before creating them on a breadboard. The Multisim circuit is shown below in Figure 17.

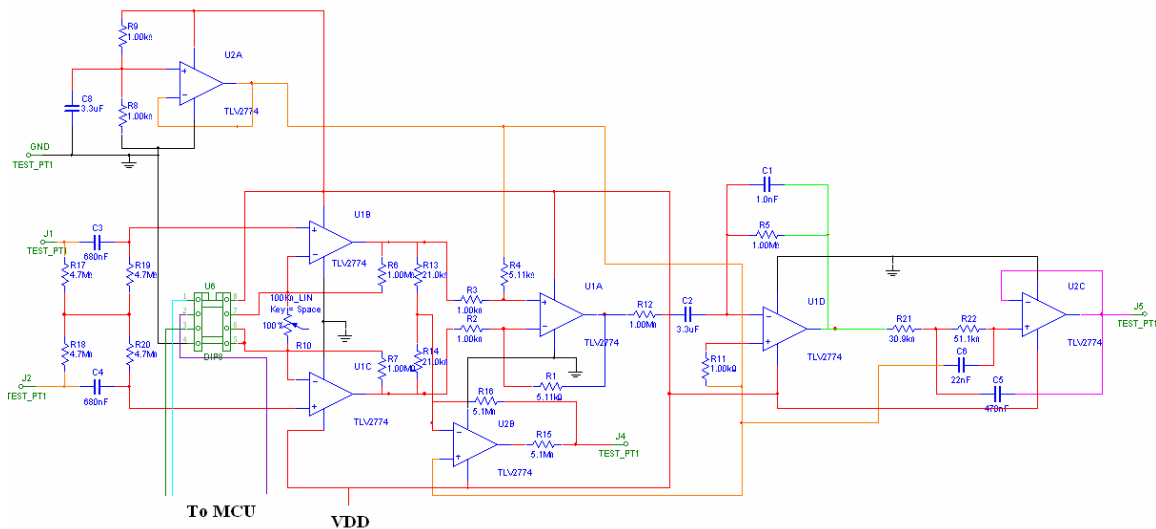


Figure 17. Multisim schematic of complete ECG circuit.

5.1/Single-supply op-amps and virtual ground

The fact that the remote unit is powered entirely by batteries introduces the additional consideration of single-supply circuitry. TLV2774 op-amps were chosen specifically for their qualities that make them ideal for a 3.3V battery-operated application:

- Single-supply
- Rail-to-rail output
- Low supply current of 1mA

In order to display the oscillating AC signal correctly, a virtual ground must be created around which the signal may oscillate from positive to negative. This is accomplished with the circuit in Figure 18.

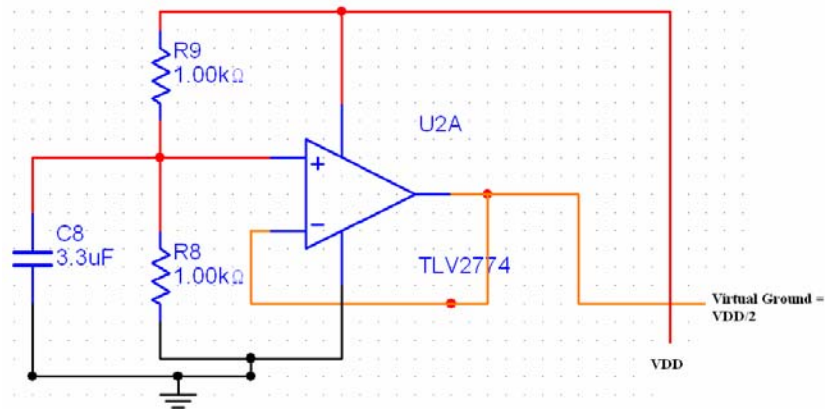


Figure 18. Virtual ground circuit

The op-amp is configured as a voltage follower with resistors R8 and R9 serving as a voltage divider to divide VDD in half.

$$\text{Virtual ground} = \left(\frac{R8}{R8 + R9} \right) VDD = \frac{1}{2} VDD$$

The capacitor is a bypass capacitor that serves to reduce the effect of non-uniform current draw on the op-amp by the rest of the circuit.⁹

5.2/Electrodes

To acquire the ECG, 3M soft cloth Red Dot electrodes are used. These are body-surface electrodes that use an Ag/AgCl interface to transmit the biopotential signal. The metal does not directly contact the skin, but a solid-state electrolytic gel is used to ensure good electrical contact without the mess of a more liquid-based gel. The backing is a breathable, stretchable cloth for patient comfort and to maintain good contact when the patient moves.

⁹ Cheever, "Single Supply Op Amps."



Figure 19. 3M Red Dot electrode.¹⁰

5.3/Instrumentation amplifier

To characterize lead I, the potential difference across the chest must be measured. To this end, an instrumentation amplifier was employed. At the front end of the amplifier, a passive AC-coupling circuit is the first step to cutting out any DC offset potential that may be present in the patient's body or at the electrode-skin interface. To avoid drawing too much current and distorting the signal, an instrumentation amplifier uses a voltage-follower-with-gain op-amp at each input, thereby ideally drawing zero current from the patient's body.

The passive AC-coupling circuit was implemented as suggested by Spinelli, et. al to maintain a high common mode rejection ratio for the instrumentation amplifier while consuming no additional power.¹¹

To reduce the amount of gain that must be obtained from the difference amplifier stage of the instrumentation amplifier, the first stage amplifies the difference signal before passing it to the second stage. The common-mode is passed through the first stage, but not amplified. The gain varies according to the 100 kΩ digital potentiometer placed between the positive terminals of the two op-amps. A digital potentiometer functions as a variable resistor whose value can be set by a microcontroller. The gain for the first stage is found to be,

Equation 1

$$V_2 - V_1 = \left(\frac{2R_2}{2R_1} + 1 \right) V_{ID}$$

With the resistor values in this configuration, the first stage ideally can have a gain of 10 to ∞ . The specific purpose and function of this component will be discussed later in Sections 5.6 and 9.2.1. Figure 20 below presents this first stage of the instrumentation amplifier.

¹⁰ Picture retrieved from <http://www.grogans.com/servlet/shop?cmd=I&id=3M9641>.

¹¹ Spinelli, et. al, 2003.

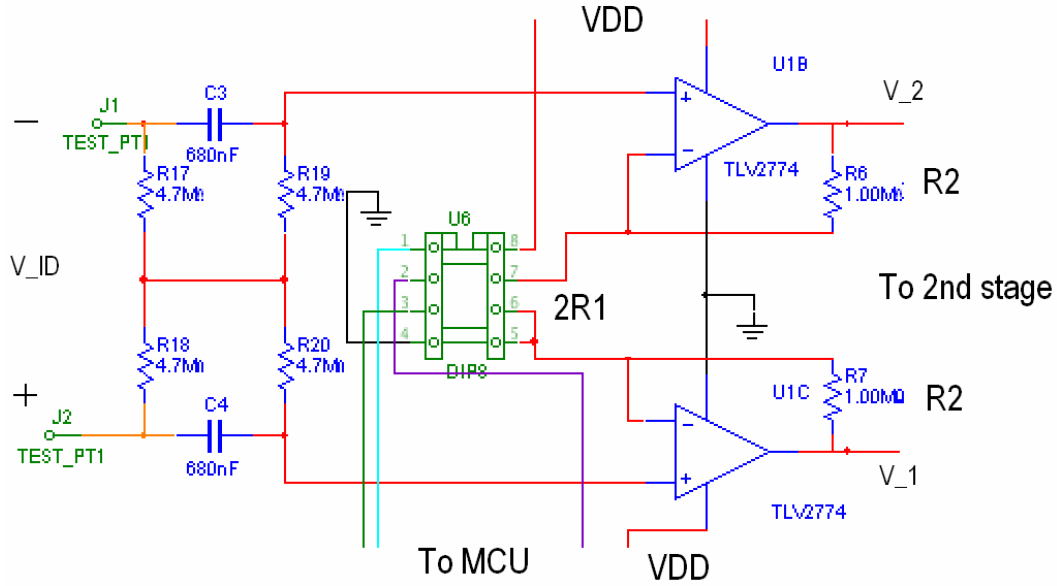


Figure 20. First stage of instrumentation amplifier

The second stage of the instrumentation amplifier further amplifies the signal and rejects the common-mode of the signal. This stage is composed of a single difference op-amp. The gain that can be obtained from this stage can be calculated as shown below (Equation 2). Combined with the gain from the first stage, the entire system has a gain from 51.1 to, again ideally, ∞ . The ECG signal ranges from 0 to around 1 mV in amplitude. Therefore, to amplify the signal to a usable range of 0 to around 2 V, a gain of approximately 2000 is necessary. Thus, the circuit was designed so that the potentiometer's value can increase or decrease so that the circuit can adapt to signals with amplitudes either greater or less than the average.

Equation 2

Second stage gain,

$$V_o = \left(\frac{R_4}{R_3} \right) V_{ID}$$

Overall gain,

$$V_o = \left(\frac{R_4}{R_3} \right) \left(\frac{R_2}{R_1} + 1 \right) V_{ID}$$

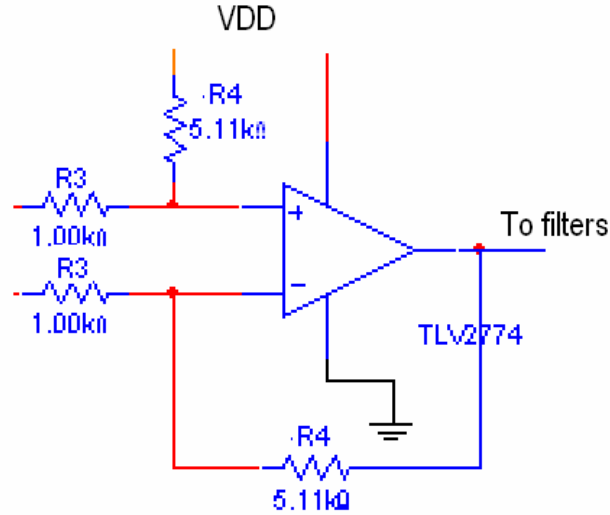


Figure 21. Second stage of instrumentation amplifier

The Common-Mode Rejection Ratio (CMRR) has been calculated to be 62.9 dB (Equation 3).

Equation 3

$$\text{CMRR} = \frac{A_{\text{DM}}}{A_{\text{CM}}} = 1391 = 62.9 \text{ dB}$$

5.4/Filters

The difference signal obtained from the instrumentation amplifier still contains a large amount of noise. Some of the noise derives from electrical interference present in the environment of the patient whether that is from appliances in the vicinity, power lines overhead, or the electrical system of the building. Also, a DC offset may be present due to non-ideal input currents in the op-amps in the circuit itself. Therefore, before the signal is of any use, it must first be filtered to remove noise as much as possible.

5.4.1/Band-pass filter

The ECG contains frequencies ranging from about 0.05 Hz to 1 kHz, but an electrocardiograph with a frequency response of 0.05 to 150 Hz can produce a well-defined ECG. This is approximately the bandwidth used for the band-pass filter (Figure 22) whose transfer function can be seen below in Equation 4.

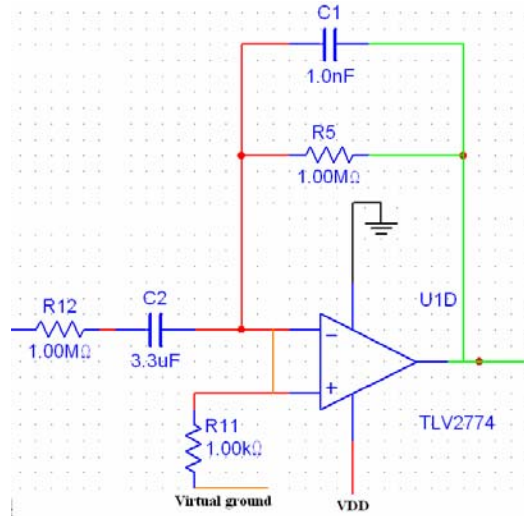


Figure 22. Band-pass filter.

Equation 4

$$H(s) = \frac{-sR_{12}C_2}{\left(1 + \frac{s}{\frac{1}{R_{12}C_2}}\right)\left(1 + \frac{s}{\frac{1}{R_5C_1}}\right)} = \frac{-sR_{12}C_2}{\left(1 + \frac{s}{\omega_L}\right)\left(1 + \frac{s}{\omega_H}\right)}$$

where, ω_L = lower threshold frequency
 ω_H = higher threshold frequency

Using the values shown in Figure 22, the cut-off frequencies, ω_L and ω_H , are 0.048 Hz and 159.2 Hz respectively. The Bode plot for this filter is shown below in Figure 23.

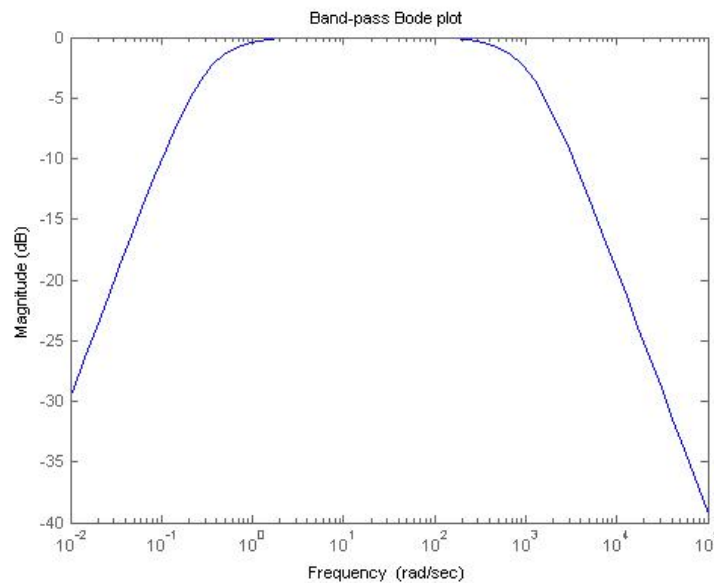


Figure 23. Bode plot for the band-pass filter.

5.4.2/ 2nd order low-pass filter

Testing showed that the band-pass filter still permitted an unacceptable amount of high-frequency noise to distort the signal. Although the band-pass filter had a cut-off at 159.2 Hz, high-frequency noise was not being attenuated sufficiently. Therefore, a second-order low-pass filter was added to the output of the band-pass filter. A Sallen-Key filter was chosen since it can perform the necessary function utilizing only one op-amp with a few passive components (Figure 24). This filter's transfer function is shown in Equation 5.

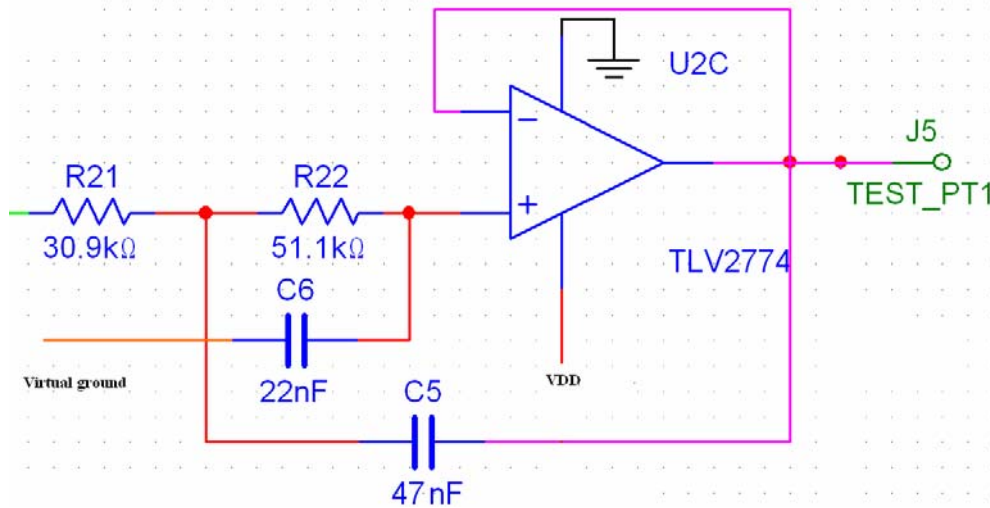


Figure 24. 2nd order low-pass filter.

Equation 5

$$H(s) = \frac{\left(\frac{1}{R_{21}R_{22}C_5C_6} \right)}{s^2 + s \left(\frac{1}{R_{22}} + \frac{1}{R_{21}} \right) \left(\frac{1}{C_5} \right) + \left(\frac{1}{R_{21}R_{22}C_5C_6} \right)}$$

So,

$$\omega_{cut-off} = \sqrt{\frac{1}{R_{21}R_{22}C_5C_6}}$$

Again, using the values from Figure 24, $\omega_{cut-off} = 785.398 \text{ rad} \cdot \text{s}^{-1}$ or 125 Hz. This cut-off frequency is different from the high-frequency cut-off of the band-pass filter. By choosing this cut-off point to be lower than the desired cut-off of 150 Hz, the attenuation at that point is greatly increased, thereby ensuring that noise at or above 150 Hz is excluded from the final signal. The Bode plot for this filter is given below in Figure 25.

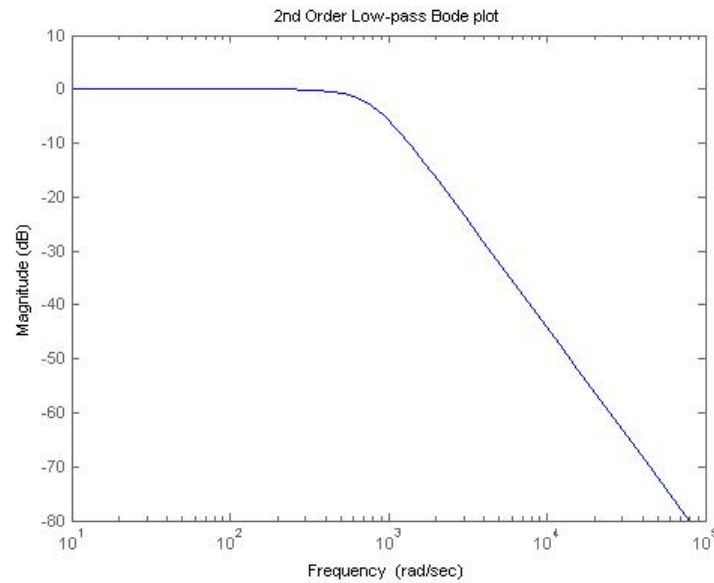


Figure 25. Bode plot for the 2nd order low-pass.

5.5/Driven-right-leg circuit

Once the signal passes through first stage of the instrumentation amplifier, it is carried along the driven-right-leg circuit to an electrode on the right leg. This circuit functions as a negative feedback loop to reduce the transmission and amplification of the common-mode portion of the signal. This is demonstrated below in Figure 26.

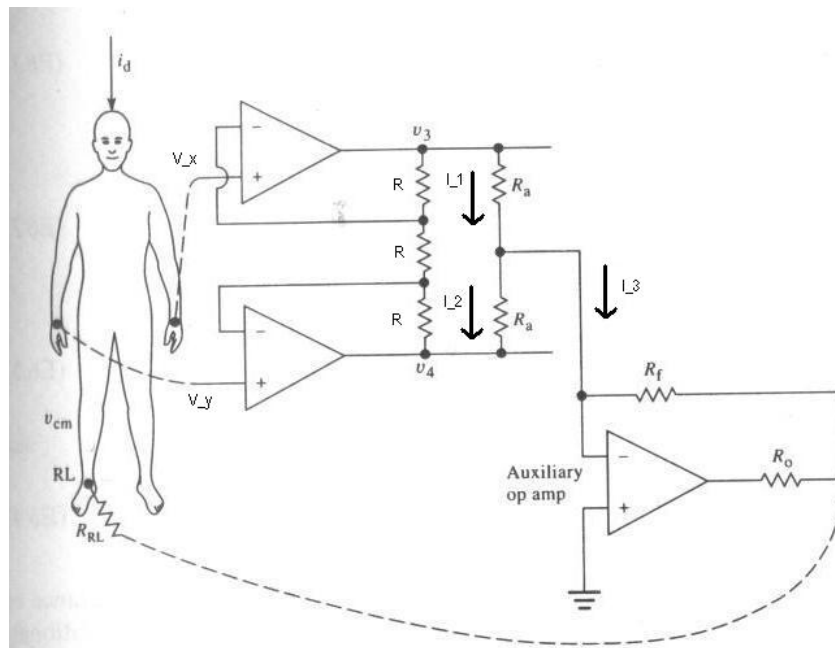


Figure 26. Driven-right-leg circuit.¹²

¹² Webster, 257.

If $V_{ID} = V_x - V_y$, then taking the common-mode voltage (V_{cm}) into consideration, we can model the input voltages as,

Equation 6

$$V_x = V_{cm} + \frac{V_{ID}}{2} \quad \text{and} \quad V_y = V_{cm} - \frac{V_{ID}}{2}$$

With the three resistors placed in series between the outputs of the two op-amps set to the value R , $V_3 = 2V_x - V_y$ and $V_4 = 2V_y - V_x$. If no common-mode voltage is present, $V_x = -V_y$, $V_3 = -V_4$ and $I_1 = I_2$, making $I_3 = 0$. In this situation, the feedback op-amp is not activated and no current travels between the circuit and the right leg.

However, in the event that a common-mode voltage is present, I_3 will be non-zero. For example, suppose that there is a common-mode voltage of 2V.

If, $V_{ID} = 4V$ (chosen for the simplification of the calculation),

$$V_x = 4V \text{ and } V_y = 0V$$

this results in,

$$V_3 = 2(4) - 0 = 8 \quad \text{and} \quad V_4 = 2(0) - 4 = -4.$$

Since the negative terminal of the feedback op-amp is ideally ground,

$$I_1 = \frac{8V}{R_a} \quad \text{and} \quad I_2 = \frac{4V}{R_a}$$

This means $I_3 = \frac{4V}{R_a}$ and this current must pass through R_f , making the output:

$$I_3 = \frac{4V}{R_a} = \frac{0 - V_{\text{output}}}{R_f}$$

thus,

$$V_{\text{output}} = -\frac{4R_f}{R_a}$$

V_{output} is connected to the patient's body via the right leg and so it actively works to lower the common-mode potential of the body to close to zero. Thus, it works to help the instrumentation amplifier refine the ECG signal and retain just the desired signal.

5.6/Adaptive gain

The voltage levels of the different waves in the ECG can be analyzed to extract medically important information. However, factors other than cardiac conditions could result in voltage differences from patient to patient. Electrode placement and skin moisture, for example, both play a role in how strong of a signal can be obtained through the body-surface electrodes. To ensure that other features of the ECG can still be detected and ease

the use of Wipmod on any patient, an adaptive gain was implemented that ensures that the ECG signal reaches a certain minimum voltage. This is accomplished with a digital potentiometer controlled by the microcontroller. The function and use of this component is described in detail in Section 9.2.1.

6/Pulse oximeter

Definitions:

Diastole- relaxation of heart muscles

Reduced hemoglobin (Hb) - hemoglobin unbound to oxygen molecule.

Oxygenated hemoglobin (HbO₂) - hemoglobin with a bound oxygen molecule.

SpO₂- pulse oximeter measurement

SaO₂- arterial blood's actual oxygen

Systole- contraction of heart muscles

The oxygen saturation of the blood is an important physiological parameter that medical personnel use as an indication of various respiratory conditions. Hemoglobin, a respiratory pigment, present in the red blood cells provides a binding mechanism for oxygen. When hemoglobin is combined with oxygen molecules, it changes color. Oxygenated hemoglobin is bright red while a deoxygenated hemoglobin molecule is bright red. This change in color is used in the application of pulse oximetry for to estimate arterial blood saturation level.

6.1/Principles of pulse oximetry

A pulse oximeter sensor shines light of two different frequencies through a capillary bed such as the finger and the ear lobe. The two frequencies of light, red (660 nm) and infra red (940 nm) are chosen so that there is a large difference in the absorption of these lights by oxygenated and reduced hemoglobin. The opaqueness of the Hb and HbO₂ for a light of certain frequency is called its extinction coefficient. The extinction coefficient of Hb and HbO₂ for various frequencies is shown in Figure 27.

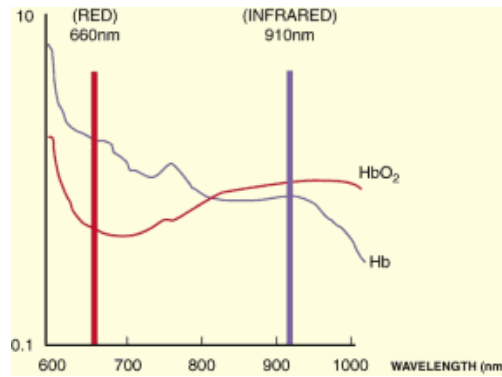


Figure 27. Extinction coefficients of Hb and HbO₂.¹³

As seen from the figure, while the extinction coefficient of HbO₂ is higher than that of Hb at infrared frequency (940 nm), the reverse is true at 660 nm. This implies that if SaO₂ increases, the absorption of infrared light will increase while the absorption of the red light will decrease.

The light transmitted through the tissue bed is absorbed by tissues and bones along with components of the blood. Therefore, it is necessary to distinguish between the tissue/bone absorption and absorption by hemoglobin. The pulse oximeter takes advantage of the pulsatile nature of the arterial blood for this purpose. The absorption of light by hemoglobin increases during systole and decreases during diastole. The total absorption of light comprises of a DC component caused by tissues, bones and non-pulsating arterial blood and an AC component due to the changes in the arterial size during systole and diastole.

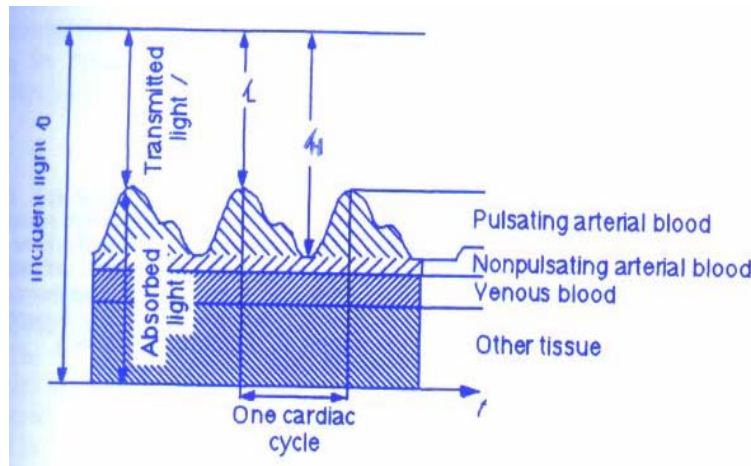


Figure 28. AC and DC components of the total absorption of LED lights. I_L represents the least transmitted light and I_H represents the maximum transmitted light in one cardiac cycle.

In order to account for different emission intensity of red and infrared LEDs, the measured light intensities have to be normalized. This normalization is done by dividing

¹³ (<http://www.oximetry.org/pulseox/principles.htm>)

the transmitted intensities with the by the peak transmitted values of each wavelength. The ratio 'R' is then calculated as follows:

$$R = \ln (I_{L,R} / I_{H,R}) / \ln (I_{L,IR} / I_{H,IR})$$

where $I_{L,R}$ = normalized least transmitted red light

$I_{H,R}$ = normalized maximum transmitted red light

$I_{L,IR}$ = normalized least transmitted infrared light

$I_{H,IR}$ = normalized maximum transmitted infrared light

6.2/Theoretical calibration curve

The functional oxygen saturation can be calculated from the ratio R using the following equation:

$$SaO_2 = \frac{\varepsilon_{Hb}(\lambda_R) - \varepsilon_{Hb}(\lambda_{IR})R}{[\varepsilon_{Hb}(\lambda_R) - \varepsilon_{HbO_2}(\lambda_{IR})R] + [\varepsilon_{HbO_2}(\lambda_R) - \varepsilon_{Hb}(\lambda_{IR})]R} \times 100\%$$

where, ε_{Hb} = extinction coefficient of Hb

ε_{HbO_2} = extinction coefficient of HbO₂

7/Pulse oximeter circuit

7.1/Some reverse engineering

Oximax pulse oximeter sensors from Nellcor are designed to be used with pulse oximetry monitors. Since no datasheets or technical information was available about the sensors, one of the sensors was taken apart and its components were tested and their operation analyzed. The sensor consisted of two LEDs, red and infrared, a photodiode and copper shielding with small slits for the photodiode. The copper shielding is placed in order to reduce error due to electromagnetic interference. Figure 29 shows the sensor and pulse oximeter sensor and its components.

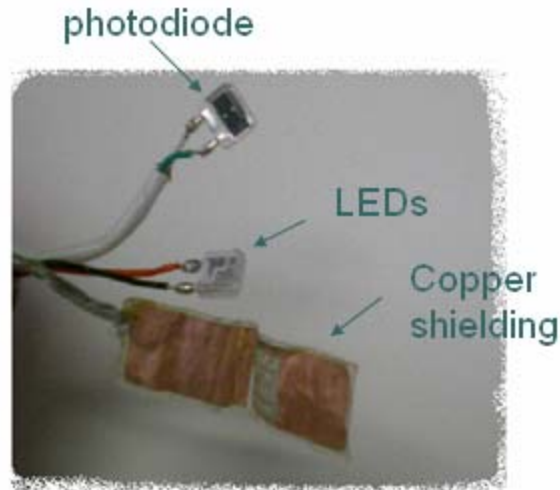


Figure 29. Pulse oximeter sensor and its components

The two LEDs are arranged in parallel to each other in opposite orientations such that only one LED turns on at a time. When light from the LED falls on the photodiode, it allows current to pass through it depending on the intensity of the light. This was verified using a resistor in series with the photodiode which resulted in a voltage drop across the resistor when the LED was turned on.

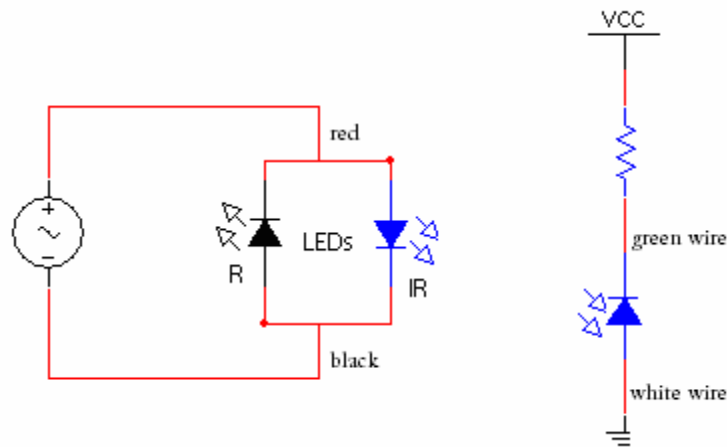


Figure 30. Arrangements of LEDs and photodiode in the pulse oximeter sensor.

The pulse oximeter sensor was equipped with a RS-232 male port to make it compatible with commercial pulse oximeter monitors. The connections of each of the pins of the port are shown in Table 1.

Pin Number	Connections	Wire color
1	Internal shielding	Large white
2	LED	Red
3	LED	Black
4	Not connected	
5	Photodiode	White
6	External shielding	
7	Copper Shielding	
8	Not connected	
9	Photodiode	Green

Table 1. RS-232 pin connections in the pulse oximeter sensor

7.2/ Block diagram of pulse oximeter circuit

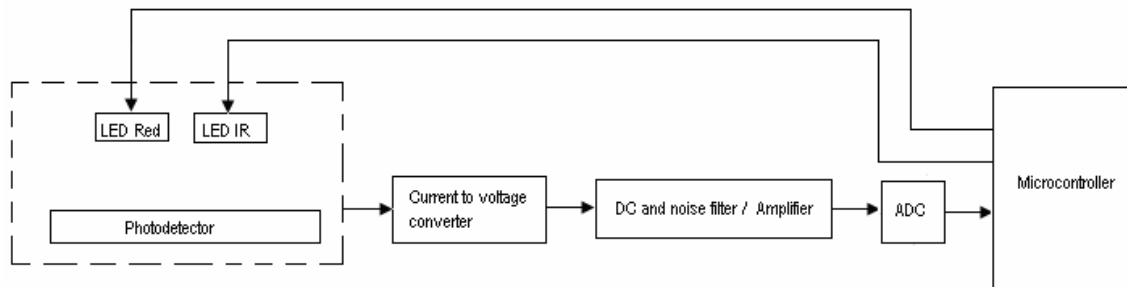


Figure 31. Block diagram for the pulse oximeter circuit.

In a pulse oximeter sensor, the timing of the two LEDs, red and infrared, is controlled such that only one of the two LEDs is on at a time. This control was acquired using a microcontroller which was programmed such that LEDs switched about 200 times per second. The output of the photodiode due to the incident light from the LEDs was then processed using analog circuitry shown in Figure 32.

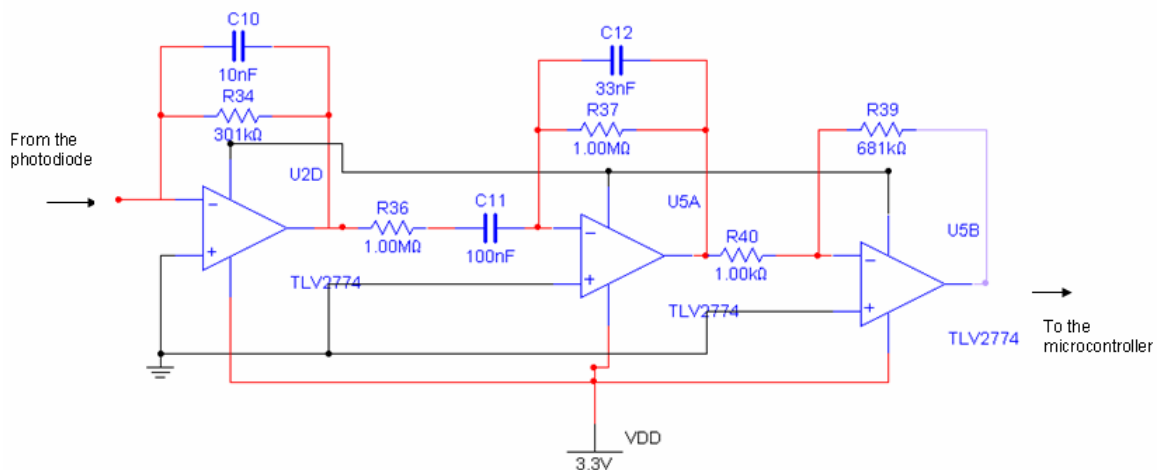


Figure 32. Multisim schematic of the pulse oximeter circuit

The first stage of the circuit is a current to voltage converter. When the light falls on the diode, the current flows through the negative feedback resistor which creates a voltage proportional to the current at the output. The frequency range of interest for a pulse oximeter signal is 0.5 Hz to 5 Hz.¹⁴ The feedback capacitor, C10, forms a low pass filter with a cutoff frequency of 5 Hz. The second stage of the circuit is a band-pass filter with the passband from 0.48 Hz to 4.8 Hz. Finally, the band-pass filtered signal is amplified to get the final output shown in Figure 33. The image shows the output waveform when only the red LED was on. This output was then sent to the microcontroller.

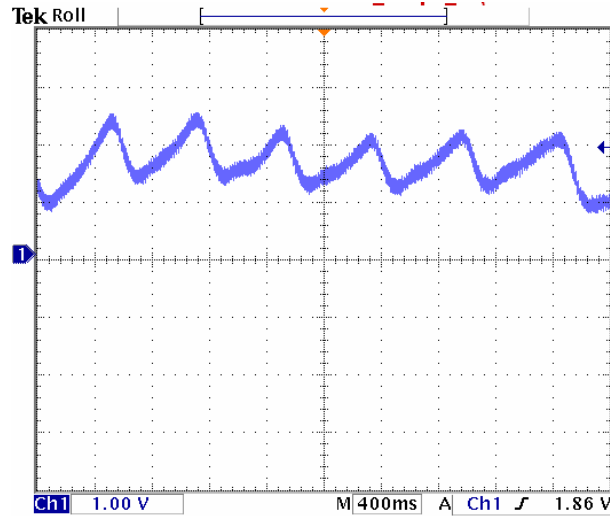


Figure 33. Output waveform of the pulse oximeter circuit for the red LED.

While the LED timing and the analog processing was tested and verified, due to time constraints, the sampling of the analog output for red and infrared light in analog to digital converter of the microcontroller was not completed.

8/ Batteries

Four 1.2V AAA Nickel Metal Hydride Energizer batteries were used to power the remote unit. The maximum dropout voltage of the 3.3V voltage regulator incorporated in the circuit is 0.7 V. Therefore, at least four 1.2V batteries were required. The maximum supply current required for the remote unit was 62.5 mA. The supply current requirements for each component of the remote unit are shown in Table 2. The batteries are 900 mAh and can therefore last for about 12 hours if the remote unit is used continuously.

¹⁴ Townsend N. 2001.

Component	Max current supply
PIC microcontroller	2 mA
Xbee Module	50 mA
Op-amps	10 mA (1mA * 10)
Digital potentiometer	0.5 mA
Total	62.5 mA

Table 2. Maximum current supply for each of the main components of remote unit.

Rechargeable batteries were used to minimize the costs, both economical and environmental, of non-rechargeable batteries. Two sets of batteries would allow the user to operate the Wipmod system for almost 24 hours, using one set while recharging the other.

9/ PIC microcontroller

9.1/ General characteristics

The basic requirements for the microcontroller to be incorporated in the remote unit are listed below. To meet these requirements the PIC16LF877A was chosen.

1. *Low operating voltage* – The maximum operating voltage of the XBee module is 3.4V. This makes it imperative for the microcontroller to operate with a source less than 3.4 V. The PIC16LF877A has a wide operating voltage range from 2.0 V to 5.5V. This feature makes it compatible with a low voltage source.
2. *8 bit analog to digital converter* – The analog to digital converter in The PIC16LF877A can be set to either an 8 bit or a 10 bit ADC.
3. *3 ADC channels* – For continuous monitoring of temperature, ECG and blood oxygen saturation, the microcontroller has to be equipped with at least three ADC channels. The PIC16LF877A has 8 ADC channels.
4. *Timers* – Timers are required for controlled sampling rate of ECG, temperature and blood oxygen saturation. The PIC16LF877A provide 3 timers.
5. *Universal Synchronous Asynchronous receiver (USART)* - This feature is required to send the sampled data to the XBee module serially for wireless transmission to the base unit.
6. *Synchronous Serial Port (SSP) with SPI* – This allows the synchronous communication between the microcontroller and a peripheral device. SPI is required for controlling the digital potentiometer for adaptive gain of ECG waveform.

9.2/Programming the PIC

The PIC microcontroller is programmed to control the physiological data collection, adaptive gain of the ECG and the patient alarm system. All of these operations require controlled timing which was achieved by the use of timers. Timers are counters that reset every time the counter overflows. For example, RTCC (Real Time Counter Clock) is an 8 bit counter which counts from 0-255 and then resets to 0. This process repeats continuously once the timer is setup. At each overflow, timers generate an interrupt. The code written within the interrupt service routine is executed when an interrupt occurs.

The three timers in PIC16LF877A are as follows:

- a. RTCC timer/ Timer 0 – an 8 bit counter
- b. Timer 1 – a 16 bit counter
- c. Timer 2 – an 8 bit counter

The oscillator clock can be scaled down by using different pre-scalar state (in RTCC) and modes (in Timer 1 and 2). For example, for a 4MHz clock, instruction cycle is $4/4 = 1$ MHz. This means that each instruction requires 4 cycles. In case of RTCC timer, if this is pre-scaled using RTCC_DIV_256, then the timer increments $(1/256)$ MHz = 3900 times per second). For an 8 bit counter, this implies that the counter overflows $3900/255 \approx 15$ times per sec, hence generating a timer which causes the interrupt routine to be executed at 15 Hz.

Each timer can be used to execute actions at different frequencies by using variables as secondary counters which decrements from its maximum value. In the project, using secondary counters, RTCC timer was used to sample ECG, temperature data and to control the buzzer in the patient alert system. Figure 34 shows the process that was implemented in the microcontroller for interrupt generation and execution of interrupt routines for a single secondary counter. As shown in the figure, some of the processes are sequential, while others are simultaneous. For example, the timers are initiated only after a start signal is received from the base station, however, the “Timer” and the “Infinite loop” processes take place simultaneously. Using a similar implementation, different counters were used for temperature sampling and patient alert system, namely *temp_START* and *buzz_START* respectively (See Appendix C).

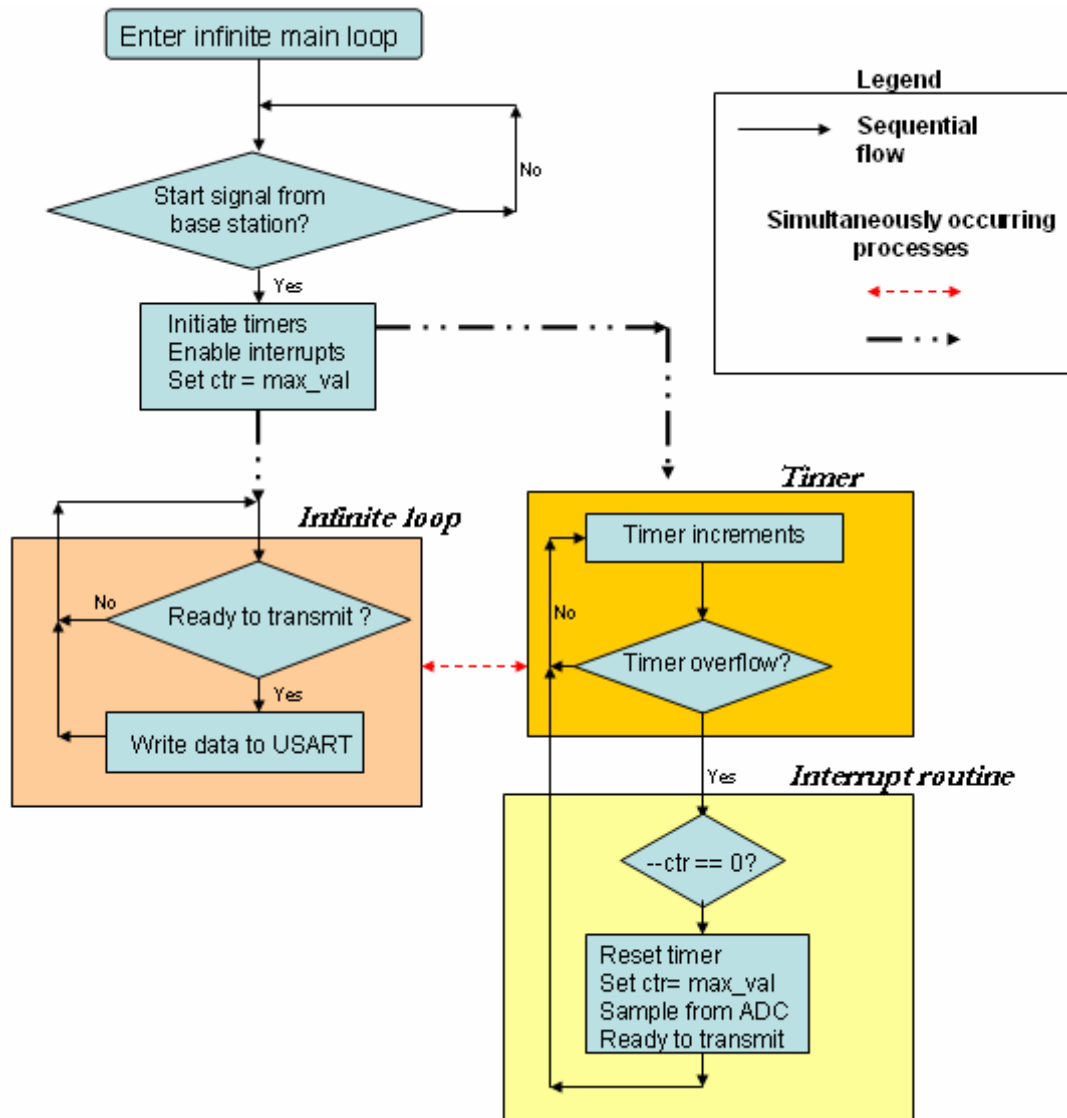


Figure 34. Flowchart demonstrating how timer/interrupts were generated to control sampling rate using counters.

For each timer, it is also possible to set the timer to start from a value other than 0 by using a `setup_timer` function. This approach was also implemented to obtain a desired sampling rate for data. In the above example, if `setup_rtcc(245)`, the interrupt routine will be executed at $3900 / (255 - 245) = 390$ Hz.

9.2.1. Digital potentiometer

The digital potentiometer, or digipot, is a peripheral component controlled by the microcontroller. As mentioned in Section 7.6, the digipot controls the adaptive gain of the electrocardiograph to allow the circuit to adapt itself to the specific conditions of the patient.

To communicate between the microcontroller and the digipot, it was necessary to use the Serial Peripheral Interface (SPI) protocol. The PIC16LF877A has hardware specifically designated for this sort of communication and so the digipot could be directly programmed without having to create additional software.

SPI works on a master-slave system in which one master can control several slaves. The master sets the clock with which the slave operates and can send instructions to all of the peripheral slaves or just one. Also, slaves can be cascaded so that one slave is controlled by the output of another. Nonetheless, in this application, only one slave is controlled by the master. The pin connections between the master PIC16LF877A and the slave MCP41100 digital potentiometer are shown below (Figure 35).

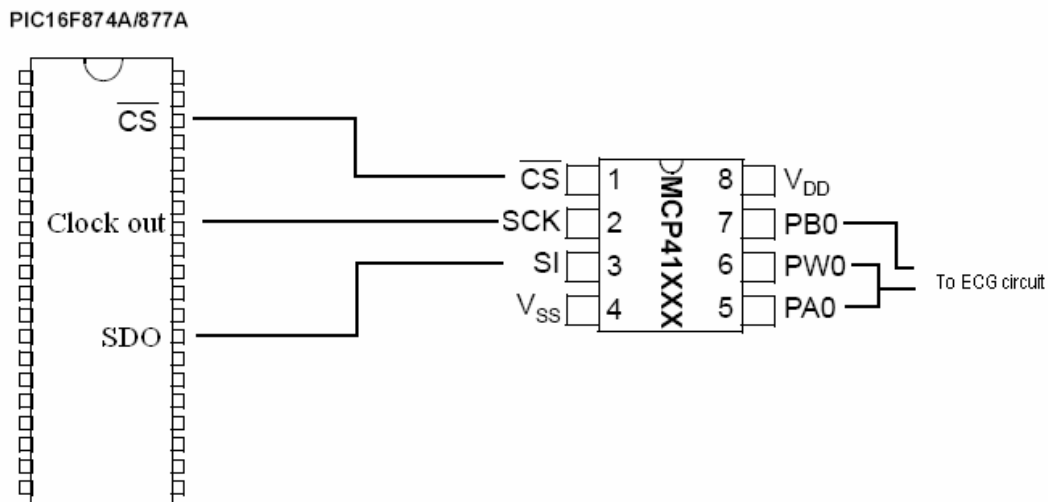


Figure 35. SPI pin connections for the digipot.

The designated functions for the pins on the digipot are tabulated in Table 3.

MCP41XXX Pins		
Pin #	Name	Function
1	\overline{CS}	Chip Select
2	SCK	Serial Clock
3	SI	Serial Data Input
4	V_{SS}	Ground
5	PA0	Terminal A Connection For Pot 0
6	PW0	Wiper Connection For Pot 0
7	PB0	Terminal B Connection For Pot 0
8	V_{DD}	Power

Table 3. MCP41100 Digital Potentiometer Pin Functions

To communicate with the digipot, the PIC must first select it by setting the chip select (CS) pin low. Then, a command byte is sent out of the Serial Data Out (SDO) pin to

instruct the digipot as to what it should do with the following data byte. For the MCP41100, the command byte can either tell the digipot to write the data byte to the potentiometer, completely shutdown, or do nothing. The data byte contains the information that sets the value of the potentiometer. Once the data byte is clocked in, the chip select must be set high again. The digipot will discard the information sent to it if the chip select pin is not set low and then high again in multiples of 16 clocks.

To adapt the gain of the electrocardiograph, the digipot is first set to a designated starting level. According to test values, the middle resistance of the digipot served as a good starting point to produce an acceptable gain. Then, the PIC monitors the output of the circuit and sets the digipot accordingly. To determine the maximum output voltage of the circuit, the PIC performs a continuous comparison of every single ECG data point collected over a period of approximately 1.5 seconds. Each data point within that period is compared to the maximum value that the PIC has seen up to that point of time within the period. Then, if the output is lower than the designated threshold, the digipot is lowered by a small amount thereby increasing the overall gain slightly. If the output is too high than another threshold, the opposite occurs. If the output remains within the designated voltage zone, nothing happens.

To clarify the function of the digipot and the algorithm for its control, the following pseudo-code is included.

Set the digipot level to an average value according to previous tests

Compare first two data points and save the greater value

For 1.5 seconds,

Compare Next data point with Current greatest value

If Next is greater than Current, replace Current with Next

Else maintain same value in Current

If Current is too low,

Set CS low

Send command byte to WRITE

Send data byte to increment the digipot by a small amount

Set CS high

If Current is too high,

Set CS low

Send command byte to WRITE

Send data byte to decrement the digipot by a small amount

Set CS high

The successful implementation of the digipot for adaptive gain of the ECG waveform is demonstrated in Figure 36 below. Immediately after the remote unit is turned on, gain for the ECG signals is small. However, the digipot responds to the amplitude of the R peak of the waveform and adjusts the gain to ensure that the peak lies within a specific range of lower and upper thresholds.

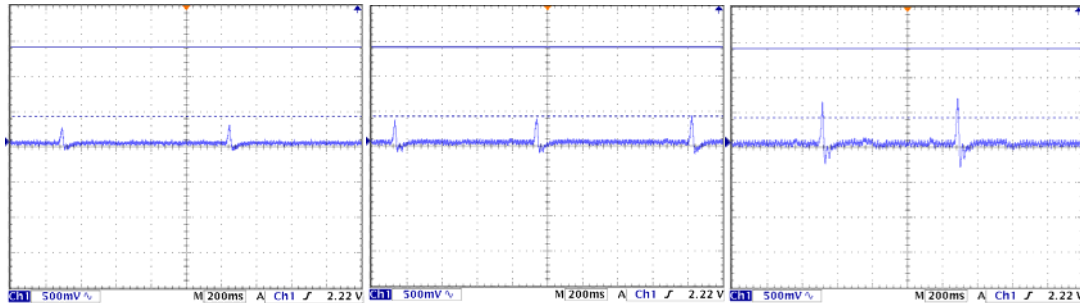


Figure 36. Three snapshots showing the adaptive gain of the ECG signal.

9.2.2 Patient alert system

This feature is incorporated in the Wipmod design to alert the patient if any of the physiological parameters that are being monitored are out of normal range. The alarm informs the patient to either contact the doctor or restrain activity that might have induced cardiac activity which poses a danger to the patient.

The alarm settings are based on the threshold values that are set by a doctor using the graphical user interface (Section 11.2.3). When the physiological parameters are beyond the threshold, the interface communicates with the remote unit wirelessly to initiate a buzzer.

The PIC microcontroller in the remote unit controls the patient alarm system. When an alert notice is received from the base unit, the microcontroller turns on the buzzer. The buzzer-snooze pushbutton available to the user allows the patient to turn off the buzzer once he or she is informed of the situation. However, after 30 seconds of “snooze” time, if the monitored parameter is still beyond the specified threshold, the alarm is re-initiated. The flowchart in Figure 37 summarizes how the alarm system works.

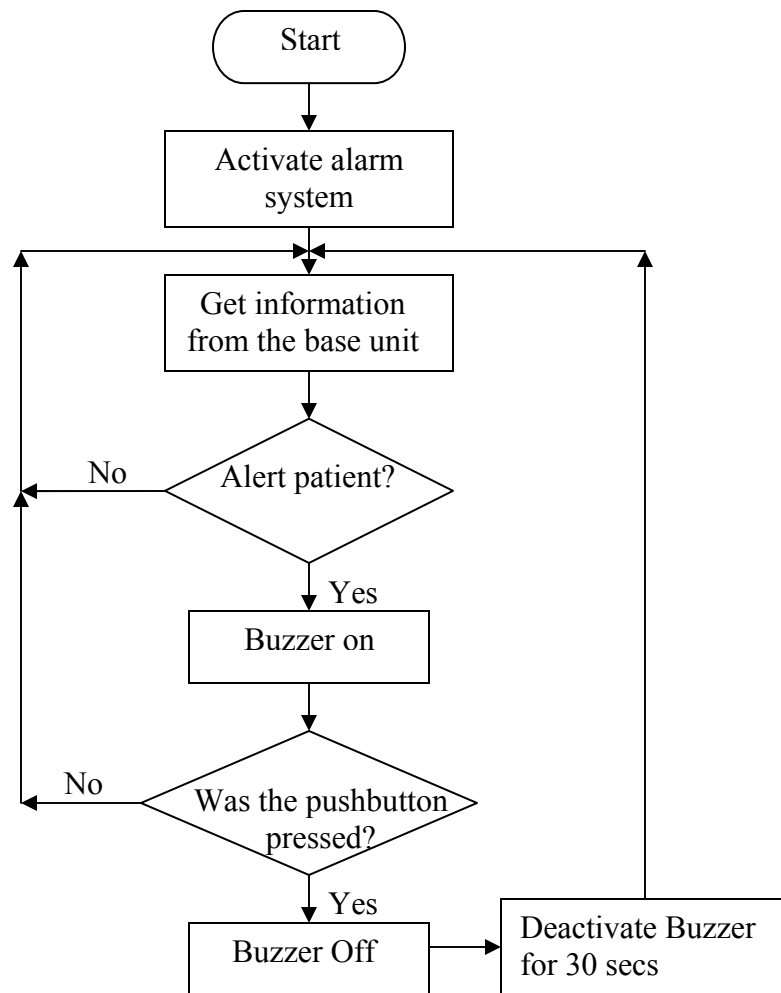


Figure 37. Flowchart describing the patient alert system.

The PIC uses timers and interrupts to implement the buzzer-snooze feature. An I/O pin, which a pushbutton is connected to, is always high until when the button is pressed. When the pin is high, the alarm system is activated. However, when the pushbutton is pressed, I/O pin is goes to 0 which initiates a counter based on the RTCC timer that deactivates the alarm system for 30 seconds. The circuit set up for the alarm system is shown in Figure 38.

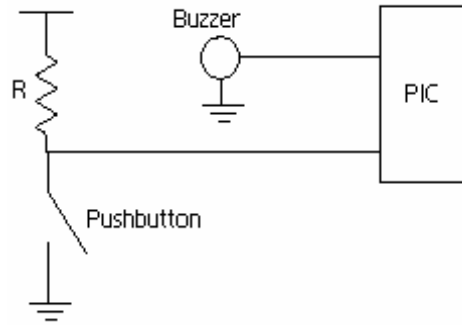


Figure 38. The circuit set up for the alert system with the buzzer-snooze pushbutton.

Table 4 below outlines the pin connections of the three physiological signals and circuit controls to the PIC microcontroller.

	Pins
ECG data	AN0 (Pin 2)
Temperature data	AN1 (Pin 3)
Pulse Oximeter data	RA2 (Pin 4)
Pulse Oximeter LEDs	RB3 (Pin 36)
	RB2 (Pin 35)
Buzzer	RB1 (Pin 34)
Buzzer pushbutton	RB5 (Pin 38)
Digipot Chip Select (CS)	RA5 (Pin 10)
Digipot Clock	RC3 (Pin 18)
Digipot SI pin	RC5 (Pin 24)

Table 4. PIC microcontroller pin connections.

10/Signal processing

10.1/Data parsing

The stream of data received by the XBee module connected to the base station contains both the ECG and the temperature data. The interface collects 3000 data points, approximately 5 seconds of data, each time it reads the serial port buffer. The parsing algorithm used on these 3000 data points is based on controlled amplification of the ECG signal by the digital potentiometer and the event flag used for the temperature data in the PIC microcontroller. The digital potentiometer in the ECG processing circuit ensures that the gain of the ECG is such that analog-to-digital converter (ADC) value is always less than 255. Therefore, if an array of 3000 data points contains a '255', the event flag, the algorithm recognizes the value that immediately follows the flag as temperature data.

10.2/Estimation techniques for characteristics of an ECG waveform

Once the wireless data has been parsed and a complete ECG data set is available, it can be processed in MATLAB to extract relevant information of medical significance. As mentioned in Section 6.2, the heart rate and the duration of the QRS complex can be analyzed to help in the diagnosis of medical conditions.

10.2.1/Heart rate

To calculate the heart rate, the R wave was used to identify a single heart beat. The QRS complex peaks just once per heart beat and so it can be used to calculate the time between heart beats. Two characteristics of the R wave were used to identify it: a voltage threshold and the slope of the surrounding points.

Since the ECG was being processed with a circuit with adaptive gain, the peaks of the QRS complex are always within the same range. Thus, a voltage threshold can be used to search for the exact R peak. Figure 39 demonstrates this first filtering step. Any points along the ECG waveform that are not above the voltage threshold are not considered as possible R peaks.

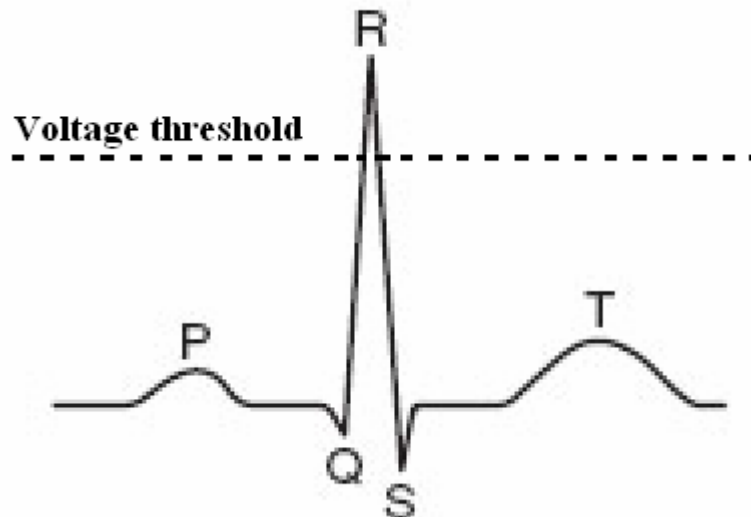


Figure 39. Voltage threshold filtering for R peak identification

The next step in finding the R peaks is to examine each point that is above the threshold and calculate the slope of the best line fit of the four points before and after the point being considered. This is done with MATLAB's *polyfit* command. The slope of the fit is then compared to two thresholds that were determined from examples of ECG data. The point that represents the R peak must have a “before” fit with a positive slope between 1 and 8 and an “after” fit with a negative slope between -1 and -8. The array index of the R

peak is then saved for later reference. To ensure that only one point per heart beat is specified, once a point is designated as the R peak, the code will ignore the next 1/10 of a second of data. Figure 40 below outlines the entire R peak search process.

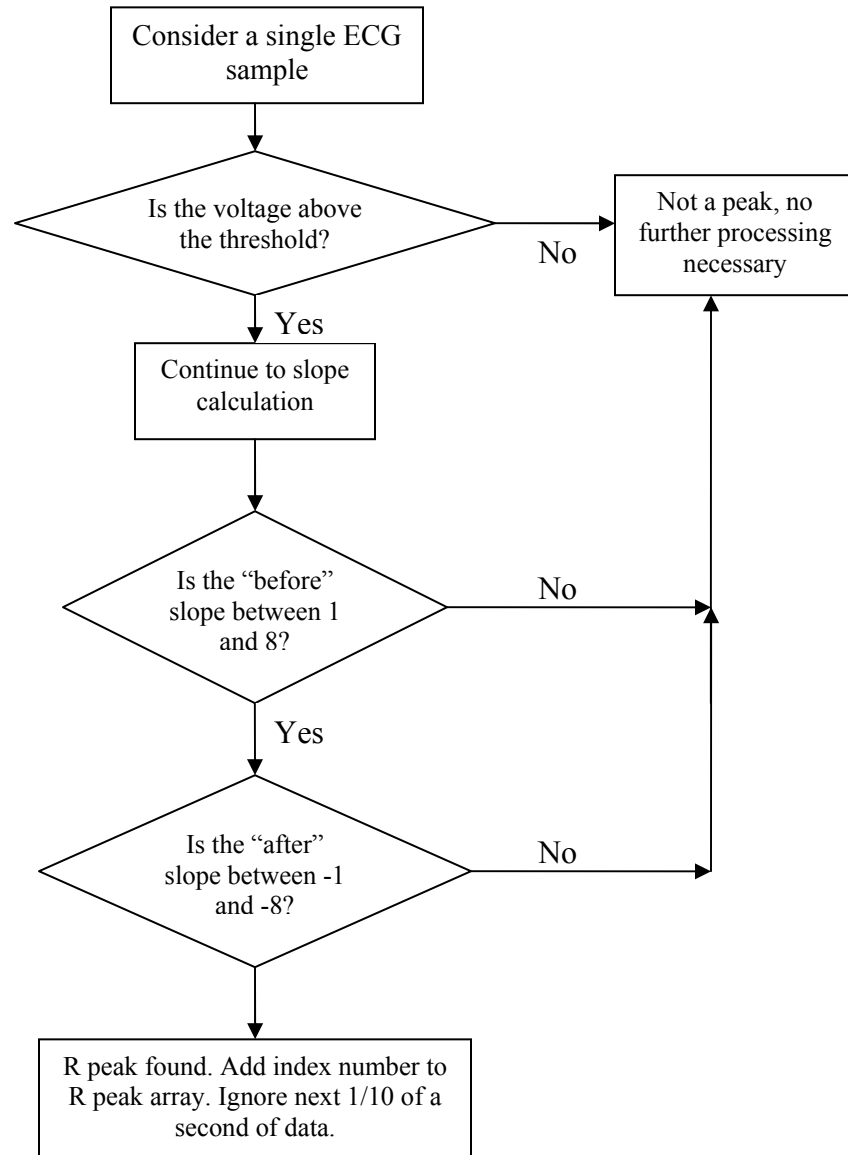


Figure 40. R peak search flow chart.

10.2.2 QRS complex width

To calculate the QRS complex width, the Q and S points must be known. As can be seen in Figure 42, these points lie on either side of the R peak and are junctions at which the slope of the ECG waveform sharply changes, with respect to the R peak, from negative to

positive. To reduce the effect of noise, the entire ECG data set is passed through a digital median filter with a window of 5 points. This process eliminates outliers that may disrupt the search for the Q and S points. It could not be performed earlier because the very nature of R peaks is exactly what the filter eliminates.

Once the R peaks have been identified, each complex is searched to find the Q and S points. A technique similar to that used to find the R peaks is employed for each. Starting from the R peak, each point of the complex on either side of the peak is examined and the slope of the best line fit of the four points on the side opposite of the R peak is calculated. If the slope reaches a certain threshold, determined from examples of ECG data, that sample is designated the Q or S point, depending on which side of the complex is being processed. For the Q point, the best line fit slope must be less than 1; for the S point, it must be greater than -0.5

To improve the search speed, a modification was attempted that used previous Q and S point locations to approximate the search start location for the next Q and S point locations. This modification averaged the R-Q and R-S distances of the previous data set and divided that distance in half. Thus, the search would start at the approximate midpoint of each slope of the QRS complex instead of at the R peak. However, after comparing processing time both with and without the modification, it was found that this addition did not show any speed improvement. Instead it seemed that variation in the processing usage of the PC had more of an effect. Therefore, to maintain as simple of a code as possible, the modification was not included and the Q and S search always start at the R peak.

This algorithm required extensive testing and revision. Since values for the threshold slopes were based on empirical data, they had to constantly be tested and modified. Also, the algorithm evolved greatly to accommodate all foreseen situations. Due to the fact that four data samples are used with the *polyfit* command, certain restrictions must be considered. For the Q point search, the first four samples before the R peak are ignored. In the S point search, the last four samples of the data set are ignored. This situation is acceptable unless the QRS begins (Q point) or ends (S point) within those eight ignored samples. These cases are treated just as if the points were truncated, as explained next.

For a given set of data, it is possible that the ECG was captured such that either the Q or S point of a given QRS complex was truncated, depending on the timing of the data set collection. In these cases, the Q and S points are set as the first or last data samples of the data set. Although this results in a shorter QRS complex width, the effects of this inaccurate measurement are mitigated by the averaging of several QRS calculations. This compromise is necessary as the first iteration of the algorithm would often freeze in the attempt to find a Q or S point that did not exist in the given data set. The final resulting algorithm and its implementation is shown in Figures 41 and 42 respectively.

Using the array indices of the Q and S points and the known sampling rate of the data, the time that passes between the two can be calculated. The average over approximately 25 seconds is then displayed in the GUI.

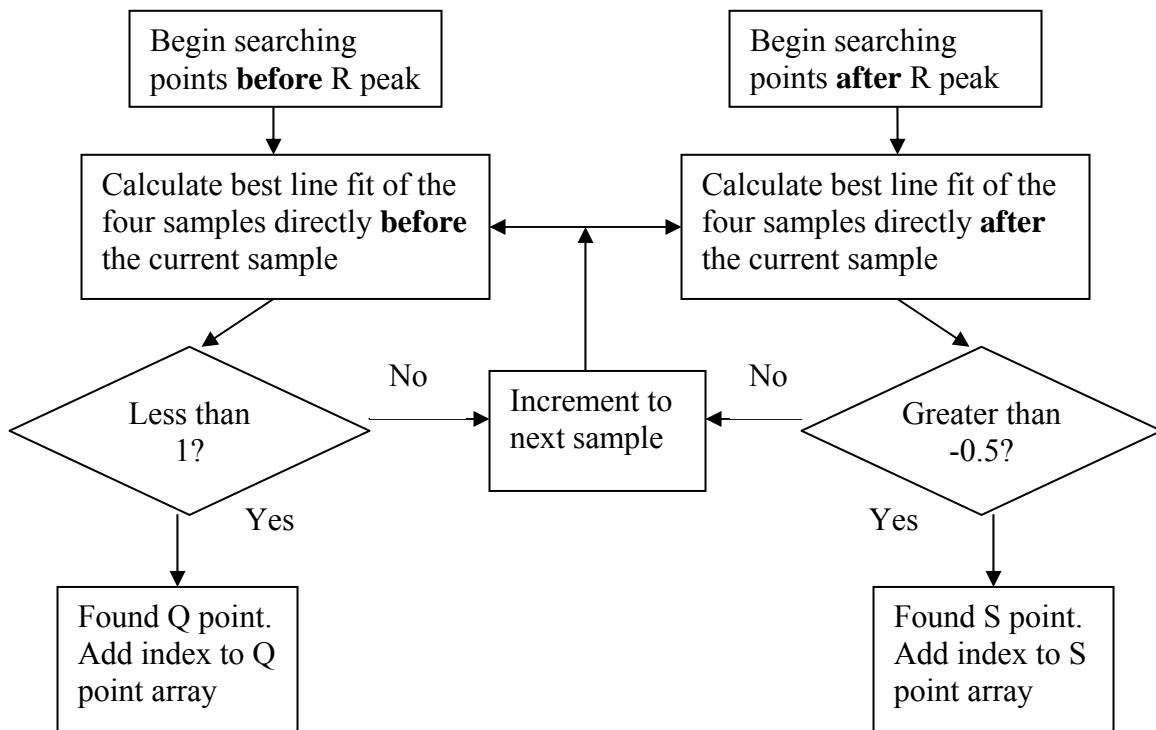


Figure 41. Q and S point search flow chart.

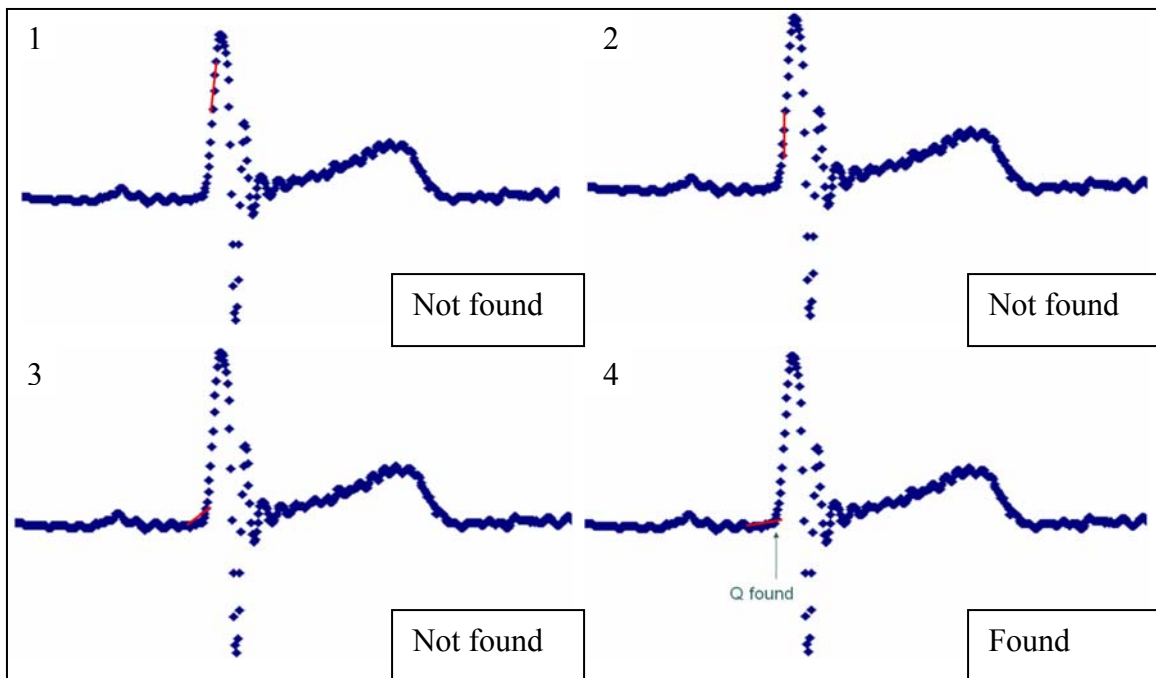


Figure 42. Example Q point search.

11/Layout and features of the graphical user interface

MATLAB offers easy-to-use resources to create graphical user interface both programmatically and by using GUIDE, a graphical user interface development environment in MATLAB. While the “drag and drop” feature of GUIDE allows one to easily create a GUI, code written to create GUI components allows a better control of the GUI. Both of these options were used, as required, to get the design features necessary for the project.

The graphical user interface allows the user to display, process and save the data received by the XBee module connected to the base station. Since Wipmod is designed to be used in a home setting, the GUI was designed to provide a user-friendly environment for any individual. The interface also has an inbuilt function that allows sharing of patient data from the base station to a doctor’s office through electronic mail.

11.1/Brief introduction to MATLAB GUI structure

In a MATLAB GUI, all the components placed in the GUI figure are automatically incorporated in the *handles* structure and are identified by the *Tag* name of each component. For instance, if a GUI figure consists of a pushbutton component with a Tag name “*Cancel_pushbutton*”, *handles.Cancel_pushbutton* is automatically generated in the *handles* structure of the figure. Each component has an associated callback function which includes the action that is executed when the component is activated in the GUI. In addition to components, handles can also store variable or arrays, also referred to as “fields” of the handles structure, which are “global” and available to all the callbacks in the GUI. Each time a field of the handles structure is changed or added, *guidata(hObject, handles)* has to be performed to update the handles structure. Further details about structures in MATLAB GUI can be found in the help files. The code, with detailed comments, is included in Appendix A.

11.2/GUI features

The GUI is divided into three main categories.

1. Main panel
2. Alarm Settings panel
3. Communication Settings panel

11.2.1/Main Panel

All the physiological parameters recorded from the sensors are displayed in this panel. These include the body temperature, the heart rate, the width of the QRS complex, the PR width and the RT width of the ECG wave of the patient. The algorithms implemented to extract information from the ECG waveform are discussed in Section 10.2. In addition to

the physiological parameters, the panel consists of a panel for ECG wave display. The ECG waveform display is almost real time as it is updated every five seconds, while the values of heart rate and widths of various parts of the ECG are updated every twenty five seconds. The communication control between the GUI and the remote unit are also incorporated in this panel.

A status bar, which gives detailed information about the path location of collected data and the status of the data collection process, is displayed at the bottom of the window to make the GUI user-friendly.

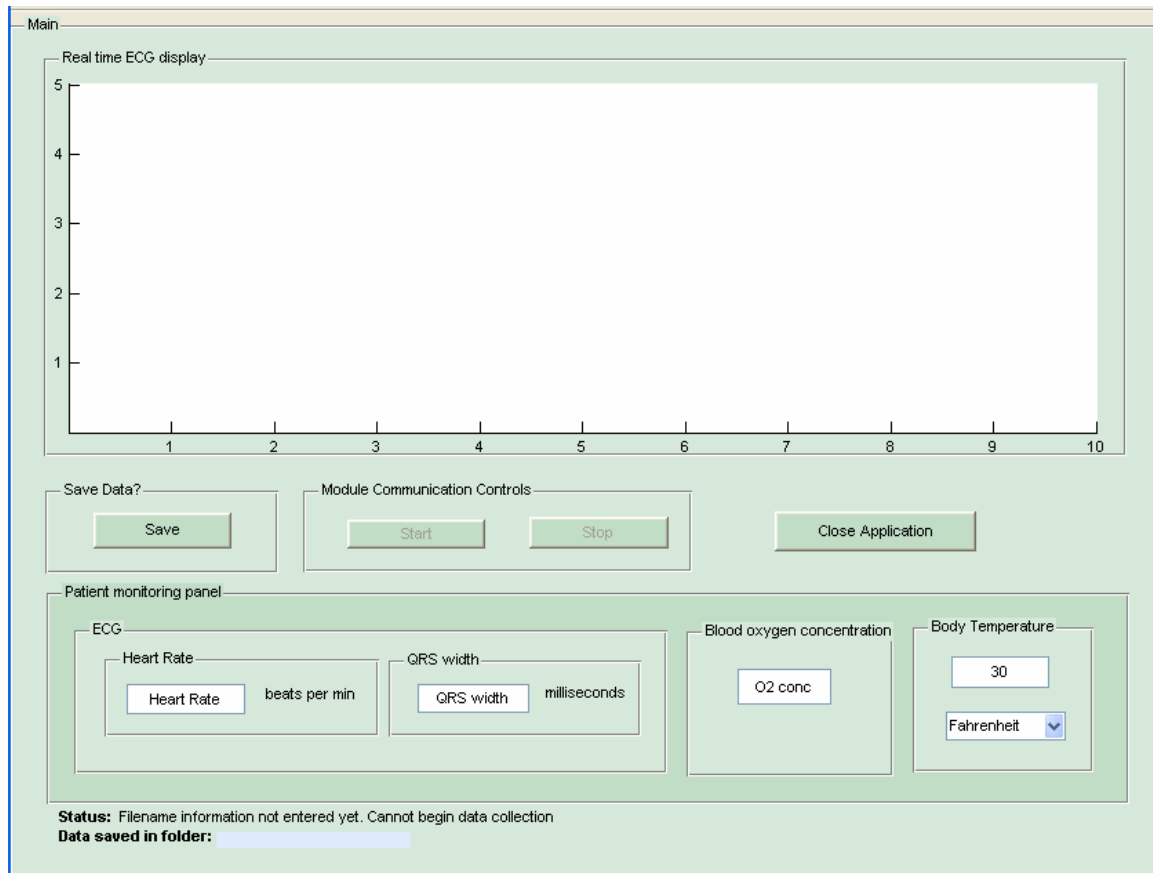


Figure 43. Main panel of the GUI.

11.2.2/Communication Settings panel

This panel contains all the fields necessary for communication with the XBee module and for sending the patient data to the doctor. On initialization of the GUI, it automatically detects the COM ports available in the computer and allows the user to select the port to which the XBee module is connected to. The server settings and email addresses required to send the collected physiological data are also specified in this panel. Under the email settings, the user needs to enter a password for the sender's email address. To ensure security, the password field is masked with asterisks. Finally, the panel also offers an "auto-email send" feature, which automatically sends the patient data by email upon end

of data collection. If the feature is disabled, GUI prompts for user permission before emailing the data.

Communication Settings

COM Port Settings

Select the port to which WipMod is connected:

COM3

Email Server settings

SMTP server smtp.gmail.com

Email Settings

Sender

Email address: wipmod@gmail.com

Password: Enter your password here

Receipient

Email address: anima.singh@gmail.com

Save email address

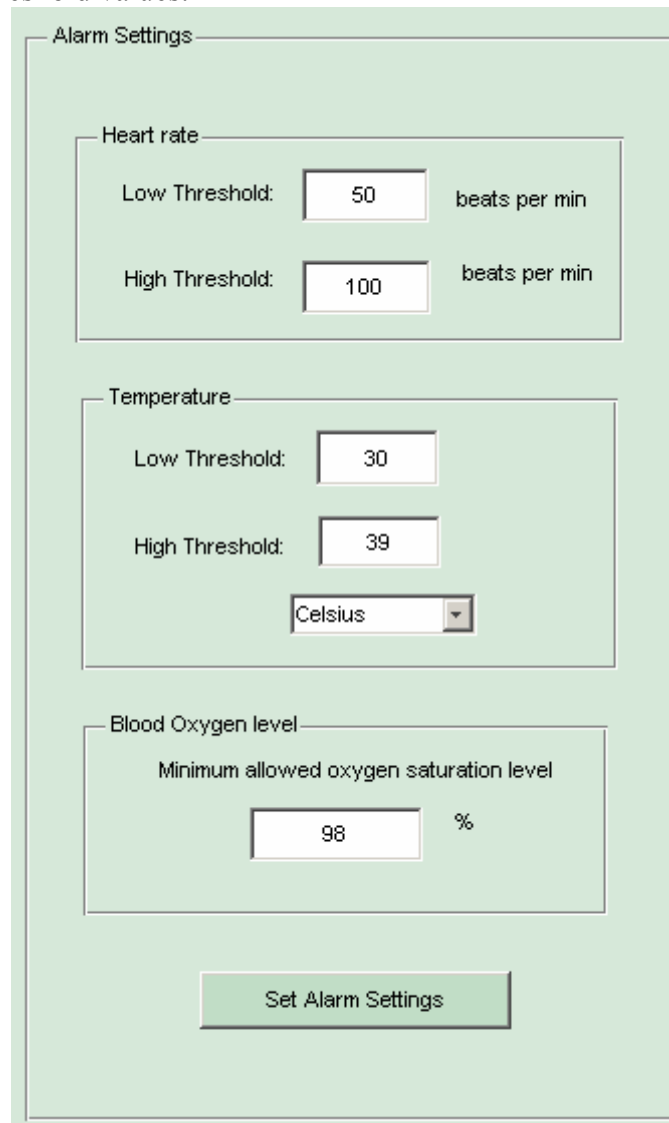
☐ Automatically send email at the end of data collection.

Figure 44. Communication Settings panel.

11.2.3/Alarm Settings panel

The normal heart rate of an individual varies depending on his or her lifestyle. For example, an athlete will have a lower resting heart rate while a person with a sedentary lifestyle will have a higher resting heart rate. This implies that for any particular person, the heart rate threshold that indicates abnormal behavior might be different from that a different individual. The alarm settings panel allows the doctor to customize the threshold parameters for the patient alarm system according to the patient's condition.

Once these threshold values are set by a doctor, the user will not have to re-enter the settings each time the application is opened. The threshold values from the previous session will be preserved and re-used. However, if need be, the panel does allow the user to re-adjust the threshold values.



The image shows a software interface titled "Alarm Settings" with a light green background. It contains three main sections for setting thresholds: "Heart rate", "Temperature", and "Blood Oxygen level". Each section has input fields for low and high thresholds and a unit selector. At the bottom is a "Set Alarm Settings" button.

Parameter	Low Threshold	High Threshold	Unit
Heart rate	50	100	beats per min
Temperature	30	39	Celsius
Blood Oxygen level	Minimum allowed oxygen saturation level		%

98

Set Alarm Settings

Figure 45. Alarm Settings panel.

12/ Acknowledgements

We would like to thank Professor Erik Cheever for being a wonderful advisor and for his help and guidance throughout the project. Also, we give many thanks to Doug Judy for his help in the shop as we assembled our prototype and to Ed Jaoudi for his patience and help in finding all the necessary components for our project. We would like to thank Nellcor and Novamed for their donations of pulse oximeters and temperature sensors respectively for the project. Finally, to all of our fellow engineers, thank you for the support, encouragement, and camaraderie throughout the past four years, from E6 to E90.

13/References

- Cheever, Erik. "Single Supply Op Amps." Retrieved from <http://www.swarthmore.edu/NatSci/echeeve1/Ref/SingleSupply/SingleSupply.html>.
- Guyton, Arthur and Hall, John. Textbook of Medical Physiology. Tenth Edition. W.B. Saunders Company, Philadelphia, PA: 2000.
- "Oximax Sensor Overview" from <http://www.nellcor.com/prod/product.aspx?S1=OXI&S2=SNR&ID=255>
- Pap et. al., "Native QRS complex duration predicts paced QRS width in patients with normal left ventricular function and right ventricular pacing for atrioventricular block." *Journal of Electrocardiology*, Vol 40, Issue 4. (360-364)
- Spinelli et. al. "AC- Coupled Front-End for Biopotential Measurements." *IEEE Transactions of Biomedical Engineering*, Vol. 50, No. 3, March 2003
- Sedra and Smith. "Microelectronics Circuits." Fifth Edition. Oxford University Press Inc.
- Townsend N. "Medical Electronics" 2001. Retrieved from http://www.robots.ox.ac.uk/~neil/teaching/lectures/med_elec/l.
- Webster, J.G. and Huhta, J.C. "60 Hz Interference in Electrocardiography." *IEEE transactions on Biomedical Engineering*, Vol. BME-20, No. 2, March 1973
- Webster, J. G. "Medical Instrumentation: Application and Design." Third Edition. Wiley, NJ: 1998.
- Webster, J. G. "Design of Pulse Oximeters." Institute of Physics Publishing, Dirac House, Temple Back, Bristol BS1 6BE, UK.
- Youngerman-Cole, Sydney. (January 18, 2006). Ambulatory Electrocardiogram. *WebMD Medical Reference*. Retrieved March 22, 2008, from <http://www.webmd.com/heart-disease/ambulatory-electrocardiogram>.

Appendix A: Wipmod version 1

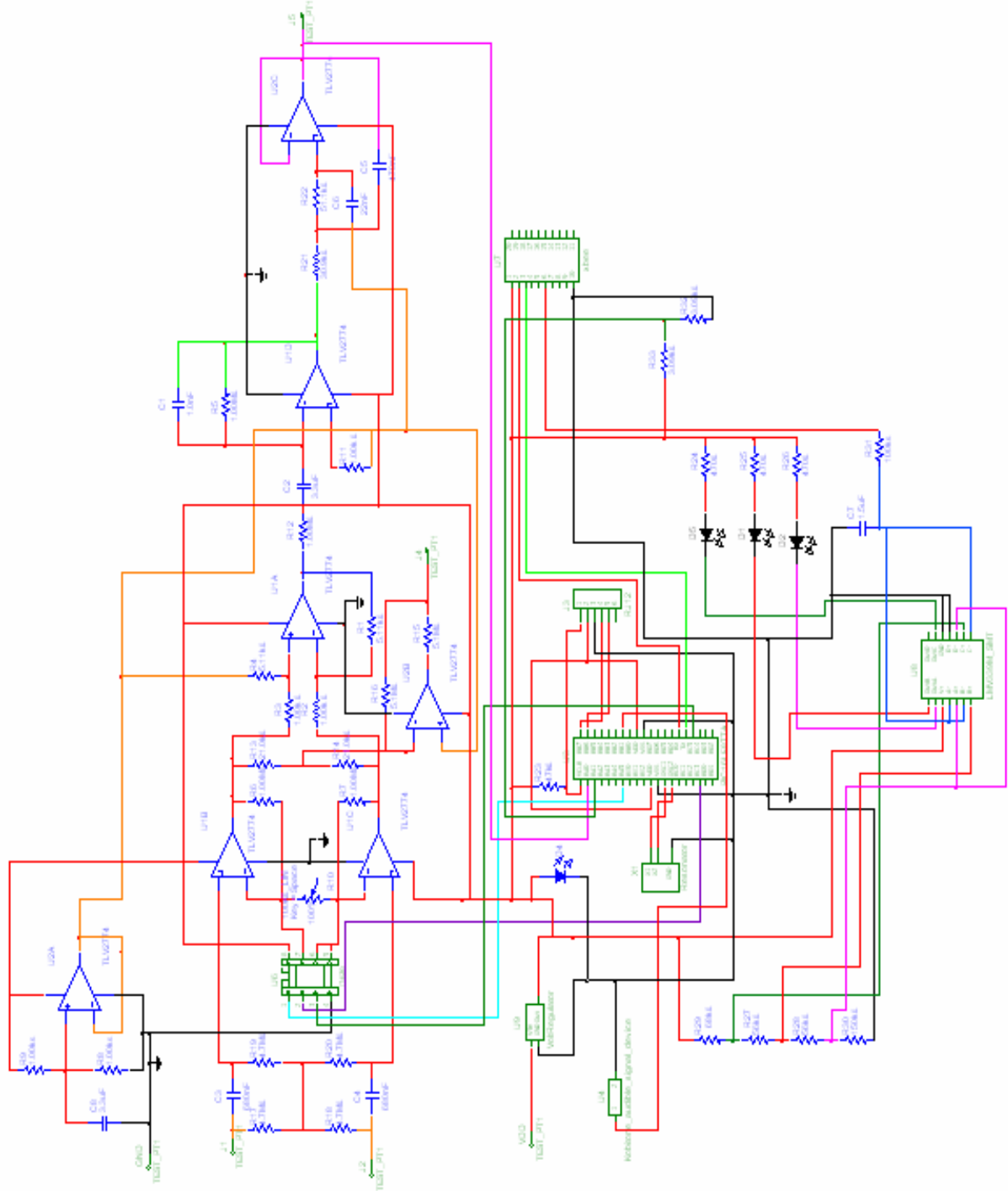


Figure 46. Complete Multisim schematic of Wipmod v. 1.0

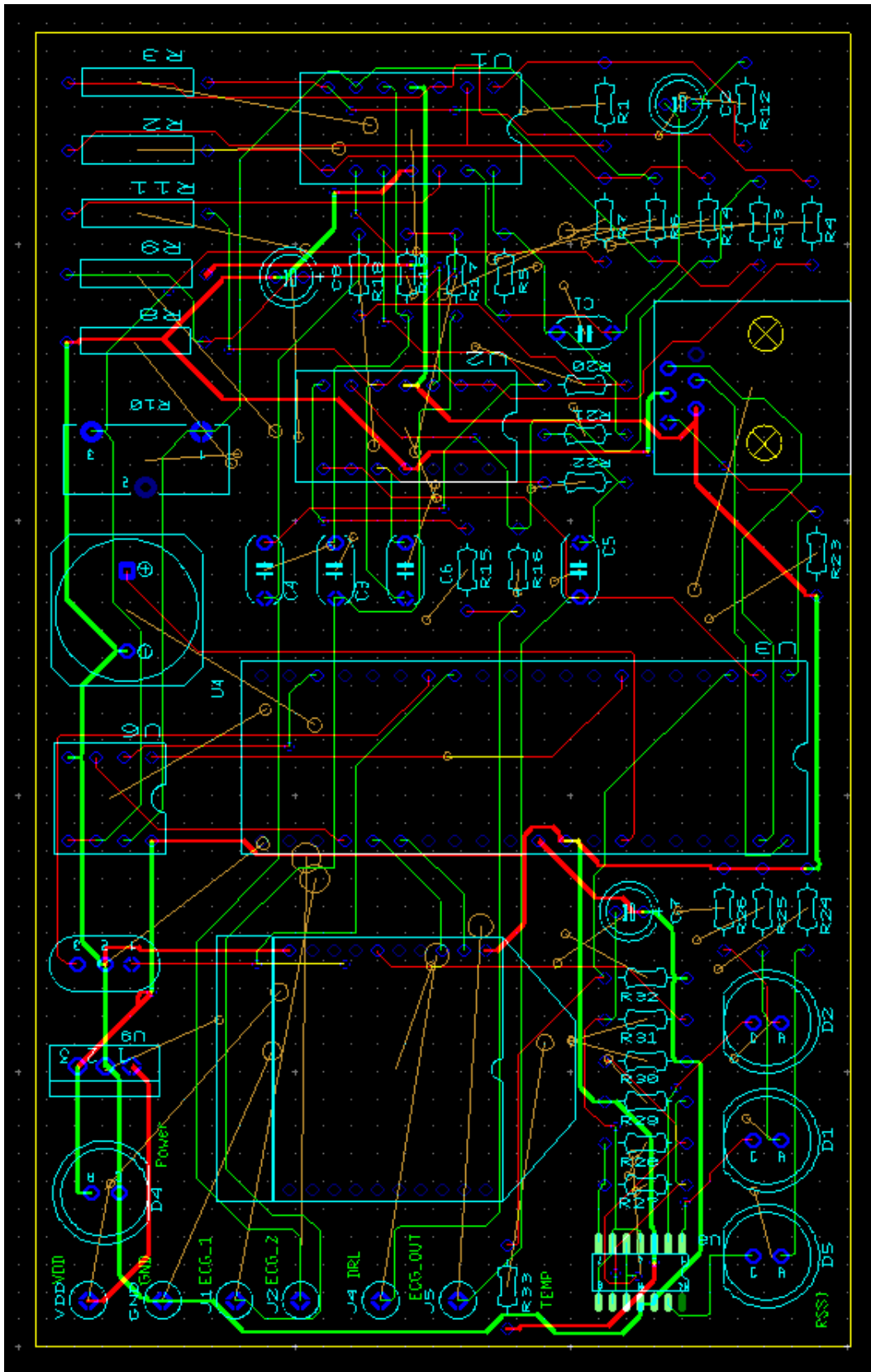


Figure 47. Ultiboard layout of Wipmod v. 1.0.

Appendix B: Wipmod version 2

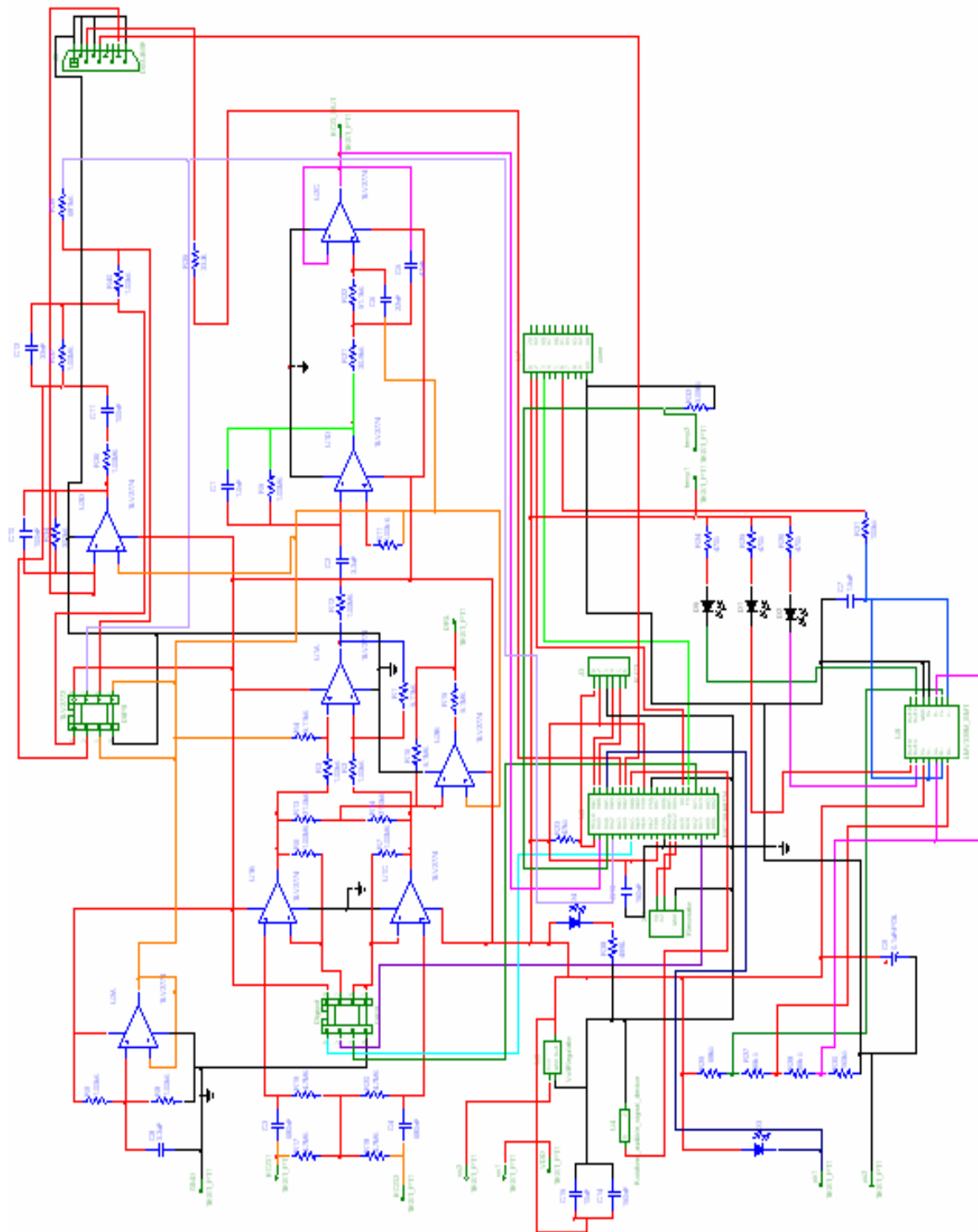
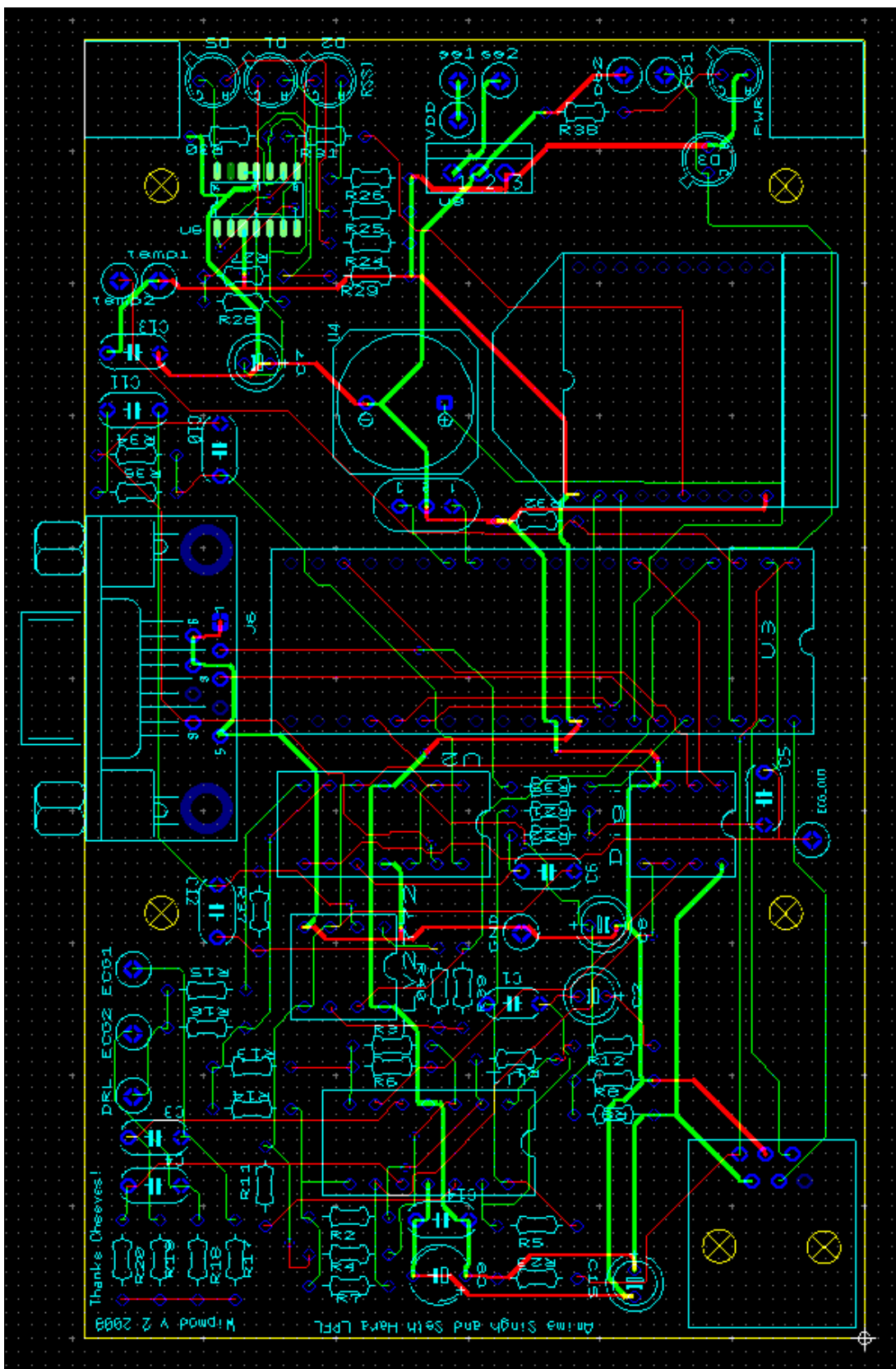


Figure 48. Complete Multisim schematic of Wipmod v. 2.0



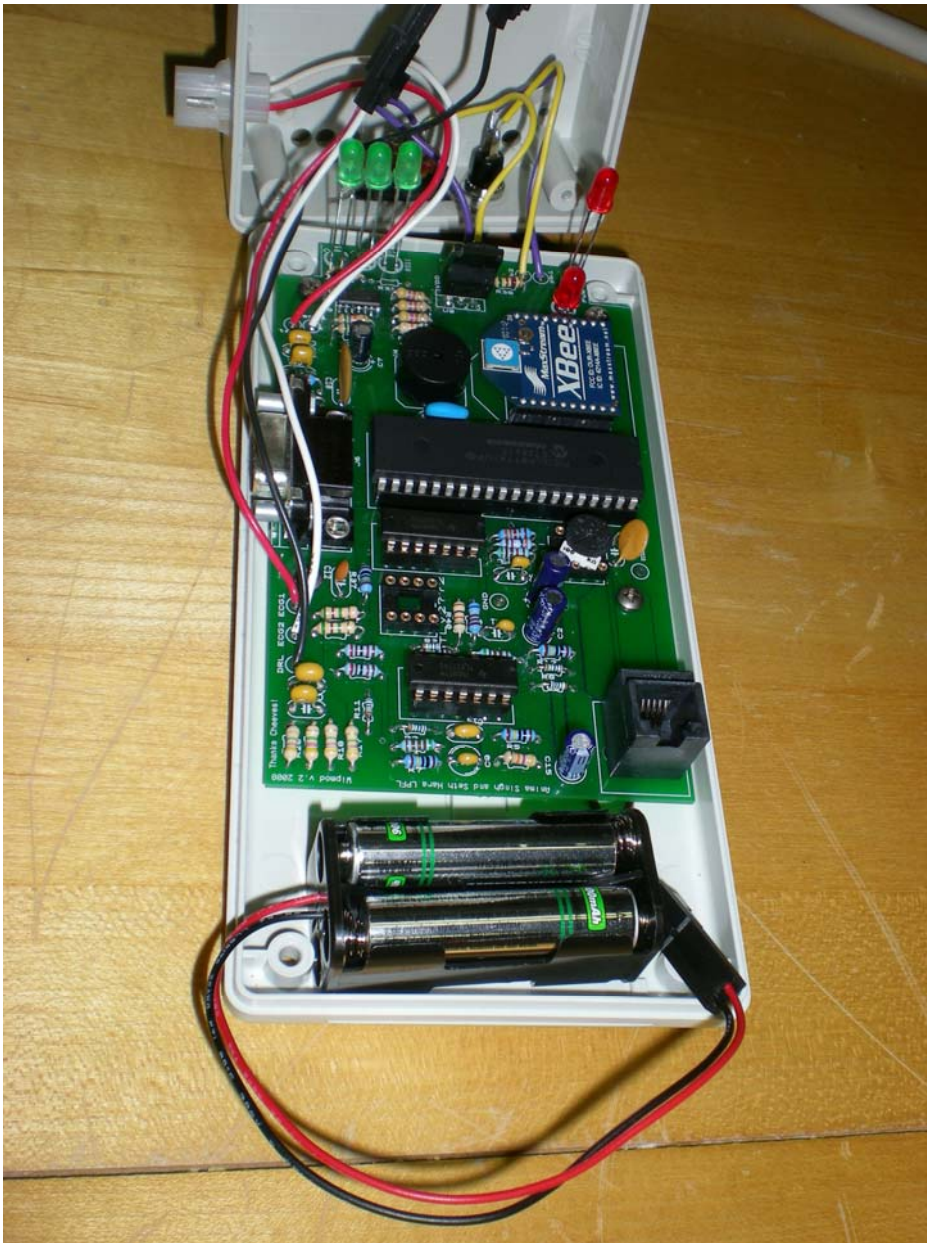


Figure 50. Interior view of the complete remote unit.



Figure 51. Exterior view of the complete remote unit with all the sensors connected.

Appendix C: PIC code

Main.c

```
#include "F:\E90\PICboard\twoChannelsVer2\main.h"
//Initialize variables
int ecg_data,temp_data;
int ecg_ready =0;
int temp_ready=0;
int flag; // flag for temperature data
#define temp_START 9000;//10= sampling at (1/10)* ECG_sampling rate.
Set it to about 10000 to sample every 15 secs.
#define buzz_START 18000;//every 30 secs

// Variables for Digipot control
#define high_start 3
#define cs_pin PIN_A5
#define write 0x11

int high_count;//for writing to digipot
int digi_marker; //Digi_marker is 1 every 1.5 secs (writes to digipot)
long int temp_count; //for temp sampling
int pulse_count;//for pulse ox sampling
```



```

//Variables Buzzer control
long int buzz_count;
long int buzz_snooze;
int buzz_flag;
int buzz_active;

//Timer 1 is for digipot control (16 bit timer)
#INT_TIMER1
void clock_isr(){
    if(--high_count==0){
        //output_high(PIN_B4);
        digi_marker=1;
        high_count=high_start;
        //delay_ms(10);
        //output_low(PIN_B4);
    }
}

// RTCC (Timer 0) is for Temp, Pulse Ox, and ECG data sampling
#INT_RTCC

clock_isr(){
    output_high(PIN_B3); //just a sanity check
    set_adc_channel(0);
    delay_us(10);
    read_adc(ADC_START_ONLY);
    while(!adc_done());

    //ECG sampling
    ecg_data=read_adc(ADC_READ_ONLY);
    ecg_ready=1;

    //For pulse ox sampling
    pulse_count=pulse_count+1;

    //!    if (pulse_count==2){ //Sampling rate of pulse ox is half the
sampling rate of ECG
    //!        pulse_ox_ready=1;
    //!        pulse_count=0;
    //!    }

    //Temp sampling
    temp_count--;

    if(temp_count==0){
        temp_ready=1;
        set_adc_channel(1); //Channel 1 is for temperature
        delay_us(10);
        read_adc(ADC_START_ONLY);
        while(!adc_done());
        temp_data=read_adc(ADC_READ_ONLY);
        temp_count= temp_START;
    }

    read_adc(ADC_START_ONLY);

```

```

    set_rtcc(250); //ECG sampling at around 600 Hz.

//Controlling the "snooze" feature of the patient alarm
    if(--buzz_snooze ==0){
        buzz_active=TRUE;
    }
//Controlling patient alarm on/off
    if((buzz_flag ==TRUE) && (--buzz_count!=0)){
        output_high(PIN_B1);
    }
    else{
        output_low(PIN_B1);
        buzz_flag=FALSE;
        buzz_count=700;
    }

    output_low(PIN_B3);
}

// Checks RX pin for incoming data and grabs it if available.
Otherwise, sets retval to 'N'
char timed_getc() {
    //unsigned int16 timeout;
    char retval;
    if(kbhit())
        retval = getc();
    else
        retval = 'N'; // 'N' for no information
    return(retval);
}

main()
{
//Initializing variables
    char PICstart, c;
    int value = 0, digipot_value = 0x80;

//Set up ADC
    setup_adc(ADC_CLOCK_DIV_2);
    setup_adc_ports(AN0_AN1_AN3);

//Set up SPI
    setup_spi(SPI_MASTER|SPI_L_TO_H|SPI_XMIT_L_TO_H|SPI_CLK_DIV_64);

//Set up the timers
    setup_timer_1(T1_internal|T1_DIV_BY_8);
    setup_counters(RTCC_INTERNAL,RTCC_DIV_256);

//Idle PIC until receive "Start" command from Matlab
    while(true){
        PICstart=timed_getc();
        if(PICstart=='Y'){
            output_high(PIN_B0);
            output_low(PIN_D7);

```

```

//Initiating the timers
    set_timer1(0);
    set_rtcc(250);

    read_adc(ADC_START_ONLY);
    enable_interrupts(INT_RTCC);
    enable_interrupts(INT_TIMER1);

    enable_interrupts(GLOBAL);

    temp_count=temp_START; // to control temperatute sampling rate
    high_count=high_start; // Digipot control sampling.
    buzz_active= TRUE; // Buzzer control sampling

    buzz_count =0;
    buzz_flag = FALSE;

//Initializing digipot to start off (Initial gain)
    output_low(cs_pin);
    spi_write(write);
    spi_write(digipot_value);
    output_high(cs_pin);

    while (TRUE){
        c=timed_getc();

//PIN_B5 is for the buzzer snooze
        if (!input(PIN_B5)) {
            buzz_active= FALSE;
            buzz_snooze= buzz_START;
        }

        if( (c=='B') && (buzz_active==TRUE)){
            //This will turn buzzer on. PIN_B1 reserved for the
buzzer.
            buzz_flag = TRUE;

        }

//Sending data over Xbee to PC
        if ((temp_ready==0) && (ecg_ready==1) ){
            ecg_ready=0;
            if(ecg_data>value){
                value = ecg_data; //Tracking the maximum value of the
ECG data for digipot settings
            }
            printf("%c",ecg_data);

        }

        if((temp_ready==1)){
            flag=255;
            printf("%c%c",flag,temp_data);
            temp_ready=0;
        }

```

```

//Adjust the digipot value according to the QRS peak value found in
line 181
    if(digi_marker){

        if(value<=200){ //If the ECG signal is lower than we
want,

            if(digipot_value > 8){
                output_high(PIN_B4);
                digipot_value = digipot_value-(4); //Decrease
the digipot value (increase gain)
            }
        }
        else if(value>=220){ //If the ECG signal is higher
than we want,

            if (digipot_value < 248){
                digipot_value = digipot_value+(4); //Increase
the digipot value (decrease gain)
            }
            else{
                digipot_value=255;
            }
        }
    }
    //Write the new value to the digipot
    output_low(cs_pin);
    spi_write(write);
    spi_write(digipot_value);
    output_high(cs_pin);
    value = 0;
    digi_marker=0;
    output_low(PIN_B4);
}
}
else{
    output_high(PIN_D7);
    output_low(PIN_B0);
}
}
}
}

```

Main.h

```

#include <16F877A.h>
#define device adc=8
#define FUSES NOWDT //No Watch Dog Timer
#define FUSES XT //Resistor/Capacitor Osc with CLKOUT
#define FUSES NOPUT //No Power Up Timer
#define FUSES NOPROTECT //Code not protected from reading
#define FUSES NODEBUG //No Debug mode for ICD
#define FUSES NOBROWNOUT //No brownout reset
#define FUSES NOLVP //No low voltage prgming, B3(PIC16) or
B5(PIC18) used for I/O
#define FUSES NOCPD //No EE protection
#define FUSES NOWRT //Program memory not write protected
#define use_delay(clock=4000000)
#define use_rs232(baud=9600,parity=N,xmit=PIN_C6,rcv=PIN_C7,bits=8)

```

Appendix D: Matlab code for GUI design

```
function varargout = GUI(varargin)
%GUI M-file for GUI.fig
%     GUI, by itself, creates a new GUI or raises the existing
%     singleton*.
%
%     H = GUI returns the handle to a new GUI or the handle to
%     the existing singleton*.
%
%     GUI('Property','Value',...) creates a new GUI using the
%     given property value pairs. Unrecognized properties are passed
via
%     varargin to GUI_OpeningFcn. This calling syntax produces a
%     warning when there is an existing singleton*.
%
%     GUI('CALLBACK') and GUI('CALLBACK',hObject,...) call the
%     local function named CALLBACK in GUI.M with the given input
%     arguments.
%
%     *See GUI Options on GUIDE's Tools menu. Choose "GUI allows only
one
%     instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help GUI

% Last Modified by GUIDE v2.5 28-Apr-2008 13:40:10

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                  'gui_Singleton',   gui_Singleton, ...
                  'gui_OpeningFcn', @GUI_OpeningFcn, ...
                  'gui_OutputFcn',  @GUI_OutputFcn, ...
                  'gui_LayoutFcn',   [], ...
                  'gui_Callback',    []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before GUI is made visible.
function GUI_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
```

```

% hObject      handle to figure
% eventdata    reserved - to be defined in a future version of MATLAB
% handles       structure with handles and user data (see GUIDATA)
% varargin      unrecognized PropertyName/PropertyValue pairs from the
%               command line (see VARARGIN)

%All other panels besides the main panel is turned off
set(handles.main_panel, 'Visible', 'on');
set(handles.settings_panel, 'Visible', 'off');

% Initialization
handles.ComPort='';
handles.Patient_ID='';

%Flag to indicate whether data collection process is ON or OFF
% false - data collection inactive; true- data collection active
handles.active= false;

% handles to store ECG and temperature data
handles.ecg_data=[];
handles.temp_data=[];

%Enabling and Disabling buttons in the main panel
set(handles.Save_pushbutton, 'Enable', 'on');
set(handles.Stop_pushbutton, 'Enable', 'off');
set(handles.Start_pushbutton, 'Enable', 'off');

set(handles.Temp_Disp, 'String', 'Temp');

% Intializes email settings
set_mailSettings(handles);

%Looks for available ports in the computer
find_ports_Callback(hObject, eventdata, handles)
handles=guidata(hObject);

% Get values saved from previous GUI session. The values are stored in
% text files in the working directory.
[HR_low,HR_high,temp_low, temp_high,ox_low] =
textread('threshold_info.txt','%s %s %s %s %s', 1);
[sender, receipient] = textread('email_info.txt','%s %s', 1);

%By default, email autosend is ON
set(handles.auto_send_checkbox, 'Value',1);

% Set values of the fields in the GUI
set(handles.sender_email, 'String',char(sender));
set(handles.receipient_email, 'String',char(receipient));

set(handles.HR_low, 'String',char(HR_low));
set(handles.HR_high, 'String',char(HR_high));
set(handles.temp_low, 'String',char(temp_low));

```

```

set(handles.temp_high, 'String', char(temp_high));
set(handles.oxygen_low, 'String', char(ox_low));

% default temp unit is celsius
set(handles.temp_unit, 'Value', 1);
set(handles.temp_unit2, 'Value', 1);

% Update handles structure
handles.output = hObject;
guidata(hObject, handles);

% UIWAIT makes GUI wait for user response (see UIRESUME)
%uiwait(handles.GUI_figure);

% --- Outputs from this function are returned to the command line.
function varargout = GUI_OutputFcn(hObject, eventdata, handles)
% varargout cell array for returning output args (see VARARGOUT);
% hObject handle to figure
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

function HR_Dispatch_Callback(hObject, eventdata, handles)
% hObject handle to HR_Dispatch (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Hints: get(hObject, 'String') returns contents of HR_Dispatch as text
% str2double(get(hObject, 'String')) returns contents of HR_Dispatch
% as a double

% --- Executes during object creation, after setting all properties.
function HR_Dispatch_CreateFcn(hObject, eventdata, handles)
% hObject handle to HR_Dispatch (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles empty - handles not created until after all CreateFcns
% called

% Hint: edit controls usually have a white background on Windows.
% See ISPC and COMPUTER.
if ispc && isequal(get(hObject, 'BackgroundColor'),
get(0, 'defaultUicontrolBackgroundColor'))
set(hObject, 'BackgroundColor', 'white');
end

```

```
end
```

```
function Temp_Dispatch_Callback(hObject, eventdata, handles)
% hObject      handle to Temp_Dispatch (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Temp_Dispatch as text
%         str2double(get(hObject,'String')) returns contents of
Temp_Dispatch as a double

% --- Executes during object creation, after setting all properties.
function Temp_Dispatch_CreateFcn(hObject, eventdata, handles)
% hObject      handle to Temp_Dispatch (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUiControlBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in Start_pushbutton.
function Start_pushbutton_Callback(hObject, eventdata, handles)
% hObject      handle to Start_pushbutton (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see
GUIDATA)handles.active = true

%disable "start" button after it is pushed until "stop" is pushed
%and enable "stop" button
set(handles.Start_pushbutton,'Enable','off');
set(handles.Stop_pushbutton,'Enable','on');

%handles.ComPort is set in find_ports_Callback
handles.port = serial(handles.ComPort,'BaudRate',9600);
%Set input buffer size
set(handles.port,'InputBufferSize',100000);

%objects pointing to the data files.
file_all= fopen('all_data.csv','w'); % stores all data (used for
debugging)
file_ECG= fopen('ECG_data.csv','w');
file_temp= fopen('temp_data.csv','w');
```



```

handles.active = true;
guidata(hObject, handles)
temp_data=[];
ecg_data=[];
pulse_ox_data=[];

% This should probably be placed in the OpeningFcn so that fopen causes
% errors we can ask the user to select the other available COM port
% try
%     fopen(handles.port);
% catch ME
%
% end

%Opens the serial port
fopen(handles.port);
flushinput(handles.port); %Clears the buffer

fprintf(handles.port, 'Y'); %Send start signal to the PIC
microcontroller

% For Heart rate estimation
count = 0;
QRS_update = -1; %initialized counter for QRS step-by-step calculation
ecg_for_R=[]; %stores ecg data used for R-peak estimation

while (handles.active == 1)

    numdata=3000; % total number of data collected from the serial port
    at a time
    out_c=fread(handles.port,numdata); % read data from the port
    out=out_c';
    fprintf(file_all,'%d,',out); %write data to file
    [max_val,index]=max(out); %find the maximum value in the array and
    its index

    %Data parsing
    %Temperature data is followed by 255. The rest of the data is ECG.
    %Pulse oximeter is not incorporated yet.

    if(max_val==255) %temp data is in the array

        if (index~=numdata)
            temp_data=[out(index+1)];
        end
        if(index~=1) % rest of the data is ecg_data
            ecg_data=[out(1:(index-1)), out((index+2):numdata)];
        else
            ecg_data=[ out(3:numdata) ];
        end

    else % no temp_data
        ecg_data=out;
    end
end

```

```

end

% *****
% Data parsing code if pulse ox was also incorporated. In this case
it is
% assumed that 254 is used as a flag for pulse ox data, 255 as a flag
for
% temperature data (as before) and the ECG data is not flagged.

% [max_val,index]=max(out);
% pulse_flags = find(out==254); % indices of the pulse_flags
% pulse_index= pulse_flags + 1; % indices of the pulse_ox data
%
%
% other_data_index = [pulse_flags, pulse_index, index+1];
%
% % collect ECG data
% for(m=1:numdata)
%
%     p = (other_data_index ==m);
%     if (max(p)~=1)
%         ecg_data =[ecg_data, out(m)];
%     end
%
% end
%
%
% % collect temp data
% if(max_val==255) %temp data is in the array
%
%     if (index~=numdata)
%         temp_data=[temp_data,out(index+1)];
%     end
%
% end
%
% % collect all pulse ox data
% for (k=1:length(pulse_index))
%     pulse_ox_data = [pulse_ox_data, out(pulse_index(k))];
% end
%
% *****

%Write to file
fprintf(file_ECG, '%d,',ecg_data');
fprintf(file_temp, '%d,',temp_data');

%Used for debugging
ecg_data';
temp_data;
% pulse_ox_data';

if(length(temp_data)~=0)

```

```

    % Temperature processing
    % y = 40.126x - 51.574 Calibration curve for the temperature
sensor

    temp_Volts= 3.3 * (temp_data/255); %Converting decimal value to
volts
    Temp_C = 40.126*temp_Volts - 51.574;%translating Voltage to
temperature
    Temp_F = Temp_C * (9/5) + 32 ; %Equation relating fahrenheit
and celsius

    % Converting string to a double
    temp_low = str2double(get(handles.temp_low,'String'));
    temp_high = str2double(get(handles.temp_high,'String'));

    %% Alarm settings check for buzzer for Temperature
abnormalities
    if (get(handles.temp_unit,'Value')==1) %If Celsius is selected
        set(handles.Temp_Disp,'String',num2str(Temp_C));
        if((Temp_C < temp_low) || (Temp_C > temp_high))
            fprintf(handles.port,'%c','B'); % 'B'- to start buzzer
        else
            fprintf(handles.port,'%c','P'); % 'P' does not mean
anything in particular.
            %It is sent so that
signal
            %strength indicator
(RSSI) blinks
            %in the remote unit. As
it
            %does so only when it
            %receives anything.

        end
    elseif(get(handles.temp_unit,'Value')==2) % If Fahrenheit is
selected
        set(handles.Temp_Disp,'String',num2str(Temp_F));
        if((Temp_F<temp_low) || (Temp_F>temp_high))
            fprintf(handles.port,'%c','B'); % same as above
        else
            fprintf(handles.port,'%c','P'); % same as above
        end
    end

end

% ECG signal processing

%Variable initialization
threshold_value = 155; % minimum amplitude threshold for R-peak
detection
QRS_update = -1; %initialized counter for QRS step-by-step
calculation

```

```

count = 0;

%expect no QRS in the another 1/10 of a sec.

current = -62; % only for initialization purposes
R_peak_index = []; % an array to keep track of the array indices of
where R-peak
                    % is located

ecg_for_R=[ecg_for_R, ecg_data]; %Data array to find the R peaks (5
screenshots)
count=count+1; % count is used to keep track that we have (5 *
numdata) datapoints for heart rate calculation
                    % In other words, heart rate update every 5
screenshots
                    % of data
QRS_update = QRS_update +1;%

if (mod(count,5)==0) %For every 5 screenshots...(5 * numdata)
datapoints
    count;
    for (i=1:length(ecg_for_R)-4)
        if (ecg_for_R(i)>threshold_value)

            if(i>4)
                %finding slope over 5 points to reduce the effect
of
                %outliers, but not get rid of them (R peaks would
be
                %removed)
                i;
                x=1:5;
                before_fit = polyfit(x,ecg_for_R(i-4:i),1); % slope
of a linear fit on 5 pts before the current data point
                after_fit = polyfit(x,ecg_for_R(i: i+4),1);% slope
of a linear fit on 5 pts after the current data point
                Beforedata= ecg_for_R(i-4:i);
                Afterdata= ecg_for_R(i:i+4);

                if((before_fit(1)>1)&& (before_fit(1)<8) && (i >
(current + 62)))
                    if((after_fit(1)<-1) && (after_fit(1)>-8))
                        R_peak_index=[R_peak_index;i]
                        current = i;
                    end
                end
            end
        end
    end
end

% Calculating Heart Rate
diff=[];

for (j=1:(length(R_peak_index)-1))
    diff(j) = R_peak_index(j+1) - R_peak_index(j);
end

```

```

end
heart_rate = 60/(mean(diff)/620) % beats per min

% % Buzzer check

HR_low = str2double(get(handles.HR_low, 'String'));
HR_high= str2double(get(handles.HR_high, 'String'));

%% Alarm settings check for buzzer for heart rate abnormalities
if ((heart_rate>HR_high)|| (heart_rate<HR_low))
    fprintf(handles.port, '%c', 'B'); % see comments above (temp
abnormalities)
else
    fprintf(handles.port, '%c', 'P');
end

count = 0; %reset count
set(handles.HR_Dispatch, 'String', num2str(heart_rate));
end

%Plotting ECG
ecg_Volts= 3.3 * (ecg_data/255); %Converting decimal value to volts
handles = guidata(hObject);
pause(1); % this pause is required to make the callback
interruptible
    % this way the data collection can be stopped using the
    % Stop_pushbutton callback
plot(ecg_Volts);
time = numdata*(1/620); %Converting number of samples to time
axis([0 numdata 0 3.5]);

% Search for Q and S points to calculate QRS duration
Q_index = []; % array containing indices of array where Q is found
S_index = []; % array containing indices of array where S is found

if (mod(count,5)==0)
% Median filter to remove outliers
ecg_for_QRS = medfilt1(ecg_for_R,5);

for(k=1:length(R_peak_index))
    L = 0;
    p = 0;
% finding Q_index
    Q_found = 0;
    while(Q_found==0)
        m = R_peak_index(k)-L;
        if (m>4)
            Q_fit = polyfit(x,ecg_for_QRS(m-4:m),1); %slope of a
linear fit
            if (Q_fit(1)<1)
                Q_index = [Q_index;m];
                Q_found = 1;
            else
                L=L+1;
            end
        end
    end
end

```

```

        else
            Q_index = [Q_index;1];
            Q_found = 1;
%           If Q is not found within the array window, which
%           should
%           only happen at the first R peak, use the first value
%           of the array. Necessary to prevent an infinite loop.
        end
    end

%   finding S_index
    S_found = 0;
    while(S_found==0)
        n = R_peak_index(k)+p;
        if (n<=(length(ecg_for_QRS) - 4))
            S_fit = polyfit(x,ecg_for_QRS(n:(n+4)),1);

            if (S_fit(1)>-0.5)
                S_index = [S_index;n];
                S_found = 1;
            else
                p=p+1;
            end
        elseif (n>(length(ecg_for_QRS)-4) &&
n<length(ecg_for_QRS))
%           To evaluate points at the end of the screenshot, use
%           previous points to calculate slope

            S_fit = polyfit(x,ecg_for_QRS((n-4):n),1);
            if (S_fit(1)>-0.5)
                S_index = [S_index;n];
                S_found = 1;
            else
                p=p+1;
            end
        else
            S_index = [S_index;n];
            S_found = 1;

%           If the S wave is not found within the screenshot, use
%           the
%           last value of the screenshot for QRS-width
%           calculation.
        end
    end
end

%   Calculate complex width
Individual_widths = ((S_index-Q_index)./620)*1000;%QRS complex
width in ms
diff = S_index-Q_index;
QRS_width = sum(Individual_widths)/length(Individual_widths);

```

```

        ecg_for_R=[];% clear out data array for next set

        set(handles.QRS_Dispatch,'String',num2str(QRS_width));
    end
%-----

end%end of while loop

%close all the txt data file.
fclose(file_all);
fclose(file_ECG);
fclose(file_temp);

zipfile_name= get(handles.File_location,'String');
zip(zipfile_name,{'ECG_data.csv','temp_data.csv'});

%Initialize email settings
set_mailSettings(handles);

auto_send = get(handles.auto_send_checkbox,'Value');

%Date and time stamp - to include int the email
date_time_stamp= datestr(now);

attachment = strcat(zipfile_name,'.zip');

%If autosend is ON
if (auto_send ~= 0)
%Send the email

sendmail('anima.singh@gmail.com',handles.Patient_ID,date_time_stamp,attachment);
end

%close ports
fclose(handles.port);
delete(handles.port);
clear handles.port;

% --- Executes on button press in Stop_pushbutton.
function Stop_pushbutton_Callback(hObject, eventdata, handles)
% hObject    handle to Stop_pushbutton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.active=false
flag='active false'
guidata(hObject, handles) ; %updates the value of handles changed
locally to be available globally.
handles = guidata(hObject); % makes a local copy of the handle

%re-enable "start" button after "stop" is pushed
set(handles.Stop_pushbutton,'Enable','off');
guidata(hObject, handles) ;

```

```

handles = guidata(hObject);

function set_mailSettings(handles)
%Define these variables appropriately:
mail = 'wipmod@gmail.com'; %Your GMail email address
%mail = get(handles.sender_email,'String');

h=gcf ; %(for * password)
%password = 'Cheever90'; %Your GMail password
password = get(h, 'UserData');

%%%%%%%%%%Print handles for test

%n='in set_mailSettings'
%handles

smtp = get(handles.smtp_server, 'String');

%Then this code will set up the preferences properly:
setpref('Internet', 'E_mail', mail);
setpref('Internet', 'SMTP_Server', smtp);
setpref('Internet', 'SMTP_Username', mail);
setpref('Internet', 'SMTP_Password', password);
props = java.lang.System.getProperties;
props.setProperty('mail.smtp.auth', 'true');
props.setProperty('mail.smtp.socketFactory.class',
'javax.net.ssl.SSLSocketFactory');
props.setProperty('mail.smtp.socketFactory.port', '465');

% --- Executes on button press in Save_pushbutton.
function Save_pushbutton_Callback(hObject, eventdata, handles)
% hObject    handle to Save_pushbutton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
SaveWindow('GUI', handles.GUI_figure);

% --- Executes when GUI_figure is resized.
function GUI_figure_ResizeFcn(hObject, eventdata, handles)
% hObject    handle to GUI_figure (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% -----
function main_Callback(hObject, eventdata, handles)

```



```

% hObject    handle to main (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.main_panel, 'Visible','on');
set(handles.settings_panel, 'Visible','off');

% -----
function settings_menu_Callback(hObject, eventdata, handles)
% hObject    handle to settings_menu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.main_panel, 'Visible','off');
set(handles.settings_panel, 'Visible','on');

% --- Executes on button press in find_ports.
%***IMPORTANT***
%THIS BUTTON IS NOT VISIBLE IN THE GUI. BUT IS REQUIRED TO FIND THE
%AVAILABLE PORTS.
%This code was obtained from
%http://www.mathworks.com/matlabcentral/fileexchange/loadFile.do?object
Id=8499&objectType=file
%Author:    Jeremy Smith
function find_ports_Callback(hObject, eventdata, handles)
% hObject    handle to find_ports (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

try
    s=serial('IMPOSSIBLE_NAME_ON_PORT');fopen(s);
catch
    lErrMsg = lasterr;
end

%Start of the COM available port
lIndex1 = findstr(lErrMsg,'COM');
%End of COM available port
lIndex2 = findstr(lErrMsg,'Use')-3;

lComStr = lErrMsg(lIndex1:lIndex2);

%Parse the resulting string
lIndexDot = findstr(lComStr,',');

% If no Port are available
if isempty(lIndex1)
    lCOM_Port{1}='';
    return;
end

```

```

% If only one Port is available
if isempty(lIndexDot)
    lCOM_Port{1}=lComStr;
    return;
end

lCOM_Port{1} = lComStr(1:lIndexDot(1)-1);

for i=1: numel(lIndexDot)+1
    % First One
    if (i==1)
        lCOM_Port{1,1} = lComStr(1:lIndexDot(i)-1);
    % Last One
    elseif (i==numel(lIndexDot)+1)
        lCOM_Port{1,1} = lComStr(lIndexDot(i-1)+2:end);
    % Others
    else
        lCOM_Port{1,1} = lComStr(lIndexDot(i-1)+2:lIndexDot(i)-1);
    end
end
handles.avail_ports= lCOM_Port;

%set(handles.ports_menu, 'String', [handles.avail_ports{2}, ...
%    handles.avail_ports{1}]);

set(handles.ports_menu, 'String', {handles.avail_ports{2}, handles.avail_p
orts{1}});

val = get(handles.ports_menu, 'Value');
str = get(handles.ports_menu, 'String');
handles.ComPort = str{val};

guidata(hObject, handles);

function smtp_server_Callback(hObject, eventdata, handles)
% hObject      handle to smtp_server (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)

% Hints: get(hObject, 'String') returns contents of smtp_server as text
%         str2double(get(hObject, 'String')) returns contents of
smtp_server as a double

% --- Executes during object creation, after setting all properties.
function smtp_server_CreateFcn(hObject, eventdata, handles)
% hObject      handle to smtp_server (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.

```

```

%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on selection change in ports_menu.
function ports_menu_Callback(hObject, eventdata, handles)
% hObject      handle to ports_menu (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns ports_menu contents
as cell array
%       contents{get(hObject,'Value')} returns selected item from
ports_menu
val = get(hObject,'Value');
str = get(hObject, 'String');

handles.ComPort = str{val};

guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function ports_menu_CreateFcn(hObject, eventdata, handles)
% hObject      handle to ports_menu (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      empty - handles not created until after all CreateFcns
called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in auto_send_checkbox.
function auto_send_checkbox_Callback(hObject, eventdata, handles)
% hObject      handle to auto_send_checkbox (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of auto_send_checkbox

function receipient_email_Callback(hObject, eventdata, handles)
% hObject      handle to receipient_email (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)

```

```

% Hints: get(hObject,'String') returns contents of receipient_email as
text
%         str2double(get(hObject,'String')) returns contents of
receipient_email as a double

% --- Executes during object creation, after setting all properties.
function receipient_email_CreateFcn(hObject, eventdata, handles)
% hObject    handle to receipient_email (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUiControlBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function sender_email_Callback(hObject, eventdata, handles)
% hObject    handle to sender_email (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of sender_email as text
%         str2double(get(hObject,'String')) returns contents of
sender_email as a double

% --- Executes during object creation, after setting all properties.
function sender_email_CreateFcn(hObject, eventdata, handles)
% hObject    handle to sender_email (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUiControlBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function password_Callback(hObject, eventdata, handles)
% hObject    handle to password (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of password as text

```

```

%      str2double(get(hObject,'String')) returns contents of password
as a double

% --- Executes during object creation, after setting all properties.
function password_CreateFcn(hObject, eventdata, handles)
% hObject    handle to password (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%      See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

%% Hide Password
%%This function is specified as a keypress-function of 'password' edit
text
%%box.
%%Downloaded from
http://www.mathworks.com/matlabcentral/fileexchange/loadFile.do?objectId=8499&objectType=file
%%Author:Jeremy Smith
function KeyPress_Function(hObject,eventdata,handles)
% Function to replace all characters in the password edit box with
% asterixes
h=gcf;
password = get(h,'Userdata');
key= get(h,'currentkey');

switch key
    case 'backspace'
        password = password(1:end-1); % Delete the last character in
the password
    case 'return' % This cannot be done through callback without
making tab to the same thing
        gui = getappdata(0,'logindlg');
        OK([],[],gui.main);
    case 'delete'
        password = password(1:end-1);
    case 'tab' % Avoid tab triggering the OK button
        gui = getappdata(0,'logindlg');
        uicontrol(gui.OK);
    otherwise
        password = [password, get(h,'currentcharacter')]; % Add the
typed character to the password
end

SizePass = size(password); % Find the number of asterixes
if SizePass(2) > 0
    asterix(1,1:SizePass(2)) = '*'; % Create a string of asterixes the
same size as the password

```

```

        set(handles.password,'String',asterix) % Set the text in the
password edit box to the asterix string
else
    set(handles.password,'String','')
end

set(h,'Userdata',password); % Store the password in its current state


% --- Executes on button press in close_app_button.
function close_app_button_Callback(hObject, eventdata, handles)
% hObject    handle to close_app_button (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
user_response = modaldlg('Title','Confirm Close')
switch lower(user_response)
case 'no'
    % take no action
case 'yes'
    % Prepare to close GUI application window
    %
    %
    %
    delete(handles.GUI_figure)
end


% --- Executes on selection change in temp_unit.
function temp_unit_Callback(hObject, eventdata, handles)
% hObject    handle to temp_unit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% Hints: contents = get(hObject,'String') returns temp_unit contents as
cell array
%         contents{get(hObject,'Value')} returns selected item from
temp_unit
if (get(hObject,'Value')==2)%if user switches from celsius to
fahrenheit
    set(handles.temp_unit2,'Value',2);


    a= get(handles.Temp_Disp,'String');
    b= get(handles.temp_low,'String');
    c= get(handles.temp_high,'String');


    tempC = str2double(a);
    tempF = tempC *(9/5)+ 32;


    tempC_low = str2double(b);
    tempF_low = tempC_low *(9/5)+ 32;


    tempC_high = str2double(c);
    tempF_high = tempC_high*(9/5)+ 32;


    set(handles.Temp_Disp,'String',num2str(tempF))
    set(handles.temp_low,'String',num2str(tempF_low))

```

```

        set(handles.temp_high, 'String', num2str(tempF_high))

elseif (get(hObject, 'Value')==1) % if user switches from fahrenheit to
celsius
    set(handles.temp_unit2, 'Value', 1);

    a= get(handles.Temp_Disp, 'String')
    b= get(handles.temp_low, 'String');
    c= get(handles.temp_high, 'String');

    tempF = str2double(a);
    tempC = (tempF-32)*(5/9);

    tempF_low = str2double(b);
    tempC_low = (tempF_low-32)*(5/9);

    tempF_high = str2double(c);
    tempC_high = (tempF_high-32)*(5/9);

    set(handles.Temp_Disp, 'String', num2str(tempC))
    set(handles.temp_low, 'String', num2str(tempC_low))
    set(handles.temp_high, 'String', num2str(tempC_high))

end

```

```

% --- Executes during object creation, after setting all properties.
function temp_unit_CreateFcn(hObject, eventdata, handles)
% hObject    handle to temp_unit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject, 'BackgroundColor'),
get(0, 'defaultUicontrolBackgroundColor'))
    set(hObject, 'BackgroundColor', 'white');
end

```

```

% --- Executes on button press in Save_email.
function Save_email_Callback(hObject, eventdata, handles)
% This button saves email addresses from previous GUI sessions

```

```

% hObject    handle to Save_email (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
fid = fopen('email_info.txt','w');

class((get(handles.sender_email,'String')));

a = get(handles.sender_email,'String');
b= get(handles.receipient_email,'String');

fprintf(fid,'%s ',a);
fprintf(fid,'%s ',b);
fclose(fid);

function edit13_Callback(hObject, eventdata, handles)
% hObject    handle to edit13 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit13 as text
%        str2double(get(hObject,'String')) returns contents of edit13
as a double

% --- Executes during object creation, after setting all properties.
function edit13_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit13 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%        See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit15_Callback(hObject, eventdata, handles)
% hObject    handle to edit15 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit15 as text
%        str2double(get(hObject,'String')) returns contents of edit15
as a double

% --- Executes during object creation, after setting all properties.
function edit15_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit15 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB

```



```

% handles      empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%      See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function QRS_Dispatch_Callback(hObject, eventdata, handles)
% hObject      handle to QRS_Dispatch (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of QRS_Dispatch as text
%      str2double(get(hObject,'String')) returns contents of QRS_Dispatch
as a double

% --- Executes during object creation, after setting all properties.
function QRS_Dispatch_CreateFcn(hObject, eventdata, handles)
% hObject      handle to QRS_Dispatch (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%      See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function HR_low_Callback(hObject, eventdata, handles)
% hObject      handle to HR_low (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of HR_low as text
%      str2double(get(hObject,'String')) returns contents of HR_low
as a double

% --- Executes during object creation, after setting all properties.
function HR_low_CreateFcn(hObject, eventdata, handles)
% hObject      handle to HR_low (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      empty - handles not created until after all CreateFcns
called

```

```

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function HR_high_Callback(hObject, eventdata, handles)
% hObject      handle to HR_high (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of HR_high as text
%         str2double(get(hObject,'String')) returns contents of HR_high
as a double

% --- Executes during object creation, after setting all properties.
function HR_high_CreateFcn(hObject, eventdata, handles)
% hObject      handle to HR_high (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function temp_low_Callback(hObject, eventdata, handles)
% hObject      handle to temp_low (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of temp_low as text
%         str2double(get(hObject,'String')) returns contents of temp_low
as a double

% --- Executes during object creation, after setting all properties.
function temp_low_CreateFcn(hObject, eventdata, handles)
% hObject      handle to temp_low (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.

```

```

%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function temp_high_Callback(hObject, eventdata, handles)
% hObject      handle to temp_high (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of temp_high as text
%          str2double(get(hObject,'String')) returns contents of
temp_high as a double

% --- Executes during object creation, after setting all properties.
function temp_high_CreateFcn(hObject, eventdata, handles)
% hObject      handle to temp_high (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in celsius_radio.
function celsius_radio_Callback(hObject, eventdata, handles)
% hObject      handle to celsius_radio (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of celsius_radio

% --- Executes on button press in fahrenheit_radio.
function fahrenheit_radio_Callback(hObject, eventdata, handles)
% hObject      handle to fahrenheit_radio (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of fahrenheit_radio

% --- Executes on button press in alarm_set_pushbutton.
function alarm_set_pushbutton_Callback(hObject, eventdata, handles)

```

```

% Saves alarm settings to a txt file so that they can be uploaded next
time
% GUI is opened.
% hObject    handle to alarm_set_pushbutton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

fid = fopen('threshold_info.txt','w');

a= get(handles.HR_low,'String');
b= get(handles.HR_high,'String');
c= get(handles.temp_low,'String');
d= get(handles.temp_high,'String');
e= get(handles.oxygen_low,'String');

fprintf(fid,'%s ',a);
fprintf(fid,'%s ',b);
fprintf(fid,'%s ',c);
fprintf(fid,'%s ',d);
fprintf(fid,'%s ',e);

fclose(fid);

function oxygen_low_Callback(hObject, eventdata, handles)
% hObject    handle to oxygen_low (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of oxygen_low as text
%        str2double(get(hObject,'String')) returns contents of
oxygen_low as a double

% --- Executes during object creation, after setting all properties.
function oxygen_low_CreateFcn(hObject, eventdata, handles)
% hObject    handle to oxygen_low (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%        See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on selection change in temp_unit2.
function temp_unit2_Callback(hObject, eventdata, handles)
% hObject    handle to temp_unit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

```

```

% Hints: contents = get(hObject,'String') returns temp_unit2 contents
as cell array
%         contents{get(hObject,'Value')} returns selected item from
temp_unit2

if (get(hObject,'Value')==2)%if user switches from celsius to
fahrenheit
    set(handles.temp_unit,'Value',2);

    a= get(handles.Temp_Disp,'String');
    b= get(handles.temp_low,'String');
    c= get(handles.temp_high,'String');

    tempC = str2double(a);
    tempF = tempC *(9/5)+ 32;

    tempC_low = str2double(b);
    tempF_low = tempC_low *(9/5)+ 32;

    tempC_high = str2double(c);
    tempF_high = tempC_high*(9/5)+ 32;

    set(handles.Temp_Disp,'String',num2str(tempF))
    set(handles.temp_low,'String',num2str(tempF_low))
    set(handles.temp_high,'String',num2str(tempF_high))

elseif (get(hObject,'Value')==1) % if user switches from fahrenheit to
celsius
    set(handles.temp_unit,'Value',1);
    a= get(handles.Temp_Disp,'String')
    b= get(handles.temp_low,'String');
    c= get(handles.temp_high,'String');

    tempF = str2double(a);
    tempC = (tempF-32)*(5/9);

    tempF_low = str2double(b);
    tempC_low = (tempF_low-32)*(5/9);

    tempF_high = str2double(c);
    tempC_high = (tempF_high-32)*(5/9);

    set(handles.Temp_Disp,'String',num2str(tempC))
    set(handles.temp_low,'String',num2str(tempC_low))
    set(handles.temp_high,'String',num2str(tempC_high))
end

% --- Executes during object creation, after setting all properties.
function temp_unit2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to temp_unit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

```

```
% Hint: popupmenu controls usually have a white background on Windows.  
%     See ISPC and COMPUTER.  
if ispc && isequal(get(hObject,'BackgroundColor'),  
get(0,'defaultUicontrolBackgroundColor'))  
    set(hObject,'BackgroundColor','white');  
end
```