University of New Mexico
UNM Digital Repository

Electrical and Computer Engineering ETDs | Engineering ETDs

6-25-2010

GUARDIAN : automated patient monitoring system

Johnny C. Silva

Follow this and additional works at: https://digitalrepository.unm.edu/ece_etds

Recommended Citation
Silva, Johnny C.. "GUARDIAN : automated patient monitoring system." (2010). https://digitalrepository.unm.edu/ece_etds/238

This Thesis is brought to you for free and open access by the Engineering ETDs at UNM Digital Repository. It has been accepted for inclusion in Electrical and Computer Engineering ETDs by an authorized administrator of UNM Digital Repository. For more information, please contact disc@unm.edu.

Johnny C. Silva
Candidate

Electrical and Computer Engineering
Department

This thesis is approved, and it is acceptable in quality and form for publication:

Approved by the Thesis Committee:

Chairperson

GUARDIAN: AUTOMATED PATIENT MONITORING SYSTEM

BY

JOHNNY C. SILVA

B.S., Electronics Engineering Technology, Oregon Institute of Technology, 2002

THESIS

Submitted in Partial Fulfillment of the
Requirements for the Degree of

Master of Science

Electrical Engineering

The University of New Mexico
Albuquerque, New Mexico

December, 2009

GUARDIAN: AUTOMATED PATIENT MONITORING SYSTEM

BY

JOHNNY C. SILVA

ABSTRACT OF THESIS

Submitted in Partial Fulfillment of the
Requirements for the Degree of

Master of Science

Electrical Engineering

The University of New Mexico
Albuquerque, New Mexico

December, 2009

GUARDIAN: AUTOMATED PATIENT MONITORING SYSTEM

by

Johnny C. Silva

B.S., Electronics Engineering Technology, Oregon Institute of Technology, 2002
M.S., Electrical Engineering, University of New Mexico, 2009

ABSTRACT

Telemedicine allows medical professionals to monitor, examine, and consult with patients remotely by transferring medical information via wireless communication technologies. This thesis describes the design and implementation of a new automated patient monitoring system.

In many cases, vital signs, location data, and other patient information are currently not tracked in a real-time automated fashion. The remote monitoring system proposed and developed in this thesis has the ability to provide doctors/healthcare professionals with 24-hour access to real-time patient information, as well as historical data of every one of their patients.

A prototype automated patient monitoring system was constructed to demonstrate general telemedicine system design concepts. The prototype consists of sensor units worn by each patient that monitor real-time vital signs information and wirelessly transmit it to a central monitoring computer (CMC) for processing. A graphical user interface displays this real-time data.

This project identified the required components of a wireless automated patient monitoring system and demonstrated one possible implementation methodology for such a system. This stand-alone prototype automated patient monitoring system was successfully designed, built, and tested.

The prototype system described in this thesis is focused on patient monitoring within a medical facility. This same system can be slightly modified to achieve different types of monitoring in different situations such as child care facilities and home healthcare. Possible future monitoring technologies are explored.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

Introduction

An important aspect of telemedicine is the ability to transfer real-time medical information via communication networks. It enables doctors to monitor, examine, and consult with patients remotely.

Telemedicine technologies can improve doctor-to-patient response time in an emergency situation and provide periodic historical vital signs data that lead to more accurate diagnoses. These technologies also allow patients to have uninhibited movement throughout a monitoring facility or their home.

Standard patient monitoring requires a nurse to periodically record a patient's vital signs. These vital signs include body temperature, pulse rate, blood pressure, and respiratory rate. Patients are interrupted from their current activities or sleep when the nurse performs these measurements. Missing one scheduled set of measurements could reduce the effectiveness of the care the patient receives.

These problems can be eliminated by incorporating an automated monitoring system that records the patient's vital signs at periodic intervals. Such a system is more reliable and less inconvenient for the patient and the medical staff.

1.1 Purpose

This thesis describes the design and implementation of a new automated patient monitoring system, named Guardian.

1.2 Scope

- Investigate the current state of patient monitoring technologies and problems these system face
- Describe current technologies available to help solve these problems
- List commercially available solutions
- Provide design and implementation of new automated patient monitoring system
- Analyze results of newly implemented system
- Propose possible improvements and future work

1.3 Acronym Definitions

CMC – Central Monitoring Computer

GDM – Guardian database manager

GPIO – General Purpose Input Output

GSP – Guardian software package

GUI – Graphical User Interface

ISR –Interrupt Service Routine

PIC – Peripheral Interface Controller

RF – Radio Frequency

UART – Universal Asynchronous Receive Transmit

WAP – Wireless Access Point

WM –Wrist Module

CHAPTER 2

Issues Facing Current Patient Monitoring Techniques

Vital signs, location data, and other patient information are currently not tracked in a real-time automated fashion. Ideally, however, doctors would like to have 24-hour access to real-time patient information and historical data on every one of their patients.

There are several limiting factors when attempting to monitor patient vital signs in and out of medical facilities. These factors vary depending on both the specific facility and the patient. Some common examples include:

2.1 Hospitals

Each time a patient enters a hospital, they fill out a medical questionnaire regarding the reason for their visit. These questionnaires can be error prone and may be difficult for patients to complete accurately. Misinformation may make it difficult for doctors to diagnose the patient's medical problem correctly. This problem is compounded each time a patient visits multiple care providers.

Much of a patient's information such as medical history, allergies, and current treatments are kept on the patient's chart. If this chart is misplaced or destroyed before the information is transferred to a computer for archiving, the information the chart contains will be permanently lost and the patient's care may be compromised. Without knowing which treatments and medications the patient has already received, there is a chance that improper care will be given or incorrect doses of medication prescribed.

When a patient is being monitored by a wired monitoring device the wires can get in the way and hinder the patient's movements. The wires can also present a choking

hazard if the patient turns over in the wrong way or could restrict blood flow if they are wrapped around a patients extremities. Also, tripping over the monitoring wires could result in serious injury to patients or medical staff.

Currently, in a hospital's new born care area, each baby has a radio frequency identification (RFID) tag attached to its ankle. RFID tags are devices used in many types of stores to ensure that shoplifters don't steal items from the store. If an item that wasn't purchased passes through the store's doors, an alarm is activated and the store's security personnel can catch the shoplifter and retrieve the stolen item. The RFID tag on each baby works in this same manner. The problem with this technology is that it can be easily circumvented by simply cutting the RFID tag off of the baby's ankle. If the RFID tag was removed, babies could be taken from the facility without the staff's knowledge.

2.2 Assisted Living Facilities

Patients in an assisted living facility must be monitored on a frequent basis. An attending nurse must find the patient to administer the patient's next dose of medication or check the patient's vital signs. If the patient is not in their room it may be difficult for the nurse to locate the patient and perform their duties.

When a nurse administers medication to a patient they log the time, type, and amount of medication given along with any other notes that are applicable. If the nurse forgets to log any of this information, or logs inaccurate information, the patient may miss a scheduled dose or get a double dose of medication. Either of these situations could compromise the health of the patient.

Many times elderly patients may fall or be in need of immediate assistance. If neither nurse nor panic button is accessible, the patient may remain in this situation until

discovered. This situation may last for minutes or even days before anyone realizes there is a problem.

Frequently, the nursing staff at assisted living facilities need to monitor the activity level of a patient to ensure that the patient doesn't over stress their body. This type of monitoring is time intensive and difficult to accomplish accurately. Without having 24-hour monitoring of the patient's vital signs, the nurse must rely heavily on the memory of the patient to provide activity level information throughout the day.

Families fear their loved ones won't be properly cared for in assisted living facilities. Even with assurances from the facility staff, this may not be enough to alleviate the family's concerns. Also, patients may be unable or unwilling to convey their quality of care.

Many potential health issues can be prevented by frequent vital sign readings. These issues may be overlooked if the medical staff's workload prevents them from taking frequent readings.

2.3 Home Healthcare

Home healthcare presents unique challenges. Costs limit the frequency of in-home care for many patients. Continuous real-time monitoring of a patient's conditions is difficult to achieve with currently available solutions. Patients must travel to a hospital for simple checkup or consultations that may only take a few minutes. Also, if a patient is alone in their home and is injured, it may be difficult for the patient to call for help.

Patients recovering from medical procedures may be kept for observation simply to monitor their vital signs. This increases the cost and inconvenience for the patient and also utilizes more of the hospital's space and resources for noncritical tasks.

Some Alzheimer or dementia patients may wander out of their home without anyone noticing. The physical safety of an Alzheimer's patient becomes a factor as well. As the patient's disease progresses, the patient is no longer able to call for assistance from neighbors or emergency personnel.

2.4 Combat Support Hospitals

When a soldier is wounded on the battle field, it is critical that the medical personnel have their accurate medical histories. This information may be hard or even impossible to access given different battle field situations. Without this information the medical personnel may be unaware of the soldier's potential allergies or previous medical conditions.

If a soldier is unable to communicate with medical personnel, the soldier's dog tag is used to verify their general personal information. More information about the soldier could aid the soldier's treatment and successful recovery.

Combat support hospitals also face the same problems presented to standard hospitals.

2.5 Child Care Facilities

Many families utilize child care facilities during working hours to provide a safe and active environment for their children. It is difficult for parents to ensure their children are active each day. Also, if the child care facility is not extremely secure, a child could wander out of the facility or be abducted from the facility without the facility's staff knowing. There is no immediate way to detect these situations.

CHAPTER 3

Available Technologies and Market Research

There are many technologies available today that can be used to eliminate the problems seen with current patient monitoring techniques. Several companies and research groups are developing patient monitoring devices to this end. This chapter explores several available technologies and describes some of the patient monitoring solutions already developed.

3.1 Available Technologies Used to Aid in Patient Monitoring

Using the following technologies, an automated patient monitoring system can be designed to eliminate many problems seen with current monitoring techniques. An ideal monitoring system includes some or all of the following technologies.

3.1.1 Measuring Vital Signs

Most basic vital signs can already be measured and converted to digital data using commercially available sensors. The sensors used for these measurements vary in complexity from the pulse rate monitors included with treadmills to a dedicated electrocardiogram machine. Common vital signs charted by healthcare staff include body temperature, pulse rate, blood pressure, and respiratory rate.

3.1.2 Digital Sampling

Once a patient's vital signs have been measured, the measurements need to be converted to digital data for analysis and logging. Some sensors have a built-in ability to convert an analog measurement into a digital signal. Other sensors require an external device to convert these signals. Modern microcontrollers are well suited to transform

these analog signals to digital signals using on-chip peripherals. [D1] Useful examples are analog to digital converters, analog comparators, Inter-Integrated Circuit (I2C) Busses, and Serial Peripheral Interfaces (SPI).

3.1.3 RF Transceiver

Once the microcontroller has converted each analog measurement to its digital equivalent, the data can be transferred from the microcontroller to an external radio frequency (RF) transceiver using any of the microcontroller's on-chip digital peripherals. The RF transceiver facilitates wireless communications between the microcontroller and other RF transceivers within range.

Many RF transceivers perform cyclic redundancy checks (CRC) to ensure that valid data is received. [D2] Also, most RF transceivers have configuration registers that enable users to change settings such as frequency, output power, and data rate. More complicated RF transceivers can also incorporate data encryption techniques to ensure secure transmissions.

3.1.4 Sensor Networks

Many microcontrollers can be linked together using hardwires or using RF transceivers to create a sensor network. This network would have one node that was designated as the network controller. All digitized data from the sensor network is sent to the network controller node for processing and transmission out of the sensor network.

To reduce out-of-network transmissions, the network controller would filter all raw data coming from the sensor network and determine which data should be sent to an external network as an emergency alert or for further processing and logging.

The network controller node is the “brain” of the sensor network. If the network controller node malfunctions, other nodes in the sensor network would have the ability to take over and become the network controller node. This type of fault-tolerant network ensures that the sensor network will continue operating properly even if there is a catastrophic failure.

3.1.5 Off-Patient Data Communication

After the network controller determines which data to send, it will forward the data to an external network or standalone device. There are many ways data can be sent to an external network. Using a wired connection is the simplest medium for data transfers. A wired connection has extremely high data rate possibilities, but limits the mobility of the user.

Alternatives to a wired connection would be using existing wireless technologies. For short range identification and tracking, the use of radio-frequency identification (RFID) can be implemented. RFID technology is usually used for transferring static data such as a serial number or product details. A different short range solution that offers more flexibility and higher data transfer rates would be to incorporate Bluetooth technology into the sensor network controller. Bluetooth is a wireless protocol for exchanging data over short distances. Bluetooth is commonly used to create personal area networks (PANs) among fixed mobile devices.

Bluetooth technology could also be used to create the sensor network itself. Many other short range wireless technologies are available including custom network protocols.

A longer range wireless solution would be to utilize the current 802.11 wireless local area network (WLAN) hardware and protocols. This network protocol, along with the hardware to interface to such a network, is ubiquitous.

For even more range a cellular network can be used. When local cellular infrastructure is not present, radio stations and satellite communications can be utilized.

Short Range Wireless (< 300 feet)	Bluetooth, Low Energy Bluetooth, Wi-Fi/802.11, Infrared , Zigbee
Medium Range Wireless (< 45 miles)	WiMax/4G, 3G, PCS, GSM, CDMA, EDGE
Long Range Wireless (> 45 miles)	GSPRS/Paging Systems, Satellite-Low-Earth Orbit, High-Earth Orbit, Licensed Radio Stations, Digital UHF/VHF Stations

Table 1: Wireless Standards by Range

All of these technologies could be incorporated into the same sensor network controller. Instead of including hardware for all of these different communication methods, the sensor network controller would incorporate a reconfigurable antenna [1] in conjunction with cognitive radios [2] to adjust itself to function with whichever network is available. This would ensure the network controller's size is minimized and increases the chance that the sensor network would have coverage at all times.

Another benefit of using wireless technologies is that as the number of sensor networks increase, each individual sensor network looks like a larger node in the external network and the effective number of available nodes in the external network increases. These larger external network nodes can work together or "cooperate" to transfer data between outlying nodes and a central monitoring computer when an outlying node doesn't have the ability to communicate with the central monitoring computer directly. In this situation, data transfers between the central monitoring computer and "out-of-range"

network nodes are relayed through “in-range” network nodes to complete the network communication path.

3.1.6 Wireless Access Point

A wireless access point (WAP) receives data from each sensor network controller via one of the communication methods described in the last section and transfers this data to a central monitoring computer (CMC) through a switching network. Wireless access points can be custom designed devices or standard devices such as WLAN routers or cellular towers. The use of wireless access points enables the coverage area of an individual system to grow with the addition of every wireless access point.

Tethering is a method that can be used by a sensor network controller to transfer data between itself and a CMC. Creating an ad-hoc Bluetooth network connection between a sensor network controller and a cell phone would allow the sensor network controller to use every cellular tower as a wireless access point for data transfers. This method of connectivity uses existing cellular networks and infrastructure to transfer bidirectional data between a standalone sensor network and any server on the internet.

Utilizing the cellular networks increases the overall effective coverage area of each standalone sensor network. There are security concerns when utilizing cellular networks. Patient data and medical records are mainly stored on local networks in medical facilities. When a patient’s medical record is transmitted through a public cellular network there is a possibility that the information can be intercepted by malicious parties. Also, the reliability of the cellular network comes into question when medical information is involved. Cell phones routinely experience dropped calls, but a loss of connection when dealing with a medical situation is unacceptable.

3.1.7 Central Monitoring Computer

A central monitoring computer (CMC) will be used as the core element of the proposed sensor monitoring system. The CMC is responsible for gathering data sent by all sensors in the system through the wireless access points. It logs all data gathered into a central database. Normally, a polling scheme is implemented to gather this data. The CMC has the ability to communicate with every wireless access point in the system. Bidirectional data transfers between the CMC and any standalone sensor network is facilitated through these wireless access points.

The CMC stores all incoming data to a database for future use. The database is stored on a central server and is available through various means.

The CMC will also perform calculations on incoming data to determine what tasks it must perform. These tasks could include requesting additional data from or sending a command to a specific sensor network. The CMC could also send an alert to an operator that there is a problem with a specific sensor network.

As the size of the entire system increases the demands put on the CMC will also increase. Additional processors and increased processor speeds may be required to accommodate system size increases. Storage capacities and data access times may also need to be improved. Also, each improvement in CMC performance will increase the power requirements of the system.

3.1.8 Display Sensor Data

While the CMC is gathering and storing all of the sensor data in the system, a graphical user interface (GUI) can be used to access the sensor data and display it to an operator in real-time. All decoded sensor data can be displayed along with limit warnings and alarms for each reading. If sensor data exceeds predefined safety alert or alarm limits,

operators are notified. Alert and alarm notifications can be as simple as flashing a light or sounding a buzzer in the GUI, to placing a phone call or sending a text message to an operator to notify them of an emergency situation. Data can also be updated in real-time to websites and displayed to operators anywhere in the world.

Another method for displaying real-time sensor information would be to use a hand-held device to request information directly from a specific sensor network controller. The hand-held device would create an ad-hoc network between itself and the sensor network controller. The hand-held device would also be able to change the sensor network's settings including custom sensor monitoring limits. Other features of the hand-held device could include logging real-time sensor data, printing sensor summaries with operator comments, and the ability for the operator to send sensor information to a main database or internet website directly for review.

3.2 Market Research

Several companies are developing patient monitoring devices using some of the technologies listed earlier in this chapter, though few of these products have seen mainstream use. This lack of product integration may be due to a combination of several factors, including cost, risk, reliability, and the fact that some doctors may be technophobic.

Chapter four proposes a remote monitoring architecture and describes a prototype implementation called Guardian. Many of the commercial products under development implement features included in the Guardian system. Guardian excels in combining these features into a complete, scalable system.

Guardian offers several advantages over commercial systems. The Guardian system is intended to be fully integrated into a facility, enables real-time monitoring, and simultaneously displays data from multiple patients. Each wrist module can be programmed with specific patient vital sign ranges, allowing medical personnel to set emergency alert levels for each patient. These parameters can be changed remotely at any time using the Guardian's graphical user interface (GUI). The Guardian system is fully bi-directional, allowing data to be transferred not only from the patient, but to the patient as well. This allows for features such as voice communication between any patient in the system and medical personnel.

Instead of simply transmitting vital signs data, the intelligent wrist modules (WMs) included in the Guardian system can preprocess data before sending it to the CMC. Each WM is assigned to a specific patient and the data that it transmits is tagged

with a unique patient identifier, allowing multiple WMs to operate with proper data association in the same facility.

Below are a few examples of current products under development.

Medic4all – aLsis
http://en.medic4all.it

Medic4all is a Switzerland based company that researches, designs, and develops many different telemedicine products. Most of their products are geared towards the consumer market.

Product Features:

- Small wearable device
- One button interface
- Transmits emergency alert wirelessly to wireless access point

Product Limitations:

- Unidirectional communication
- No patient data transferred with button press
- No severity indicator
- No unique identifier, single patient use

Product Use:

- Home healthcare

Medic4all - Watch Me
http://en.medic4all.it

Product Features:

- Single lead ECG sensor
- Heart Rate sensor
- Heart Rhythm Regularity monitor
- Stores vital signs data locally
- Wirelessly transmits data to a wireless access point

Product Limitations:

- Unidirectional communication
- No unique identifier, single patient use
- No emergency alert button
- No patient specific measurement limits

Product Use:

- Home healthcare

Medic4all - MedicGate
http://en.medic4all.it

Product Features:

- Wireless access point
- Enables communication between wireless monitoring devices and the central monitoring center

- Data communication to central monitoring center via telephone lines, internet or cellular network

Product Limitations:

- Unidirectional communication from patient monitoring devices to monitoring center

Product Use:

- Home healthcare
- Medical facility

CSIRO Information and Communication Technologies (ICT) Center – ECHONET
http://research.ict.csiro.au

CSIRO is an Australia based company that focuses its ICT research in the commercial medical field.

Product Features:

- Mobile
- Broadband communication
- Real-time video
- Real-time measurements
- Virtually brings a remote specialist to the point of care
- Enables image-based diagnostic procedures such as echocardiography

Product Limitations:

- Requires medical personnel to operate equipment
- Only dedicated measurements available, not easily expandable
- Not automated
- Large size

Product Use:

- Medical facilities

MAQUET - AV Conference Premium
http://www.maquet.com

MAQUET is a Swedish based group of companies with three specialty divisions for Critical Care, Cardiovascular, and Surgical Workplaces. This company has developed a telemedicine solution for its surgical workplace division.

Product Features:

- Records vital signs information
- Records audio and video
- Broadcasts all data live to remote facilities
- Bidirectional audio communication between facilities
- Touch screen user interfaces

Product Limitations:

- Requires medical personnel to operate equipment
- Not automated
- No mobility
- Large size

Product Use:

- Medical operating rooms

NASA - Telemedicine Instrumentation Pack
http://lsda.jsc.nasa.gov

NASA is an agency of the United States government. They develop and use telemedicine systems to monitor astronauts during training and throughout space missions.

Product Features:

- Vital signs monitoring
- Record heartbeat, electrocardiographic, and pulse-oximeter data
- Transmit data to remote locations in real-time
- Physical examination imaging capabilities
- Screens for signs of airborne mucous membrane irritation and injuries

- Accurately assess skin responses for immune function testing

Product Limitations:

- Requires medical personnel to operate equipment
- Large size
- Set measurement capabilities
- No patient specific limit settings

Product Use:

- NASA internal use

MobiHealth – Mobile
http://www.mobihealth.com

Mobihealth is Dutch company that develops mobile sensors, on-line software, and software designed specifically for use by healthcare professionals.

Product Features:

- Wireless sensors worn on patient's body
- Multi-lead ECG sensor
- Multi-channel EMG sensor
- Pulse rate sensor
- Oxygen saturation (SpO2) sensor
- Respiration monitor
- Core/skin temperature sensor
- 9-hours battery life during real-time measurements and transmission

- Patient feedback using vibration, audio, and graph display
- Bluetooth

Product Limitations:

- Unidirectional communication
- No patient specific measurement limits
- Requires external device to function
- Requires medical professionals to interpret measurement data

Product Use:

- Medical facility
- Home healthcare

SensWear - Body Monitoring Solution
http://www.sensewear.com

SensWear is a U.S. base company that develops wearable body monitoring systems that are designed to help people lose weight, improve performance and live healthier lifestyles.

Product Features:

- Wireless sensor unit worn on patient's arm
- Real-time data displayed on patient's wrist display
- Monitors energy expenditure
- Monitors activity level

- Monitors sleep efficiency

Product Limitations:

- No data transmission to external networks
- No vital signs measurements
- Used primarily for motion measurements

Product Use:

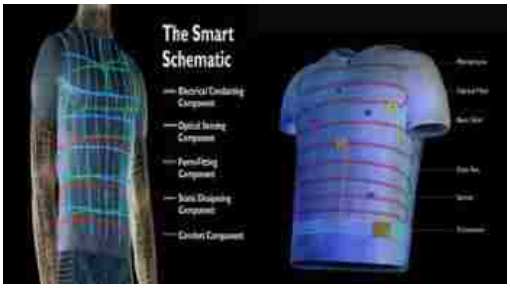
- Home healthcare

Sensatex – SmartShirt
http://www.sensatex.com

Sensatex, Inc. is a life science technology company focused on the development of Smart Textile Systems. They have developed the Sensatex SmartShirt which is a wearable Smart Textile unisex T-shirt designed to acquire physiological information and movement data from the human body.

Product Features:

- Wearable T-shirt
- Conductive fiber/sensor system
- Real-time sensor measurements and transmission



- Heart rate sensor
- Respiration sensor
- Body temperature sensor

- Activity level monitoring
- Wireless communication via Bluetooth or ZigBee

Product Limitations:

- Unidirectional communication
- No patient specific measurement limits
- Set measurement capabilities
- Comfort concerns

Product Use:

- Medical facility
- Home healthcare

Cardiomems Inc. - EndoSure Wireless AAA Pressure Sensor
http://www.cardiomems.com

Cardiomems Inc. is a U.S. based company that develops small implantable sensors designed to improve the management of severe chronic cardiovascular diseases such as heart failure and aneurysms.

Product Features:

- Permanently implanted into the cardiovascular system
- Monitors cardiac information
- Blood pressure measurement
- Heart rate measurement
- Small size

- Durable
- No wires or batteries
- Sensors transmit real-time data to an external electronics module using inductively coupled power and data

Product Limitations:

- Data only viewable and recordable while being actively monitored with an external device in a medical facility
- Requires surgical implantation
- No emergency notification to medical personnel when patient is outside medical facility
- Set measurement capabilities

Product Use:

- Medical facilities

Integrated Sensing Systems (ISSYS) -Wireless, Batteryless Cardiac Sensor
http://www.mems-issys.com

ISSYS is a U.S. based company that develops implantable intelligent MEMS-sensor-based systems to enhance the quality of medical treatment. Their MEMS cardiac sensor is

used for patients diagnosed with Congestive Heart Failure (CHF).

Product Features:

- Minimally invasive Implanted sensor (catheter delivered)
- Long-term operation (>10 years)
- No wires or batteries
- Real-time cardiac pressure measurements
- Monitor measurements at home
- Transfer data to physicians via telephone line or internet
- Collect information from sensor using hand-held inductively coupled reader

Product Limitations:

- Data only viewable and recordable while being actively monitored with an external device
- Requires surgical implantation
- No emergency notification to medical personnel when patient is outside medical facility
- Set measurement capabilities

Product Use:

- Medical facility
- Home healthcare

CHAPTER 4

Proposed Remote Monitoring System Architecture and Prototype Implementation

This chapter will outline the details of a proposed remote monitoring system architecture. It will then describe the operation of the overall proposed system and each system component. A prototype implementation of an automated patient monitoring system will be described. All aspects of the prototype system will be addressed including overall system design, hardware implementation, software implementation, data polling methods, and communication data structures. Operation of the Graphical User Interface (GUI) will also be explained.

4.1 Proposed Remote Monitoring System Architecture

The proposed remote monitoring system consists of sensor units worn by each patient that monitor real-time vital signs information and wirelessly transmit it through wireless access points to a central monitoring computer (CMC) for processing.

The proposed system architecture is scalable in both the number of patient's and number of monitoring areas employed. Each monitoring area can support multiple sensor units.

Medical personnel monitor all patients simultaneously on the CMC and are notified of any emergency events that occur. Using this type of automated monitoring allows a minimal number of medical personnel to comprehensively address the needs of large number of patients.

A block diagram of the proposed system is shown in figure 1 below.

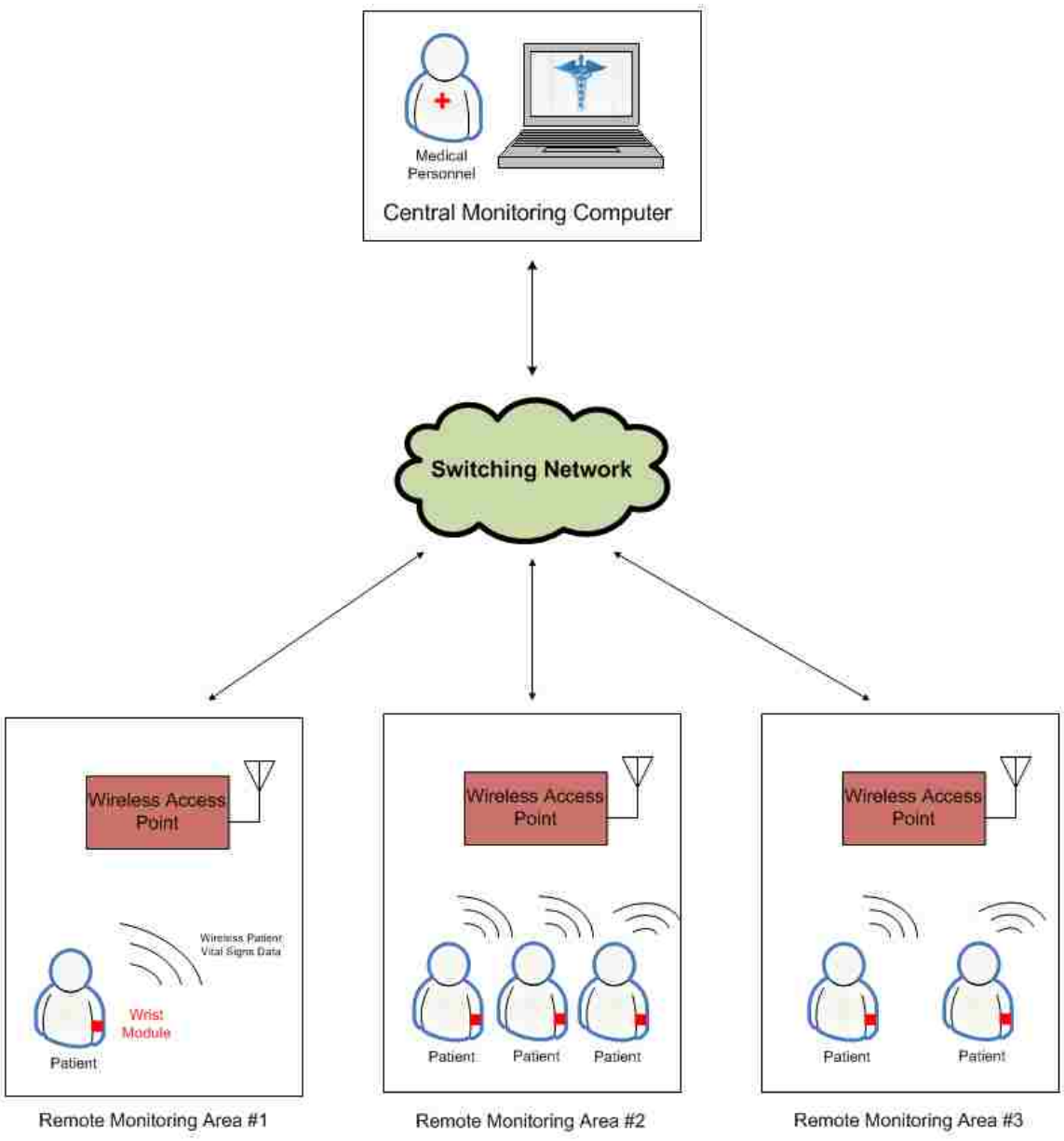


Figure 1: Proposed System Block Diagram

4.1.1 Central Monitoring Computer

The central monitoring computer (CMC) will be in control of the entire system. The CMC will communicate through a switching network with wireless access points to poll each patient's sensor unit requesting the patient's current vital sign measurements. A graphical user interface (GUI) will be running on the CMC. The GUI will display each

patient's real-time vital signs, location information, and emergency events to medical personnel operating the system.

The CMC will also log and maintain all patient information in a database for future use.

4.1.2 Wireless Access Points

Range limitations, output power limits, unknown monitoring area layouts, scalability, and many other factors make it very difficult for the wearable sensor units to communicate directly with the central monitoring computer. A solution is to incorporate wireless access points (WAPs) into the system. The WAPs relay commands and data between the CMC to the wearable sensor units. They allow the central monitoring computer to monitor many locations without having to communicate with the wearable sensor units directly. Also, a patient's location can be determined using each WAP's coverage area. Multiple WAPs are connected to the CMC via a switching network. The complexity of the switching network will vary based on the requirements of the overall system.

4.1.3 Wearable Patient Sensor Unit

Each patient will wear a mobile wireless unit that monitors and transmits their vital signs and other patient data back to the central monitoring computer. This wireless unit will also receive commands from the central monitoring computer and respond appropriately. In the proposed system these wearable sensor units are worn on each patient's wrist and called Wrist Modules (WMs).

WMs will monitor each patient's standard vital signs. The WMs will also be equipped with an array of patient specific sensors that can be easily added or removed

from the WM depending on each patient's medical requirements. The reconfigurable nature of the WMs allows the proposed system to easily be used in other monitoring situations.

The proposed system enables bidirectional variable data rate communication between WMs and the CMC. This enables the CMC to reprogram WMs while they are being worn by patients. New measurement rates, data rates, or emergency limits can be reconfigured based on the patient's needs. Using these capabilities also allows for a bidirectional audio communications link between WMs and the CMC. This feature can be used in many monitoring conditions to quickly evaluate the severity of a situation.

Each WM will have the ability to communicate directly with a handheld device. The handheld device has the same functionality as the CMC. It is used by medical personnel during exams or consultations to display a patient's real-time vital signs data. Also, if necessary, medical personnel can change a patient's WM configuration to emphasize specific measurements.

4.2 Prototype Hardware

A prototype automated patient monitoring system was constructed to demonstrate some general telemedicine system design concepts described in chapter 3. The prototype system is designed for use in a medical facility environment, but could be applied to many other monitoring situations. The prototype consists of a central monitoring computer (CMC) that polls multiple patient's wrist modules (WMs) requesting current measurements from each. The CMC communicates with each WM through a simple

switching network and two wireless access points (WAPs). Three wrist modules were created that monitor basic vital signs.

4.2.1 Central Monitoring Computer

The central monitoring computer (CMC) is a stand-alone computer that runs a graphical user interface (GUI) named Guardian. The CMC must be running a Windows operating system and have a LabVIEW 8.6 runtime engine installed in order to run the Guardian software. The only other requirement for the CMC is that it must have at least one USB port available for connection with the switching network. The connection from the CMC to the switching network is made using a USB to RS-232 converter.

4.2.2 Switching Network

The prototype system was implemented with two wireless access points (WAPs) so a complicated switching network was not required. The UART-TX signals coming from each WAP are active low. An analog OR gate was constructed using two diodes and a resistor as seen in figure 2 below. The OR gate allows each WAP UART to communicate directly to the CMU without creating bus contention between WAPs.

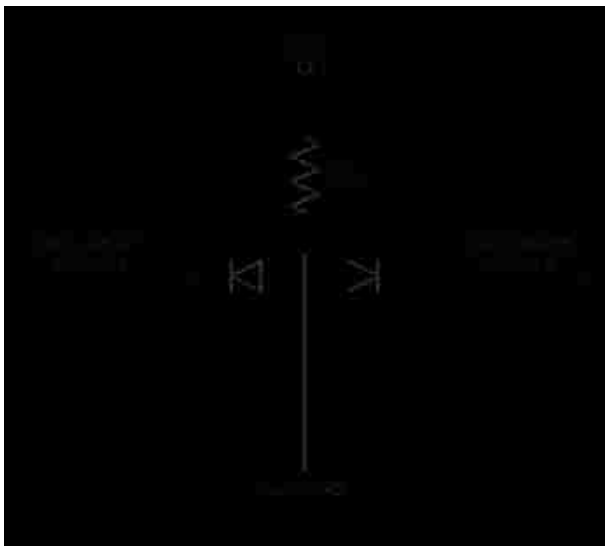


Figure 2: Switching Network Schematic

4.2.3 Wireless Access Point

The wireless access point (WAP) is comprised of a PIC microcontroller [D1] and a 2.4GHz wireless transceiver [D2]. The PIC's UART enables bidirectional data transfers between the CMC and the WAP via the switching network. Five general purpose input/output (GPIO) pins form the PIC are connected to the wireless transceiver. These GPIO lines control the transceiver and enable bidirectional data transfers between the WAP and any wrist module (WM) that is within the WAP's coverage area. A simple block diagram of the WAP can be seen in figure 3 below.

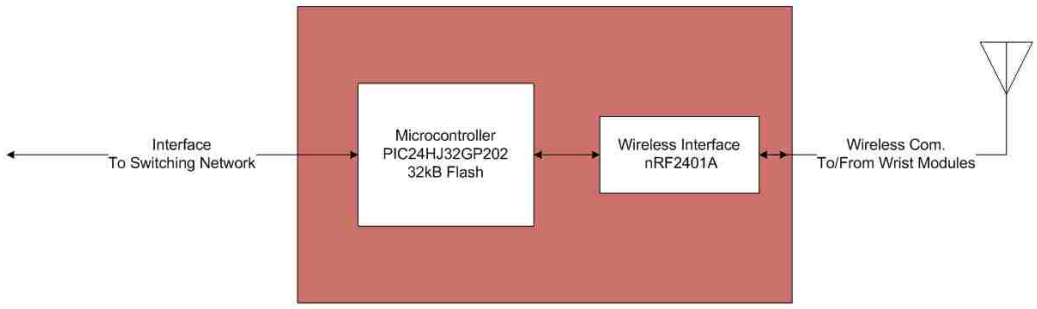


Figure 3: Block Diagram of a Wireless Access Point (WAP)

Each WAP also has two light emitting diode (LED) indicators that show status information about the WAP's operation. The first LED is yellow and indicates an RF data transfer between the WAP and a WM. The second LED is green and toggles between on and off states; each transition indicates the WAP received a valid response from the WM with which it was trying to communicate.

A schematic of the WAP can be seen in figure 4 below.

4.2.3.1 WAP Schematic

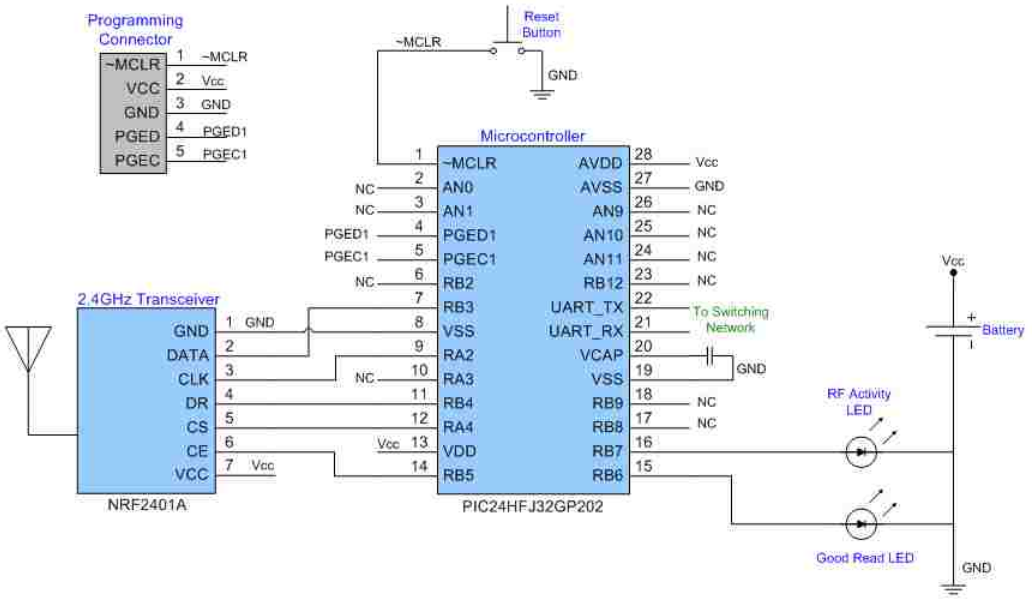


Figure 4: Schematic of Wireless Access Point (WAP)

The schematic shown in figure 4 depicts the connections between the WAP's microcontroller and its peripherals. The microcontroller is connected to a 2.4GHz transceiver, a programming connector, and two LEDs. The microcontroller's UART transmit and receive lines are connected directly to the switching network.

4.2.3.2 Transceiver

The transceiver [D2] that was chosen for the prototype system has the following specifications.

Manufacturer – Nordic Semiconductor
Transceiver Type – nRF2401A
Frequency – 2.4 GHz – 2.524 GHz (ISM Band)
Channels – 125
Modulation – GFSK
Output Power – -20dBm to 0dBm (Prototype uses **0dBm**)
Data Rate – 1Mbps or 250kbps (Prototype uses **250kbps**)
Range – 50ft indoors
Address and CRC computation (Prototype uses **8-bit address** and **16-bit CRC**)
Incorporates ShockBurst technology

4.2.3.2.1 ShockBurst

The ShockBurst technology [D2] in the transceiver uses an on-chip FIFO to clock in data at a low data rate and transmit the data at a very high rate. This enables the transceiver to reduce operational power consumption.

When the transceiver is configured in ShockBurst mode, TX or RX operations are conducted in the following manner.

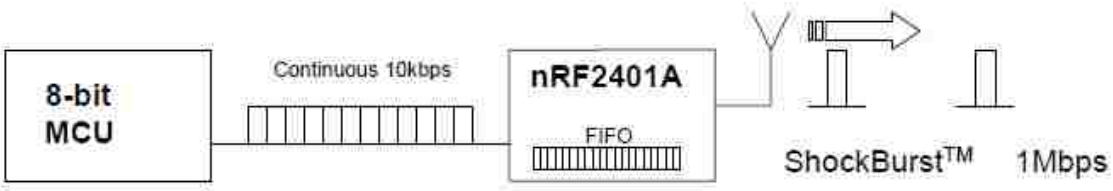


Figure 5: Clocking in Data with MCU and Sending with ShockBurst

The benefits of using the ShockBurst mode on current consumption can be seen in figure 6 below.

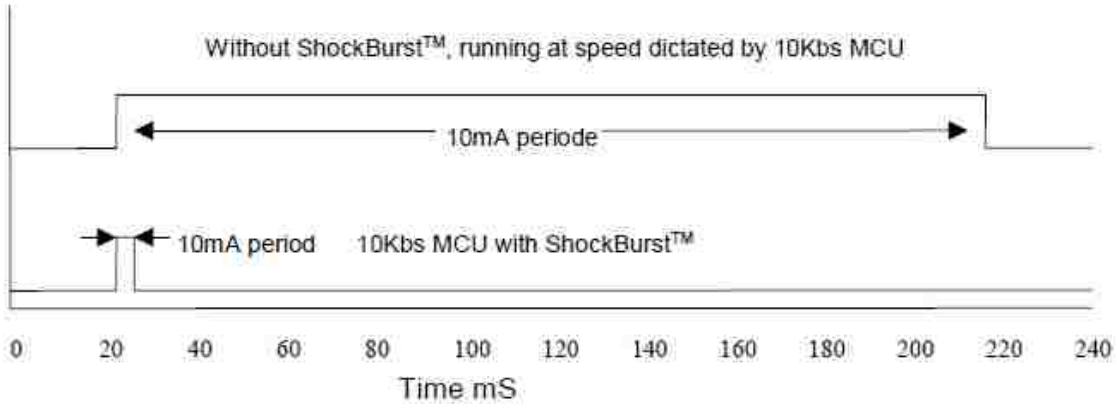


Figure 6: Current Consumption with & without ShockBurst

4.2.3.2.2 Wireless Data Package Description

The transceiver uses the following data package structure for ShockBurst communication.

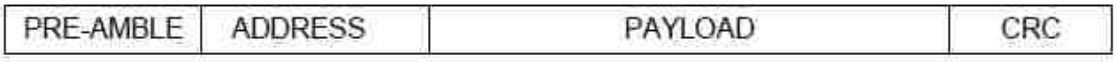


Figure 7: ShockBurst Data Package Diagram

This data package is divided into 4 sections. Each of these sections is described in table 2 below.

1. Preamble	• Preamble is 8 bits in length and is automatically added to the data packet in ShockBurst mode. (Used in prototype system)
2. Address	• 8 to 40 bits (Prototype uses 8-bit mode)
3. Payload	• Data to be transmitted • Payload size is 256 bits minus the following: Address: 8 to 40 bits (Prototype uses 8-bit), CRC 8 or 16 bits (Prototype uses 16-bit) • Prototype overall payload size = 256 – 8 – 16 = 232 bits or **29 Bytes**
4. CRC	• 8 or 16 bits • Optional (Prototype uses 16-bit mode CRC)

Table 2: ShockBurst Data Package Description

The prototype system simply loads data to be transmitted into the payload section of the data package and the transceiver automatically adds the remaining sections before transmission. The maximum payload size using an 8-bit address and a 16-bit CRC is 232 bits or 29 bytes.

When the transceiver receives a data package, it verifies all data package information and sends only the payload data to the microcontroller for processing.

4.2.3.2.3 Channel Allocations

The transceiver can be programmed to communicate using one of 125 different RF channels. Channels above 83 can only be utilized in certain territories (ex: Japan). The prototype system therefore only utilizes channels through 83.

RF transceiver channels are defined as:

*RF_Channel = 2400MHz + (Chan_Num * 1.0MHz)*

So, for the transceiver to communicate on channel 30, the RF_Channel is set to 2430MHz.

Using the above equation, the prototype system defines three channels that it uses for all data communication.

Prototype Channel Allocations		
Channel Name	**Channel Number**	**Channel Frequency**
Beacon	1	2401MHz
Broadcast	30	2430MHz
Response	32	2432MHz

Table 3: Prototype Channel Allocations

4.2.3.3 WAP Picture and Description

A picture and description of a fully assembled WAP can be seen in figure 8 below.

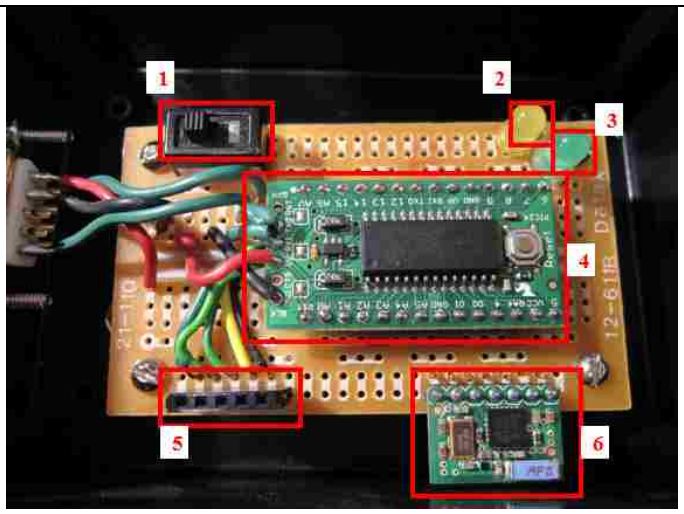


Figure 8: Picture of Fully Assembled Wireless Access Point (WAP)

Description

1 – On/Off power switch
Provides power for the entire WAP board.

2 – RF activity LED
On when the RF transceiver is transmitting or receiving data to/from any WM.

3 – Good read LED (Toggle)
Toggles between on/off states each time a valid response is received from any WM.

4 – PIC Microcontroller [D1]
Microcontroller running at 40MIPS that controls all functions of the WAP.

5 – Programming connector
Allows in-circuit serial programming (ICSP) of the PIC microcontroller.

6 – 2.4GHz transceiver [D2]
Enables RF data transfers between the WAP and any WM within the WAP's coverage area.

4.2.4 Wrist Module

The wrist module (WM) is comprised of a PIC microcontroller [D1], a 2.4GHz wireless transceiver [D2], an accelerometer [D3], a temperature sensor [D4], and an input button. Five general purpose input/output (GPIO) pins form the PIC are connected to the wireless transceiver. These GPIO lines control the transceiver and enable bidirectional data transfers between the WM and any wrist wireless access point (WAP) that is within the WM's RF range.

Each WM is setup to read the value of three 12-bit analog inputs:

- Accelerometer to monitor emergency fall events and WM orientation
- Temperature sensor to monitor patient's body temperature
- Battery voltage to ensure proper operating conditions

A simple block diagram of the WM can be seen in figure 9 below.

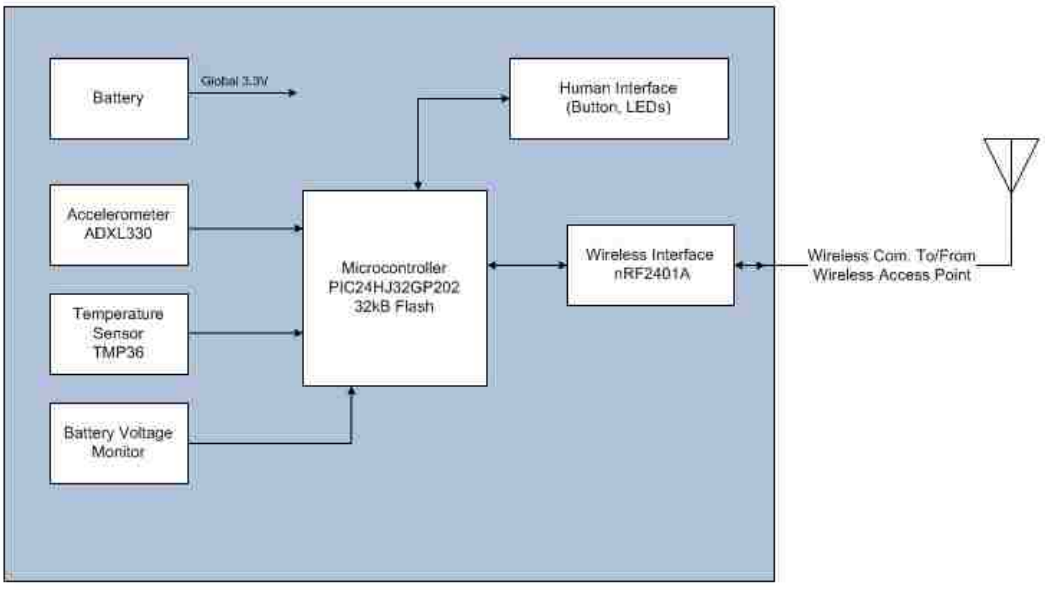


Figure 9: Block Diagram of a Wrist Module (WM)

Each WM also has an emergency event reset button and two light emitting diode indicators that provide the patient with status information. The first LED is yellow and indicates an emergency fall event has occurred. This LED will flash repeatedly when activated. The patient can reset the emergency fall event by pressing the emergency event reset button. The second LED is green and indicates that the WM is within a WAP's coverage area. This LED will remain on while within any WAP coverage area otherwise it will turn off.

A schematic of the WM can be seen in figure 10 below.

4.2.4.1 WM Schematic

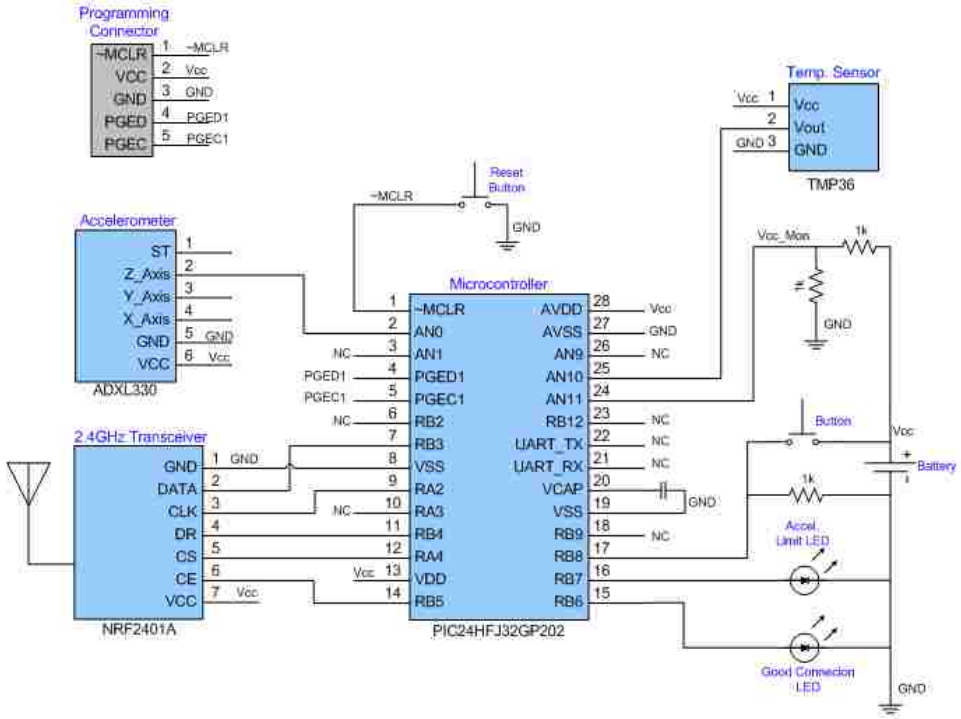


Figure 10: Schematic of Wrist Module (WM)

The schematic shown in figure 10 depicts the connections between the WM's microcontroller and its peripherals. The microcontroller is connected to a 2.4GHz

transceiver, programming connector, accelerometer, temperature sensor, voltage monitor, emergency event reset button, and two LEDs.

4.2.4.2 Transceiver

The same transceiver [D2] is used for the WM as the WAP.

4.2.4.3 WM Picture and Description

A picture and description of a fully assembled WM can be seen in figure 11 below.

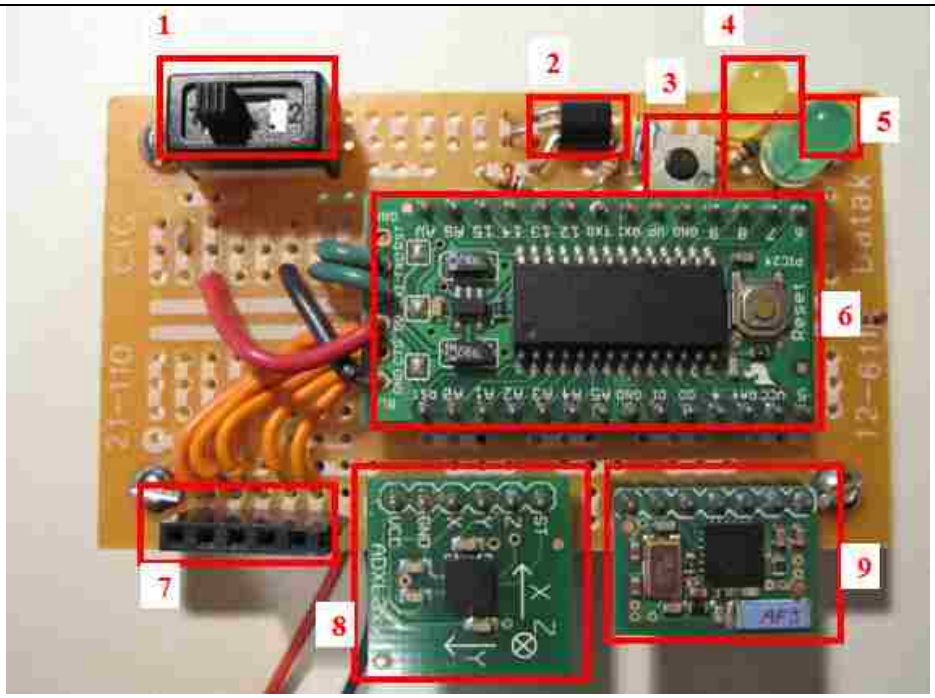


Figure 11: Picture of Fully Assembled Wrist Module (WM)

Description

1 – On/Off power switch
Provides power for the entire WAP board.

2 – Temperature sensor [D4]
Measures patient's body temperature in degrees Fahrenheit.

3 – Emergency event reset button
Resets emergency status of the WM. Demonstrates basic user input function.

4 - Emergency event indicator LED
Repeatedly flashes when accelerometer limit of +/- 2g is exceeded

5 - Good connection LED
On when WM within WAP coverage area, off when WM out of WAP coverage area
6 – PIC Microcontroller [D1]
Microcontroller running at 40MIPS that controls all functions of the WM.
7 – Programming connector
Allows in-circuit serial programming (ICSP) of the PIC microcontroller.
8 - +/- 3g 3-axis accelerometer [D3]
Only the z-axis of the accelerometer is used as both a "Fall Sensor" and to determine the WM orientation. The z-axis is normal to the WM board's surface and measures gravity to determine board orientation.
9 – 2.4GHz transceiver [D2]
Enables RF data transfers between the WM and any WAPs within range.

4.3 Prototype Software

There are three different pieces of software that were written for the prototype system: Guardian software package, Embedded WAP code, and Embedded WM code. The Guardian Automated Patient Monitoring System is comprised of all three pieces of software operating together.

4.3.1 Guardian Software Package

The Guardian software package (GSP) was written in LabVIEW 8.6 and is comprised of a Graphical User Interface (GUI), Database Manager, and Communications Engine. The GSP runs on the CMC and allows operators to view real-time patient data, add, edit, and remove patients from the system, view and edit patient database files, and send commands directly to a specific patient to reset an emergency event status.

4.3.1.1 Guardian GUI

The Guardian GUI is the front end control and monitoring interface used by system operators. The GUI includes many different displays that allow operators to add, edit, and view patient information in the system. When the GUI is first launched, the **Home Screen** is displayed. The **Home Screen** displays real-time vital signs information

for every patient in the system and provides control buttons to allow operators to maneuver easily through the program. The **Home Screen** can be seen in figure 12 below.

Figure 12: Guardian GUI - Home Screen

The **Home Screen** includes the following components:

- Current Time
  - Displays the current CMC time in a 24-hour **HH:MM:SS** format.
  - Used for log file creation and time tagging of log entries.
- User Help area

- Displays help information for any control or display on the **Home Screen** when an operator points to the control or display.

- Real-Time Patient Display area
  - Displays the following real-time vital signs information for every patient in the system:
    - Wrist Module Number
      - Assigned to each patient's WM when the patient is added to the system.
    - Location
      - Indicates which WAP coverage area the patient is in currently. (WAP-#)
    - First Name
    - Last Name
    - Body Temperature (ºF)
    - Accelerometer Value (g)
      - Used for emergency fall event limits and WM orientation.
    - Vcc Monitor
      - Monitors the battery voltage of the WM in Volts.
    - Accelerometer Limit
      - Indicates emergency fall event status. (0 – No event, 1 – Event detected)
    - Button State
      - Indicates patient's input button status. (0 – Button not pressed, 1 – Button pressed)
    - Last Updated

- Indicates when the patient's vital signs information was last updated. (24-hour HH:MM:SS)

▪ Primary Nurse

- Operator can sort all display data at any time by clicking on the column header of the desired sort column. The first click sorts data in ascending order, second click sorts data in descending order.
- Automatically sorts patient information based on emergency event status or out-of-range status. Patient data indicating an emergency event or an out-of-range event will be moved to the top of the display. Patient data indicating an emergency event will have priority over an out-of-range event and will therefore be placed above the out-of-range patient data.
- If there is an emergency event indicated by any patient data, an emergency event window will appear allowing the operator to clear the emergency event for the patient. The details of the emergency event will be added to the patient's real-time log file. The emergency event window can be seen in figure 13 below.

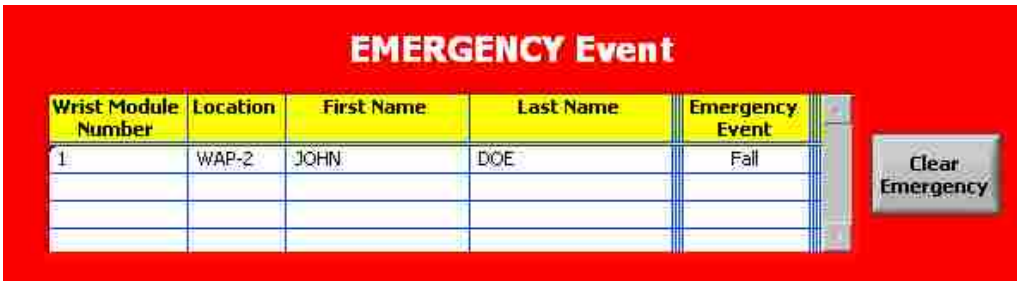


Figure 13: Emergency Event Window

- Operator Control Buttons
    - Allows operators to perform program functions.
- Program Status Indicators
    - Displays system communication and logging status information.

Each operator control button on the **Home Screen** performs a specific function. The outcome of pressing each of these buttons is described below.

Home Screen >> Add Patient Button

Figure 14: Add Patient Screen

Pressing the **Add Patient** button opens the screen shown in figure 14 above. This screen allows an operator to add patients to the Guardian patient database. The following information is entered for each new patient:

- Wrist Module number
  - Uniquely assigned to each new patient
- First Name
- Last Name

- Patient Picture
- Current Medications
- Primary Nurse
- Doctor Notes

The operator has two options after new patient information has been entered.

Add Patient Screen >> Add Patient

- Adds all of the new patient's information to the patient database
- Creates a directory structure for the new patient
- Create new time and date stamped log file for the new patient

Add Patient Screen >> Cancel

- Doesn't add patient information to database.
- Closes the **Add Patient Screen** and returns to the **Home Screen**.

Home Screen >> Edit Patient Button

Figure 15: Edit Patient Screen

Pressing the **Edit Patient** button opens the screen shown in figure 15 above. This screen allows an operator to change any of the information for an individual patient that was entered when the patient was added to the database.

The operator has two options after the patient information has been changed.

Edit Patient Screen >> Save Changes

- Changes patient's modified information in the patient database.

- Creates a new directory structure and time and date stamped log file for the patient if the patient's first or last name is changed.

Edit Patient Screen >> Cancel

- Doesn't change patient information in database information.
- Closes the **Edit Patient Screen** and returns to the **Home Screen**.

Home Screen >> Remove Patient

After selecting a patient from the real-time display on the **Home Screen**, pressing the **Remove Patient** button will remove the selected patient from the system. The patient's real-time data will no longer be requested, displayed, or logged in the system. All of the patient's previously logged vital signs data will not be removed from the GSP database.

Home Screen >> Patient Data

Figure 16: Patient Data Screen

Pressing the **Patient Data** button opens the screen shown in figure 16 above. This screen allows an operator to view real-time trending data for individual patients. The patient data screen includes the following data:

- Wrist Module Number
- First Name
- Last Name
- Patient Picture
- Body Temperature (ºF)

- Plot of real-time trending data
- Numeric display of current body temperature
- Numeric display of average body temperature for trending plot

• Accelerometer Value (g)

- Numeric display of accelerometer value
- Three LED indicators displaying WM surface orientation (Up, Side, Down)

• Vcc Monitor

- Numeric display of battery voltage of the WM

• Accelerometer Limit

- LED indicator displaying emergency fall event status

• Button State

- LED indicator displaying patient's input button status.

The operator has three options after viewing a patient's real-time trending data.

Patient Data Screen >> More Info.

- Pressing the **More Info.** button on the **Patient Data Screen** opens the following screen.

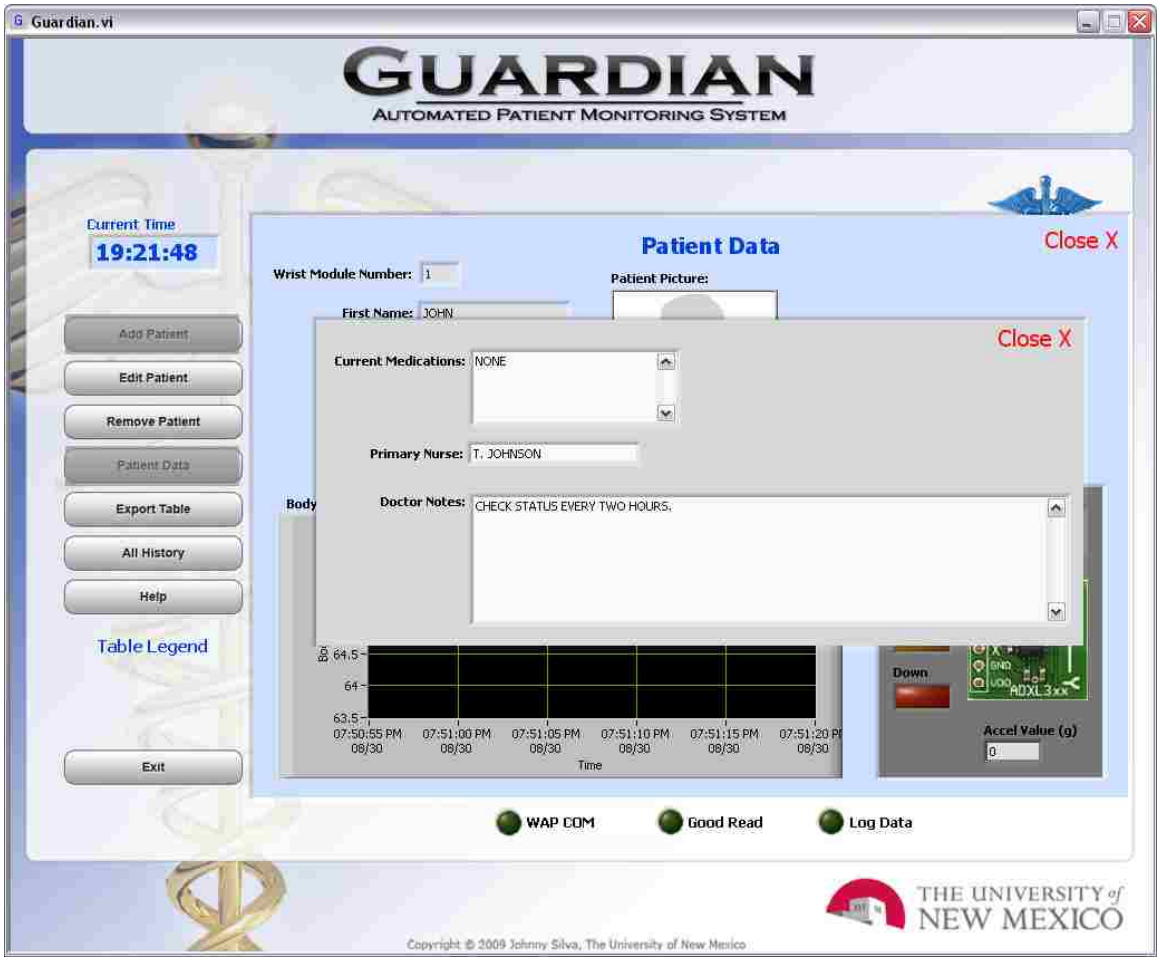


Figure 17: Patient Data Screen – More Info.

The **More Info. Screen** displays the following pieces of patient information.

- Current Medications
- Primary Nurse
- Doctor Notes

The operator can return to the **Patient Data Screen** by pressing the red **Close X** button at the top right of the **More Info. Screen**.

Patient Data >> History

- Pressing the **History** button on the **Patient Data Screen** opens the following screen.

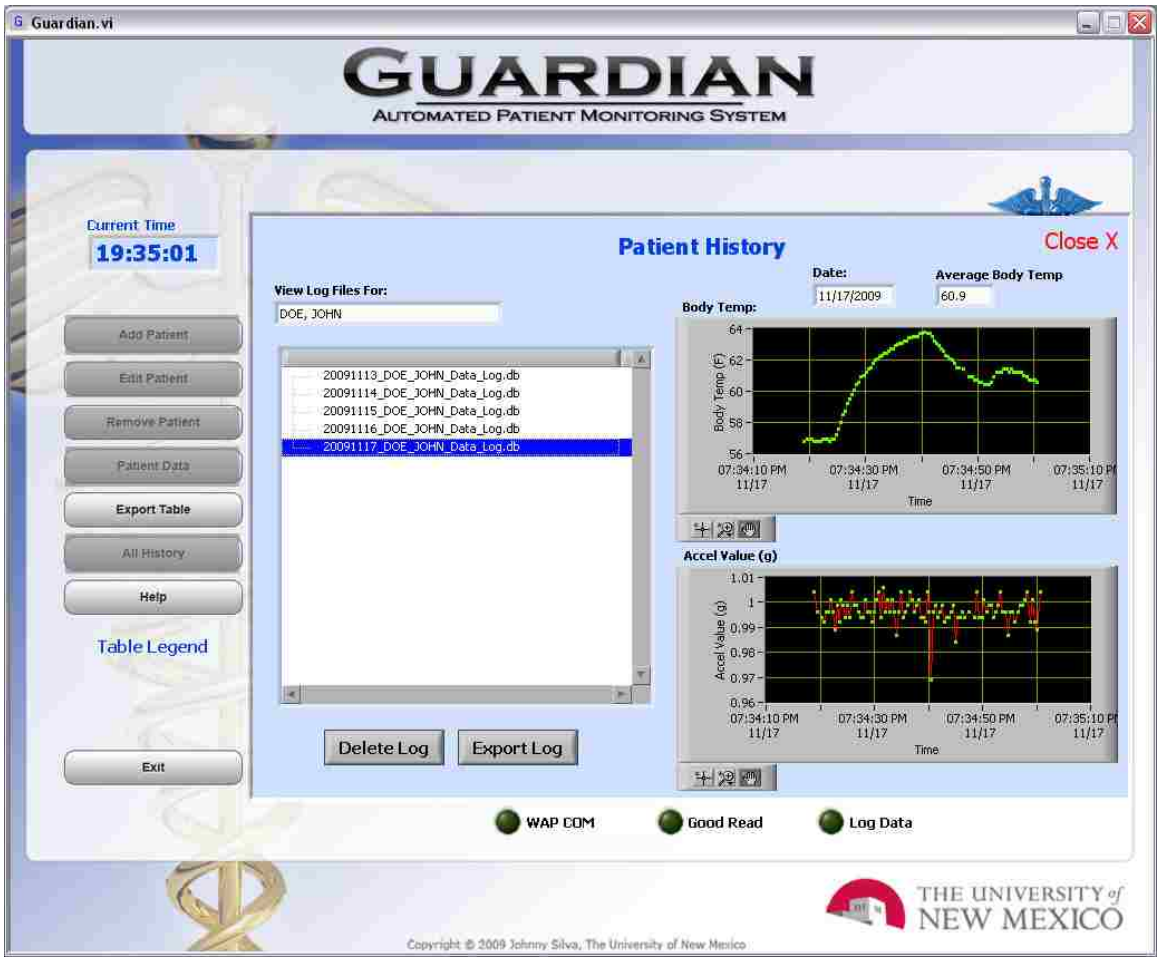


Figure 18: Patient History Screen

The **Patient History Screen** displays the following pieces of patient information.

- Patient's name (Last, First)
- List of all log files for current patient
- Date of selected log file

- Numeric display of average body temperature for all readings is selected log file
- Plot of all temperature readings in selected log file
- Plot of all accelerometer readings in selected log file

The operator can delete any of the log files in the displayed list by selecting the desired log file and pressing the **Delete Log** button.

Pressing the **Export Log** button will export the selected log file data to Microsoft Excel.

The operator can return to the **Patient Data Screen** by pressing the red **Close X** button at the top right of the **Patient History Screen**.

Patient Data Screen >> Close X

- Closes the **Patient Data Screen** and returns to the **Home Screen**.

Home Screen >> Export Table

Pressing the **Export Table** button will export all of the data in the real-time display on the **Home Screen** to Microsoft Excel.

Home Screen >> All History

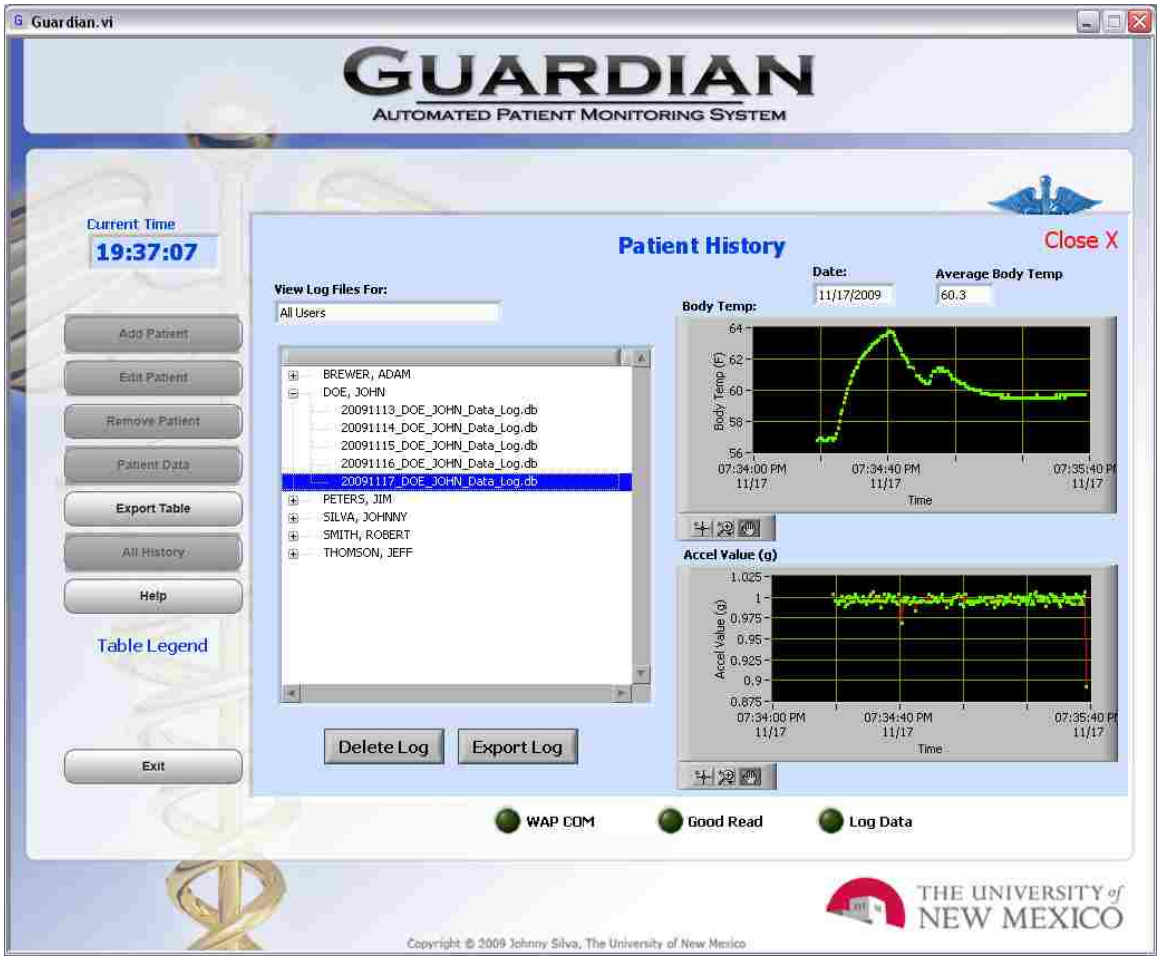


Figure 19: All History Screen

The **All History Screen** is the same as the **Patient History Screen** but incorporates the ability for operators to view, export, or delete any patient log file in the entire patient database.

The operator can return to the **Home Screen** by pressing the red **Close X** button at the top right of the **All History Screen**.

Home Screen >> Help

Pressing the **Help** button will open a web page to guide system operators through the features and operation of the Guardian GUI.

Home Screen >> Table Legend

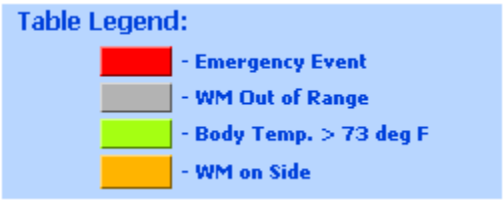


Figure 20: Real-Time Display Table Legend

Pointing to the **Table Legend** label displays a legend in the **User Help** area of the **Home Screen**. The legend describes the coloring of the data rows in the real-time display on the **Home Screen**. Each color indicates a different patient limit condition.

Home Screen >> Exit

Pressing the **Exit** button closes the Guardian GUI and stops all patient data queries and data logging.

4.3.1.2 Database Manager

The Guardian database manager (GDM) handles the creation and maintenance of the entire Guardian patient vital signs database. The GDM creates a database directory structure along with time and date stamped log files for every patient that is entered into the system. Each log file is updated in real-time to ensure minimal loss of data in the

event of a system failure. Each day the GDM automatically creates new log files for every patient in the system.

All data logging done by the Guardian GUI is done through a GDM data submission. The GUI submits a data packet to the GDM containing all received patient data from its current patient query. The GDM interprets the data and adds it to the correct patient log file. Data requests made by the GUI also are done through the GDM. When an operator requests history information for a specific patient, the GDM retrieves the information and passes it to the GUI for display and/or export to Microsoft Excel. An operator can also view the entire patient database by selecting the **All History** button on the **Home Screen** of the Guardian GUI. When the GDM receives this data request, it loads the entire database into memory and sends all database information to the GUI.

The GDM is also used in the GUI when patient log files and directories are deleted. A delete command is issued by an operator and the GDM removes the specified patient's log file or entire log directory from the patient vital signs database.

4.3.1.3 Communications Engine

The communications engine was written using National Instrument's LabVIEW software. It polls through every patient in the system retrieving each patient's real-time vital signs information and checking for emergency events. It also interfaces with the GUI and database manager.

Polled patient information is stored locally on the CMC or on a remote server. A flowchart defining operation of the communications engine can be seen in figure 21 below.

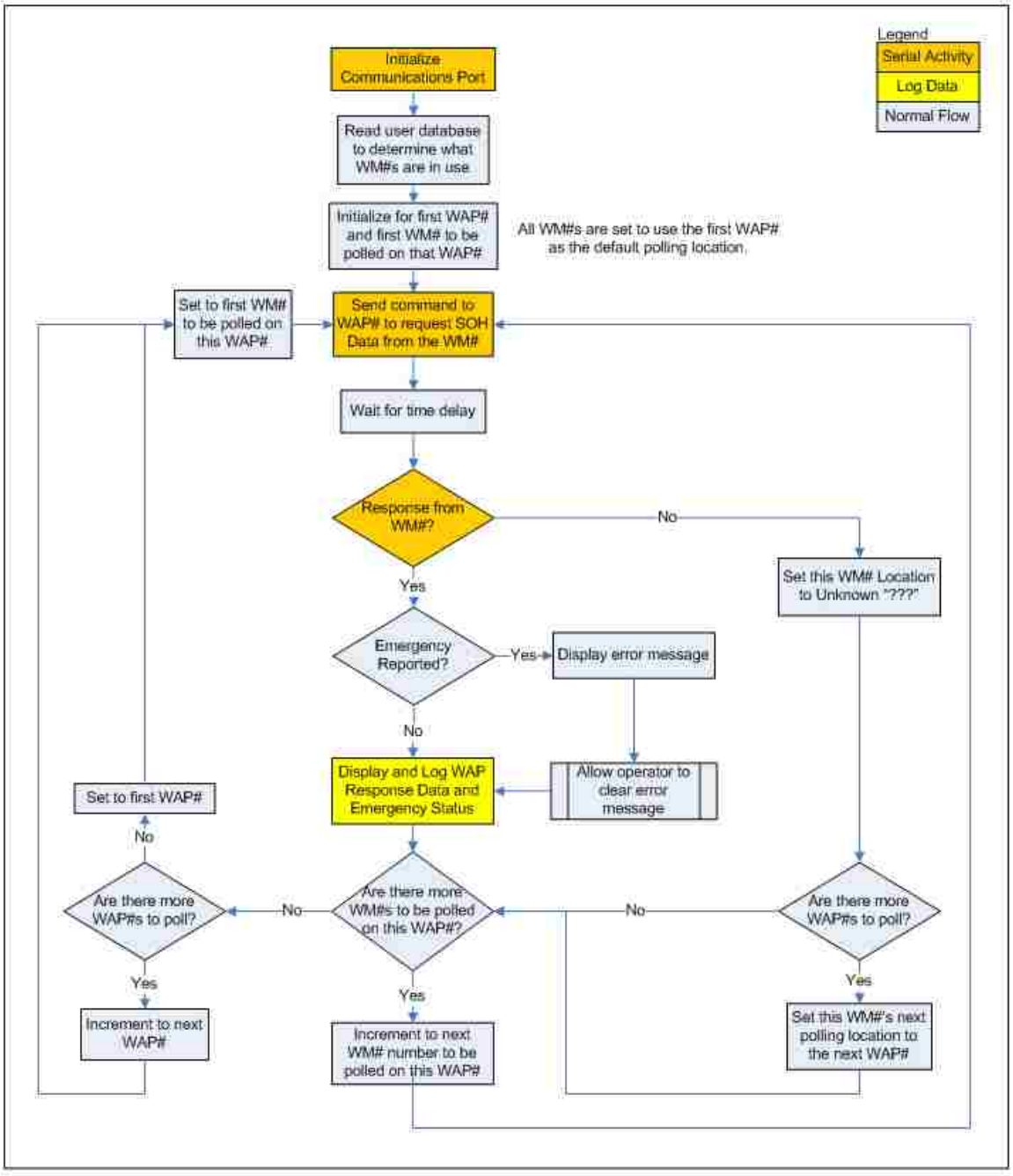


Figure 21: Guardian Communications Engine Flowchart

4.3.1.3.1 Communications Engine Flowchart description

The Guardian communications engine performs the following operations to provide a fully automated patient monitoring system.

1. Initialize wireless access point (WAP) communication port.
2. Read patient database to determine what WM#s are in use.
3. Initialize the polling for the first WAP# in the system and the first WM# to be polled on that WAP#.
4. Send command to current WAP# to request SOH data from current WM#.
5. Wait for time delay.
6. Response from the WM#?
    a. If yes, check to see if that WM# has an emergency to report.
        i. If yes, display error message on the central monitoring computer
            1. Allow operator to clear the error message (this can happen asynchronously any time after this point)
        ii. Display and log WAP response data and emergency status on the central monitoring computer.
    b. If no, set this WM# location to "???"
        i. More WAP#s to poll?
            1. If yes, set this WM#'s next polling location to the next WAP#.
7. More WM#s to be polled on this WAP#?
    a. If yes, increment to the next WM# to be polled on this WAP#.
        i. Return to step 4.
    b. If no, check to see if there are more WAP#s to poll.

i. If yes, increment to the next WAP#.

ii. If no, set WAP# to the first WAP# in the system.

iii. Set to the first WM# to be polled on this WAP#.

8. Return to step 4.

4.3.1.3.2 Polling Schemes

The Guardian communications engine collects patient vital sign data from all patients in the system using a continuous round-robin polling routine. Two different polling methods were investigated to determine what impact changing the number of wireless access points (WAPs) and the number of patients in the system would have on the total time required to complete one polling cycle. Both of the methods below describe a steady-state polling routine without WMs moving between WAP coverage areas.

Polling Method-1:

To complete one polling cycle the system polls the first WAP and requests current vital signs data from **all** patient wrist modules (WMs) in the system. The system then polls the next WAP and again requests data from **all** patient WMs. The polling continues until all of the WAPs in the system are polled.

When any WM is in a shared WAP coverage area, this polling method allows multiple WAPs to access the WM. The WM location is updated in the GUI to indicate the shared coverage area.

The problem with this method is that the overall time required for a single polling cycle is directly proportional to the product of the number of WAPs in the system and

the number of patients in the system. Therefore this method is not efficient for large systems.

Polling Method-2:

To complete one polling cycle the system polls the first WAP and requests current vital signs data only from the patient WMs that are within its coverage area. The system then polls the next WAP and again requests data only from the patient WMs that are within its coverage area. The polling continues until every WM is polled. The communications engine keeps track of which WMs are within each WAP's coverage area.

This method doesn't need to poll every WAP in the system to complete a cycle. If there aren't any WMs in a WAP's coverage area then the WAP is not polled.

This method is much faster than method-1 because the overall time required for a single polling cycle is only proportional to the number of patients in the system and has no relation to the number of WAPs.

The prototype system's communications engine uses this method of polling.

Results for both of these methods are plotted in figure 22 below.

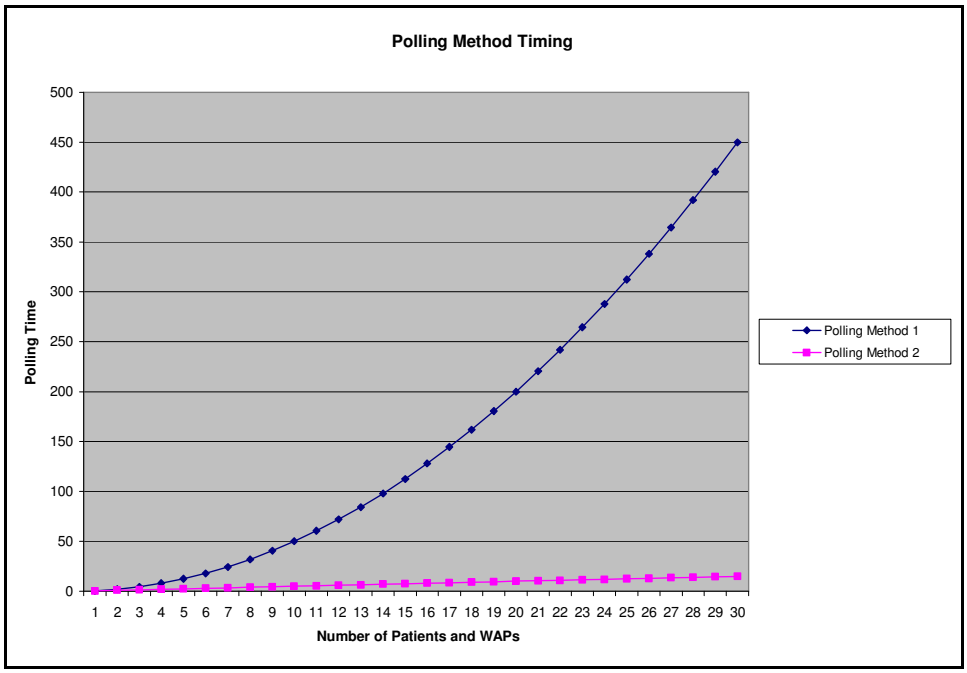


Figure 22: Polling Method Timing Plot

Future Polling Methods

Figure 22 illustrates that polling method 2 is much more time efficient than method 1. However, there are still issues with this polling method. By implementing a non-interruptible constant polling routine, the time needed to poll every patient in the system will always be proportional to the number of patients in the system. This may not be a problem when there are a small number of patients in the system, but for larger facilities this would be unacceptable.

Eliminating the polling routine all together is one method of alleviating this time issue. Implementing a WM interrupt driven data transfer protocol would allow the number of patients in the system to increase exponentially with minimal impact to overall data collection time. Using this type of protocol, each WM would collect patient vital signs data and send it to the CMC periodically using a request interrupt. The CMC would

wait for interrupt requests and service the interrupts in a FIFO manner. The CMC would only request data form a specific WM upon operator request. All emergency events would be transferred to the CMC using a higher level interrupt than the normal vital signs data requests ensuring that all emergency situations are addressed immediately.

4.3.2 Embedded WAP code

The WAP software was developed in C and was programmed into the WAP microcontroller using Microchip's integrated development environment (IDL) called MPLAB. A flowchart and description of the WAP microcontroller code can be seen below.

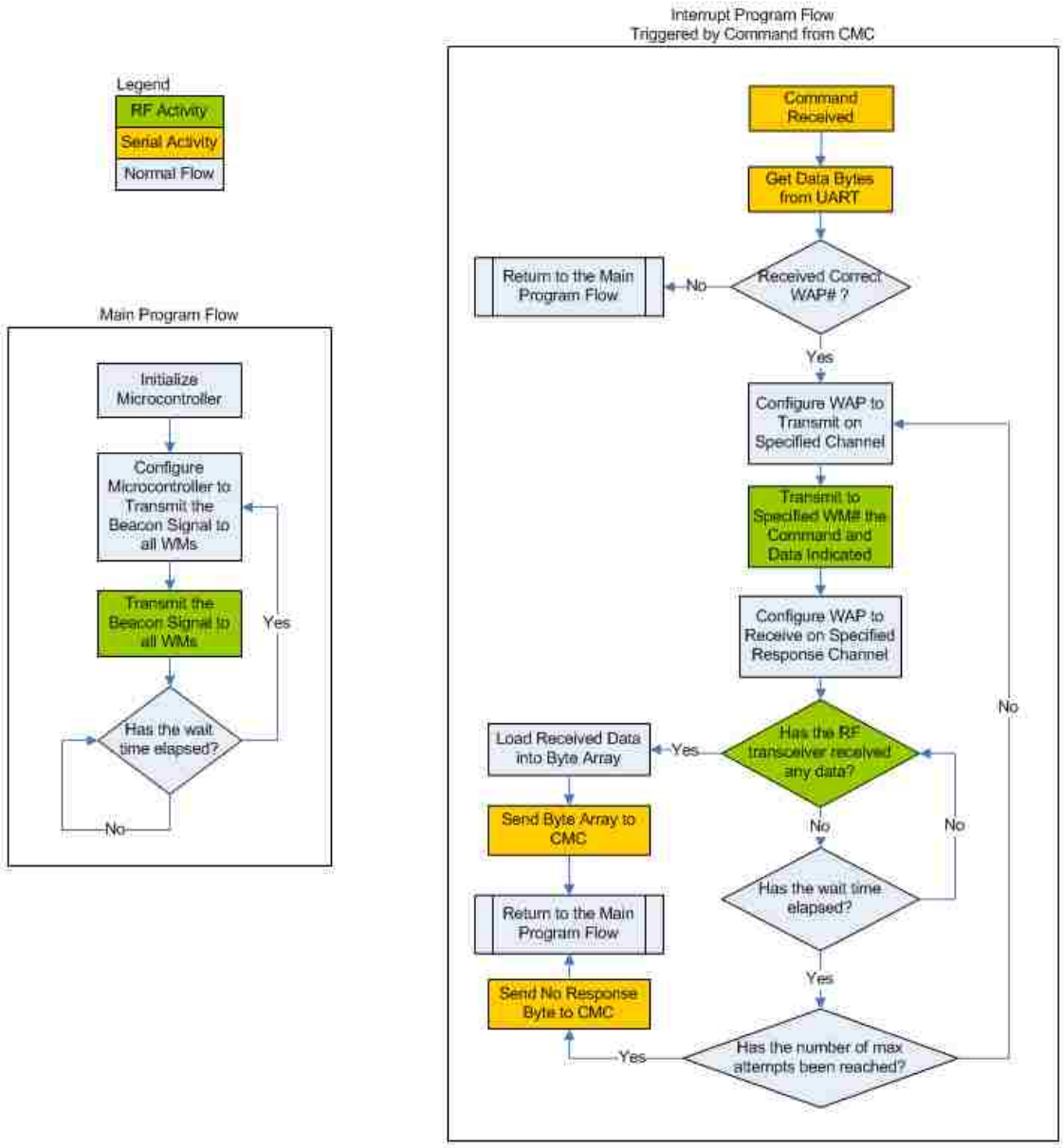


Figure 23: Embedded WAP code Flowchart

4.3.2.1 WAP Flowchart description

The WAP is mainly an interrupt driven program. It will send out a beacon signal in its main loop every **1.2 seconds** to notify the wrist modules that they are within range of a WAP. Commands form the CMC interrupt this loop and are serviced in the order they are received.

4.3.2.1.1 Microcontroller Initialization

When the WAP is first turned on the microcontroller will complete the following actions.

1. Microcontroller will be powered
2. All necessary internal registers of the microcontroller will be initiated to perform the following functions
    a. Five GPIO lines are initialized for communication via the wireless transceiver
    b. The receive channel of the microcontroller's UART is setup as an external interrupt
    c. Internal Timer
        i. Each time the timer overruns it generates an interrupt that resets the timer and starts it again.

4.3.2.1.2 Main Program Flow

1. Configure the transceiver to transmit information on the "Beacon" channel.
2. Send out the beacon signal to wrist modules on the "Beacon" channel.
3. Wait **1.2 seconds**.
4. Go to step 2.

At any point during steps 2 through 4, a command received from the CMC can initiate the Received CMC command interrupt service routine (ISR).

4.3.2.1.3 Received CMC Command Interrupt

1. Check to see if the command was meant for this WAP
    a. If the command is meant for this WAP, determine the type of command sent from the CMC
    b. Create valid wireless data packet and transmit to specified wrist module
    c. If forwarded command requires a response from the wrist module
        i. Configure transceiver to receiver
        ii. Wait 1 millisecond for response from wrist module
        iii. If there is no response
            a) If there was no response after 10 tries
                a. Configure transceiver to transmit
                b. Send “No Response” message back to CMC
                c. Return to main program flow
            b) Else
                a. Configure transceiver to transmit
                b. Forward command to wrist module again
                c. Go to 1.c.i.
        iv. If there is a response
            a) Forward response message to CMC
            b) Return to main program flow
2. Else
            a. Return to main program flow

The associated C code for section is included in **Appendix A**.

4.3.3 Embedded WM code

The WM software was developed in C and was programmed into the WM microcontroller using Microchip's integrated development environment (IDL) called MPLAB. Each wrist module has a designated "wrist module number" associated with it that is entered into the main computer control program database when the module is assigned to a new patient.

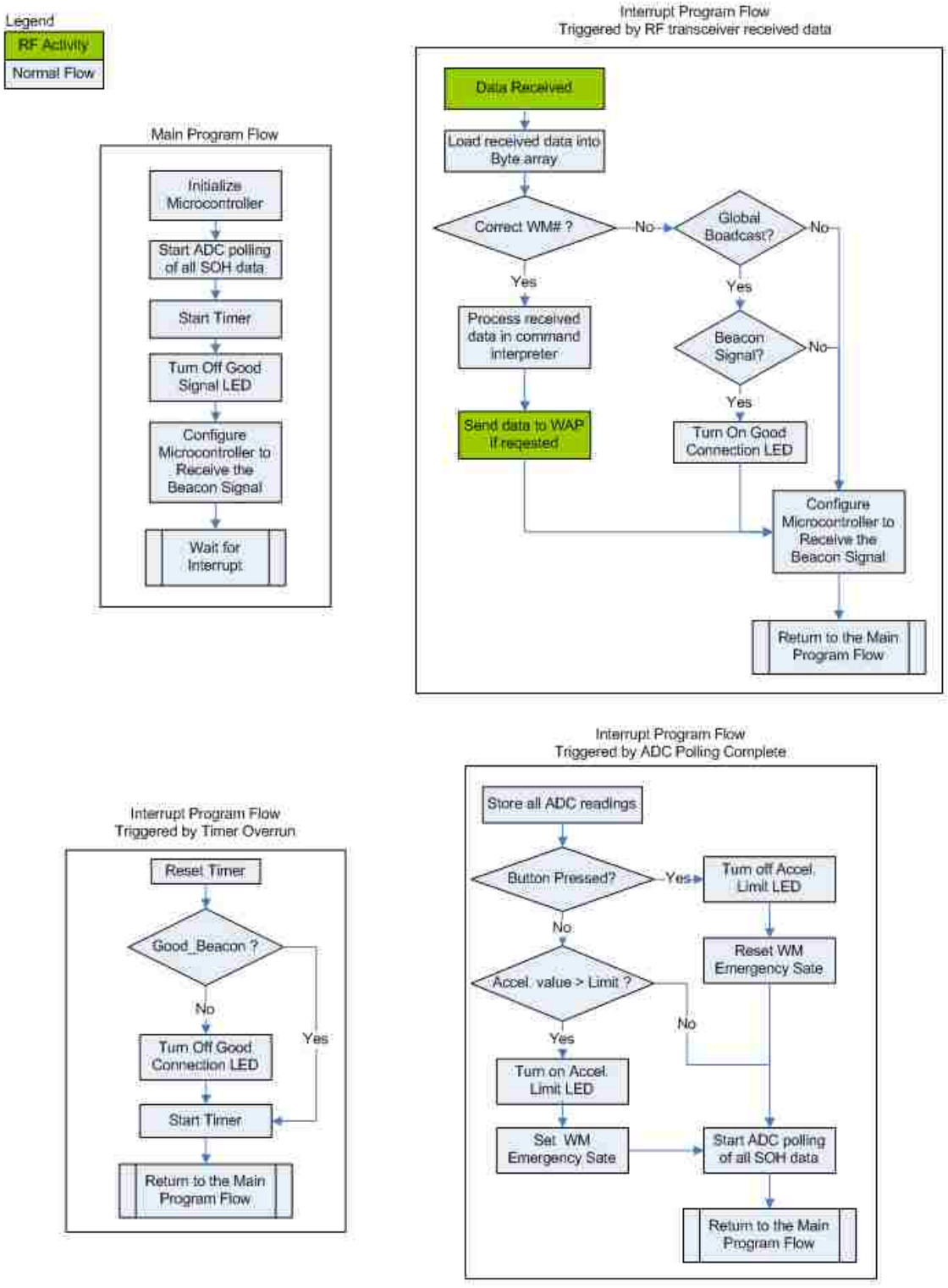


Figure 24: Embedded WM code Flowchart

4.3.3.1 WM Flowchart description

4.3.3.1.1 Microcontroller Initialization

When the wrist module is first turned on the microcontroller will complete the following actions.

1. Microcontroller is powered on
2. All necessary internal registers of the microcontroller are initialized to perform the following functions
   d. Three analog to digital converter (ADC) channels are initialized to monitor the following signals in a polling scheme. When the ADC is done polling it generates an internal interrupt where the readings can be used. At the end of each of these interrupts the ADC polling is started again. Once this process is started it will continue until the microcontroller is powered off.
      i. Body Temperature
      ii. Fall sensor (Accelerometer)
      iii. Wrist module battery voltage
   e. Five GPIO lines are initialized for communication via the wireless transceiver
      i. The GPIO input connected to the Data Ready (DR) line of the wireless transceiver is associated with an external interrupt. This interrupt is triggered whenever the transceiver receives new valid data.
   f. GPIO input for Button status
   g. GPIO output for Good Connection LED
   h. GPIO output for Accelerometer Emergency event LED
   i. Internal Timer

i. Each time the timer overruns it generates an interrupt that resets the timer and starts it again.

4.3.3.1.2 Main Program Flow

In the main program loop the wrist module waits for either an internal or external interrupt to occur. There are three possible interrupts that can occur.

1. Timer Overrun (Internal)
2. ADC Polling Complete (Internal)
3. New wireless message received (External – via data ready line from transceiver)

Each one of these interrupts has its own Interrupt Service Routine (ISR) that is called when a specific interrupt occurs. The new message received ISR processes and interprets State of Health (SOH) data requests and other general commands that are issued from the monitoring computer (CMC). These requests and commands are serviced as requested. The main program loop will have the following form.

1. Configure the transceiver to receive information on the "Beacon" channel.
   a. This channel also receives commands from the CMC and interrupts the program flow to service the commands.
2. Start the internal timer.
3. Start the ADC polling routine.
4. Wait for internal or external interrupt.
5. Go to step 4.

4.3.3.1.3 Timer Overrun Interrupt

Once the internal timer is started it will overrun in **13ms** and carry out the following operations.

1. Every 255 times the timer overruns (**3.3 seconds**) the microcontroller will check to see if it has received a good beacon signal.
    a. If the microcontroller has not received a good beacon signal it will turn off the Good Connection LED.
2. Reset timer
3. Start timer
4. Return to main program flow

4.3.3.1.4 ADC Polling Complete Interrupt

The ADC polling routine takes **88us** (**11.4kHz**) to complete. Each time the polling completes, this ISR is started and the following operations are executed.

1. Read the following voltages that were just polled on the ADC and store them in local variables.
    a. Body Temperature
    b. Fall sensor (Accelerometer)
    c. Wrist module battery voltage
2. Verify that the accelerometer value is within the proper +/- 2g limits.
    a. If the accelerometer reading is outside the limits set the **Emergency Event** flag.
3. Check Button Status
    a. If the Button is being pressed clear the **Emergency Event** flag.

4. Start ADC polling routine.
5. Return to main program flow.

4.3.3.1.5 New wireless message received Interrupt

This interrupt is triggered whenever the transceiver asynchronously receives new valid data. When this occurs this ISR is started and the following operations are executed.

1. Read new data bytes from the wireless transceiver.
2. Check to see if the new data bytes are meant for this Wrist Module (WM) by checking the included WM number.
    a. If the command is meant for this WM, take the appropriate action required by the command. These commands are discussed further in the communications protocol section.
3. If the new data bytes aren't meant for this WM, check to see if they are meant as a global broadcast message to all WM numbers.
    a. If the new data is a global broadcast, check to see if it is a beacon signal.
        i. If the new data is a beacon signal, turn on the Good Wireless Connection LED.
4. Set the wrist module transceiver to receive information on the "Beacon" channel.
5. Return to main program flow.

The associated C code for section is included in **Appendix B**.

4.3.4 System Interface Description

Commands from the CMC are sent to each WM through a WAP. The response from the WM is sent back through the WAP to the CMC. This data transfer sequence is shown in figure 25 below.

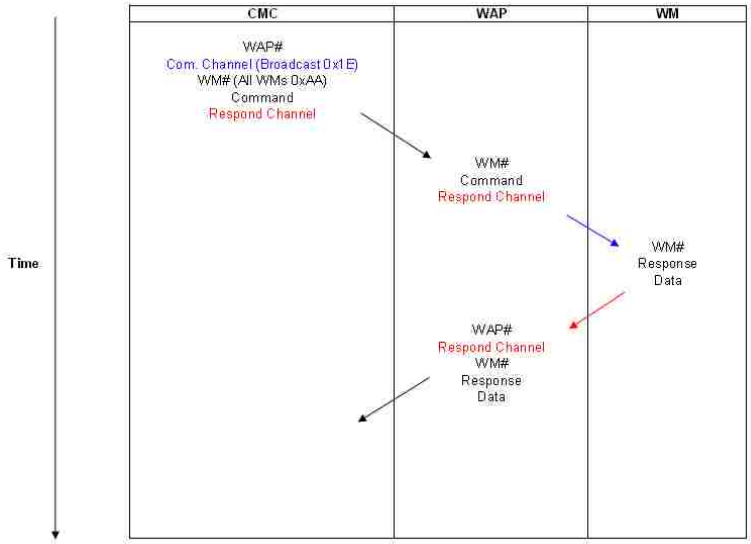


Figure 25: Prototype System Data Sequence Diagram

4.3.4.1 CMC ↔ WAP Data Structure

Data transfers between the CMC and any WAP must be in the following form.

CMC ↔ WAP Data Structure		
Data Field	**Bytes**	**Description**
WAP #	1	WAP used for command and/or data transfers
Com. / Response Channel	1	• CMC → WAP: Channel number WAP is to use when communicating with WM • WAP → CMC: Channel number WM responded on
WM #	1	WM to be commanded
Command / Response	1	• CMC → WAP: Command to be sent to WM • WAP → CMC: Response form WM
Response Channel / Data	Variable	• CMC → WAP: Channel number WM must respond on • WAP → CMC: Data response from WM

Table 4: CMC to WAP Data Structure

4.3.4.2 WAP ↔ WM Data Structure

Data transfers between the WAP and any WM must be in the following form.

WAP ↔ WM Data Structure		
Data Field	**Bytes**	**Description**
WM #	1	• WAP → WM: WM to be commanded • WM → WAP: WM that responded
Command / Response	1	• WAP → WM: Command to WM • WM → WAP: Response form WM
Data	Variable	• WAP → WM: Channel number WM is to responded on • WM → WAP: Data response from WM

Table 5: WAP to WM Data Structure

4.3.4.3 Prototype Command Definitions

The following commands are used in the prototype system for data transfers.

Command Definitions			
Command From Computer		**Response From WM**	
Name	**Value**	**Name**	**Value**
Check for Emergency Event	0x10	Emergency	0x01
Data Request	0x20	Requested Data Response	0x02
Clear Emergency Event	0x40		
Command from WAP			
Beacon	0x30		

Table 6: Prototype Command Definitions

CHAPTER 5
Conclusions & Future Work

5.1 Conclusions

This project identified the required components of a wireless automated patient monitoring system and demonstrated one possible implementation methodology for such a system. This stand-alone prototype automated patient monitoring system was successfully designed, built, and tested.

The final system prototype hardware includes two wireless access points, three wrist modules, and a switching network. Using this hardware in conjunction with the associated embedded microcontroller software and the Guardian graphical user interface allows a centralized monitoring computer to wirelessly collect and display three patients' real-time vital signs information from two different monitoring areas and log all of the received information to a central database.

5.1.1 System Successes

The following sensors were able to be incorporated into the final system.

- Temperature Sensor
    - Monitors patient's body temperature in degrees Fahrenheit.
- Accelerometer
    - Provided for patient fall detection and WM orientation.
- Wrist Module battery voltage monitor
    - Used to ensure proper WM operating voltage.

- Patient input button
    - Used as a simple proof of concept user interface and to clear emergency events.

A wireless link between wrist modules and wireless access points was successfully implemented.

All data from each wrist module is successfully stored in a main patient database on the central monitoring computer.

User friendly graphical interface makes patient monitoring and database file review and export extremely simple.

5.1.2 Possible System Improvements

The following improvements of the prototype design should be addressed for more reliable operation.

- Limited wireless range
    - Potentially use Bluetooth, WLAN, ZigBee wireless hardware, or Cellular modem.
- Weak wireless communication protocol
    - Potentially use standard protocol such as 802.11
- Battery type and life
    - Potentially use 2 AA rechargeable batteries without voltage regulator.

- Size
    - All system sensors and interfaces could be integrated onto a custom circuit board and installed in a custom package that is able to be worn on a patient's wrist.
- Sensors
    - Potentially include more sensors on each wrist module.
- Single Channel Operation
    - More effective monitoring could be archived if the WAPs and WMs communicated on multiple channels. This would allow for an interrupt driven communications system.
- Channel dedication
    - Each WM should have an allocated communications channel.

5.2 Future Work

Many research and development opportunities for future work are available in the field of telemedicine. Some of these possible topics include:

- Use implanted microelectronic sensors and interface these sensors with external electronics for wireless transfer and processing.
- Use an inductively coupled battery charging system
- Use inductive coupling for implanted sensor power and data communication
- Reduce power used by the wrist module by implementing techniques such as a low power sleep mode.

- Develop and incorporate power scavenging techniques to improve battery life such as body heat, solar, and kinetic energy converting electronics.
- Add the capability of bidirectional audio communication between each patient and the central monitoring computer.
- Incorporate innovative techniques for measuring common vital signs such as calculating blood pressure from the patient's pulse velocity.
- Incorporate a reconfigurable antenna and a cognitive radio into each wrist module so it can communicate using multiple communication networks and frequencies.

Given more time and funding to continue work on this project I would first layout a custom board for each wrist module and wireless access point that includes all required circuitry for complete operation. I would also replace the wireless transceivers that are in the current prototype with WLAN 802.11 transceivers. Finally, I would incorporate a bidirectional audio communications feature to each wrist module.

APPENDICES

Appendix A

WAP Embedded C code

```
/*********************************************************************
System: Guardian - Wireless Vital Signs Monitoring System
Program: Wireless Access Point Code
Author:  Johnny Silva
Microcontroller: PIC24HJ32GP202
*********************************************************************/

#include "p24HJ32GP202.h"
#include "Guardian.h"
#include "uart.h"

// Internal FRC Oscillator
_FOSCSEL(FNOSC_FRC); // FRC Oscillator
// Clock Switching is enabled and Fail Safe Clock Monitor is disabled
_FOSC(FCKSM_CSECMD & OSCIOFNC_OFF  & POSCMD_NONE);
// OSC2 Pin Function: OSC2 is Clock Output
// Primary Oscillator Mode: Disabled

_FWDT(FWDTEN_OFF); // Watchdog Timer Enabled/disabled by user software


/*************************************************************/
#define WAP_NUM         0x01 // Set Wireless Access Point Number/
/*************************************************************/

unsigned char Delay_Length = 2;
unsigned char Read_Bytes[PAYLOAD_LEN] = {0};
unsigned char Write_Bytes[PAYLOAD_LEN] = {0};
unsigned char First_Config = -1;
unsigned char Config_Byte = 0x00;
unsigned char Command = 0x00;
unsigned char Response_Channel = 0;
unsigned char WM_Num = 0;
unsigned char Config_Setup[14] = {0};
unsigned char UART_ReceivedChar[16] = {0};
unsigned char UART_Bytes_Received = 0;
unsigned int Timer_Rollover_Cnt = 0;

#if WAP_NUM == 0x01 // WAP #1 ?
      unsigned int Delay = 0;
#else
      unsigned int Delay = 50;
#endif

// Function Prototypes
void Init_Clock();
void Init_Timer1();
void Init_UART();
```

```
void Init_Ports();
void Configure_Transceiver(unsigned char TX_RX, unsigned char Channel);
void Transmit_Data(unsigned char Command, unsigned char Channel);
void Receive_Data();
void Delay_us(int Delay_Cnt);


int main(void)
{
      unsigned char index = 0;

      Init_Clock();
      Init_Timer1();
      Init_UART();
      Init_Ports();

      //*************************************
      // Transceiver Setup Configuration Array
      //*************************************
      // Data bits 111-104 Max data width on channel
      // 1 (excluding CRC and adrs) is 232 (29 Bytes)
      Config_Setup[0] = PAYLOAD_LEN * 8; // Set payload byte length

      // Data bits 103-64 Channel 2 address - we don't care
      // set it to 200
      Config_Setup[1] = 0;
      Config_Setup[2] = 0;
      Config_Setup[3] = 0;
      Config_Setup[4] = 0;
      Config_Setup[5] = 200;

      // Data bits 63-24 Channel 1 address - set it to 17
      Config_Setup[6] = 0;
      Config_Setup[7] = 0;
      Config_Setup[8] = 0;
      Config_Setup[9] = 0;
      Config_Setup[10] = 17;

      // Data bits 23-16 Address width and CRC
      Config_Setup[11] = 0x23; // 8 bit address, 16 bit CRC,
                               // CRC Enabled

      // Data bits 15-8
      Config_Setup[12] = 0x4F; // 1 Channel, ShockBurst Enabled,
                               // 250kbps, 16MHz Crystal,
                               // 0dBm output power

      // Data bits 7-0
      Config_Setup[13] = 0x02; // Channel 1, Transmit
      //*************************************
      // End Setup
      //*************************************
```

```
	// Wait so Beacon signal doesn't start at the same time...
	while(Timer_Rollover_Cnt < Delay);

	Timer_Rollover_Cnt = 0;

	while(1) // MAIN LOOP
	{
		if(Timer_Rollover_Cnt > 100)
		{
			// Send out one Beacon signal every second
			WM_Num = BROADCAST_WM_NUM; // Set to transmit to
						   // every wrist module

			for(index = 0; index < 5; index++)
				Transmit_Data(WAP_BEACON, BEACON_CHAN);

			Timer_Rollover_Cnt = 0;
		}
	}
} // End main


void Init_Clock()
{
	// Configure Oscillator to operate the device at 40Mhz
	// Fosc= Fin*M/(N1*N2), Fcy=Fosc/2
	// Fosc= 7.37*43/(2*2)=80Mhz for 7.37 input clock
	PLLFBD=41;		// M=43
	CLKDIVbits.PLLPOST=0;	// N1=2
	CLKDIVbits.PLLPRE=0;	// N2=2
	OSCTUN=0;		// Tune FRC oscillator, if FRC is used

	// Disable Watch Dog Timer
	RCONbits.SWDTEN=0;

	// Clock switch to incorporate PLL
	__builtin_write_OSCCONH(0x01); // Initiate Clock Switch to
				       // FRC with PLL (NOSC=0b001)
	__builtin_write_OSCCONL(0x01); // Start clock switching
	while (OSCCONbits.COSC != 0b001); // Wait for Clock switch to
					  // occur

	// Wait for PLL to lock
	while(OSCCONbits.LOCK!=1) {};
}


void Init_Timer1()
{
	// Setup timer for 40M/8 = 5M clock (200ns per count)
	// Setup timer to roll-over at 65536 (13.1ms)

	T1CONbits.TON = 0;	// Disable Timer
	T1CONbits.TCS = 0;	// Select internal instruction
				// cycle clock
	T1CONbits.TGATE = 0;	// Disable Gated Timer mode
	T1CONbits.TCKPS = 1;	// Select 1:8 Prescaler
```

```
	TMR1 = 0x0000;          // Clear timer register
	PR1 = 0xFFFF;           // Load the period value
	IPC0bits.T1IP = 7;      // Set Timer1 Interrupt Priority Level
	IFS0bits.T1IF = 0;      // Clear Timer1 Interrupt Flag
	IEC0bits.T1IE = 1;      // Enable Timer1 interrupt
	T1CONbits.TON = 1;      // Start Timer
}


void Init_UART() {
	//Configure U1MODE
	U1MODEbits.UARTEN = 0;  // Bit15 TX, RX DISABLED,
	                        // ENABLE at end of func
	U1MODEbits.USIDL = 0;   // Bit13 Continue in Idle
	U1MODEbits.IREN = 0;    // Bit12 No IR translation
	U1MODEbits.RTSMD = 0;   // Bit11 Simplex Mode
	U1MODEbits.UEN = 0;     // Bits8,9 TX,RX enabled, CTS,RTS not
	U1MODEbits.WAKE = 0;    // Bit7 No Wake up
	U1MODEbits.LPBACK = 0;  // Bit6 No Loop Back
	U1MODEbits.ABAUD = 0;   // Bit5 No Autobaud
	U1MODEbits.URXINV = 0;  // Bit4 IdleState = 1  (for dsPIC)
	U1MODEbits.BRGH = 1;    // Bit3 4 clocks per bit period
	U1MODEbits.PDSEL = 0;   // Bits1,2 8bit, No Parity
	U1MODEbits.STSEL = 0;   // Bit0 One Stop Bit

	U1BRG = BRGVAL; //115200 (Eq. 17-2)

	// Load all values in for U1STA SFR
	U1STAbits.UTXISEL1 = 0; // Bit15 Int when Char is transferred
	U1STAbits.UTXINV = 0;   // Bit14 N/A, IRDA config
	U1STAbits.UTXISEL0 = 0; // Bit13 Other half of Bit15
	U1STAbits.UTXBRK = 0;   // Bit11 Disabled
	U1STAbits.UTXEN = 0;    // Bit10 TX pins controlled by periph
	U1STAbits.UTXBF = 0;    // Bit9 *Read Only Bit*
	U1STAbits.TRMT = 0;     // Bit8 *Read Only bit*
	U1STAbits.URXISEL = 0;  // Bits6,7 Int. on character recieved
	U1STAbits.ADDEN = 0;    // Bit5 Address Detect Disabled
	U1STAbits.RIDLE = 0;    // Bit4 *Read Only Bit*
	U1STAbits.PERR = 0;     // Bit3 *Read Only Bit*
	U1STAbits.FERR = 0;     // Bit2 *Read Only Bit*
	U1STAbits.OERR = 0;     // Bit1 *Read Only Bit*
	U1STAbits.URXDA = 0;    // Bit0 *Read Only Bit*

	RPINR18bits.U1RXR = 10; // Set RP10 to UART RX
	RPOR5bits.RP11R = 3;    // Set RP11 to UART TX

	// Enable RX Interrupt
	IPC2bits.U1RXIP = 6;    // Set Interrupt Priority level
	IFS0bits.U1RXIF = 0;    // Clear the Recieve Interrupt Flag
	IEC0bits.U1RXIE = 1;    // Enable Recieve Interrupts

	U1MODEbits.UARTEN = 1;  // And turn the peripheral on

	U1STAbits.UTXEN = 1;
}
```

```
void Init_Ports()
{
      TRISA = 0xFFFF;    // Set all bits on port A to inputs
      _TRISA2 = 0;       // Set RA2 to output
      _TRISA4 = 0;       // Set RA4 to output

      AD1PCFGL = 0xFFFF; // all PORTA = Digital

      TRISB = 0x0000;    // Set all bits on port B to outputs
      _TRISB0 = 1;       // Set RB0 to input
      _TRISB1 = 1;       // Set RB1 to input
      _TRISB10 = 1;      // Set RB10 to input
      _TRISB15 = 1;      // Set RB15 to input

      _TRISB13 = 1;      // Set RB13 to input
      _TRISB14 = 1;      // Set RB14 to input
      _TRISB4 = 1;       // Set RB4 to input for RX_DR1
}


// 2.4G Configuration - Transceiver
void Configure_Transceiver(unsigned char TX_RX, unsigned char Channel)
{
    unsigned char index = 0, index_2 = 0, index_3 = 0;
    unsigned char temp = 0;

    // During configuration of the receiver, we need RX_DATA
    // as an output
      _TRISB3 = 0; // Set RB3 to output for RX_DAT config

      if(TX_RX) // Transmit
      {
            Config_Byte = Channel & 0xFE;
      }
      else // Receive
      {
            Config_Byte = Channel | 0x01;
      }

    if(First_Config)
      {
            Delay_us(4000);   // 4ms Delay

            // Config Mode
            CE = 0; CS = 1;

            Delay_us(5);      // 5us Delay

            // Shift out all 14 Bytes of data to cofig the transceiver
            Config_Setup[13] = Config_Byte;

            // Clock in configuration data
            for(index = 0 ; index < 14 ; index++)
            {
                  temp = Config_Setup[index];

                  for(index_2 = 0 ; index_2 < 8 ; index_2++)
```

```
                {
                        TX_DAT = temp >> 7;

                        // Delay
                        for(index_3 = 0; index_3 < Delay_Length;
                            index_3++)
                              Nop(); // 25ns

                        CLK = 1;

                        // Delay
                        for(index_3 = 0; index_3 < Delay_Length;
                            index_3++)
                              Nop(); // 25ns

                        CLK = 0;

                        temp <<= 1;
                }
        }
}
else
{
        // 50ns Delay
        Nop(); // 25ns
        Nop(); // 25ns

        // Config Mode
        CE = 0; CS = 1;

        Delay_us(5); // 5us Delay

        // Only shift out Config_Byte
        for(index = 0 ; index < 8 ; index++)
        {
                TX_DAT = Config_Byte >> 7;

                // Delay
                for(index_2 = 0; index_2 < Delay_Length; index_2++)
                        Nop(); // 25ns

                CLK = 1;

                // Delay
                for(index_2 = 0; index_2 < Delay_Length; index_2++)
                        Nop(); // 25ns

                CLK = 0;

                Config_Byte <<= 1;
        }
}

// Configuration is actived on falling edge of CS
CE = 0; CS = 0;

if(TX_RX == 0)
```

```
	{
		// After configuration of the receiver, we need
		// RX_DATA as an input
		_TRISB3 = 1; // Set RB3 to input for RX_DAT

	// Start monitoring the air
	CE = 1;

	Delay_us(205); // 205us Delay
	}

	First_Config = 0;
}


// This sends out the data stored in the Write_Bytes
void Transmit_Data(unsigned char Command, unsigned char Channel)
{
	unsigned char index = 0, index_2 = 0, index_3 = 0;
	unsigned char temp = 0, rf_address = 0;

	Configure_Transceiver(TRANSMIT, Channel);

	// Erase the current data array so that we know we are
	// sending real data
	for(index = 0 ; index < PAYLOAD_LEN; index++)
	  Write_Bytes[index] = 0x00;

	// 50ns Delay
	Nop(); // 25ns
	Nop(); // 25ns

	RF_ACTIVE = 1;

	// Load Data Array
	Write_Bytes[0] = WM_Num;
	Write_Bytes[1] = Command;
	Write_Bytes[2] = Response_Channel;

	CE = 1;

	Delay_us(5); // 5us Delay

    // Clock in address
    rf_address = 17;
    for(index = 0 ; index < 8 ; index++)
    {
        TX_DAT = rf_address >> 7;

		// Delay
		for(index_2 = 0; index_2 < Delay_Length; index_2++)
		  Nop(); // 25ns

		CLK = 1;
```

```
            // Delay
            for(index_2 = 0; index_2 < Delay_Length; index_2++)
              Nop(); // 25ns

        CLK = 0;

        rf_address <<= 1;
    }

    // Clock in the Write_Bytes
    for(index = 0; index < PAYLOAD_LEN; index++) // PAYLOAD_LEN bytes
    {
        temp = Write_Bytes[index];

        for(index_2 = 0; index_2 < 8; index_2++) // One bit at a time
        {
            TX_DAT = temp >> 7;

                  // Delay
                  for(index_3 = 0; index_3 < Delay_Length; index_3++)
                    Nop(); // 25ns

              CLK = 1;

                  // Delay
                  for(index_3 = 0; index_3 < Delay_Length; index_3++)
                    Nop(); // 25ns

              CLK = 0;

            temp <<= 1;
        }
    }

      CE = 0; // Start transmission

      TX_DAT = 0; // Set TX_Data low

      Delay_us(1000);

      RF_ACTIVE = 0;
}


// This will clock out the current payload into the Read_bytes
void Receive_Data()
{
      RF_ACTIVE = 1;

    unsigned char index = 0, index_2 = 0, index_3 = 0;
    unsigned char temp = 0;

    CE = 0; // Power down RF Front end
```

```
    // Erase the current data array so that we know we are
    // looking at actual received data
    for(index = 0; index < PAYLOAD_LEN; index++)
        Read_Bytes[index] = 0x00;

    // Clock in data
    for(index = 0; index < PAYLOAD_LEN; index++) // PAYLOAD_LEN bytes
    {
        for(index_2 = 0; index_2 < 8; index_2++) // 8 bits each
        {
            temp <<= 1;
            temp += RX_DAT;

                  // Delay
                  for(index_3 = 0; index_3 < Delay_Length; index_3++)
                    Nop(); // 25ns

            CLK = 1;

                  // Delay
                  for(index_3 = 0; index_3 < Delay_Length; index_3++)
                    Nop(); // 25ns

            CLK = 0;
        }

        Read_Bytes[index] = temp; // Store this byte
    }

    CE = 1; // Power up RF Front end

    RF_ACTIVE = 0;
}


void Delay_us(int Delay_Cnt)
{
      int Start_Time = 0;
      int Delta_Time = 0;

      Start_Time = TMR1;

      while((int)(Delta_Time/5) < Delay_Cnt) // Convert Delta_Time to
                                             // microseconds
      {
            Delta_Time = TMR1 - Start_Time;
      }
}


//******************** ISRs ***********************
void __attribute__((interrupt, no_auto_psv)) _T1Interrupt(void)
{// RTC Timer

      // Reset Timer
      IFS0bits.T1IF = 0; // Clear interrupt flag
    T1CONbits.TON = 0;   // Disable timer
```

```
    TMR1 = 0; // Reset timer count

    Timer_Rollover_Cnt++;

      T1CONbits.TON = 1; // Enable timer
}


void __attribute__((interrupt, no_auto_psv)) _U1RXInterrupt(void)
{// RX Control Computer

      IEC0bits.U1RXIE = 0; // Disable Recieve Interrupts
      IFS0bits.U1RXIF = 0; // Reset interrupt flag

      char Done = 0;
      char Attempts = 0;
      unsigned char Channel = 0;
      unsigned char UART_Index = 0;

      // Read character from UART1
      UART_ReceivedChar[UART_Bytes_Received] = U1RXREG;

      UART_Bytes_Received++;

      if(UART_Bytes_Received > 4)
      {
            UART_Bytes_Received = 0;

            if(UART_ReceivedChar[0] == WAP_NUM)
            {
                  Channel = UART_ReceivedChar[1];
                  WM_Num = UART_ReceivedChar[2];
                  Command = UART_ReceivedChar[3];
                  Response_Channel = UART_ReceivedChar[4];

                  DATA_RECEIVED = 0; // Clear DATA_RECEIVED flag

                  while(!Done)
                  {
                        Transmit_Data(Command, Channel);

                        Configure_Transceiver(RECEIVE,
                        Response_Channel);

                        Delay_us(1000); // Wait for response from WM

                        if(RX_DR1)
                        {
                              Receive_Data();

                              // wait if the buffer is full
                              while(U1STAbits.UTXBF);

                              U1TXREG = WAP_NUM;
                              // wait if the buffer is full
                              while(U1STAbits.UTXBF);
```

```
                                    U1TXREG = Response_Channel;

                                    for(UART_Index = 0 ; UART_Index <
                                        PAYLOAD_LEN; UART_Index++)
                                    {
                                          // wait if the buffer is full
                                          while(U1STAbits.UTXBF);

                                          U1TXREG = Read_Bytes[UART_Index];
                                    }

                                    Done = -1;

                                    // Set DATA_RECEIVED flag
                                    DATA_RECEIVED = 1;
                              }
                              else
                              {
                                    if(Attempts++ > 10)
                                    {
                                          // Send Error back to CMC
                                          // wait if the buffer is full
                                          while(U1STAbits.UTXBF);
                                          U1TXREG = 0xDE;
                                          // wait if the buffer is full
                                          while(U1STAbits.UTXBF);

                                          U1TXREG = 0xAD;
                                          Done = -1;
                                    }
                              }
                        }

                  }
            }

            IEC0bits.U1RXIE = 1; // Enable Recieve Interrupts
}
//***************************************
// END ISRs
//***************************************
```

Appendix B

WM Embedded C code

```
/*****************************************************************
System: Guardian - Wireless Vital Signs Monitoring System
Program: Wrist Module Code
Author:  Johnny Silva
Microcontroller: PIC24HJ32GP202
*****************************************************************/

#include "p24HJ32GP202.h"
#include "Guardian.h"


// Internal FRC Oscillator
_FOSCSEL(FNOSC_FRC); // FRC Oscillator
// Clock Switching is enabled and Fail Safe Clock Monitor is disabled
_FOSC(FCKSM_CSECMD & OSCIOFNC_OFF  & POSCMD_NONE);
// OSC2 Pin Function: OSC2 is Clock Output
// Primary Oscillator Mode: Disabled

_FWDT(FWDTEN_OFF); // Watchdog Timer Enabled/disabled by user software


/*************************************************************/
#define WM_NUM          0x01 // Set Wrist Module Number        /
/*************************************************************/

unsigned char Delay_Length = 2;
unsigned char Read_Bytes[PAYLOAD_LEN] = {0};
unsigned char Write_Bytes[PAYLOAD_LEN] = {0};
unsigned char First_Config = -1;
unsigned char Config_Byte = 0x00;
unsigned char Response = 0x00;
unsigned char Config_Setup[14] = {0};
unsigned char Allocated_Channel = 0;
unsigned int Accel_Limit = 0; // Set to ADC voltage limit
unsigned char No_Beacon_Cnt = 0;
int Accel_Value = 0;
int Temp_Value = 0;
int Vcc_Value = 0;
int ADC_Buffer_Sum = 0;
unsigned char ADC_Average_Number = 8;
unsigned char ADC_Average_Count = 0;
unsigned int ADC_Average = 0;
unsigned int Timer_Rollover_Cnt = 0;
unsigned char Emergency_Event = 0;


// Function Prototypes
void Init_Clock();
void Init_Timer1();
void Init_Ports();
```

```
void Init_ADC();
void Configure_Transceiver(unsigned char TX_RX, unsigned char Channel);
void Transmit_Data(unsigned char Response, unsigned char Channel);
void Receive_Data();
void Delay_us(int Delay_Cnt);


int main(void)
{
      int index = 0;

      Init_Clock();
      Init_Timer1();
      Init_Ports();
      Init_ADC();

      //*************************************
      // Transceiver Setup Configuration Array
      //*************************************
      // Data bits 111-104 Max data width on channel 1
      // (excluding CRC and adrs) is 232 (29 Bytes)
      Config_Setup[0] = PAYLOAD_LEN * 8; // Set to PAYLOAD_LEN
                                         // byte transfer

      // Data bits 103-64 Channel 2 address - we don't care,
      // set it to 200
      Config_Setup[1] = 0;
      Config_Setup[2] = 0;
      Config_Setup[3] = 0;
      Config_Setup[4] = 0;
      Config_Setup[5] = 200;

      // Data bits 63-24 Channel 1 address - set it to 17
      Config_Setup[6] = 0;
      Config_Setup[7] = 0;
      Config_Setup[8] = 0;
      Config_Setup[9] = 0;
      Config_Setup[10] = 17;

      // Data bits 23-16 Address width and CRC
      Config_Setup[11] = 0x23; // 8 bit address, 16 bit CRC,
                               // CRC Enabled

      // Data bits 15-8
      Config_Setup[12] = 0x4F; // 1 Channel, ShockBurst Enabled,
                               // 250kbps, 16MHz Crystal,
                               // 0dBm output power

      // Data bits 7-0
      Config_Setup[13] = 0x02; // Channel 1, Transmit
      //*************************************
      // End Setup
      //*************************************
```

```
	for(index = 0; index < WM_NUM; index++)
	{
		GOOD_CONNECTION_LED = 1;

		while(Timer_Rollover_Cnt < 10);
		Timer_Rollover_Cnt = 0;

		GOOD_CONNECTION_LED = 0;

		while(Timer_Rollover_Cnt < 10);
		Timer_Rollover_Cnt = 0;
	}


	// Setup WM to receive on allocated Beacon channel
	Configure_Transceiver(RECEIVE, BEACON_CHAN);

	while(1) // MAIN LOOP
	{
		if(Timer_Rollover_Cnt > 10)
		{
			if(Emergency_Event)
				ACCEL_LIMIT_LED = !ACCEL_LIMIT_LED;

			Timer_Rollover_Cnt = 0;
		}
	}

}// End main


void Init_Clock()
{
	// Configure Oscillator to operate the device at 40Mhz
	// Fosc= Fin*M/(N1*N2), Fcy=Fosc/2
	// Fosc= 7.37*43/(2*2)=80Mhz for 7.37 input clock
	PLLFBD=41;                      // M=43
	CLKDIVbits.PLLPOST=0;           // N1=2
	CLKDIVbits.PLLPRE=0;            // N2=2
	OSCTUN=0;                       // Tune FRC oscillator,
	                                // if FRC is used

	// Disable Watch Dog Timer
	RCONbits.SWDTEN=0;

	// Clock switch to incorporate PLL
	__builtin_write_OSCCONH(0x01); // Initiate Clock Switch to
	                               // FRC with PLL (NOSC=0b001)
	__builtin_write_OSCCONL(0x01); // Start clock switching
	while(OSCCONbits.COSC != 0b001); // Wait for Clock switch to
	                                 // occur

	// Wait for PLL to lock
	while(OSCCONbits.LOCK!=1);
}
```

```
void Init_Timer1()
{
      // Setup timer for 40M/8 = 5M clock (200ns per count)
      // Setup timer to roll-over at 65535 (13.1ms)

      T1CONbits.TON = 0;      // Disable Timer
      T1CONbits.TCS = 0;      // Select internal instruction cycle
                              // clock
      T1CONbits.TGATE = 0;    // Disable Gated Timer mode
      T1CONbits.TCKPS = 1;    // Select 1:8 Prescaler
      TMR1 = 0x0000;          // Clear timer register
      PR1 = 0xFFFF;           // Load the period value
      IPC0bits.T1IP = 7;      // Set Timer1 Interrupt Priority Level
      IFS0bits.T1IF = 0;      // Clear Timer1 Interrupt Flag
      IEC0bits.T1IE = 1;      // Enable Timer1 interrupt
      T1CONbits.TON = 1;      // Start Timer
}


void Init_Ports()
{
      TRISA = 0xFFFF;   // Set all bits on port A to inputs
      _TRISA2 = 0;      // Set RA2 to output - CLK
      _TRISA4 = 0;      // Set RA4 to output - CS

      TRISB = 0x0000;   // Set all bits on port B to outputs
      _TRISB0 = 1;      // Set RB0 to input - PGED1
      _TRISB1 = 1;      // Set RB1 to input - PGEC1
      _TRISB10 = 1;     // Set RB10 to input - UART_RX
      _TRISB15 = 1;     // Set RB15 to input - LED1 Red

      _TRISB8 = 1;      // Set RB8 to input - BUTTON_1
      _TRISB13 = 1;     // Set RB13 to input - Vcc_Mon
      _TRISB14 = 1;     // Set RB14 to input - Temp_Mon
      _TRISB4 = 1;      // Set RB4 to input - RX_DR1

      RPINR0bits.INT1R = 4; // Set RX_DR1 as INT1

      // SETUP RX_DR1 INTERRUPT (INT1)
      INTCON2bits.INT1EP = 0; // INT1 INTERRUPT ON POSITIVE EDGE
      IFS1bits.INT1IF = 0;    // CLEAR INTERRUPT INT1 FLAG
      IPC5bits.INT1IP = 6;    // SET INT1 PRIORITY
      IEC1bits.INT1IE = 1;    // ENABLE INT1 INTERRUPT

      // ADC Interrupt priority
      IPC3bits.AD1IP = 1;
}

void Init_ADC()
{
      AD1CON1bits.FORM   = 0; // Data Output Format: Integer
      AD1CON1bits.SSRC   = 7; // Sample Clock Source: Internal counter
                              // ends sampling and starts conversion
                              // (auto convert)
      AD1CON1bits.ASAM   = 1; // ADC Sample Control: Sampling begins
                              // immediately after conversion
      AD1CON1bits.AD12B  = 1; // 12-bit ADC operation
```

```
      AD1CON2bits.VCFG = 0;  // Converter Voltage Reference
                             // Configuration bits (VREFH = Avdd,
                             // VREFL = Avss)
      AD1CON2bits.CSCNA = 1; // Scan Input Selections for CH0+ during
                             // Sample A bit
      AD1CON2bits.CHPS  = 0; // Converts CH0
      AD1CON3bits.ADRC = 0;  // ADC Clock is derived from Systems Clock
      AD1CON3bits.ADCS = 25; // ADC Conversion Clock Tad=Tcy*(ADCS+1)=
                             // (1/40M)*26 = 650ns (1.54Mhz)
                             // ADC Conversion Time for 12-bit
                             // Tc=14*Tad = 9.1us
                             // Time to convert 3 ADC channels 3*Tc =
                             // 3*9.1us = 27.3us
      AD1CON3bits.SAMC = 31; // Auto Sample Time bits 31*Tad = 20.15us
                             // Time to sample 3 ADC channels 3 *
                             // 20.15us = 60.45us
                             // So ADC interrupt happens every 88us or
                             // 11.4kHz
      AD1CON2bits.SMPI = 2;  // 3 ADC Channels are scanned

      // AD1CSSL: A/D Input Scan Selection Register
      AD1CSSL = 0x0C01; // Enable AN0, AN10, and AN11 for channel scan

      // AD1PCFGL: Port Configuration Register
      AD1PCFGL = 0xF3FE; // AN0, AN10, and AN11 as Analog Input

      IFS0bits.AD1IF   = 0; // Clear the A/D interrupt flag bit
      IEC0bits.AD1IE   = 1; // Do Not Enable A/D interrupt
      AD1CON1bits.ADON = 1; // Turn on the A/D converter
}


// 2.4G Configuration - Transceiver
void Configure_Transceiver(unsigned char TX_RX, unsigned char Channel)
{
    unsigned char index = 0, index_2 = 0, index_3 = 0;
    unsigned char temp = 0;

      // During configuration of the receiver, we need RX_DATA as an
      // output
      _TRISB3 = 0; // Set RB3 to output for RX_DAT config

      if(TX_RX) // Transmit
      {
            Config_Byte = Channel & 0xFE;
      }
      else // Receive
      {
            Config_Byte = Channel | 0x01;
      }

    if(First_Config)
      {
            Delay_us(4000); // 4ms Delay

            // Config Mode
            CE = 0; CS = 1;
```

```
        Delay_us(5); // 5us Delay

        // Shift out all 14 Bytes of data to cofig the transceiver
        Config_Setup[13] = Config_Byte;

        // Clock in configuration data
        for(index = 0 ; index < 14 ; index++)
        {
              temp = Config_Setup[index];

              for(index_2 = 0 ; index_2 < 8 ; index_2++)
              {
                    TX_DAT = temp >> 7;

                    // Delay
                    for(index_3 = 0; index_3 < Delay_Length;
                        index_3++)
                          Nop(); // 25ns

                    CLK = 1;

                    // Delay
                    for(index_3 = 0; index_3 < Delay_Length;
                        index_3++)
                          Nop(); // 25ns

                    CLK = 0;

                    temp <<= 1;
              }
        }
}
else
{
        // 50ns Delay
        Nop(); // 25ns
        Nop(); // 25ns

        // Config Mode
        CE = 0; CS = 1;

        Delay_us(5); // 5us Delay

        // Only shift out Config_Byte
        for(index = 0 ; index < 8 ; index++)
        {
              TX_DAT = Config_Byte >> 7;

              // Delay
              for(index_2 = 0; index_2 < Delay_Length; index_2++)
                Nop(); // 25ns

              CLK = 1;
```

```
                        // Delay
                        for(index_2 = 0; index_2 < Delay_Length; index_2++)
                          Nop(); // 25ns

                        CLK = 0;

                        Config_Byte <<= 1;
                }
        }

        // Configuration is actived on falling edge of CS (page 10)
        CE = 0; CS = 0;

        if(TX_RX == 0)
        {
                // After configuration of the receiver, we need RX_DATA
                // as an input
                _TRISB3 = 1; // Set RB3 to input for RX_DAT

                // Start monitoring the air
                CE = 1;

                Delay_us(205); // 205us Delay
        }

        First_Config = 0;
}


// This sends out the data stored in the Write_Bytes
void Transmit_Data(unsigned char Response, unsigned char Channel)
{
        unsigned char index = 0, index_2 = 0, index_3 = 0;
        unsigned char temp = 0, rf_address = 0;

        Configure_Transceiver(TRANSMIT, Channel);

        // Erase the current data array so that we know we are
        // sending real data
        for(index = 0 ; index < PAYLOAD_LEN; index++)
          Write_Bytes[index] = 0x00;


        // 50ns Delay
        Nop(); // 25ns
        Nop(); // 25ns

        //Load Data Array
        Write_Bytes[0] = WM_NUM;
        Write_Bytes[1] = Response;

        switch(Response)
        {
                case(WM_EMERGENCY): //0x01
                        Write_Bytes[2] = 0xAA;
                        Write_Bytes[3] = 0x55;
                break;
```

```
        case(WM_SOH_RESPONSE): //0x02
              //High Byte
              Write_Bytes[2] = (char)((Temp_Value & 0xFF00) >> 8);
              //Low Byte
              Write_Bytes[3] = (char)(Temp_Value & 0x00FF);
              //High Byte
              Write_Bytes[4] = (char)((Accel_Value & 0xFF00) >> 8);
              //Low Byte
              Write_Bytes[5] = (char)(Accel_Value & 0x00FF);
              //High Byte
              Write_Bytes[6] = (char)((Vcc_Value & 0xFF00) >> 8);
              //Low Byte
              Write_Bytes[7] = (char)(Vcc_Value & 0x00FF);
              //Send Emergency_Event state
              Write_Bytes[8] = Emergency_Event;
              //Send BUTTON_1 state
              Write_Bytes[9] = BUTTON_1;
        break;
  }

CE = 1;

Delay_us(5); // 5us Delay

// Clock in address
rf_address = 17;
for(index = 0 ; index < 8 ; index++)
{
    TX_DAT = rf_address >> 7;

    // Delay
    for(index_2 = 0; index_2 < Delay_Length; index_2++)
      Nop(); // 25ns

    CLK = 1;

  // Delay
  for(index_2 = 0; index_2 < Delay_Length; index_2++)
    Nop(); // 25ns

  CLK = 0;

  rf_address <<= 1;
}

// Clock in the Write_Bytes
for(index = 0; index < PAYLOAD_LEN; index++) // PAYLOAD_LEN bytes
{
    temp = Write_Bytes[index];

    for(index_2 = 0 ; index_2 < 8 ; index_2++) // One bit at a time
    {
        TX_DAT = temp >> 7;

        // Delay
        for(index_3 = 0; index_3 < Delay_Length; index_3++)
          Nop(); // 25ns
```

```
            CLK = 1;

            // Delay
            for(index_3 = 0; index_3 < Delay_Length; index_3++)
              Nop(); // 25ns

            CLK = 0;

            temp <<= 1;
        }
    }

    CE = 0; // Start transmission

      TX_DAT = 0; // Set TX_Data low

      Delay_us(1000);
}


// This will clock out the current payload into the Read_Bytes
void Receive_Data()
{
    unsigned char index = 0, index_2 = 0, index_3 = 0;
    unsigned char temp = 0;

    CE = 0; // Power down RF Front end

    // Erase the current data array so that we know we are looking at
    // actual received data
    for(index = 0 ; index < PAYLOAD_LEN; index++)
        Read_Bytes[index] = 0x00;

    // Clock in data
    for(index = 0 ; index < PAYLOAD_LEN; index++) // PAYLOAD_LEN bytes
    {
        for(index_2 = 0 ; index_2 < 8 ; index_2++) // 8 bits each
        {
            temp <<= 1;
            temp += RX_DAT;

            // Delay
            for(index_3 = 0; index_3 < Delay_Length; index_3++)
              Nop(); // 25ns

            CLK = 1;

            // Delay
            for(index_3 = 0; index_3 < Delay_Length; index_3++)
              Nop(); // 25ns

            CLK = 0;
        }

        Read_Bytes[index] = temp; // Store this byte
    }
```

```
    CE = 1; // Power up RF Front end
}


void Delay_us(int Delay_Cnt)
{
      int Start_Time = 0;
      int Delta_Time = 0;

      Start_Time = TMR1;

      while((int)(Delta_Time/5) < Delay_Cnt) // Convert Delta_Time to
                                             // microseconds
      {
            Delta_Time = TMR1 - Start_Time;
      }
}


//******************** ISRs ***********************
void __attribute__((interrupt, no_auto_psv)) _T1Interrupt(void)
{// RTC Timer

    // Reset Timer
    IFS0bits.T1IF = 0; // Clear interrupt flag
    T1CONbits.TON = 0; // Disable timer
    TMR1 = 0;          // Reset timer count

      // Check Good Connection status
      if(No_Beacon_Cnt++ > 254)
      {
            No_Beacon_Cnt = 0;
            GOOD_CONNECTION_LED = 0; // Disable Good Connection LED
      }

      Timer_Rollover_Cnt++;

      T1CONbits.TON = 1; // Enable timer
}


void __attribute__((interrupt, no_auto_psv)) _INT1Interrupt(void)
{// RX_DR1 INTERRUPT (INT1)

      unsigned char index = 0;

      Receive_Data();

      if(Read_Bytes[0] == WM_NUM) // Is this transmission meant for
                                  // this wrist module?
      {
            switch(Read_Bytes[1]) // WAP_Command
            {
                  case(WAP_DATA_REQUEST):
                        Allocated_Channel = Read_Bytes[2];
```

```
                        for(index = 0; index < 5; index++)
                              Transmit_Data(WM_SOH_RESPONSE,
                                          Allocated_Channel);

                        // Setup WM to receive on allocated Beacon
                        // channel
                        Configure_Transceiver(RECEIVE, BEACON_CHAN);
                  break;
                  case(WAP_CLEAR_ACCEL_EVENT):
                        // Clear Emergency Event flag
                        Emergency_Event = 0;
                        // Turn off ACCEL_LIMIT_LED
                        ACCEL_LIMIT_LED = 0;
                  break;
            }
      }
      else if(Read_Bytes[0] == BROADCAST_WM_NUM) // 0xAA
      {// Message sent to all wrist modules
            switch(Read_Bytes[1]) // WAP_Command
            {
                  case(WAP_CHECK_EMERGENCY):
                        if(Emergency_Event)
                        {
                              Allocated_Channel = Read_Bytes[2];
                              Transmit_Data(WM_EMERGENCY,
                                          Allocated_Channel);
                        }
                  break;
                  case(WAP_BEACON):
                        No_Beacon_Cnt = 0;
                        // Enable Good Connection LED
                        GOOD_CONNECTION_LED = 1;
                  break;
            }
      }

      // Setup WM to receive on allocated Beacon channel
      Configure_Transceiver(RECEIVE, BEACON_CHAN);

      IFS1bits.INT1IF = 0;    // CLEAR INTERRUPT INT1 FLAG
}

void __attribute__((interrupt, no_auto_psv)) _ADC1Interrupt(void)
{// ADC1 Interrupt
      long Result_32 = 0;
      int Result_16 = 0;
      long Accel_Scaled_SF = 20122;
      int Accel_Scaled_Bias = 4935;
      long Temp_Scaled_SF = 12000;
      int Temp_Scaled_Bias = 1000;
      long Vcc_Scaled_SF = 6631;
      int Vcc_Scaled_Bias = 0;


      // Convert Accel. - 0.001
      Result_32 = ADC1BUF0 * Accel_Scaled_SF;
      Result_16 = Result_32 >> 13;
```

```
        Result_16 -= Accel_Scaled_Bias;
        Accel_Value = Result_16;

        if(BUTTON_1) // Check BUTTON_1 state
        {
              Emergency_Event = 0;   // Clear Emergency Event flag
              ACCEL_LIMIT_LED = 0;   // Turn off ACCEL_LIMIT_LED
        }
        else if(Accel_Value > 2000)  // Check Accel Limit 2g
              Emergency_Event = 1;   // Set Emergency Event flag
        else if(Accel_Value < -2000) // Check Accel Limit 2g
              Emergency_Event = 1;   // Set Emergency Event flag


        // Calculate Average Temperature - 0.05
        ADC_Average_Count++;
        ADC_Buffer_Sum += ADC1BUF1;

        if(ADC_Average_Count >= ADC_Average_Number)
        {
              ADC_Average = ADC_Buffer_Sum/ADC_Average_Number;
              ADC_Buffer_Sum = 0;

              // Convert Temperature
              Result_32 = ADC_Average * Temp_Scaled_SF;
              Result_16 = Result_32 >> 13;
              Result_16 -= Temp_Scaled_Bias;
              Temp_Value = Result_16;
        }

        if(ADC_Average_Count >= ADC_Average_Number)
        {
              ADC_Average_Count = 0;
        }


        // Convert VCC - 0.002
        Result_32 = ADC1BUF2 * Vcc_Scaled_SF;
        Result_16 = Result_32 >> 13;
        Result_16 -= Vcc_Scaled_Bias;
        Vcc_Value = Result_16;


        IFS0bits.AD1IF   = 0; // Clear the A/D interrupt flag bit
}
//***************************************
// END ISRs
//***************************************
```

Appendix C

Guardian.h file used in WAP and WM Embedded C code

```
#define Fosc 80000000 //Internal Clock Freq.
#define Fcy  Fosc/2
#define Baudrate 115200
//Define UART buadrate register value
#define BRGVAL ((Fcy/Baudrate)/4)-1

#define TRANSMIT  1
#define RECEIVE   0

//Define inputs/outputs
#define GOOD_CONNECTION_LED LATBbits.LATB6      //LED_2 - WM
#define DATA_RECEIVED LATBbits.LATB6            //LED_2 - WAP
#define ACCEL_LIMIT_LED LATBbits.LATB7          //LED_3 - WM
#define RF_ACTIVE LATBbits.LATB7                //LED_3 - WAP
#define BUTTON_1 PORTBbits.RB8

//Port Names for transmitter
#define CLK LATAbits.LATA2
#define CS LATAbits.LATA4
#define TX_DAT LATBbits.LATB3
#define RX_DAT PORTBbits.RB3
#define RX_DR1 PORTBbits.RB4
#define CE LATBbits.LATB5

#define BEACON_CHAN           0x01
#define BROADCAST_WM_NUM      0xAA

//Max PAYLOAD_LEN is 29 bytes
#define PAYLOAD_LEN 10  //Set payload length in number of bytes

//Define Wireless Access Point Commands
#define WAP_CHECK_EMERGENCY         0x10
#define WAP_DATA_REQUEST            0x20
#define WAP_BEACON                  0x30
#define WAP_CLEAR_ACCEL_EVENT       0x40


//Define Wrist Module Commands
#define WM_EMERGENCY                0x01
#define WM_SOH_RESPONSE             0x02
```

Appendix D

General Information

Sample of logged patient vital signs data

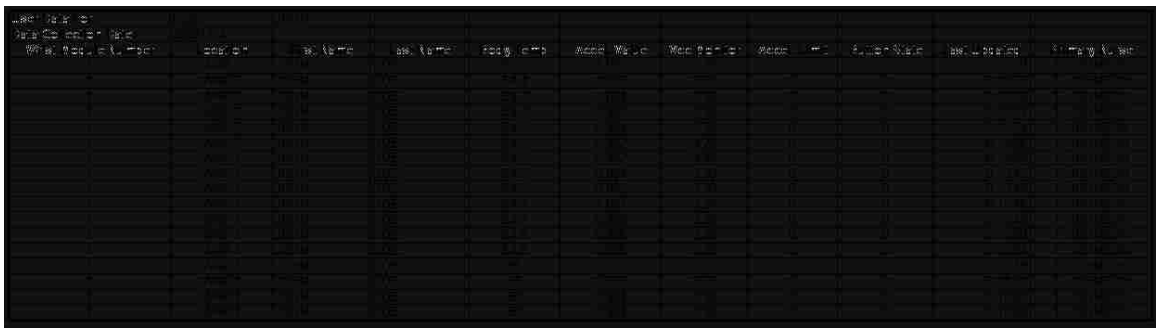


Figure 26: Sample Log Data

Sample of Guardian database directory structure.

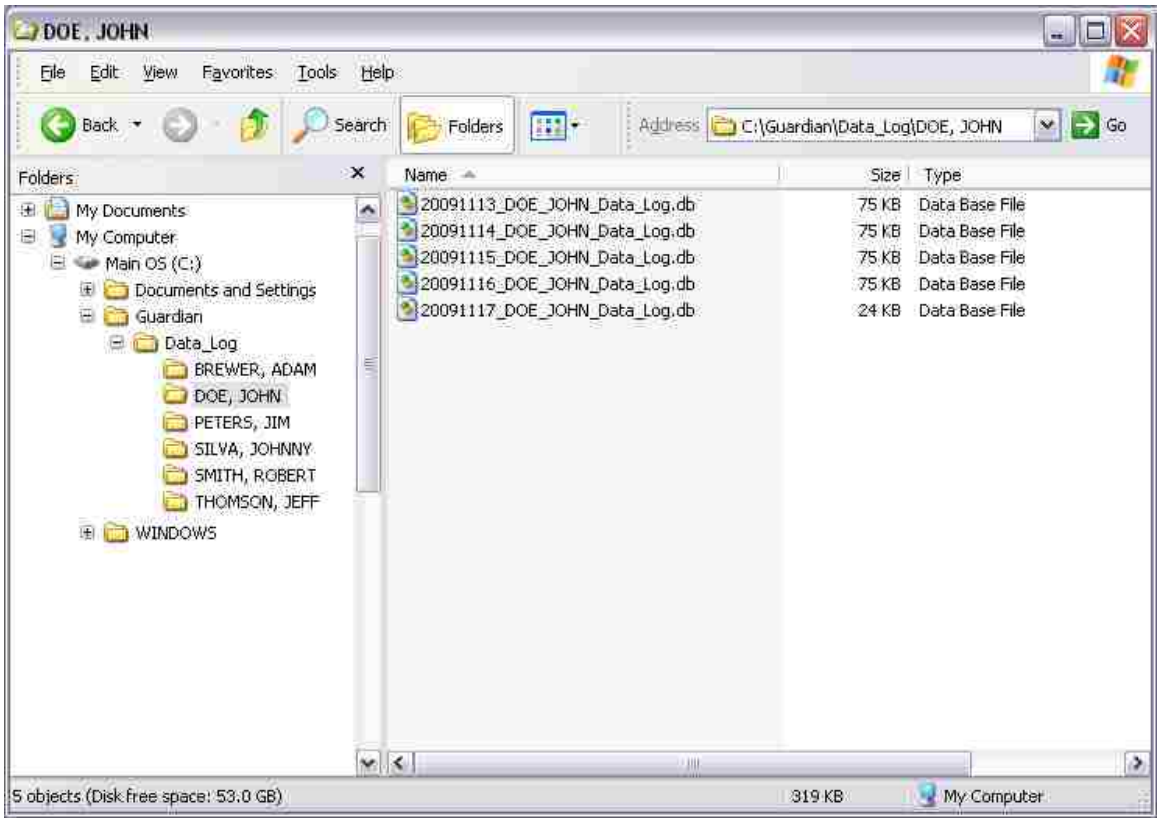


Figure 27: Sample Database Directory Structure

REFERENCES

[1] D. Piazza, N.J. Kirsch, A. Forenza, R.W. Heath, and K.R. Dandekar, "Design and Evaluation of a Reconfigurable Antenna Array for MIMO Systems," IEEE Transactions on Antennas and Propagation, vol. 56, issue 3, pp. 869-881, Mar. 2008

[2] Zhao Youping, Mao Shiwen, J.O. Neel, and J.H. Reed, "Performance Evaluation of Cognitive Radios: Metrics, Utility Functions, and Methodology," Proceedings of the IEEE, vol. 97, issue 4, pp. 642-659, Apr. 2009

[3] J.E Cabral. Jr., and Yongmin Kim, "Multimedia systems for telemedicine and their communications requirements," IEEE Communications Magazine, vol. 34, issue 7, pp. 20-27, Jul. 1996

[4] Y.B. Choi, J.S. Krause, Seo Hyewon, K.E. Capitan, and Chung Kyusuk, "Telemedicine in the USA: standardization through information management and technical applications," IEEE Communications Magazine, vol. 44, issue 4, pp. 41-48, Apr. 2006

[5] M. Krol, "Telemedicine," IEEE Potentials, vol.16, issue 4, pp. 29-31, Nov. 1997

[6] O.R.L. Sheng, P.J.-H Hu, Wei Chih-Ping, and Ma Pai-Chun, "Organizational management of telemedicine technology: conquering time and space boundaries in health care services," IEEE Transactions on Engineering Management, vol. 46, issue 3, pp. 265-278, Aug. 1999

Datasheet References:

[D1] Microchip Technology Incorporated, "PIC24HJ32GP202/204 and PIC24HJ16GP304," Datasheet, Preliminary Release, rev. B, Jun. 2008

[D2] Nordic Semiconductor ASA, "Single chip 2.4 GHz Transceiver nRF2401A," Product Specification, rev. 1.0, Dec. 2004

[D3] Analog Devices Incorporated, "Small, Low Power, 3-Axis ±3 g iMEMS© Accelerometer ADXL330," Datasheet, rev. 0, Mar. 2006

[D4] Analog Devices Incorporated, "Low Voltage Temperature Sensors TMP35/TMP36/TMP37," Datasheet, rev. E, Aug. 2008