5-2004

A Fleet Tracking System using G.P.S. and Radios

John Mulholland
Utah State University

Matthew Warner
Utah State University

Matthew Waldron
Utah State University

Eric Widdison
Utah State University

A Fleet Tracking System using G.P.S. and radios

By

Matthew Warner
Matthew Waldron
John Mulholland
Eric Widdison

Thesis submitted in partial fulfillment
of the requirements for the degree

of

UNIVERSITY HONORS
WITH DEPARTMENT HONORS

in

Computer Engineering

Approved:

Thesis/Project Advisor	Department Honors Advisor

Director of Honors Program

UTAH STATE UNIVERSITY
Logan, UT

Year of Graduation 2004

Honors Senior Project Final Report:

A Fleet Tracking System using G.P.S. and radios

Presented by:
Matthew Warner
Matthew Waldron
John Mulholland
Eric Widdison

November 27, 2002

Senior Project
What I did
John Mulholland

My job in the senior project team was to provide all of the high level software. This consisted of providing a Graphical User Interface (GUI), providing the interfaces for the user to input commands, the execution of commands, and communicate with the PK-96 through the serial port. I was also the final tester.

In order to provide the GUI, I used Microsoft Foundation Classes (MFC). I acquired some knowledge from working at the Space Dynamics Lab (SDL). The rest I had to learn. The GUI displayed information about where the boats were, where the fish had been caught, and a map of the area. I created most of this code myself but was able to find some free code on the internet which I slightly modified to provide the zooming in and out of the map.

There were many options that the user had. He could change what area his screen covered. He could also load a map and set which area that map covered. He could display waypoints at desired locations on the map. When the data was received it needed to be recorded. It was saved to a file at a rate the user specified.

The data was requested and then transmitted using RS232 protocol. I did this by modifying some code that I had used in another project. This data was maintained in a buffer until the proper data was found. The data was then passed to the rest of the program.

Due to the fact that I was doing the software, I became the final tester. I had previously tested my software using simulated data but since accepting the software is the last thing done, I was the final tester. This is always difficult because other team members often get their part done right before a deadline and that leaves me to do last minute integration testing.

I also created a terminal emulator for the people designing the hardware. This program proved very useful in providing knowledge of if the hardware was working. It was much more difficult to use the main software due to the fact that it might be my problem or theirs if it doesn't work.

As an experienced software developer I was able to advise my team leader in decisions regarding how long it takes to do things. It make really only take two hours but you should schedule more time because other problems will probably arise.

Abstract:

Doc Warner's Alaska Fishing LLC is a growing business operating out of Excursion Inlet, Alaska. Based in Bountiful, Utah, this company has experienced large amounts of growth in the past few years, and as a result has needed to make quick adaptations to accommodate larger groups of people fishing with them. Doc Warner's offers a unique experience in that they do not charter boats, but rather allow their guests to captain the boats. The number of vessels that might be fishing at any time has increased from four to more than twenty in the last three years. This large increase has made it difficult for the staff to watch over each individual boat, and the business feels that its customer service is declining from the high standard it desires to give. For example, if a boat has engine troubles, it is often over an hour before the staff is able to discover this and assist the guests. Doc Warner's would also like a way to alert their guests to current fishing "hot-spots". Other problems include guests wandering too far from the camp and potentially heading into dangerous waters, or guests staying out too late after curfew. Doc Warner's would like a system that will give them the location of each of their boats at any given time.

Table of Contents

List of Figures

Acknowledgments:

Special thanks to Scott Poulsen for helping us enter the world of radio communications and getting us started in the right direction. We also want to thank Kent Porter of Advanced Communications Inc., Ogden UT for his help in setting up the radios and allowing the use of his frequencies for testing of this project. Development of our assembly code was facilitated by the many hours of labor that Byon Garrabrandt put into designing interfaces to the PIC-E. His programs helped us to create our own custom interfaces. And finally, we want to thank Doc Warner's for funding this project, and our wives for putting up with the late nights we put into it.

Introduction

"I must down to the seas again, to the lonely sea and the sky,
And all I ask is a tall ship and a star to steer her by."
–*John Masefield*
Sea Fever

Doc Warner's Alaska Fishing LLC is a company that provides boats and accommodations to people interested in fishing the waters of Excursion Inlet, Alaska. Due to recent growth, the company has had to make many changes recently to be able to provide for an increasing clientele. This includes increasing the size of their fleet from four to twenty boats. With the increase in boats, it has become much more difficult to monitor the guests and provide for their safety. If something goes wrong, it could take a very long time for the staff to become aware of the problem, much less find the troubled boat and render aid. To solve this problem, Doc Warner's would like a way to monitor the location of each boat, and communicate with them in case of trouble.

The concept for a fleet monitoring system at Doc Warner's began in the summer of 1999 after a boat told those who were fishing with them that they would be staying out until curfew to catch the last few fish on their limit. They wandered from their original fishing area and decided to go a different direction than they had said they would be. There is a boat curfew imposed a short time before the sun sets, because as twilight approaches it is extremely difficult to see obstacles in the water and distinguish the shoreline from the water. Doc Warner's wants to allow their guests the most enjoyable fishing experience possible, while still maintaining their complete safety. Unfortunately it is difficult to convince the sometimes foolhardy guests of the dangers that are in the waters of Alaska. As the curfew came and went this boat had still not

returned and the staff became extremely worried. Boats were sent out to look for them, however as twilight approached clouds filled the sky and it darkened quickly. Fortunately the group guests in the boat returned to the pier with hungry stomachs and a large number of fish. As they returned, they passed between the search pattern being followed by the staff boats, and were undetected. The search boats continued searching for them for nearly two hours, risking their own safety by being on the water after dark. Later that night when all the boats had finally returned a discussion ensued on the dangers of the situation. The staff boats were without radios, and therefore were unable to be notified of the return of the missing vessel. Had the staff boats waited another ten minutes they would have seen the vessel and wouldn't have put themselves in danger's way. However, if the guest's boat had of suffered an engine malfunction, they could have been swept into dangerous waters and as their expected location was far from their actual one, it would have been nearly impossible to find them before nightfall.

In this situation no one was harmed, but a need for a method to track the locations of the boats was made apparent to all who were involved. Different methods were discussed for solving the issue, however the greatest ideas came from the guests themselves. In surveys of the guests conducted after their week-long stay it was discovered that most guests did not feel that they were ever placed in dangerous situations, however a number mentioned that they would have liked to have had radios for communicating with their fishing buddies or camp. Marine Band VHF radios were proposed to solve this situation, however the amount of training required for the guests to use them properly made them impractical. An opportunity was seen by one of the long-time staff members, Matthew Warner to solve all of these problems with a single solution. By implementing a fleet tracking system which would allow voice communications as

well as report the locations of the boats, Doc Warner's would be able to better serve their guests and improve the safety of their fishing experience. The system has many features that are beneficial over a simple voice radio. By constantly reporting the boat's current location, even in the event of total power failure on the boat the staff will still know where to look for them.

Due to the remoteness of the area, cell phones and radio are the only means of reliable communication. A standard commercial fleet monitoring system using the satellite telephone "Satcomm" system is prohibitively expensive. Doc Warner's would also like a system that allows limited two way communication between the boats and the home base, in the form of an alert to the boats if they've wandered too far away. A "panic" button on the boats could also alert the base of any problems that the guests are having. The ultimate purpose of this system is to transmit data between G.P.S. receivers mounted on each of the boats and a central monitoring station. The central receiver will then decode the signal and display it using a customized program written for that purpose. As all G.P.S. receivers have industry standard outputs, it is only a matter of converting the digital coordinates into an analog signals that can be transmitted, received, and converted back into a format that is readable by the hardware that will interface with the computer.

The data stream leaving the G.P.S. must be edited to contain a programmer defined command word and only the data pertinent to this project. The command word has unused status bits can be encoded at a future date to provide customized information from each boat as the company sees necessary. The edited data is then transmitted when a request is received from a base computer. The base computer will receive the data, plot it on the screen over a map of the

area, and save the data for future analysis. Voice communication will also be allowed to provide contact in emergencies.

One problem faced by the designers is that the lodge is situated within a narrow inlet that has tall mountains on either side. The main fishing areas are located on the far side of these mountains. As a result, line of sight transmissions are impossible. In order to solve this, one proposal has been to place a repeating station on a small island located at the mouth of the inlet. This island has line of sight coverage to both the fishing areas and the lodge. Preliminary testing carried out in the summer of 2001 showed that using commercially available amateur radios and repeaters operating on a business band frequency could be used. This testing also showed that the repeater is an effective solution. Further testing with higher wattage radios in the summer of 2002 found that by using higher wattage radios adequate coverage can be obtained without a repeater.

To distinguish between the signals from each individual boat each boat will be encoded with a specific digital signature. The "receiving" antenna on the computer will transmit this code with a hand shake signal to each boat as it becomes necessary to locate each of them. Each boat will then recognize when its signature has been transmitted, and respond by transmitting the appropriate data.

Doc Warner's would also like their system to have the ability to track where different types of fish are caught by their guests. This data would be combined in a database in the central computer. When compared with weather reports and tide charts, the data would allow a more accurate prediction of where fishing will be best at a given time and place. The database could be accessed to show the demographics of the fish caught over a given period of time, thus

identifying immediate hot spots and allowing Doc Warner's to more accurately track fish migration through the fishing grounds. A system that monitors current boat status, such as fuel level, and motor on/off could eventually be added to allow for more complete monitoring of the boats.

Due to the size of this project a number students were involved to accomplish it. One student worked on the development of the software for the base computer and interfacing it through a serial port. Another student handled the many communications problems presented in this project. The third team member did the micro-controller programming. The fourth team member handled and interfaced of the many different hardware components, as well as assisted the other three when addition assistance was required.

The budget goal for this project is to implement the system at a cost of less than $1000.00 per boat. There will be additional costs involved with the repeater and the base computer, we hope to keep these additional costs below $4000.00.

The reusability of this system in other applications is also being explored by the team to potentially market this product. There has already been interest in it expressed by other companies.

This paper discusses the development and implementation of this system. It describes in detail the steps that were taken in the design of this project. It also includes flow charts describing the advances in the design as hardware was chosen and other decisions made. Specifications for some of the devices are included. Due to the size of the software it is not found in this text, however a simple flowchart of the computer software is.

Preliminary Design

Problem Analysis

There are two main goals that Doc Warner's needed accomplished. First of all, Doc Warner's desired to increase the safety of their customers. This was to be done by having each boat transmit its position while maintaining the ability to communicate by voice to people within the Lodge. There are over twenty boats in the fleet at Doc Warner's and each boat will need both data and voice communications. Secondly Doc Warner's desired to improve the fishing experience for each guest. Doc Warner's will accomplish this by monitoring the location and type of fish caught by the guests and enabling the boats to communicate this information to each other. A log will be kept detailing the species and location for each fish caught by the individual boats. This data needs to be displayed in a real time manner so that Doc Warner's will be able to alert the guests of the current fishing "hot spots". Access to the fish log history will also make it possible to discover trends in the fishing locations.

In order to accomplish these tasks software needed to be created which allows the user to track the boats and other information in real time on a map of the area. This software will also allow the user to access the historical fishing data. All communications between this software and outside hardware will be through the serial port.

In order to know the location of the boats and where they catch fish, a G.P.S. receiver must be placed on each boat. This G.P.S. signal must be received by a device, decoded by it, and then transmitted to the computer. This transmission will be accomplished using business band UHF radios. We needed to determine what type and wattage of radio to use and whether or not a

repeater is necessary to provide adequate coverage of the fishing area. A method was also needed to interface the radio to the computer. Protocols to handle all the communication between different subsystems had to be developed or learned.

Other decisions made by the design team included what data to transmit and how often each boat will communicate with the base computer.

Summary of Specifications

- The cost is to be minimized. There are many similar systems currently in use in the world, but they are prohibitively expensive to a small business like Doc Warner's.
- Marine quality materials must be used. Any devices used must have integrity against weather conditions and the corrosive properties of salty sea water. There should be a minimum of student manufactured parts in order to promote robustness.
- The product must be user friendly. It should operate autonomously with no need for outside interference. However, it should be able to compensate for any interference caused by the users.
- The project conform to FCC regulations and laws. Frequencies will be licensed for the area in which the radios will be used.
- The system must cover the entire fishing area that is accessible by the guests. As this area contains many small islands and narrow bays it must be determined which radios are sufficient in strength, and if a repeater must be installed to cover any dead spots where the base computer can't reach.
- The product must not compromise the safety of the guests. It must not use excessive amounts of battery power thereby stranding the guests with dead batteries.

- The base computer must have a variable frequency between communications with the boat. This is to allow the system to be calibrated to operate at maximum efficiency in the field. It also allows the system to be used in different settings in the future.
- Interface to the radios as well as home base computer need to be upgradeable as to what data is transmitted. This is to allow the system to be applied in different settings in the future.
- The system needs to have a method for recording the location where fish are caught, plotting it real time on the base computer, and store these locations for future data analysis. It should distinguish between halibut and salmon.
- The system needs to have "Panic" button of some sort. This is a method whereby a user in distress can communicate to the home base. Rather than put an actual button on the boats this criteria has been changed to allowing voice communication without permanently disrupting the data stream from the boats.

Discussion of Main Features of the Design Problem

This design problem was initially broken down into three subsystems. These consisted of the G.P.S., Radio, and Base Computer subsystems. While each of the subsystem of this project contains a number of different and unique features of the design problem, they are all intricately interwoven by the common methods they use to communicate between them.

The G.P.S. Subsystem presented particular challenges because it required the learning of two distinct protocols. In order for the boats to "know" their location, they must each have a G.P.S. receiver on board.. This G.P.S. receiver must communicate with a base computer located at Doc Warner's fishing lodge through two-way UHF radios. Most G.P.S. systems output a

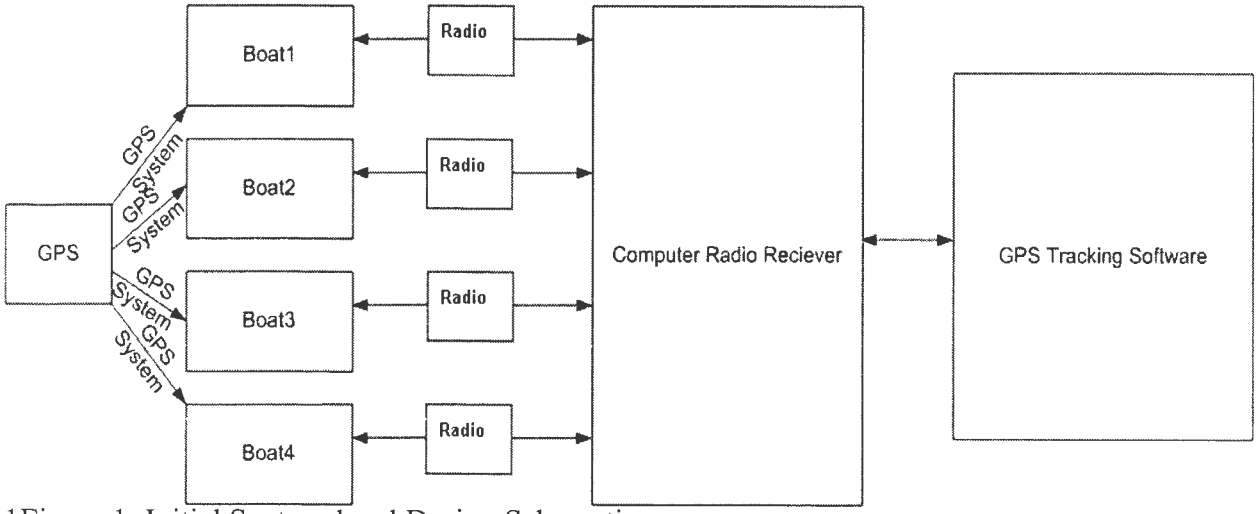
standardized NMEA data string using a RS-232 protocol. Both the NMEA data string and the RS-232 protocol needed to be learned. Only certain portions of the NMEA string are relevant to this project. As a result decisions must be made regarding whether or not to keep extraneous portions of the data string, or transmit the entire thing. The boats require two buttons to be mounted on them. These buttons will correspond to the specific species of fish most frequently caught at Doc Warner's and, when pressed, will cause the current location to be transmitted to and recorded on the base computer as a location where that specific species of fish was caught. This creates a potential for multiple boats to transmit simultaneously, or cross talk. A method to prevent this must be designed. The frequency with which each boat communicates with the base computer must also be determined, so that adequate memory storage can be installed on each boat for anything stored there. The G.P.S. must both be programmable and have separate I/O connections for these buttons, or a device is required that is capable of communicating with the G.P.S. receiver, interpreting its output, and forwarding the compressed data to the radio. When many of the challenges presented by this subsystem are overcome they will define how other challenges in the other subsystems will be handled.

The Radio Subsystem compromises the radios and the protocols they used for communication. This subsystem required the determination of what type and wattage of radio to use to provide adequate coverage of the fishing area. In the event that the radios could not reach every location the placement of repeaters was considered. Most important is the need for a reliable communication protocol between the radios. As this protocol will be transmitted from one radio to the other it will be difficult to debug. As a result this protocol must be both well

designed and well understood. This radio system should also function as an emergency contact for the Lodge in the event any boat suffers a catastrophic failure.

The Base Computer Subsystem consists of a method or device to send and receive signals between the radio and the base computer. This subsystem must organize the receiving pattern to prevent crosstalk while still providing adequate coverage of the fleet of boats. It must also allow for voice communication to be carried on the same radio channel without interfering with the transmission of data. In the event the data transmission is interrupted, the computer should be able to automatically continue its normal functions when the radio channel is clear. Software will interpret the received signals as G.P.S. coordinates from each boat and plot them over a map of the area. It should distinguish between the coordinates of fish caught and the coordinates of the boat's current location on the map. The map will only display current locations of boats and fish caught within the last few minutes or hours as defined by the end user. This software will have to be custom designed and written in order to have the ability to carry out the desired functions and interface with the radios using the serial port of the computer. The computer's serial port communicates using the RS-232 protocol. A method or device will need to translate the inputs to and outputs from the radio to this format. The computer will also store the locations where Doc Warner's guests have traveled and have caught fish. This will enable the data to be analyzed at a later date to discover fishing trends. Many parameters, such as the sampling rate and delay time between retransmission attempts, will be adjustable by the end user, to better customize the project in the field for maximum efficiency.

Summary of Technical Approach



1Figure 1: Initial System-level Design Schematic.

We began the project by drawing a simple flowchart of the necessary subsystems for this project (Figure 1), and then we looked at what would be required to design each subsystem. We refined the flowchart as we found components that would work to solve our problems. We researched industry to find parts that could meet our objectives, and then decided (based on cost and availability) which parts we wanted to use and which ones we wanted to build.

Initially we recognized the extreme difficulty in designing our own G.P.S. receivers and radios. As a result we chose to implement those portions of this project with existing technologies. A method was needed to control when the G.P.S. data would go to the radio, and edit out all but the desired portions of the G.P.S. data string. To reduce the number of parts in the system we first investigated programmable G.P.S. receivers but we found them to be prohibitively expensive. This created a need to insert an interfacing system capable of reading G.P.S. strings, interpreting G.P.S. strings, storing them for a specified amount of time, and then transmitting them to the radio.

Terminal Node Controller

Building our own interface between the radio and G.P.S. instead of using an existing one was the lowest cost alternative available. We began researching the feasibility of using a microcontroller to accomplish the task. As we considered many different makes and models of microcontrollers, the one we found that would best have suited our plans was the PIC18C442. We then undertook to learn the PIC assembly language, and studied this chip in particular to understand how we could handle the two different type of I/O we would be needing from our interface device. The G.P.S. outputs in a standard RS-232 format, while the radio is completely analog. We researched radios with a digital I/O port; however they proved to be prohibitively expensive. At this point we had to determine the protocol we would use for communication over the radio. As we were planning to build the system in it its entirety we realized that we could define our own method for the analog communication. We considered a method using a single frequency that would turn on and off, much like a digital signal. After further consideration however We realized the best method was using a frequency switching pattern. Using a pattern where it would switch between two frequencies would reduce the erroneous transmissions due to static or interference. We even designed and simulated high order filters to interpret these frequencies.

As we continued researching the capabilities of the PIC18C442 microcontroller we encountered a number of obstacles. First of all, we would need to build a reliable circuit board to contain it and the elements needed for it to operate. Secondly, it has only a single I/O port, which would be used for the G.P.S. Interfacing with the radio would require a series of special circuits to transform the analog wave forms to digital signals and vice versa.

We researched the methods available to us for manufacturing circuit boards on campus. We discovered that while there is the machinery to do so, there are few faculty who actually know how to use it. Also, our lack of experience with the PIC18C442 made it difficult to know how and where to insert resistors and capacitors to match impedance and prevent damage to the circuit. As a result, using it would significantly increase the amount of time required for the project. Although the use of the microcontroller was initially the least expensive method for implementing this project, we soon realized that the amount of testing required of our circuits in order to achieve the robustness desired was monumental. The time requirements became more than the scope of this project allowed.

Realizing that the radio channel would also be desired for voice communication as well as data communication we worked on designing filters to split the signals that would be sharing the channel. We investigated the normal frequency range of human speech. Finding that it normally falls between 300 to 3000 Hertz, we considered using the lower band of the radio channel, using low-pass and high pass filters to separate the signals. This didn't provide sufficient bandwidth for the data payload that we needed. We also considered using a narrow-band range (or two) somewhere in the upper frequencies of the radio channel. The filters in this case were not responsive enough for frequency shift keying (where two frequencies, f0 and f1, are used to represent bit values of 0 or 1), and the filters couldn't remove enough noise for pulse code modulation to be successful. Furthermore, as we researched further the radios available we discovered that the radios filter out all frequencies outside of the normal range of human speech as a form of noise rejection. The best solution we could find was to develop a protocol for polling the boats from the base computer so that we can prevent any boat from having to wait too

long for the channel to be free before it uploads its data. The computer would need to track who had gone the longest since the last upload, as well as detect when the channel was free to use.

To handle the I/O with the radio we began searching for existing methods in use. We found some inspiration in a device called the TinyTrakII, built and distributed by Byon Garrabrant (www.byonics.com). The TinyTrakII is a small device that interfaces with G.P.S. and transmits at set intervals to the radio using a Ham radio standard protocol called AX.25. It outputs to the radio using a specialized bridging circuit which allows four pins changing at certain intervals to approximate a sine wave. Unfortunately the TinyTrakII is unable to handle input from the radio, and so it not suitable for our design concept.

The AX.25 protocol is similar to the idea we had developed independently, in that it switches between two frequencies. However the similarities end there. As a packet communication protocol it has built-in complex checksums, as well as identifiers of who sent the packet and who is supposed to receive it. It even includes a list of repeaters that are asked to pass it on, and which ones have already passed it. A large amount of study was devoted to understanding this protocol. After studying AX.25 we decided to use it in our device for a number of reasons. The Ax-25 is an industry standard, allowing us to use existing technologies in our project. Our protocols lacked the robustness of the AX.25, particularly in dealing with interference and lost packets.

With the discovery of the TinyTrakII and the AX.25 protocol, we began looking at other existing solutions to our interface problem. The TinyTrakII was designed to have Terminal Node Controller or TNC on the receiving end that would interface with the computer. TNC's are used by the Ham radio world as a simple interface for computers and radios. They handle

encoding, checksum, and even are capable of digipeting (repeating a signal sent from one radio so it will be heard by another) and storing mail. Most TNC's also have the ability to detect when the radio channel busy and wait until it is unused before transmitting their data. While most TNC's are both too expensive and too specialized to be adapted to our use, we began searching for one that we could program to do what we needed. After number phone calls and e-mails to various manufacturer's we were eventually recommended to the Tucson Amateur Packet Radio web site (www.TAPR.com) where we found a small programmable TNC called the PIC-E for eighty dollars. After serious consideration we decided that the PIC-E would not only be the most cost effective way to go, but also the quickest. We then redirected our focus to learning the assembly language used in the PIC16F84 which is the heart of the PIC-E. TAPR has available on their website assembly programs for the PIC-E to both interface with a G.P.S. and a radio through the AX.25 protocol, however they use the available RAM very inefficiently. It became necessary for us to write our own versions of these programs to accomplish our design objectives. We were able to put our research with the PIC18C442 to use in writing the code for the PIC16F84 as there are few differences between their languages.

Due to the exposed nature of the PIC-E we were also able to determine two unnecessary devices on the circuit board (a switch and an LED) that we can connect to external buttons to be pressed when the fishermen catch a salmon or halibut.

Our next step was to determine how to interface the Radio and the base computer. At this point we knew we would need something that could interpret and send the AX.25 protocol as well as communicate with the computer via a serial port using the RS-232 protocol. It became obvious that we needed another TNC; however we were faced with a serious design decision

determining whether to use a commercial model or the programmable PIC-E. One commercial TNC we considered was Timewave Inc.'s PK-96. The PK-96 has a buffered input and output to the computer. It also has a built in channel busy detection circuit. This meant that in the event the radio was busy, the TNC would hold data until it was free to transmit. By listening for channel activity through the PK-96 the computer gained the ability to wait until the channel was free before polling a boat. The TNC also has the convenient function of holding data received from the radio until the computer is able to receive it. This allows for lag in the computer response time, thereby preventing possible errors. The PK-96 also gives us an easy method for testing the PIC-E programs we've written by providing a method for interpreting the signal that we know functions as it should.

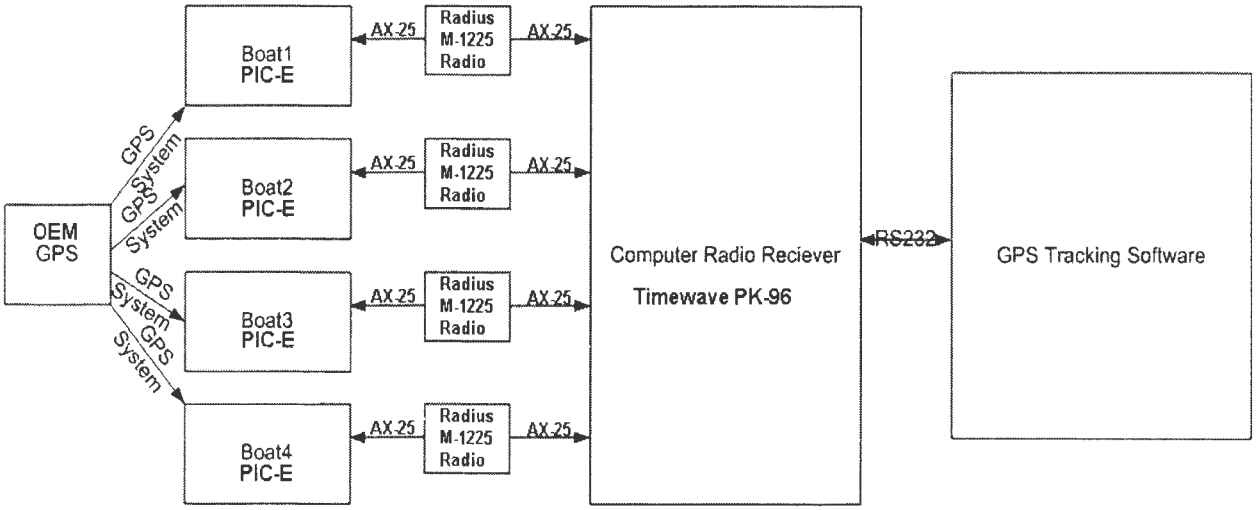
Radio

We spoke with the owner of the local Motorola service shop in Ogden, Utah and discussed our project and radio needs. He recommended using a Radius series mobile radio. The Radius mobile radio series include a number of different radios, each programmable with a different number of channels. The radio with the largest capacity is the M-1225. It can have up to twenty channels programmed into it. The cost between the M-1225 and the other models was negligible and after speaking with the customer we decided that for future expansion possibilities we would use the M-1225. The M-1225 Radius mobile radios are tunable between ten to forty watts. One serious concern was whether or not these radios could reach all of the fishing locations without the need for an expensive repeater installed on a central island in the fishing areas. To test this we installed three radios on boats at Doc Warner's in Excursion Inlet Alaska, and another radio where the base computer will be located. We then tested the range to different

locations using different antennas of voice communication. We discovered that a forty Watt radio with a 0 dB ground plane generating antenna has adequate coverage. We decided for added signal strength that all of the boats that can will use 5 dB antennas that must be mounted on a separate ground plane. It was also determined that the deep cycle marine batteries and the generators in the motors used on the boats at Doc Warner's are sufficient that the drain of the radio does not threaten the safety of the guests. The Motorola service shop owner also recommended the UHF frequencies that would work best in the environment the system is intended to be placed in. We had to purchase specific frequencies for the area they are intended to be used in, and we obtained permission from the Motorola service shop owner to use one of his local frequencies to test our system.

G.P.S. Receiver

In researching G.P.S. systems we found first of all that a programmable G.P.S. receiver with antenna is quite costly. However, a standard G.P.S. receiver (with built in antenna) can be purchased for under $200.00. These systems are completely self- contained, weatherproof, and output a standard NMEA G.P.S. string every second without the need for configuration. Due to the ease of use, these units seemed ideal for our project. However later interviews with professionals in the field yielded information regarding OEM G.P.S. units with waterproof patch antennas that could be installed at half the cost and even allow for dual antennas to create a differential G.P.S. system with greater accuracy. As this D.G.P.S. system is beyond the scope of our project we did not choose to install that system, however it could be considered for future development in the project.



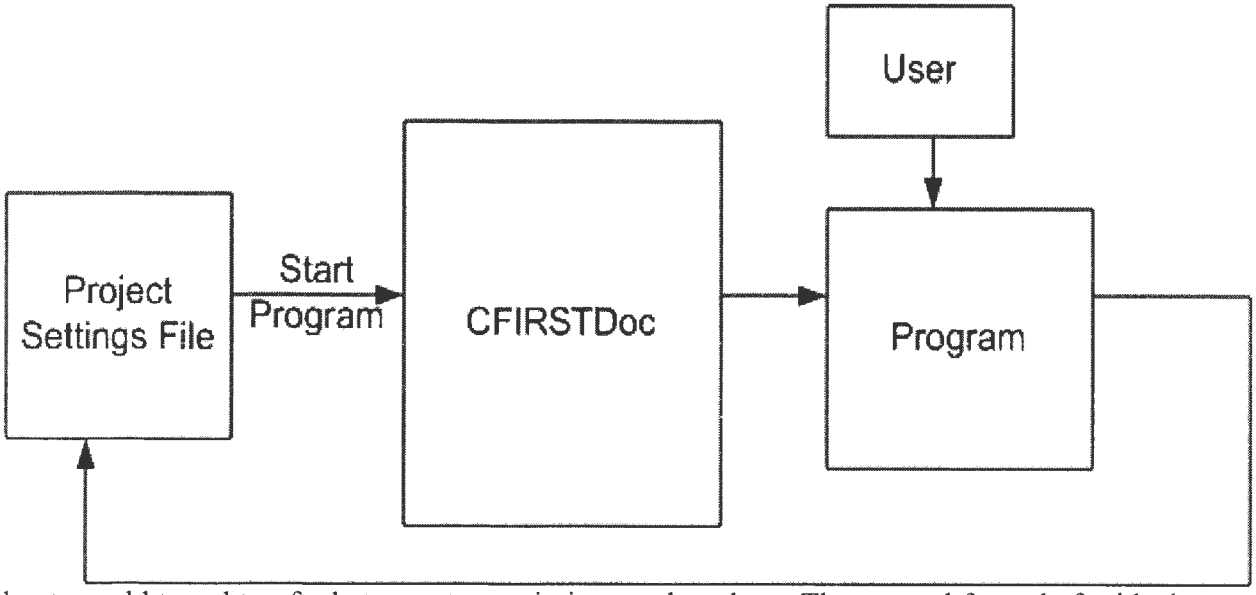
convey commands for functions (such as polling, data transmission, and retransmission), compress and block out data for transmission, identify and resolve errors, and handle frame control. In designing this protocol, we also tried to identify some bandwidth for transmission and design filters to implement this physical link layer. Unfortunately, our early protocols could barely deal with several of these issues, and the realizable bandwidth for a shared data/voice channel was not sufficient for our needs. As we identified TNC's as a viable interface between the radios and the data devices (G.P.S. receivers or the base computer), we recognized the AX.25 protocol as a much more robust solution for transmitting our desired data. As a result, our original protocol was streamlined to include only a command word with identifier flags for the data, and the compressed G.P.S. data stream.

The available program memory and RAM on the PIC-E chips was very limited. As a result, we needed an algorithm that gave good compression without requiring significant software or memory to implement it. The resulting protocol consists of blocks made up of one byte of command word (for polling boats or requesting a re-transmission) or 9 bytes of data. Data blocks can then be transmitted consecutively in an AX.25 packet. The command byte is

2Figure 2: Final System-level Design Schematic

actually found in each block, either as the entire block or to identify the data contents. A command block can take on a hex value of C2 (to request retransmission of last packet) or C3 (to poll a boat). If the command byte is in a data block, then it can take on three hex values: 80 (current location), 82 (location of salmon catch), and 83 (location of halibut catch). The remaining eight bytes of the data block consists of the latitude and longitude NMEA ASCII data compressed into binary format. That amounts to one byte for degrees, one byte for minutes, and two bytes for fractions of minutes (NMEA output takes the form DDD:MM.mmmm). The hemisphere is stored as the least significant bit of the minutes byte (so that byte reads 0MMMMMMH in binary).

The remaining controls are handled in the AX.25 protocol, which includes station identifiers for the sender and recipient (as well as for any digipeater stations), block framing, and error detection and correction. In introducing the AX.25, we found an interesting effect. One key design concern was frequency of polls. If the polls are a few minutes apart, then we can keep reasonably close tabs on the boats. If the delay between polls exceeds ten minutes, then the



boats could travel too far between transmissions and get lost. The general formula for ideal polling time is

(# of boats)*((propagation delay)+(data payload)/(bandwidth)).

With our original protocol, this amounted to (20 boats)*(10 sec/boat+(50 bytes/boat)/(300 bits/second)). This gave us polling times in the neighborhood of 4 minutes, which is good. The AX.25 protocol has a much greater payload (about 100 bytes/boat), but it also uses a faster transmission rate (about 1200 baud, which is partly due to not sharing the bandwidth with voice communication), so the actual polling time is reduced.

Software

The design of the computer software began with a flow chart showing the basic functionality of the program (Figure 3)

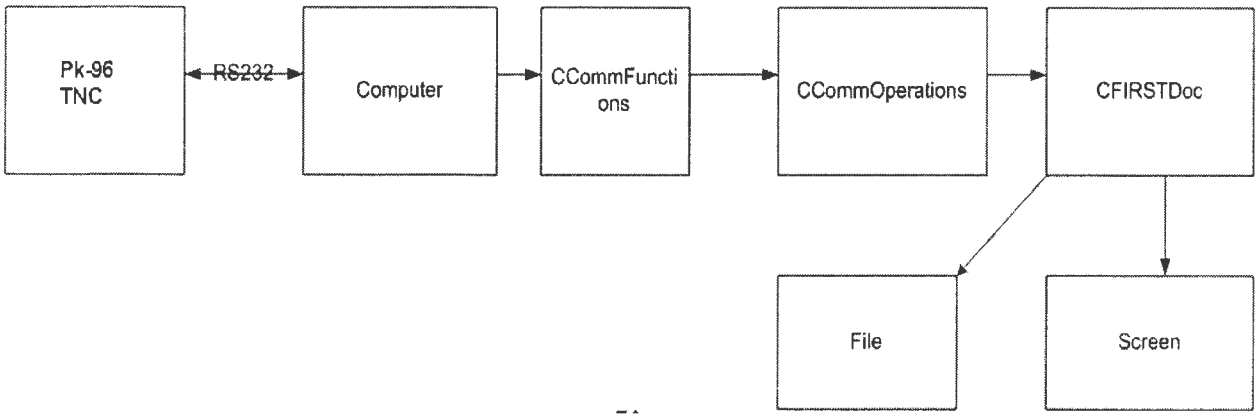
. Next, it was decided what data structures and graphical tools would be necessary to carry out these functions. Then the necessary changes to the data structures to make the program work correctly were made as the program was written. As the code was developed, it was tested for functionality. The final code needed to receive data from the TNC, process it and then output it to the screen and to a file so that a user could track the boats in real time and have a history of fishing activity. This flow of data is shown in Figure 4. Finally, features were added and tested to assure that they functioned correctly.

3Figure 3: Software System Flowchart

Summary of Preliminary Design Solution

By using an OEM G.P.S. system we were able to reduce the costs of the hardware needed for the project as well as size of the exposed components. This is important for future developments that may include Differential G.P.S. systems, and wearable vests with a built in tracking system.

Through the PIC-E TNC we were able to communicate with the RS-232 protocol to the G.P.S., parse the desired portions of the G.P.S. string, convert the data from ASCII to a binary representation, encode according to the AX.25 protocol, and transmit it to the radio. We are also



able to receive commands through the radio in the AX.25 format and react to them. By streamlining existing code, and addition of our own we were able to allow the PIC-E to store up to five locations before transmission.

The AX.25 protocol is a reliable method for transmitting data from radio to radio. By using AX.25 in our system we were able to take advantage of existing technologies for testing. It also made available many sources for help on our code.

The M-1225 Radius radios allow for future expansion to a wide number of frequencies. They are also powerful enough to transmit a forty Watt signal which can reach the base computer from any of the fishing grounds. Due to the standardization of the Motorola radios, the use of the M-1225 Radius also allows for other radios to be substituted in its place, even portable hand-held radios. This allows for many future developments and branches in this system.

Timewave Inc.'s PK-96 TNC provides a simple method to send and receive signals from the radio to the computer. It buffers both the incoming and outgoing signals from the computer and prevents transmission of the data when the channel is busy. At $200 it is relatively inexpensive to install on the base computer, however it's too bulky and the cost is too high to install it on each of the boats. It also provided an easy method for testing the signals being sent by the PIC-E TNC during development.

The cost for parts on each boat is around $850. For the base computer costs approached $1200. This was well below the goals set at the beginning of this project.

System Design

Implementing the PIC-E Through Code

The PIC-E was chosen as the means for interfacing the G.P.S. and the radio in a sufficiently economic manner. The PIC-E was designed for Ham radio enthusiasts as a means to communicate digitally over ham frequencies. The PIC-E uses the MX614 modem to interface the PIC 16F84 with the radio. In our situation the business band radio is easily used in place of the Ham radio. The PIC-E also has a serial port to connect to a G.P.S. output. It was designed for use with the Tripmate, but was easily converted to be able to handle the Garmin 35LP that we are using. The Tripmate and the Garmin and both G.P.S. systems that operate under NEMA standards but are sold by different manufacturers. The difference between the two will be explained later in the paper.

The PIC-E uses a PIC 16F84 to connect to the various components, and make sense of the incoming signals. It has the following ports as described in the Microchip Datasheet.

PORTA is a bi-directional I/O port:

Pin RA0 is port 17 and is used for I/O with TTL voltage levels.
Pin RA1 is port 18 and is used for I/O with TTL voltage levels.
Pin RA2 is port 1 and is used for I/O with TTL voltage levels.
Pin RA3 is port 2 and is used for I/O with TTL voltage levels.
Pin RA4/T0CKI is port 3 and is used for I/O with ST voltage levels. It can also be selected to be the clock input to the TMR0 timer/counter. Output is open drain type.

PORTB is a bi-directional I/O port. It can be programmed for internal weak pull-up on all inputs. All pins can be used for input and output and TTL voltage levels. RB0, RB6, RB7 can also accept ST voltage levels.

Pin RB0/INT is port 6. It can also be selected as an external interrupt pin.
Pin RB1 is port 7
Pin RB2 is port 8
Pin RB3 is port 9
Pin RB4 is port 10. Interrupt on change.
Pin RB5 is port 11. Interrupt on change.
Pin RB6 is port 12. Interrupt on change. Serial programming clock.

Pin RB7 is port 13. Interrupt on change. Serial programming data.

The modem converts the received radio signal to a digital signal. The rxd line from the modem carries a TTL logic signal, which toggles when receiving a 0, and remains at the same level when receiving a 1. The modem is configured to receive and transmit at 1200 bps, though it has the ability to work at higher speeds. The rxd line from the modem was connected to RB4. Port B was chosen for the ability to use a function provided by the microprocessor that recognizes a change in any port B pin. The RBIF flag is set to indicate a change. This function is used to check for a toggle on the rxd line from the MX614. RB5 is connected to txd line on the modem. This bit is used to serially output the information to the MX614, where it is converted for sending over the business band frequency. RB2 is connected to M0 and RB1 is connected to M1 of the modem. These bits are used to configure the modem for transmitting, or receiving.

The G.P.S. is connected to the microprocessor on port A. RA0 is used for the serial input and is connected to the transmit line of the G.P.S.. RA1 is designed for transmitting configuration bits to the G.P.S., though we have not used it, and have left it unconnected. This is the only interface needed between the G.P.S. and the PIC.

The PIC is connected to the radio as well. RA2 is connected to PTT_OUT of the radio. This is the Push To Talk pin. It is akin to pushing the Microphone button to talk, and needs to be asserted when transmitting.

An LED was attached to RB0, and an unimplemented function was the toggle switch attached to RA4.

The PIC-E also has an RJ-45 connector for the Microphone. This allows the radio to be used for voice communications as well as digital.

The PIC-E was exactly what we needed to accomplish this project in the amount of time allotted. All of the pieces fit together and the only thing needed was the code to implement the required tasks. These tasks, as stated earlier, are:

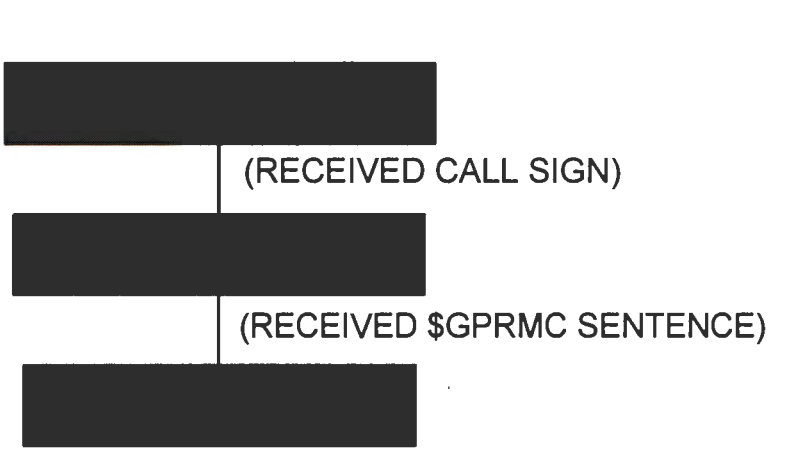
1. Track Doc Warner's 20+ fishing boats within Excursion Inlet Alaska.
2. Allow limited 2-Way voice communication between guests and home base.
3. Track the location of fish caught, and distinguish between Halibut and Salmon. Transmit the information over a business band radio.

The code that is needed will be implemented with the PIC 16F84 microprocessor. This PIC has available 1K of program memory, 64 bytes of RAM, and 8 bytes for a program stack. This is not much RAM. The G.P.S. compression algorithm compressed one location to 9 bytes. Assuming we wanted to store the location of 4 fish caught, and the current boat location, 45 bytes would be used. This left only 19 bytes for variables needed in the code.

The code needed to perform three main functions. These functions are packet receiving, serial receiving, and packet sending (Figure 5). The three main functions can be described as follows:

PACKET RECEIVING

The Packet Receiving function needs to listen for the unique call sign for the individual boat. Home Base



will be using a PK 96 Node Packet Controller to send and receive information over the radio.

The Node Packet Controller will be sending information using the AX.25 protocol. The protocol was explained by Csaba Gajdos as follows:

> The name AX.25 originates from the recommendation X.25 of CCITT, adding letter A that stands for Amateur; AX.25 is therefore Amateur packet radio link layer protocol. These are the main differences between the two protocols:
>
> * the address field has been expanded to include radio Ham calls (every Ham has an international callsign, and Hams must always identify themselves in their conversations by means of their callsign).
> * it has been added the possibility to use UI frames (Unnumbered Information), that is unnumbered packets; usually packets are numbered to restore the sending sequence.
>
> The purpose of this protocol is to define the frame structure and to set the requirements of the station that sends or receives that frame or packet. Every packet, besides the data, contains other auxiliary and control informations, so that every packet includes all needed informations to reach its destination. This addressing technique allows packet radio stations to share the same frequency without interferring with each other; every station can monitor all the traffic in the frequency channel, or filter only activity related to one or more stations, ignoring the rest.
>
> Without entering in the details, we here mention the main features of the protocol.
>
> Each packet is composed of the following fields:
>
> * FLAG: is an identifier that marks the start and the end of each packet.
> * ADDRESS: contains informations needed to route the packet, and it can contain 2 to 10 Ham calls. A secondary identification, or SSID, can be added to each call.
> * CONTROL: here are contained some control informations, as the kind of packet, the number of the packet, and much more.
> * PROTOCOL ID (PID): this field is included only in type I (information) or UI (unnumbered information) packets. It represents the kind of net protocol used.
> * INFORMATION (I): this field contains data to be sent (up to 256 bytes). OSI system superior levels can use part of this bytes as service information of their own.
> * FRAME CONTROL SEQUENCE (FCS): is a number calculated by the receiver to control the integrity of the packet; it uses algorythm HDLC. Protocols used in packet radio are normally mastered by TNC. It is also possible to implement these protocols using the right PC software; in any case all procedures are authomatic and do not require the operator's intervention. What can be seen on the screen of a PC (which, connected to a TNC, becomes a normal terminal) is a good approximation of what is passing through

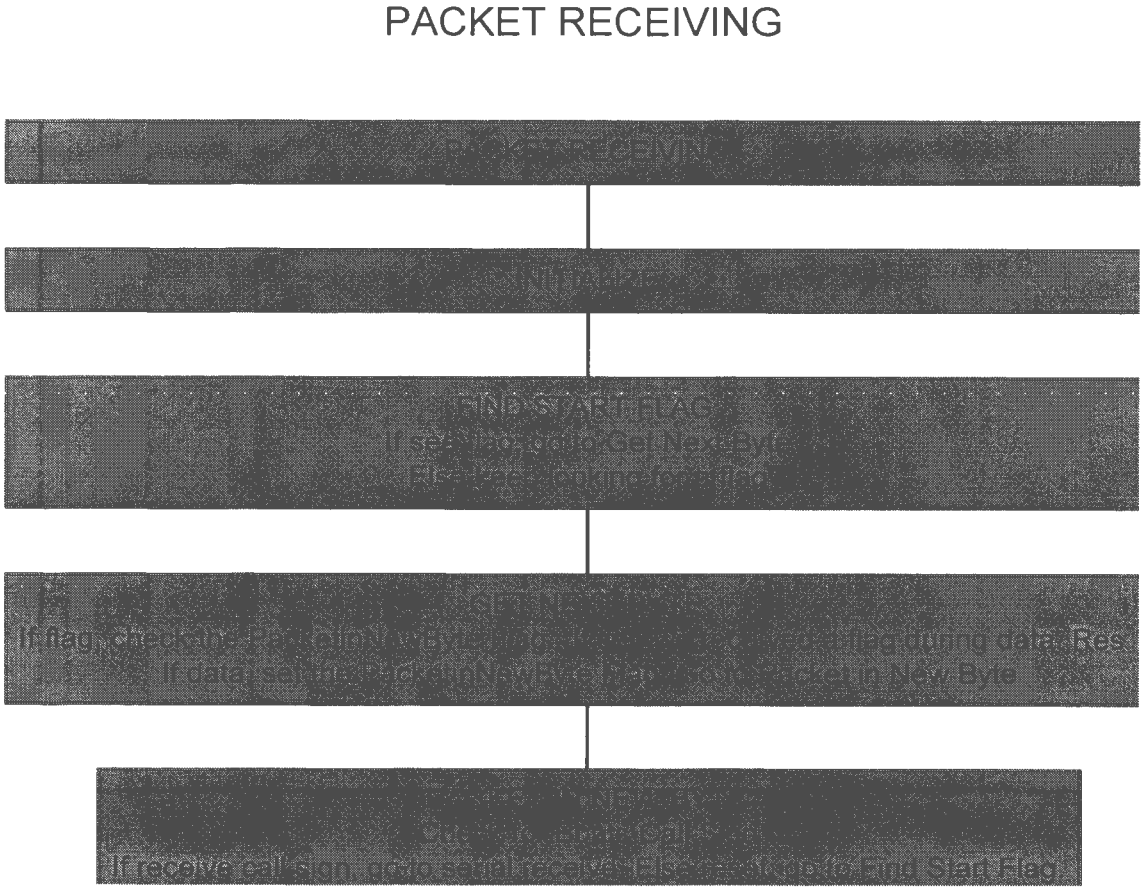
the connection level. The screen becomes in this case the presentation level. [http://www.qsl.net/yo5ofh/doc/AX.25%20protocol%20.htm]

The packet receiving function must therefore have the ability to receive packets in the AX.25 protocol as mentioned above. Tuscon Amateur Packet Radio (TAPR) provides code at their website (http://www.tapr.org) that implements the AX.25 protocol on the PIC-E. This code was used as a basis for the packet receiving code implemented in our project. There were many major changes required to change the functionality to meet the goals of our project, however the available code provided invaluable programming lessons, as well as a foundation to build on.

Packet receiving required a timing sequence for receiving the serial bits from the modem. As explained earlier, the rxd line from the MX614 toggled to indicate a 0, and remained in its current state to indicate a received 1. The speed of the transmission is 1200 bps. This meant that the packet receive bit, RB4, would have to be monitored for toggles. As previously mentioned the PIC has a function that flags a change in the state of any Port B bit. While receiving a packet the only Port B bit that should change is RB4. This allowed the use of the RBIF flag. The number of clock cycles between bits needed to be calculated to allow for correct timing in the receiving process. The PIC operates on a 10 MHz. clock cycle. The transmission rate is 1200 bps. There will be =8333.33 clock cycles between bits. The program must take this into account. This also allows for quite a bit of time to perform logic on the incoming data between bits.

$$\frac{1}{1200} \qquad \frac{1}{10 \times 10^{6}}$$

The packet receiving algorithm consists of initialization, searching for a start flag, receiving serial bytes, and looking for the correct call sign. It is outlined in Figure 6.



6Figure 6: AX.25 Packet Receiving Algorithm Flowchart.

Initialize:

The initialization for packet receiving would have to initialize the port bits involved. The modem is configured using the Mode 0 and Mode 1 pins for receiving at 1200 bps. The TMRO function is initialized to set a flag after the time between bits has passed. The RBIF function is initialized to set the RBIF flag when a Port B pin changes states. The interrupts are disabled during packet receive. When these items have been initialized the PIC is ready to begin reading in packets.

The packets will be coming in the AX.25 protocol. Therefore the start of a packet will be the flag. The flag is H'7E'. This is a 0 followed by six 1's followed by a zero.

Find Start Flag:

The code starts out looking for the start flag. When six 1's are received the next character expected is a 0. If a zero is received it is a flag and it is known that data could follow. There may be many flags that follow, as it is common practice to send 40 start flags at the beginning of a packet, but when the flags end the data is read.

Get Next Byte:

Upon reading the 5^{th} 1 following the first flag the next byte is monitored for bit stuffing. If there is a 6^{th} then we will know we are still receiving flags. Upon receiving the first non-flag the following bytes will have to un-stuffed. There should no longer be six 1's in a row while data is being received. This is due to bit stuffing. Bit stuffing ensures that there are no flags during the data. If five 1's are encountered sending an AX.25 packet the byte is stuffed. A 0 is inserted after the fifth 1 to avoid this byte accidentally being misinterpreted as a flag. This requires that the inserted 0's be taken out when receiving a packet. Upon reading five 1's in a row the following 0 will be removed. If six 1's are received it is known to be the end flag. This flag signifies the end of the transmission. If a flag is not seen a byte has been received. The byte is stored and sent to Packet In New Byte.

Packet In New Byte:

Because there is enough time between bits, the received byte is compared to the expected byte in the boats call sign. The AX.25 protocol requires the transmission to be in the form: Flags, Destination, Source, and Data, followed by Flags. This is generalized, as there is more required but this will be sufficient to explain the function of the code. Following the start flags the boat will be looking for it's call sign. This is the destination field in the AX.25 protocol. If

it's call sign is not recognized, it resets and begins looking for start flags. Each byte received is compared to the boats call sign. The call signs are defined as BT00XX, where XX will be the boats number. This is currently in the range of 1-20. When the last character of the call sign has been received the Serial Receiving function is called to get the current G.P.S. location.

SERIAL RECEIVING

The serial receiving function will reads in the G.P.S. location from the Garmin 35LP, filters out the unwanted information and sends the desired data through the Widi_compression algorithm.

The Garmin 35LP operates under the NMEA standards. These standards require that the G.P.S. output the required NMEA sentences every second. The time elapsed between the start of the first transmission to the start of the following transmission for each NMEA sentence is 1 second. These sentences are output serially at 4800 bps. The string that we are interested in is the $GPRMC string. This is the recommended minimum specific string. This string contains the latitude and longitude in the form we desire, along with other unneeded information such as altitude. RA0 of the PIC is connected to the transmit pin of the Garmin. The PIC-E's layout allows the Garmin to receive its power through the DB9 connector that connects the G.P.S. to the PIC-E. This allows for a relatively simple connection. The received pin of the Garmin is not connected because it is not needed. The default output of the Garmin is what we want.

Upon connecting the Garmin to the PIC-E we experienced difficulty receiving the transmitted sentences. We tested the circuit by burning the PIC with working code available from TAPR that would simply read in the G.P.S. location and transmit it. This did not work either. At this point it was discovered that the V+ pin of the Garmin had only 1.1 volts. This

was much lower than the required operating voltage of the 4.0 V needed. The PIC-E had a 10K Ohm resistor in line with the V+ pin. This resistor had a drop across it of 3.8 volts. This meant that the impedance of the Garmin was much lower than that of the Tripmate that the PIC-E was designed for. The specs for the Garmin showed that it had a maximum current draw of 140mA. With a minimum of 4.0 V at the input, we calculated that the resistor needed to be less than 6.5 Ohms. The resistor was removed, and the G.P.S. began to transmit the NMEA sentences every second as expected.

The serial receiving flow chart is shown in Figure 7.

Main:

Main initializes for receiving the serial transmission of the G.P.S.. The NMEA receiving counts, and modes are initialized. RA0, and RA1 are initialized as input and output. The command word is initialized for the time being as a boat location. When the functionality is added to record the location of fish caught, the distinction will be made between locations as to which are fish caught and which are current locations. The Buffer is initialized to point to the start of the buffer. Serial Receive is called once the initializations are complete.

Serial Receive:

Serial Receive checks RA0 for the start bit. The Garmin transmits using RS232 standards. Each character will be 9 bits. The first bit is the start bit, which is asserted high. The following 8 bits are the data followed by two stop bits, which are low. The Serial Receive therefore looks for the start bit. Then the Serial Receive Loop is called.

ReceiveLoop:

The timing is done using a loop which decrements a count each time through the loop. Receiving at 4800 bps there are 2083.33 clock cycles between bits.

When the start bit is received the code waits for 1/3 of the time between bit transmissions to ensure that we are far enough away from the edge of the bit change. Nine bits are subsequently read into an eight-bit register. This rotates the start bit off the end of the register leaving the 8 data bits we desire. The received byte is then sent to the NMEA decoder.

NMEA Decoder:

The NMEA decoding function relies on the NMEA standards used by the Garmin. The $GPRMC sentence is output once every second. Knowing this, it is possible to look for the $GPRMC title that precedes this string. An example of this sentence is:

$GPRMC,191103,V,4137.8942,N,11150.7949,W...

The title is followed by the 6-digit time, followed by the validity, followed by the Latitude and the Longitude. Commas separate all modes. The NMEA decoder looks for the title. When the title is received it looks for the comma, followed by the 6-digit time, followed by a comma. If the comma is expected and it does not get the comma the NMEA decoder resets the count and starts again. After the second time a comma has been read, the decoder implements a lookup

table to keep track of how many bytes to read in for the mode it is in. Upon seeing a comma the mode is incremented. The NMEA mode keeps track of the commas to know where, in the NMEA sentence, it is. There is also a count to keep track of how many bytes it has received in the current mode. This keeps the decoder aligned with the incoming string. As the byte is received it is immediately sent to the Widi_compress algorithm.

Store Compressed Position:

We have sufficient time between bytes to decode and compress. The byte is decoded and the compressed data is stored in the buffer. Once the byte is decoded and stored when appropriate, we return to Serial Receive and wait for the start bit indicating the start of the next byte.

Once we have reached the end of the Longitude and we have the compressed position stored in the buffer, Packet Send is called. When the fish caught buttons have been implemented the distinction will have to be made here whether to send the buffer or go to Packet Receive.

PACKET SENDING

The MX614 is used to send the data. The data is sent at 1200 bps. The timing is accomplished using the TMR0 interrupt. The TMR0 function sets a flag after a designated number of clock cycles have passed, and when the interrupt is enabled it calls an interrupt. This provides the functionality needed to send 1200 bps to the modem.

The PK 96 will be expecting AX.25 protocol messages. This will require the outgoing transmission to conform to the protocol. The algorithm can be found in Figure 8.

Out Main:

RA4 is initialized to allow PTT to be pressed during sending. The buffer end address is stored, and the modem is initialized to transmit at 1200 bps. The interrupts are disabled at this point. Once the initialization is complete, Packet Send is called.

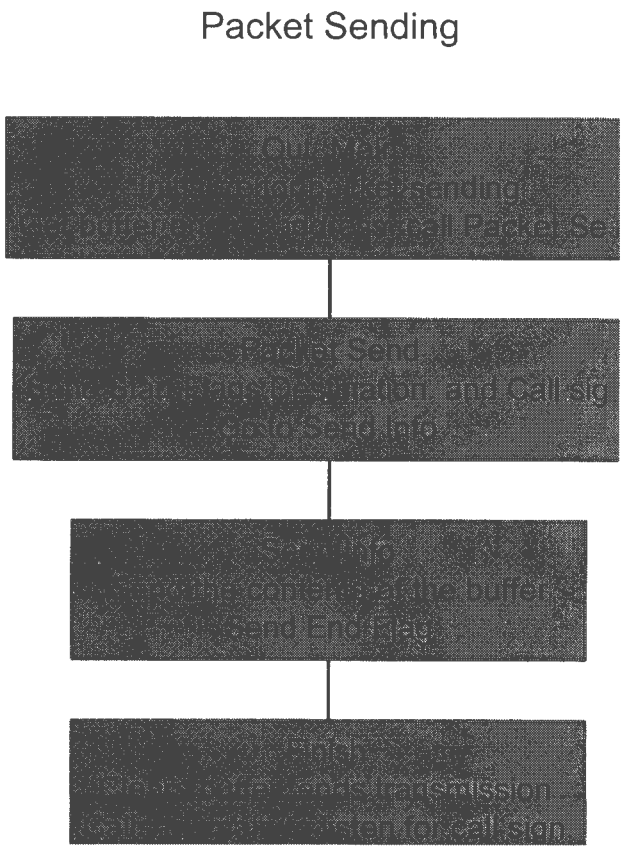


Figure 8: AX.25 Packet Sending Algorithm.

Packet Send:

PTT is asserted high to begin the transmission. The TMR0 interrupt is initialized and the interrupt enabled. 40 start flags are sent. These are followed by the Destination. The destination is DWBAS0, which will remain the same for all boats. This is followed by SSID, which is a control byte for the address. The call sign for the boat sending the packet is then sent in the form BT00XX, followed by the SSID. What is actually sent will be slightly different. The next byte sent will be the Protocol ID, or PID. Once these things have been sent to conform to the AX.25 protocol we can go to Send Info.

Send Info:

The buffer can now be sent. The buffer is sent from the bottom to the top. The first item in the buffer will be the first out. The buffer will contain up to 4 fish caught locations and will always end with the current location. Each location is nine bytes long. Each time a byte is transmitted

from the buffer the pointer is incremented and the address is checked with the last address of the buffer. When the addresses are the same it jumps to Finish.

Finish:

Sends the checksum, the end flag, and sets the buffer length to zero. The PTT is now released, and the TMRO interrupt is disabled. The task of retrieving the G.P.S. coordinates, compressing them, and sending the compressed location is completed. The code now goes back to Packet Receiving and looks for its call sign to start the process again.

DESIGN PROCESS

The code was not built in one fell swoop. The code was broken down into individual functions, and each function was coded and tested. When each function was debugged separately, the functions were pieced together one function at a time.

Writing and Compiling the Code:

Due to the RAM and memory constraints it was decided that, to know the exact memory use, assembly language programming would be necessary. The Compiler used was Microchips MPLAB. The code was written and compiled in MPLAB, which produced the Hex file. The Hex file was what was needed to burn the PIC. MPLAB was available from Microchip's website at no cost.

Programming the PIC:

The PIC-E was designed to connect to the serial port of a computer to allow the PIC 16F84 to be programmed using a Ludi Pipo style programmer. The voltage levels produced by the serial port are sufficient to program the PIC. We discovered that a -Windows 95, or 98 operating system was required to burn the program. The program that we used was called Pic Prog. This program

provides a simple GUI for easy operation, and proved to be the simplest to understand. This program uses a single buffer to read and store the hex file. The program can then be burned. The Pic Prog will inform the user following the burn whether it was successful or not.

There are many programs that are available that perform the same function. The Pic Prog was explained in the PIC-E manual, and proved the easiest to use. The ability to easily program the PIC from almost any computer proved extremely useful as the code was developed and tested.

The first function to be built was named Blue. This clever name came about because the PIC-E used to test this code had a blue capacitor.

Blue:

The function that blue was to accomplish was to transmit an AX.25 sentence that would be recognized by the PK 96. This seemed simple enough, though there were many intricacies that were missed as the code was written. The main problems came about because of a misinterpretation of the AX.25 protocol. Once the discrepancies were resolved the problems filtered out fairly quickly.

As Blue began to be pieced together and tested, an LED connected to RB0 was recognized as a valuable testing tool. This LED was known as VALID_OUT. Blue started by initializing for packet sending. After initialization, VALID_OUT was set. This let us know that we were reaching this point in the code. The code was added upon piece by piece until we were able to transmit a packet.

The code was tested by burning the code in the PIC, connecting the PIC to one radio, connecting the PK 96 to another radio and connecting the PK 96 to a computer running

Hyperterm. By setting up Hyperterm to connect to the PK 96 we could see the incoming data being sent by Blue. The sent packets were in the form:

Destination–SSID–Source–SSID–PID–Data–Checksum

In our case this was: DWBAS0 0 BT0001 1 <UI> Data Checksum. Where the Data was whatever we placed in the buffer. The above packet was preceded by 40 start flags and followed by a stop flag.

Now we could work on packet receiving, or Brown.

Brown:

Once again this clever name was chosen for the color of the capacitor on the PIC-E it was tested on. Brown's function was to receive a packet transmitted from Hyperterm and recognize its unique call sign.

The code began by looking for a start flag. Upon receipt of the start flag the VALID_OUT LED was cleared. This showed that the code was able to read in a character and recognize it as a flag. The code continued to be built using the same approach. VALID_OUT was used to test the code and locate errors. Code was added to recognize when the start flags end and the destination starts. The PIC-E used for testing was given the call sign BT0001. Brown would check each incoming byte with the corresponding byte in the call sign. When the last byte was reached and the call sign was recognized the code was considered finished. Any information that followed including the checksum was ignored and the program immediately exits the packet receiving function. This greatly reduced the size of the code.

At this point two of the three main functions required were working separate from the other. It was decided to combine the two functions and create a polling system. The new code

would start in Brown listening for its call sign. Upon seeing its call sign the code would call blue and transmit the contents of the buffer. This would be similar to the functionality required by the final system. The new code was affectionately titled BrownAndBlue.

BrownAndBlue:

Combining the two functions proved to be somewhat difficult. This time the code was put together with only a few command changes and tested as a whole. Upon the first test run nothing seemed to be happening. Through researching the PIC and reviewing the code it was discovered that the memory organization needed to be altered. Though the problem was solved by accident at this instance, it was later realized what the problem was.

The PIC 16F84 memory is organized in pages. There are 4 pages of program memory, each page containing 255 bytes. The program count is therefore 12 bits. The PCL (8 bits) addresses the 255 bytes on the current page and the PCLATH (4 bits) addresses the page. The data lookup tables use a command that adds an offset to the PCL to point to the desired byte in the table. It was not known at the time, but when the offset was added to the PCL, the PCLATH register had to be loaded with the current page. If it is not loaded it automatically loads page 0. When the two functions were combined the data lookup tables were now on page 1. Because we did not initialize the PCLATH before adding the offset to the PCL, we were jumping to the wrong address. We encountered this problem later on in the development, and it was then that the situation was fully understood.

The problem was accidentally corrected in BrownAndBlue. When this problem was gone there were only minor issues to resolve before the polling function worked. The packet was sent from Hyperterm, through the PK 96, received by the PIC-E and was responded to with

a transmission of the contents of the buffer. Now that the polling was functioning it was apparent that we were making some significant headway. There was one function remaining to be added. This was serial receiving. Now the name for serial receiving was up in the air. We had used up all of the capacitor colors, and we could choose any name we desired. We chose Red.

Red:

Serial receiving needed to read in the G.P.S. position transmitted by the Garmin. The $GPRMC sentence contained much more data than was desired. The unneeded information would have to be filtered out. This still left a fairly large sentence. The position would have to be read into the buffer. The buffer size is 45 bytes. By compressing the position down to 9 bytes, which included the command word designating whether it is the boats location or the type of fish caught at that location, 5 locations could be stored in the buffer. It was decided that 5 locations would be sufficient for the frequency of polling. The serial receive would also have to compress the position.

A function was written to implement the compression. This function can be viewed separately from the serial receive. The serial receive was initially implemented along with the compression algorithm. The code was initially written in its entirety and tested as a whole. When the initial tests failed the code was simplified to read the G.P.S. straight into the buffer until the buffer was full. When this version was tested there was again failure.

At this point the circuit was tested and it was discovered that the voltage at V+ on the G.P.S. was only 1.1V. When the voltage supply problem was fixed the code was again tested and found to work. The Widi_compress algorithm remained unimplemented and the program

read in the G.P.S. information until the buffer was full. It was decided to add red to BrownAndBlue as is. We would then be able to poll the PIC-E and have it respond with the uncompressed G.P.S. sentence. This new code was to be named BrownAndBlueAndRed. This made perfect sense, but the name proved to be too long for MPLAB and it was changed to BrownAndRed.

BrownAndRed:

Adding Red to BrownAndBlue went without incident. The serial receiving function was added to the program and soon the uncompressed NMEA sentences were being sent to the PK 96 and output onto the Hyperterm screen. This would occur after a poll from home base. The three main functions were now working correctly. Now the compression algorithm needed to be added to the serial receive function.

NMEA Decoding, and Compressing:

The first object was to locate the desired NMEA sentence. This was the $GPRMC sentence. Once the code was able to recognize the title it was possible to grab only the data we needed. The NMEA sentence could be divided into modes, with each mode separated by a comma. The modes each have a specific length in bytes. The length of each mode is found by reading the table for the current mode. The data lookup table is a function that is called with an offset in the W register. The function adds the offset to the current program count to jump to the desired location in the table. The function returns with the data from that location in the W register. When a byte is received the count is compared with the mode length by retrieving it from the table and comparing it with the current count.

The compression was added to BrownAndRed one piece at a time. The title recognition was added without a hitch. However, when it came time to implement the data lookup table, there were problems. The VALID_OUT LED was put to work debugging the code to find out where the problem lay. The problem was tracked to the data lookup table, and even further to the command for the lookup table that adds the offset to the PCL. It was at this point that we learned to initialize the PCLATH prior to adding the offset. This was explained in the BrownAndBlue section of this report. With this problem was resolved it seemed that there could be no more set backs finishing the compression.

The only remaining challenge was implementing a couple of compression functions that edited a byte in the buffer, which had been received several bytes previous, depending on the current byte received. This required moving the buffer pointer to the desired buffer location, changing that byte if needed and then returning the pointer to the end of the buffer. Several methods were tried and tested until the method that worked was found. The decimal number of address locations to move the buffer was subtracted and added to the buffer pointer. Once this worked the compression algorithm finished itself. The code was tested and found to work. Red was programmed onto the PIC-E to transmit the un-compressed position, and then BrownAndRed was burned onto the PIC. The uncompressed data was compressed by hand to compare to the data received from BrownAndRed. There was a perfect match. There was great rejoicing, and much dancing that ensued. Following the dancing the PIC-E was polled to ensure the results were the same. This time two bytes of compressed data were missing.

It was strange that the compression algorithm would work for a while and then stop working correctly. Red was programmed into the PIC-E and the G.P.S. position was sent again

un-compressed. The position was checked with the previous and it was discovered that the bytes that were missing were ASCII control characters. It was assumed that the compression algorithm was working fine and Hyperterm could not print these characters. This would later be tested using John's terminal program.

The only function left to implement is the Fish caught push buttons. The method for implementing the fish caught buttons has been researched, and the buttons will be added prior to implementing the system.

The current working code worked well enough using Hyperterm. The program was now ready for more extensive testing with the base station computer program. Details of these tests can be found in the section on system testing.

G.P.S. Selection

There were several steps in selecting a G.P.S. unit to use in this system. First, consideration was given to designing a G.P.S. receiver. It quickly became apparent that this requiring designing antennas, circuitry, and software capable of decoding the position from the satellite data. That would be quite a project in its own right, and therefore was well beyond the scope of this project. It was also realized that a manufactured unit would be much more robust in terms of performance than a student-designed unit would be.

Once the determination was made to use a commercial unit, the selection of a G.P.S. receiver was greatly simplified by the relatively small number of manufacturers of OEM devices. Particularly, the only company we could find that sold receivers suitable for this project was Garmin. They offer several products, including the Garmin25 and the Garmin35LVC. Both devices are capable of providing differential G.P.S. (DGPS) data, and both are capable of

interfacing directly with the PIC-E without any additional circuitry to interface the serial connection or provide power. The 25 is a circuit board model containing the circuitry necessary for decoding the G.P.S. satellite signals, but which doesn't include any cabling or antenna. The 35LVC is a self-contained unit, including a connection cable and antenna.

Since both devices are so similar, the main factor in deciding which model to use is cost. The 15L is available for $120, however, it requires an external antenna, and a data cable, which cost $20 each, so the total cost for the unit comes to $160. The 35LVC costs $170, but it comes with all of the necessary cabling. With prices so similar, we decided to go with the 35LVC, which also is enclosed and waterproof.

The Garmin Trakpak 35 actually consists of a variety of models, each with a different application. The Garmin 35PC is designed for connections directly to the serial port of a computer and accepts either True RS-232 voltage levels or TTL voltage levels. The Garmin 35 HVx is designed for systems with an input voltage between 6 and 40 volts. It comes in either the HVC or HVS models which accept TTL or true RS-232 logic levels respectively. The low power version, the Garmin 35 LVx, also comes in HVC and HVS models each with the same characteristics as their higher power counterparts. The Garmin 35 LVx accepts voltages between 3 to 6 volts, with a maximum efficiency occurring for an input of 4 volts.

Protocols

WIDI G.P.S. Compression Protocol

The NEMA $GPMRC string is normally outputted by the G.P.S. once per second and contains the Universal Time Code(UTC), the current or last known latitude and longitude, and the current bearing and speed. Each of these parts of the G.P.S. string are separated by commas

to delimit them. For this project not all of the G.P.S. information was necessary to accomplish our task. This presented the design team with some choices. Should the entire G.P.S. string be transmitted? When the data is stored, will it be compressed or left in the form that it originated in?

As the RAM on the microcontroller was limited to only 68 bytes, which is less than the length of the G.P.S. string, it was decided that most or part of the string would need to be removed. Later as the assembly code was being written it was discovered that 22 bytes would be needed of the RAM to operate properly. This left only 46 bytes for storage. As each G.P.S. location required 19 bytes and another byte was allocated for a command word to be stored along with it, we discovered that we could only store 2 locations before we would run out of memory. This was not acceptable for our design parameters, which required that the boat be able to store multiple locations with the catch data. Each character in the G.P.S. string is in the ASCII format, which could require up to eight bytes just for the longitude. By converting the numbers into binary we were able to compress the data to 9 bytes. As only one of the numbers stored could exceed 128 we decided to adopt a 7 bits per byte policy. While this caused extemporaneous data to be transmitted, thus lowering the data density marginally, it matched up with the default reception characteristics of the PK-96 and simplified the decoding process greatly. For future implementations of this project a PIC-E would be substituted for the PK-96 and that feature could be removed if desired.

In order to fit the relevant data to nine bytes, a special encoding algorithm had to be developed. This algorithm was based on the concept that each section delimited by commas in the $GPMRC string can be treated as a separate mode. A counter is kept to track which mode

the program is currently in. Based on the mode the program uses a data lookup table to determine how many of the bytes after the first comma contain desired data. After reading in the determined number of bytes, the program then looks for the next comma before advancing to the next mode. Through the use of these modes we are able to call the read and decode functions for only the bytes relevant to this project. A counter called "count" is used to track which of the 19 bytes of data are currently being read in and determines the proper encoding function to call. These functions are called Widi_one through Widi_twenty inclusively. Widi_one stores the command word before the first byte of latitude is received. In Widi_two the byte is received and has the decimal number 48 subtracted from it to convert it from ASCII to its decimal equivalent. This number is then used with a lookup table to create the effect of multiplication by a factor of ten. This value is then stored in Widi_temp. Widi_three repeats the process of subtracting 48 from the byte, and adds the resulting value to what has been stored in Widi_temp. At this point the entire latitude has been read in and converted from two ASCII numbers to a single binary number that ranges from zero to ninety. This resulting binary number is subsequently stored in memory and the offset, Widi_inf_count, is incremented to point to the next available location in memory. This process is repeated with all the bytes read in with the exception of the Longitude and the Hemisphere.

As the Longitude can go up to 180 degrees, the highest order bit is removed and placed in the lowest order bit of the command word, to ensure that only seven bits are used in each byte.

The minutes can only go up to a value of sixty, which means that only six bits are needed to represent them. The extra bit of memory in these locations is allocated to the hemisphere.

The hemisphere is located in the lowest order bit and the minute data is shifted to the left one bit. The location of the hemisphere bit was chosen due to the ease of encoding and decoding it.

The floating point decimal values of the minutes are accurate to the fourth digit. These digits are broken up into two groups of two digits, each with a maximum value of ninety-nine. They are stored with an assumed decimal point before the first group. This decimal point is later added by the decoding software.

The format for the algorithm is given in the table below.

Memory Map for Data Stored by Bit								Pointers to Memory Locations
0	C	C	C	C	C	C	G	"Widi_inf_base
0	T	T	T	T	T	T	T	
0	M	M	M	M	M	M	H	
0	D	D	D	D	D	D	D	
0	D	D	D	D	D	D	D	
0	G	G	G	G	G	G	G	
0	M	M	M	M	M	M	H	
0	D	D	D	D	D	D	D	
0	D	D	D	D	D	D	D	
								"Widi_inf_base+Widi_inf_count

LEGEND:
C=Command Bit
T=Latitude Bit
G=Longitude Bit
M=Minute Bit
H=Hemisphere Bit(E=1,W=0,N=1,S=0)
D=Minute Floating Point Value Bit

AX.25 Link Layer Protocol

The AX.25 Link-Layer Protocol provides a method for nearly error-free communication between the boats and the base computer. AX.25 Link-Layer Protocol is capable of transmitting three different types of data blocks, called frames. Each frame is made up of several smaller groups called fields. For this project the simplest frame, called an Unnumbered Information (UI)

frame, was all that was needed to both poll the boats and transmit G.P.S. locations. The UI frame consists of 6 fields, each containing a specific number of bytes. The first and last fields are flag fields and consist of a single byte with a hex value of 7E (01111110). The flags denote the beginning and ending of each frame. In order to distinguish flags (which have six consecutive ones), the AX.25 protocol does not allow any other fields to contain more than five ones in sequence. In fact, whenever five ones that aren't in a flag are to be transmitted, a zero is automatically inserted after them. On the receiving end, any zero that occurs after five ones is removed. This bit stuffing causes the flags to be a unique occurrence of more than five ones, and therefore easily detectable.

Flag	**Address**	**Control**	**PID**	**Info.**	**FCS**	**Flag**
01111110	112/560 Bits	8 Bits	8 Bits	N*8 Bits	16 Bits	01111110

The front flag field is followed by the Address Field. The Address Field is encoded with both the destination and source call signs for the frame. These call signs consist of upper-case alpha and numeric ASCII characters only. If repeaters are used to propagate the signal then their call sign is also included in the Address Field. Every character in the call signs are shifted to the left. The low order bit then becomes an indicator of whether or not there is more data to follow within the Address Field. If the low order bit equals one then the Address Field has ended.

Each call sign is followed by a special byte which contains information that describes how the frame should be handled. This special byte is referred to as the Secondary Station Identifier or SSID. The SSID allows up to sixteen different stations to utilize the same call sign and still be distinguished one from another. It also allows a transmission to track the path it has taken through repeater stations. As repeaters were not used for this project and multiple call

signs were used, the functionality of the SSID was not needed, and was set to a hex value of 30 in all cases except the final one where it was set to 31.

The base computer has been assigned a call sign of DWBAS0, and each boat a call sign of BT00XX. (XX denotes the boat number.) While the PIC-E is fully capable of completely decoding AX.25 packets it receives, we decided that for this application all the PIC-E had to do was listen for its own call sign. Once this call sign is received, it can ignore the rest of the packet and prepare to transmit the response. This response is a fully encoded UI frame that contains the address of the base computer and the sending boat followed by the data stored on the PIC-E.

Following the last SSID is a control word which specifies both the type of frame being transmitted as well as the intended function of the information contained within it. For the case of the UI frames that are being transmitted in this project the command word is set to a hex value of 03 which denotes an Unnumbered Information frame.

Immediately following the Control Field is the Protocol Identifier (PID) field. This field identifies the type of layer 3 protocol being used if any. In the case of this project the PID was set to a hex value of F0 which designates no layer 3 protocol implemented.

The Information field is the next field transmitted from the PIC-E. It contains the compressed data that has been collected on the boat. Due to the receiving characteristics of the PK-96 only seven bits of data are stored in each byte of the information field.

With the exception of the FCS all bytes transmitted are transmitted low order bit first.

The last field before the Flag field marking the end of the frame is the Frame-Check Sequence (FCS) which is a sixteen bit checksum calculated by both the sender and receiver of a frame. It is used to insure that the frame was not corrupted by the medium used to get the frame

from the sender to the receiver. The AX.25 protocol is described in detail at the web site http://www.tapr.org/tapr/html/Fax25.html. From examples given there the following method was developed.

First, the crc_hi and crc_lo bytes are set to a hex value of FF. Their values are calculated using inverted logic and they are also stored backwards. This facilitates their transmission as a separate algorithm for transmission didn't need to be developed. The rest of this discussion on them will describe the way that the algorithm sees them, not as they will be interpreted by the receiving system.

The bit that is to be transmitted is separated from the rest of the byte. If this bit is one then the low order bit of the FCS is toggled. Due to the method of storing the byte reversed this low order bit will be seen as the high order bit by the receiving system. Whether or not this bit is toggled, the entire FCS is then shifted to the right, effectively dividing it by two. If there is a remainder when this divide is executed then the crc_hi and crc_lo bytes are exclusively or'ed with a hex value of 84 and 08 respectively. This creates the effect of raising the value of the crc to a polynomial. This entire process is repeated for every bit that is to be transmitted. When the time comes for the FSR to be transmitted then the crc_hi and crc_lo bytes are exclusively or'ed with a hex value of FF to invert their values back to positive logic.

The following section of assembly code accomplishes the generation of the FSR as described above:

```
      movlw H'FF'
      movwf packetin_crc_lo
      movwf packetin_crc_hi

AX25_Send_Data
      movwf packetin_data
      movlw 8
```

```
      movwf packetin_flag_count
AX25_Send_Data_Loop
      rrf   packetin_data, W  ;get a bit for CRC
      movl  1
      xorwf packetin_crc_lo, F
      clrc
      rrf   packetin_crc_high,F
      rrf   packetin_crc_lo, F
      skpc
          goto    AX25_CRC_Done
      movlw H'08'
      xorwf packetin_crc_lo, F
      movlw H'84'
      xorwf packetin_crc_hi, F
AX25_CRC_Done
      rrf   packetin_data, F
      call  AX25_Send_Bit
      decfsz      packetin_flag_count, F
         goto AX25_Send_Data_Loop

      movlw H'FF'
      xorwf packetin_crc_lo, F      ; invert values of crc low for
transmission.
      xorwf packetin_crc_hi, F      ; invert values of crc high for

transmission
```

The receiving system calculates the FSR for an incoming packet in the same way, replacing packetin_data with the incoming byte. In the case of this design using the PIC-E there was no need to implement a receiving FSR check as the boats look only for their call sign.

Software Development

The development of the software for the base station posed some unique design challenges. The software was to be fairly complex, offering many features and yet allowing users to easily navigate its options, but the greatest challenge came in developing the software in parallel with the hardware development. The software had to be written from the anticipated hardware specs and features. As a result, the software engineering approach was planned around system specs as they were anticipated to become available.

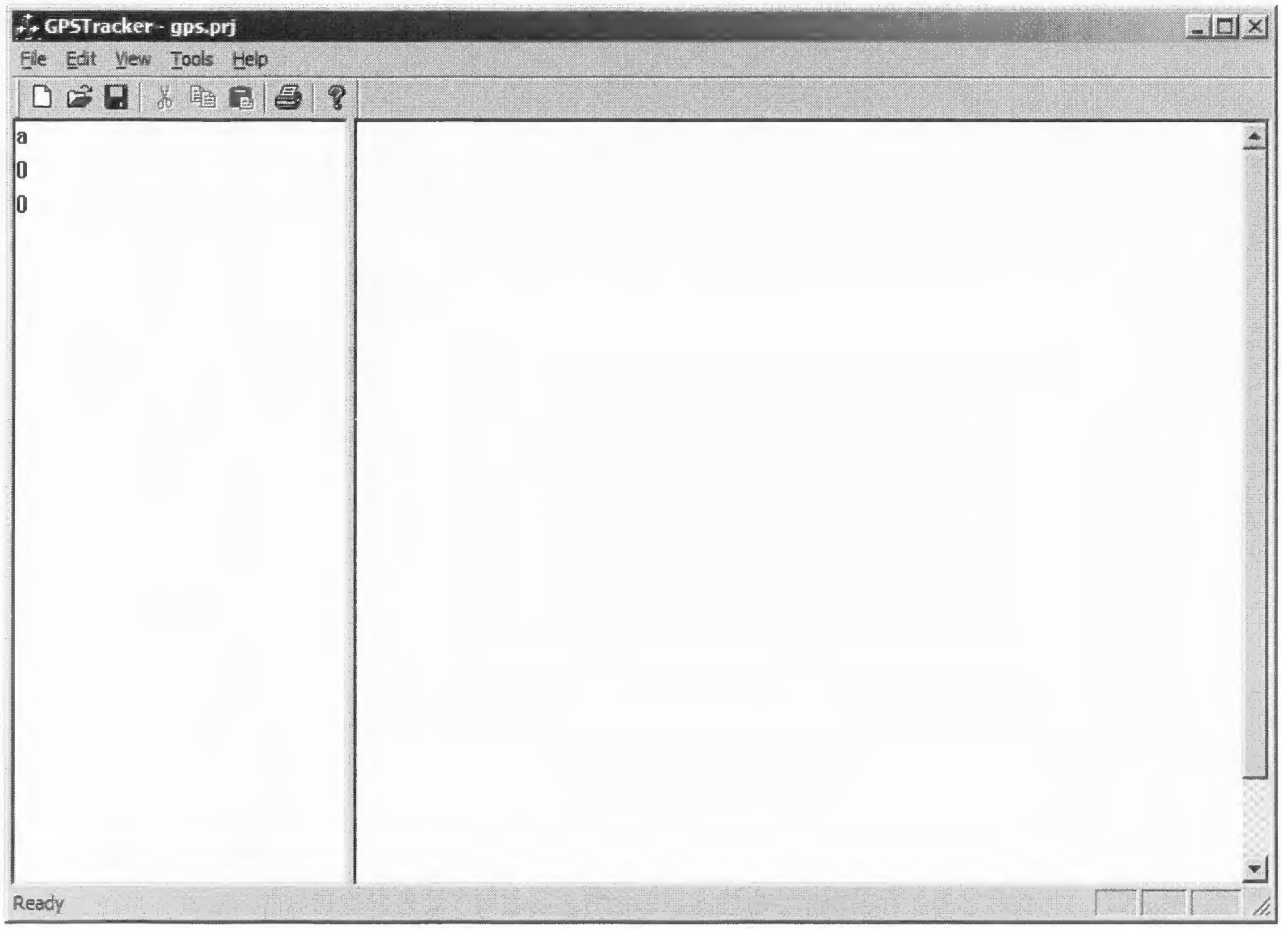
The communications, data requests, and data receiving were by far the functions most prone to change depending on the hardware. It was decided that both the hardware and clearly defined data structures would be needed for these sections. Therefore, they were held back as long in the development process as possible. It ended up being a good decision.

There were many tasks to be completed in designing the software. The first step was to create a shell program that would serve as a framework for the rest of the project. Next came developing data structures for storing boat and fish positions, and creating algorithms for calculating boat speed. A utility to merge and export data files was created. Also, user controls regarding data collection and plotting options were added. Next came the ability to load and display stored data. The next feature was polling and response. Finally, the ability to overlay maps was added.

Program Creation

At the time it wasn't sure what hardware interface would exist for the software. Because of this difficulty the software had to be developed in a very generic manner and only needed an adapter piece of software to complete the connection. This adapter piece would function as a translator between the data as it was input and the data as it was to be utilized by the program. With this approach the rest of the software could be designed and tested with minimal modifications needed when the hardware was added.

Due to the versatility of the language and the writer's familiarity with it, C++ was selected for this project. One particularly useful feature of the language are the Microsoft Foundation Classes (MFC), which includes tools for creating GUIs. Another advantage was that



framework for the program. Figure 9

shows boat 'a' at 0° N, 0° E.

All of the settings are stored in files so the user wouldn't need to reenter them every time the program was run. First among these settings is a list of all the boats to track, which is stored in its own file. To distinguish the boats, each one needed a number, name, and color. The user was therefore given the option of editing these attributes, as well as changing the total number of boats. Figure 10 demonstrates how this data is input.

9Figure 9: Basic Window with Boat 'a' at 0° N 0° E.

Processing Boat Data

The next step was to read in data from generated files, sort it, and then write it to files. The information is stored in files in a tab-delimited format. This was chosen because then if the files are opened in some other program (like a word processor or spreadsheet), the information could easily be displayed without much additional formatting. Particularly, this meant that the data could easily be imported by Excel., which has many useful data analysis tools and graphing options. That meant that an analysis package for the fish tracker software would not be needed at this time. It would be possible to add one later, if it were desired.

The formatting of the data in the files is actually very similar to the formatting of the NMEA string (an interesting case of the format going full circle). If boat 0 was at 50.03960° W and 49.99440° N, and the data was created December 31, 2001 at 16:01:45 with no fish caught, the data file line would read

```
0     50.03960   49.99440   2001 12    31    16    01    45          0
```

If a fish were caught, the format would be similar. The following shows that a salmon was caught at 98.7276° W,

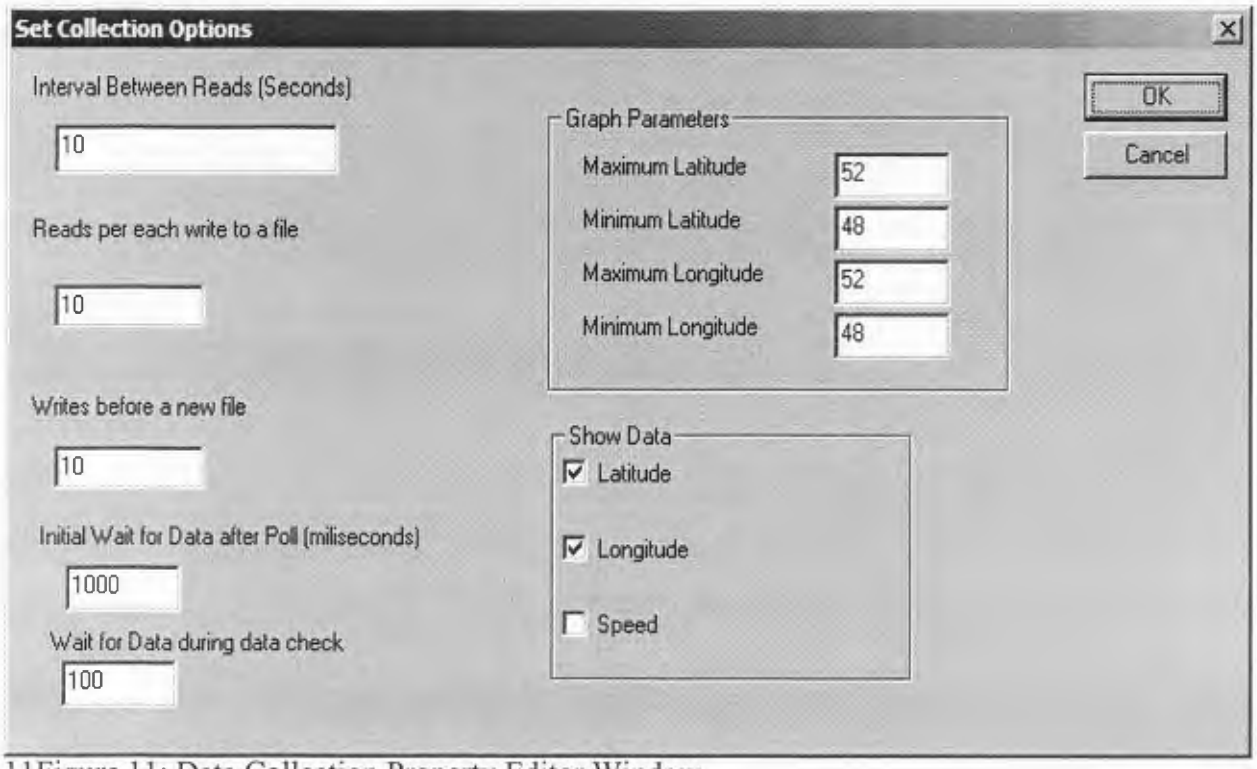
49.2475°N on January 10Figure 10: Boat Property Editor Box.

1st, 2002 at 17:51:45. The leading zero is the indicator of the type of fish.

```
0     98.7276    49.2475    2002 01    01    17    51    45
```

Once data had been gathered from boats (or from the data generator), it would be possible to compute the velocity. Initially it was thought that this would be a simple matter of subtracting the coordinate values of successive positions and dividing by the time between samples. It turned out that the computed value was useless. Not only was it not in any useful units (such as miles or kilometers per hour), but unless the coordinates are at the equator, the number of kilometers per degree is very different for latitude and longitude. At this point, no effort has been made to correct the error, but it is anticipated that an improved algorithm will be included in a future release.

The collection of data is the main purpose of this software. There are several parameters that a user can set to determine how the data is collected. These include polling frequency, the number of polls between data being archived, and the number of data points to be stored per file. A window that allows these properties to be edited is shown in Figure 11. The window also



11Figure 11: Data Collection Property Editor Window.

allows for a range of coordinates to be displayed. That will come into play when maps are added later.

As data is collected, it is important that the position of the boats be displayed in real time. The color properties stored for each boat comes into play here. Each boat is displayed on the window using the color assigned in the boat properties. As the information is updated with successive polls, the motion of the boats can be seen and tracked visually. Data is then read in and the boats can be seen moving around the screen. If the boats leave the active window, it can be resized so that it contains the current boat positions.

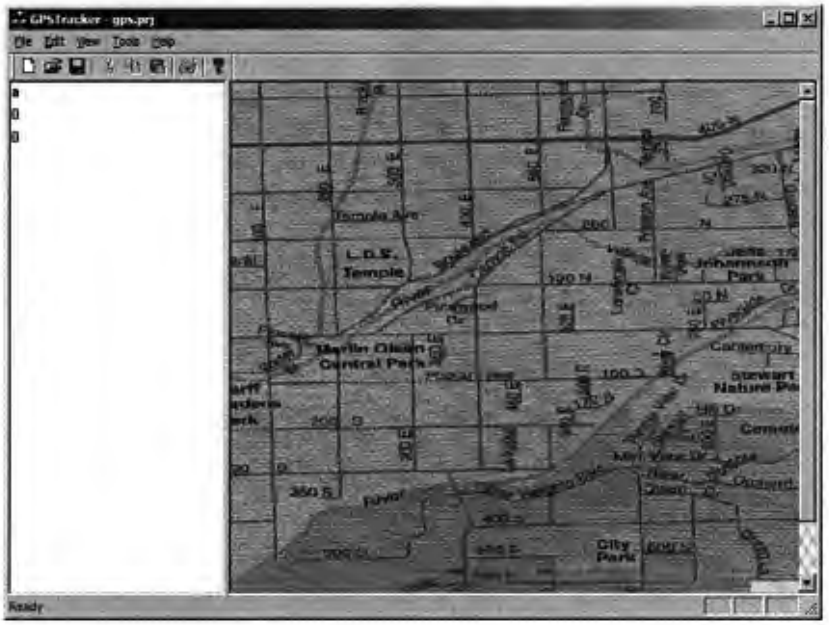
The next task was to merge data files. Depending on the user settings, a large number of data files can be created. Manually merging these files would be very time consuming, but necessary in order to look at the data in a third party program. A simple function was created to

merge these files. Merging can be done on boat data or fish data. The option is also given to delete the original files after they are merged. The program reads in each selected file into a huge buffer and writes it to the new file.

The next step was to plot where fish have been caught. This allows users to determine which areas are fishing hot spots. The user is allowed to select which files he wishes to display. Right now the program only displays one pixel per fish so it is hard to see for a small amount of fish. This can easily be changed if a new graphic is provided. The next addition was to add a plot of the history. This would basically do a replay of the files. When a fisherman comes back after a day of fishing he can see where he has been. This history will act as a journal.

Map Overlay

At this point, it was necessary to delay working on the communication protocols and address displaying a map. One option was to have the programmer personally create a file that contained borders of the water. It was then decided it would be much easier to load a bitmap containing the map. The difficulty with the doing it this way was gaining the ability to move and resize the map according to the current window size. To resolve this, an algorithm was used that allows portions of the image to be scaled and then displayed. The program had to figure out where to put the top left corner of the bitmap and what size to make it. This depends on where the user wants to view and what

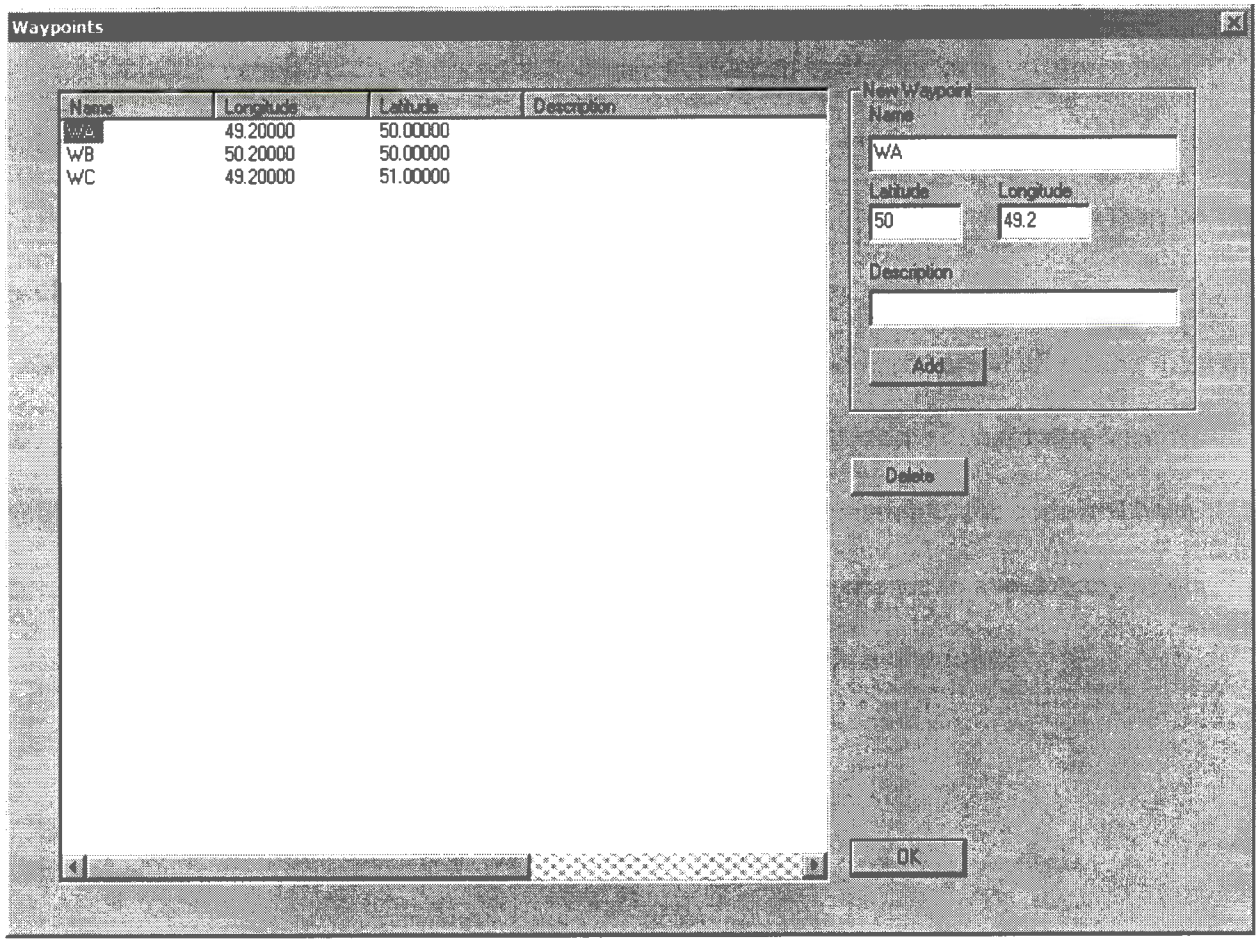


area the map covers. Figure 12 shows how a map appears in the right hand window.

In the early stages of the project, consideration was given to giving customers the option of setting and following waypoints. While in the process of working on the hardware it was determined that this was not feasible at the present time, they can still be used as reference markers on a map. The program already had the functionality to find a point. For a waypoint, the program displays the name of the waypoint where the graphic would go if it was a boat. Figure 13 shows a dialog box for creating waypoints.

Communication Software

At this point, the hardware was specified enough that the issues of communications could be addressed. For early development and debugging, a separate terminal program was made to read and write to the COM port. With this program, a great deal of exploration was done on the



function of serial ports and modem communications. This terminal program is able to send and receive data and display it on the screen. It isn't a very complex program but it did what was needed to streamline the process of debuging communications. It could also be used in future as an all-purpose terminal program (similar to Windows Hyperterm).

At this stage, the easy stuff was out of the way and it was time to form the communication interface in the program. A circular buffer was created to hold the data so it always had room for the nearly constant flow of data. After data was requested, it would be received in the circular COM buffer. The program would then parse the information to extract the coordinate data. The boats are configured to transmit any fish data before the current location. This was done so that upon receiving a block with a current location, the program would know that it had read all the data in the packet. The program then looks ahead in the buffer to make sure that it has finished reading the necessary data.

Once the data is received, then the software developed earlier kicks in. It stores the values, displays the numerical coordinates in the left pane of the window, and plots the position of the boat in the right pane. The data is buffered to be stored to data files, which can then be read back to analyze the behavior of both boaters and the fish they pursue.

The biggest difficulty arose from the fact that there was no hardware at all during most of the project. It was necessary to plan ahead and work on parts that were not hardware specific until the hardware was available. By the time the hardware was available, the code had been developed and tested with simulated data. We used previous experience to avoid many of the possible pitfalls and even gained some experience to avoid pitfalls in the future.

Looking back on the project, the software portion was planned and executed well. Of course there were things that should have been done better. More effort to validate the function of the software with the entire group would have helped to avoid some small problems. Other problems might have been caught if the entire team had reviewed the algorithms earlier, but instead had to wait until the system tests at the end of the project. In those cases, however, it is unlikely that extensive debugging efforts early in the project would have yielded any overall benefits, since it would have pulled the rest of the team from their design responsibilities. Overall, the software design was the smoothest part of the project.

System Testing

Stationary:

The PK 96 was connected to a laptop using John's terminal program and the polling command was tested. The two bytes were once again found to be missing. The compression algorithm was checked for errors and certified to be correct. This led us back to the fact that the characters were ASCII control characters.

Researching the receiving protocol of the PK 96 it was discovered that we could view the incoming data as hex characters as well. When the PK 96 was set up for this mode the missing characters could be seen. The problem was solved by changing the mode of the PK 96 which allowed the control characters to pass through.

When John was able to read in the compressed position and was ready for changing G.P.S. coordinates it was time to change the setup.

Mobile:

The radio was set up in a car. Using a power adapter for the 12V power connector in the car to power the radio, the PIC-E was set up and powered to test on a mobile object. The PIC-E has an RJ-45 jack for connecting the microphone to allow for voice communications. The microphone was connected and the group at home base was contacted over the radio to send a poll. The poll was received, recognized, and responded to by the PIC-E. The transmission from the PIC-E could be heard at home base, but the packet was not recognized by the PK 96. The problem was eventually traced to the microphone. When the microphone was unplugged the transmission was again recognized by the PK 96. Unplugging the microphone has initially been our solution to the problem, though a better means will need to be developed.

Once the polling command was in working order the car was driven to provide changing coordinates. We communicated by voice to verify the direction of travel with the coordinates being received. The polling was not 100% effective but the poll would receive a response approximately 75% of the time. This still allows data to be collected frequently enough that a boat is not likely to get lost.

A map of Cache Valley was taken from the phone book and scanned into a computer. Corners were chosen for the final display map and then the vehicle with the system installed was driven to those locations in order to get their longitudinal and latitudinal coordinates. These coordinates were then use to place the map and calculate the vehicle's position on the map. Small adjustments were then made to compensate for imperfections in the map in order to better display the motion of the vehicle on streets. The final system shows the position of a car on the map accurately enough that a person can determine where the car is and where it is going. We

were quite pleased with the test results and feel confident that this will meet the needs of Doc Warner's.

Project Scope

Summary of Project Tasks

The road that this project has followed was a long and winding one. Work progressed from efforts to design each component of the system from scratch to intense efforts to integrate the different components and debug the software responsible for the connections. Over that course, the design emphasis changed from determination of design parameters and feasibility to selection of components based on the design parameters, to extensive work making modifications to the devices selected in order to achieve the desired functionality.

In the early stages of the project, the intention was to design each component of the system (except the radios, which would be a separate and larger project in their own right). Several methods were considered for communicating between the radios and the computers or microcontrollers on each end. These methods all included filtering techniques to separate the radio channel into voice and data channels. All either required too much bandwidth (cutting into the voice channel) or forced the data rate to unacceptably slow rates. In addition, it was realized that the homemade protocol was not robust enough to handle significant data loss on the channel.

Upon realizing the inability of our designs to adequately meet the needs of the project, we turned to the protocols and devices developed for packet radio. Packet radio is a system for transmitting data over radio, and is popular among amateur radio operators. It uses devices called Terminal Node Controllers (TNCs) to interface computers and radios, and a protocol (AX.25) to format the data for transmission. It was clear that this technology could be used in implementing the system. Additionally, much work had already been done in configuring TNCs to interface with G.P.S. units and transmit location. At this stage, the radios, TNCs (the PK-96

and PIC-E), and G.P.S. unit were obtained. The task that remained was to reconfigure these devices to allow for polling and response, as our system required, and also to compress the data so that the polling process could run in a more timely manner.

From that point, the work has been one of intense software development, accompanied by some modification of internal protocols and hardware. There are two major programming aspects to this project. The first is programming the microprocessor on the PIC-E to handle compression and storage of the G.P.S. data, and to respond to polling from the base station. This was a daunting task, as all of the software for the PIC-E is in assembly language. Ultimately, this is where most of the work during the last stages of the project was focused. The other major programming aspect was developing the software for the base station. This software needed to be able to track the boats, as well as log their positions and handle the polling. This would have been a big bottleneck for the project if the programming had begun after the rest of the system was working. Fortunately, it was possible to develop much of the code for this program even before packet radio was considered in the design.

The final stage of the project up to this point has been testing the system. This began over the summer of 2002, when the radios were taken to Doc Warner's in Alaska and tested for range and sound quality. Other aspects of the system that have been successfully tested include the polling function, the G.P.S. compression and transmission, and the tracking abilities of the base station software.

Future Developments

At this stage, the project has produced a functional prototype. From here, there are several tasks remaining to meet the original design criteria and prepare the system for

implementation in Alaska. These include building a housing for the system, adding other tracking functions, and configuring the software for the site in Alaska.

The physical installation of the system will present some challenges. The combination of wind, cold, salt water, and human interaction will quickly take its toll on any components that aren't completely waterproofed. The problem is complicated by the heat generated by the radios and TNC. While the ambient temperature of the operating environment will probably be cool enough to keep the system from overheating, there will need to be an additional heat sink attached to the PIC-E.

The original design called for buttons which users could push to indicate that they had caught fish. For a while, it seemed that, while the PIC-E was programmable, it lacked any available ports to connect these buttons to. In the late stages of this project, a set of ports that are used for programming the PIC-E, and so were believed unusable, were determined to be perfect for this application. Code has already been developed to implement the interrupts required for these buttons, but time delays and testing concerns have made it unfeasible to have them ready on the current prototype. Another item that was discussed early in the project was tracking the engine status. The engines on the boats have outputs for this type of feature, but at this point, no effort has been made to incorporate this into the design. Due to limitations on memory and inputs with the current PIC-E chips, adding this feature would have to wait for a second generation system. Monitoring the engine status was also deemed unnecessary by the customer as any motor malfunctions could be immediately reported via the radios installed for this project.

Lessons Learned

It is impossible to work on a project for over a year without learning a few things. Some of these lessons could have saved us some pain earlier on this project, while others just would have helped us to know where to direct our focus. The biggest lessons were that a project is easier completed in very small steps. Early in this project, effort was made to complete entire sections of the project (such as modifying the code for the PIC-E to parse the G.P.S. data, store it, and transmit it in response to polls). These were just too much to chew on at once, and the project became bogged down trying to identify where bugs were. Since so much was being changed between testing, it was nearly impossible to determine where things were going wrong. To our credit, while it took a lot of effort, the work that had been done while we were in this mind set was eventually included in the project, usually with only slight modifications.

Another important thing learned on this project had to do with build/buy decisions. In the early stages of the project, we were determined to design all of the components for the system. As problems were encountered and anticipated, it was realized that we couldn't design components as well as companies and organizations that had already put substantial effort into their designs. While it may have cost less for us to build these components, much was gained in terms of robustness. One significant aspect of design where this type of decision is easier is that of implementing existing algorithms. After attempts to develop a transmission protocol, it was realized that an existing protocol (AX.25) would provide the necessary functionality. In cases where some technology already exists, much is gained in terms of functionality and compatibility by using it.

Special Details

One detail of importance was a modification needed to power the Garmin 35 G.P.S. from the PIC-E. Upon connecting the G.P.S. to the PIC-E we were unable to get a signal from it. Testing quickly showed that the input impedance of the Garmin 35 LVC was much lower than what the PIC-E was designed to interface with and the input voltage had dropped from 5.0 V to 0.9 V. After pouring over the schematic it was discovered that a 10 K Ohm resistor was in series with the Voltage Regulator and the power connection to the G.P.S. Technical specifications where quickly looked up for the Voltage Regulator and we discovered that it was capable of outputting more than enough current for the system we've designed, however the additional draw of the G.P.S. would require a heat sink to be placed on the Voltage Regulator to control the heat build-up. It was determined that for the input impedance of the G.P.S. that would need to matched is on the order of 6 Ohms. As it was so small it was decided that a wire would simply be put in the place of the 10 K Ohm resistor. Once this was done the system began to work flawlessly.

Product Life-cycle

The units installed in the boats in Alaska will definitely need regular servicing. The elements of wind, rain, cold, and salt water could potentially corrode exposed parts in a season. While the enclosures for the radios and hardware will need to protect the equipment as much as possible, they will also need to be accessible so that parts can be repaired and replaced. The life cycle of this equipment will probably only be a few years, although replacing components as they fail could probably extend that life to five or more years. Fortunately the only parts that are exposed to the elements are the microphone and antenna which have a relatively inexpensive replacement cost compared to the rest of the system.

Miscellaneous

G.P.S. Pricing

After comparing the different G.P.S. systems available for our use and determining that the Garmin TrakPak G.P.S. 35 was the desired unit for our project we began to search online for the best price available. As it was discovered that the prices tend to fluctuate no fixed supplier could be determined who can give the best price at any future time. For this reason there will be a need to look into sale prices and bulk rates when ordering the remaining G.P.S. antennas.

The Tucson Amateur Packet Radio web site and personnel were of invaluable service to our project. Not only did they provide the equipment for our TNCs, but they also provided vast stores of information which aided us in the design and implementation of this system. As the original implementers of the AX.25 Link-Layer protocol the web site was extremely helpful in learning how to communicate with little loss of data. We are also extremely grateful for the help of Byon Garrabrandt for answering programming questions we had and helping us to understand the intricacies of the PIC16f84.

TNC Options

The choice to use the PK-96 to connect to the computer was made to simplify design and testing of the system. The PK-96 provided fully function method to test the capacity of the PIC-E to both recieve and transmit AX.25 packets. As a fully functional TNC the PK-96 aided the study of packet radio systems. However, the PK-96 contains a large number of features that are completely unneccesary and even complicate this project.

During the developmental process it was determined that certain settings needed to be changed to allow for smooth operations. First of all, the PK-96 defaults to only recieve standard ASCII characters 7 bits in length. This meant that it ignored all data that used all 8 bits in a byte. Fortunately there was only one byte in the WIDI compression that even needed the eighth bit, and it could be broken up and stored in the spare bits of the control byte.. Special commands also had to be used in order to send the desired types of packets to their desired locations. In order to transmit a UI frame the intended reciever had to be stored in the UNPROTO variable inside the PK-96. This was accomplished by sending the command "UNPROTO BT00xx"(where x is the boat number).

To actually transmit the frame a single letter K was then sent, followed by two line feeds then a break character. Immediately after this the boat responds with the data it has stored on board. This same sequence is repeated for every poll of each boat. As testing progressed it was discovered that certain standard ASCII characters that were transmitted by the PIC-E were being filtered out by the PK-96. These bytes had ASCII values less than 32, which are used for formatting and control commands (such as carraige return or backspace). Fixing this bug was the last small hurdle crossed before the system was able to function properly in tests.

Originally, the PK-96 was chosen to make testing the prototype easier. Since then, we've been able to get the same functionality out of simpler and less costly TNC's (such as the PIC-E). Therefore, in further applications of this project, the PK-96 will be replaced by a custom built TNC similar to the PIC-E.

Environmental Issues

Any one who has spent time near a sea water environment will tell you that stainless steel is just a myth. It just rusts slower. As the very air is permeated with salt water, and this project will spend the majority of its life in a boat on the water, it is extremely important that measures be taken to increase the longevity of the project. During testing in Alaska in the summer of 2002 it was determined that an electricians common antioxidant such as NoAlox worked to prevent the contacts within the antenna from rusting. By sealing base of the antenna with silicon caulking the contacts are doubly protected. The microphones were also directly exposed to the elements and it was determined that they function well however their RJ-45 connector is prone to extreme corrosion if exposed directly to water. To prevent ruining this connection the connector must be placed in a dry location, and periodically cleaned with a chemical aerosol electrode cleaner. The radios proved themselves to be both resistant to the weather and the mistreatment that they were subjected to. Further protection will be supplied to the radios by special enclosures that will keep them dry but still accessible.

Legal Issues

As only certain frequencies are allowed to be used for digital communication, it was necessary to apply for FCC licensing for frequencies that would serve this purpose. Through the efforts of this group two frequencies were purchased by Doc Warner's in 2002 for their area in Alaska for voice and data communications. Permission was obtained to use the license of another business' frequency that is licensed for northern Utah to allow legal testing of the prototype of the system in Logan.

Customer Support

The system is designed to be as user friendly as possible. The average user will see nothing more than two inconspicuous buttons and an antenna on the console of their boat. They will probably forget about the system completely until they return to camp and are presented with a map of where they were fishing that day and the locations of every fish that was reported caught plotted on that map. The guests will then have the ability to plan their days around the spots that are currently the "hottest". They will also be able to see the times that fish were caught, and be able to predict the best times to be fishing. The system requires only the push of the button every time that they catch a fish to accomplish all this.

This system will allow the staff to track current fishing trends, and communicate with boats that are less successful than others to help them have a more successful fishing experience. Having voice communication over the radios in 2002 allowed for much of this to occur. Doc Warner's is extremely concerned for the safety of their guests, and are especially worried about guests who choose to stay out past the boat curfew at night when it becomes difficult to spot obstructions in the water. This system will allow the staff to immediately locate the errant boat and assist them in returning to camp safely. The radio will also allow emergency communication if there are ever any problems with the boat.

Project Management and Cost Analysis

Project Management Summary

Completed Tasks

1. A working prototype has been built.
   a. Two fully functional PIC-E's without an enclosure that operates to design parameters has been constructed
   b. Waterproof momentary contact buttons to be used as Fish-Caught buttons have been selected.
2. Software for PIC-E has been written
   a. All necessary software has been written for PIC-E
   b. Special attention was paid to the potential of future upgrades in the development of this software.
3. Computer software has been written for base computer
   a. Software has been revised numerous times after inspections by the customer in order to better meet their needs.
4. Licensing
   a. Doc Warner's purchased the rights to two frequencies for the use of voice communication and this project.
5. Preliminary testing done in Alaska to determine Repeater need
   a. Range of forty watt radio determined to be sufficient without the need for an additional repeater.
   b. Connectivity problems in corrosive salt water environment solved
   c. Testing at this early stage also demonstrated to the customer the value of the system and it improved the funding situation.
6. Future markets of the product have been explored
   a. Conversations with various professionals have demonstrated an interest in the installation of similar systems for home and business use.
   b. Market niche defined as the producers of a fully customizable fleet tracking/monitoring system.
   c. Funding committed by Doc Warner's to assist in development of business opportunities.

Remaining tasks

1. Build production model and subject it to environmental testing in Alaska
   a. Select Enclosure for Radios that are mounted on boats that don't have dry storage
   b. Select best enclosure for PIC-E.
2. Implement fully Fish-Caught buttons
   a. Determine mounting specifications

b. Complete testing of system to insure robustness and longevity without needing a user reset.

3. Constructing PIC-E's for additional boats
   a. Purchase PIC-E's.
   b. Modify PIC-E's circuit layout as needed, including adding a heat sink
   c. Program PIC-E's
   d. Install a PIC-E on each boat
4. G.P.S.
   a. Purchase necessary connectors and mounting hardware
5. Develop User Interface instructions for both base computer and boats
   a. Simple instructions for fishermen
   b. Simple instruction set for base computer
   c. Advanced user instruction set for base computer for data analysis
6. Radios
   a. Purchase remaining radios and antennas
   b. Install radios on boats
7. Estimated Man-Hours needed to complete remaining tasks: 100 hours

Cost Summary

As the concept for this project was developed, the customers of Doc Warner's were consulted for their opinions as to what they would like to see this system be capable of. One of the men consulted is a managing engineer for Questar Corporation in Salt Lake City. After he had heard about the system that we were attempting to design, he asked one of his engineers to put together a bid and basic proposal for the system. Their proposal implemented a communication system with a phone patch attached to the base radio. This would allow the guests of Doc Warner's to make phone calls on the radios as well as communicate freely amongst themselves. The system would be mounted on wearable vests that would require recharging each night. This proposal carried with it a $76,000 bid. Concerns about this design were the potential for damage inherent with the vest, and the need to charge the batteries when a minimal power system is all that operates at night at Doc Warner's. Due to the remoteness of the

site, electricity is supplied by large diesel generators. These generators do not run between 10:30 p.m. and 5:00 a.m. to conserve fuel.

The preliminary concepts for our project allowed us to create a bid for $52,410. As the project developed we discovered that some items that were included in the bid were not necessary and were removed. The adjusted bid became $46,100. Our proposal was accepted due to the lower price tag and the ability to configure the system to capture more than just current location. With the completion of the project we discovered that our final production cost is $26,120. This is far below our bid, and even further below Questar's. An itemized budget can be found in Appendix B.

Facilities and Personnel

The staff at Doc Warner's will benefit greatly from this system, and only a few key people will need to be trained to maintain this system. This primarily includes the Maintenance Manager and on-site Boat Mechanic. They will need to understand basic electrical precautions that must be taken in a sea-water environment in order to preserve the components of this system. They will also need to be capable of replacing components of the system that may have failed for any reason.

Another person that will need to be trained is the Store Manager, who will manage the data stored on the base computer. They will need to be familiar with the methods for displaying and printing different data. While the program does allow for high level changes to be made through the menus, these configurations should only need to be made once, and then never changed again. This eliminates the need for specialized knowledge on the programming and configuration of the system. A base radio station has already been installed with a 12' antenna

elevated nearly 60' in the air and designed to have an extended range. The computer network that will host the fleet monitoring system has also already been installed.

Conclusion

Purpose of report

This report provides a thorough documentation of the engineering design processes that were used to solve the problem. It discusses at length the design solutions that were found in order to successfully implement this system. This documentation also briefly mentions some of the design alternatives that were considered and were not chosen. Nor can the amount of knowledge and experience gained be adequately expressed in so few pages. A simple solution that can be written in a single sentence often required hours of meditation and weeks of research to discover. Most importantly, through all of our study and research we gained the knowledge that we can confront a seemingly impossible task, and overcome the obstacles along the way. Through personal initiative, each member of this design team has had the opportunity to become thoroughly exposed to subjects within the realm of engineering that our traditional schooling did not provide. While the value of this system to Doc Warner's is great, the greater value for those who designed it lies in the creative learning process that was executed to accomplish our goal. This project has left an indelible imprint upon all who were involved in its journey to completion.

Objectives of project

The initial objective of this project was to provide a means whereby Doc Warner's could locate their fishing vessels. As solutions were discussed, it became apparent that a project of this type could accomplish much more than had initially been desired. Not only could a vessel be located, but a record of its movements could be kept. Data describing the location and type of

fish caught could also be stored. The data could also be displayed in a real time format, thus enabling the staff at Doc Warner's to identify the best fishing grounds for a given day quickly. This solution also allowed the use of voice communication in the event of an emergency, thus allowing quick responses to any emergencies on the water.The system designed also met it's objective of being extremely inexpensive when compared to similar systems available commercially. Another bid by a professional communications firm for the same system carried with it a $76,000 price tag. Our bid was much lower at $52,166. The actual cost for this finalized system is just over half of what we bid. As cost was one of the main design constraints, this project has far surpassed the goals that were set for it.

Robustness was a large concern for this project, especially considering the harsh environment that it will operate in. The only way to gain experience with the robustness of an element of the system would be to build it and test it. Therefore an existing piece of hardware that had already been through the rigorous testing processes would be preferable to the many learning models that we would have developed. The decision was then made to try to use existing hardware modified as necessary for this project. This was a way to assure the quality of the components and also reduce our costs. As there was less time needed to be devoted to testing the robustness, and there was not a need to make new models, time and money were saved.

The software also had to have a large amount of robustness, be able to handle interrupted and partial data, and provide a user interface that would not allow for the system to be corrupted by user errors. On the boats the only interface with the system are the buttons to record fish data. On the computer there are a number of different systems administrator commands available, but these will never need to be accessed by the end users. The system is designed so that the delay

times can be minimized between transmissions, to maximize the efficiency. These delays are all changeable at the base computer, however these settings can be saved and after they have been set the system can be run without modifying them. The only functions needed by the typical user will be simple functions to access and print data. Other functions have been protected in such a manner that the system cannot be damaged or altered by actions taken by the user. Thus, the software is fool-resistant, and can be used by someone who has not had intense training in its operation, which was another design goal.

Once the project concept was developed, hardware and protocols were selected. The limitations of the components placed constraints on the design. This included the amount of data that could stored and transmitted, the format which it would be transmitted between each device, and the manner in which it could all be accomplished. Each device had to interface successfully with the next without a loss of the pertinent data. Negotiating these connections was a significant challenge that had to be overcome at every stage of the project.

As the project neared completion, continued input from the customer refined and improved the product. Changes in the computer display were made after reviews from the customer. Further improvements will be made in the future to ensure that this product will always meet the needs of the customer. Future customers will also be able to count on the continued support of the design team to create the perfect fleet monitoring system for their needs.

Appendix A

Materials List

1. Motorola Radius 1225 UHF 20 channel radio
    a. Programmed with frequencies and desired settings
    b. Includes the following items
        i. Includes power cable
        ii. Fuse
        iii. Necessary connective hardware for power cable.

2. Antenna
    a. 5 dB gain 5/8 wave antenna(part #RAE 4014A)
    b. antenna cable(part #0180300B02)
    c. antenna mount(optional)

3. Garmin G.P.S. Trakpak 35
    a. Requires female DB9 connector

4. PIC-E
    a. Requires assembly, Kit includes:

i.	4	1.2K ohm resistor
ii.	3	2.2k ohm resistor
iii.	9	10k ohm resistor
iv.	2	22k ohm resistor
v.	3	100k ohm resistor
vi.	1	10k ohm Trimpot
vii.	5	0.1uF Mylar capacitor
viii.	2	1uF Electrolytic capacitor
ix.	1	100uF Electrolytic capacitor
x.	1	1N34 Germanium Diode
xi.	3	1N4002 Silicon Diode
xii.	2	1N4148 Silicon Diode
xiii.	1	1N5231BDICT-ND 5.1V Zener Diode
xiv.	1	Green LED
xv.	2	Yellow LED
xvi.	1	Red LED
xvii.	1	PIC 16F84 I.C.
xviii.	1	MX614 I.C.
xix.	1	2N3904 NPN Transistor
xx.	1	7805 Voltage Regulator
xxi.	1	3 MHz Ceramic Resonator
xxii.	1	10 MHz Ceramic Resonator
xxiii.	2	16-pin DIP IC Socket

xxiv.	1	18-pin DIP IC Socket
xxv.	1	16-pin Machine Tooled DIP IC Socket
xxvi.	3	1x2-pin male header
xxvii.	2	1x3-pin male header
xxviii.	3	1x8-pin male header
xxix.	12	2-pin Jumper, Push-On
xxx.	1	DB9 Female PCB right angle mount
xxxi.	1	DB9 Male PCB right angle mount
xxxii.	2	8-pin RJ-45 jack PCB right angle mount
xxxiii.	1	2.1 mm Coaxial Jack
xxxiv.	1	2.1 mm Coaxial Plug
xxxv.	1	Printed Circuit Board
xxxvi.	1	Cable Assembly with RJ-45 connectors
xxxvii.	2	Miniture Toggle Switches
xxxviii.	4	4-40 x 3/8" scres
xxxix.	4	4x40 nuts
xl.	1	Solid Wire

5. PK-96
    a. Comes with connector from PK-96 to radio, however an RJ-45 plug needs to be attached to one end of it.
    b. Requires 12V 500 mA transformer.
    c. Null modem cable (DB9 to DB25)

6. Enclosures
    a. Exact specifications on enclosures have yet to be determined.

Appendix B

Project Budget

Qty.	Description	Projected Unit Cost0	Actual Unit Cost	Projected Total	Actual Total
	MOBILE STATIONS				
20	Motorola Radius 1225 Mobile Radio with 5dB 5/8 wave antenna	$850	$615.50	$17,000	$12,310
20	Speaker Microphone	$67	$67	$1,340	$1,340
20	Node Controller	$399	$60	$7,980	$1,200
20	GPS Antenna with Decoder	$150	$170	$3,000	$3,400
20	Interface Cables	$50	$3	$1,000	$60
5	Waterproof Enclosure For Radio	$0	$100	$0	$500
20	Enclosure for PIC-E	$0	$12	$0	$240
40	"Fish Caught" buttons	$0	$5	$0	$200
	TOTAL COSTS FOR MOBILE STATIONS			**$30,320**	**$19,250**
	BASE CAMP				
1	Motorola Radius 1225 Mobile Radio	$850	$574	$850	$574
1	Desktop Microphone	$51	$40	$51	$40
1	Power Supply	$296	$30	$296	$30
1	Antenna and Line Kit	$500	$200	$500	$200
1	Node Controller	$399	$200	$399	$200
	TOTAL COSTS FOR BASE STATION			**$2,096**	**$1,044**
	REPEATER				
1	GR 1225 Repeater	$1,995	$0	$1,995	$0
1	Duplexer	$400	$0	$400	$0
1	Repeater Controller Cable	$73	$0	$73	$0
1	Preselector	$336	$0	$336	$0
1	ZR 310 Repeater Controller	$950	$0	$950	$0
1	I/C Repeater Controller	$950	$0	$950	$0
2	Antenna and Line Kit	$500	$0	$1,000	$0
3	RSS Software	$300	$0	$900	$0
	TOTAL COSTS FOR REPEATER			**$6,604**	**$0**

TOTAL EQUIPMENT COSTS				**$39,020**	**$20,294**
NON EQUIPMENT COSTS					
30	Hourly Rate for Installation Cost	$95	$95	$2,850	$2,850
TOTAL COSTS FOR LABOR				**$2,850**	**$2,850**
SUB-TOTAL				**$41,870**	**$23,114**
SHIPPING & HANDLING				**$1,335**	**$1,335**
TOTAL				**$43,205**	**$24,479**
SALES TAX				**$2,895**	**$1,641**
GRAND TOTAL				**$46,100**	**$26,120**

DEVELOMENTAL COSTS	
ESTIMATED MAN-HOURS FOR DEVELOMENT	
Research and design determination	250 hours
Base Computer Program development	300 hours
PIC-E assembly and programming	700 hours
Prototype testing	40 hours
Preliminary Radio installation and troubleshooting	50 hours
TOTAL MAN-HOURS	**1,340 hours**
Equipment and supplies used in developmental process	Less than $300

Appendix C

Connections and Schematics

This section includes schematics and pinouts that are necessary to build and test this system.

PIC-E Signals		Microphone Jack Signals	Wire Color	
9	Ground	8V	Brown	8
10	+5 Volts	Not Used	Brown/White	7
11	MIC Ground	Hook	Green	6
12	MIC Audio	Ground	Blue/White	5
13	RXA	Mic Audio	Blue	4
14	MIC PTT	Radio PTT	Green/White	3
15	Radio Ground	SCI	Orange	2
16	MIC Power	Handset RX Audio	Orange/White	1

Table: PIC-E interface to radio through microphone jack.

PK96	Color	Radio
1 TX	Blue	5 Mic Audio
2 GND	Blue/White	4 Ground
3 PTT	Grn/White	6 Mic PTT
4 RX	Orange/White	8 Handset Rx AUD
5 SQ	N.C	

Table: PK96 to Radio Connection

14Figure 14: Schematic of PIC-E.