

Design and Implementation of a Real-time, Chemical Sensor Network

by

Joseph Y. Wong

Submitted to the Department of Electrical Engineering and Computer Science

in Partial Fulfillment of the Requirements for the Degrees of

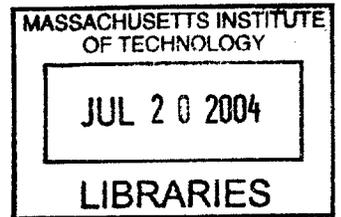
Bachelor of Science in Electrical Engineering and Computer Science

and Master of Engineering in Electrical Engineering and Computer Science

at the Massachusetts Institute of Technology

May 20, 2004 [June 2004]

Copyright 2004 Joseph Y. Wong. All rights reserved.



The author hereby grants to M.I.T. permission to reproduce and distribute publicly paper and electronic copies of this thesis and to grant others the right to do so.

Author _____
Department of Electrical Engineering and Computer Science
May 20, 2004

Certified by _____
Professor Harold F. Hemond
Thesis Supervisor

Accepted by _____
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

Design and Implementation of a Real-time, Chemical Sensor Network

by

Joseph Y. Wong

Submitted to the Department of Electrical Engineering and Computer Science

in Partial Fulfillment of the Requirements for the Degrees of

Bachelor of Science in Electrical Engineering and Computer Science

and Master of Engineering in Electrical Engineering and Computer Science

at the Massachusetts Institute of Technology

May 20, 2004

Copyright 2004 Joseph Y. Wong. All rights reserved.

Abstract

Current methods of environmental chemical data collection are limited in both time and space. This limited set of data inhibits researchers from fully understanding the chemical processes occurring in water bodies. In order to gain further insight, it is necessary to develop a new method of data collection that will increase the amount of data collected many times over. The proposed project will deploy a real-time, chemical sensor network at Upper Mystic Lake. Nodes in the network will collect and store massive amounts of chemical data for subsequent analysis by researchers. The major phases of the project include specification of the network, physical construction of network nodes, software development for control of nodes, and testing of network performance. The work will be completed in the Ralph Parsons Laboratory at MIT.

Thesis Supervisor: Prof. Harold Hemond
Title: Director, Ralph M. Parsons Laboratory

Acknowledgements

First off, I would like to thank my thesis advisor, Prof. Harry Hemond, who has provided me great guidance throughout this whole project. This project wouldn't have gotten off the ground without him. I would also like to thank Rob who took the time out to tutor me on the ways of the MOOS. I'd also like to thank Terry who did most of the mechanical work needed for this project and also helped with testing of the network.

Also in line for acknowledgement is my family who have always been there for me. I wouldn't have gotten to where I am now without them. Finally, I would like to thank Margaret who makes an often stressful and difficult place like MIT a happy one.

Table of Contents

- 1. Introduction**
 - 2. System Overview**
 - 3. Research Activities**
 - 4. MOOS Overview**
 - 4.1 Why MOOS?
 - 4.2 MOOS Basics
 - 4.3 Multiple MOOS communities
 - 5. MOOS at Upper Mystic Lake**
 - 5.1 Variable Sources and Destinations
 - 5.2 Design Critique
 - 6. GPS Interface**
 - 6.1 Hardware Interface
 - 6.2 NMEA Message Protocol
 - 6.3 Software Implementation - iUMLGPS
 - 7. Hydrolab Interface**
 - 7.1 Hardware Interface
 - 7.2 Communication Protocol
 - 7.3 Software Implementation - iHydrolab
 - 8. Thermistor Chain Interface**
 - 8.1 Thermistor Chain Hardware
 - 8.2 Thermistor Chain Software
 - 9. pShoreCommand**
 - 10. Buoy Prototype**
 - 10.1 Power Supply Design
 - 10.2 Rose Enclosure Penetrations
 - 10.3 Internal Enclosure Packaging
 - 11. Test Results**
 - 11.1 Multi-MOOS Test
 - 11.2 Upper Mystic Lake
 - 12. Future Work**
-
- Appendix A Linux and Networking Configuration**
Appendix B Thermistor Circuit Power Supply Calculations

List of Figures and Tables

Figure 2-1	Overview of nodes and connections
Figure 2-2	Depiction of Odyssey-class AUV
Figure 2-3	Simple block diagram of buoy components
Figure 3-1	Block diagram of simplified buoy
Figure 4-1	Typical MOOS Community
Table 4-1	Contents of a MOOS message
Figure 5-1	UML MOOS architecture
Table 5-1	MOOS data publishers and subscribers
Table 5-2	MOOSApp publishing frequencies
Figure 6-1	Magellan GPS 310
Figure 6-2	Power and data cable for GPS
Table 6-1	GPS wire to DB-9 pinout
Table 6-2	Magellan GPS 310 NMEA messages
Table 6-3	NMEA 0183 \$GPGGA format
Table 6-4	Meaning of \$GPGGA data
Figure 7-1	Hydrolab MiniSonde
Figure 7-2	Contact configuration of MiniSonde's connector
Figure 7-3	Contact configuration of DB-9 connector
Table 7-1	Pinout of MiniSonde bulkhead to DB-9
Figure 7-4	Example session between MiniSonde and Hyperterminal
Figure 7-5	Example MiniSonde initialization session
Figure 7-6	Typical raw characters from MiniSonde
Table 7-2	MiniSonde tokens and corresponding MOOS variables
Figure 8-1	Overview of computer, circuit, and chain interfaces
Figure 8-2	Front side of thermistor circuit board
Figure 8-3	Back side of thermistor circuit board
Figure 8-4	Pin configuration of Maxim 4638
Table 8-1	Pin description of Maxim 4638
Table 8-2	Pin description for Maxim 6682
Figure 8-6	Path from R- to GND
Figure 8-7	Reduced path from R+ to GND
Figure 8-8	Voltage divider formula
Figure 8-9	Thermistor chain to circuit board overview
Figure 8-10	Mapping of thermistors to bulkhead connector pins and to Ethernet pins
Figure 8-11	Thermistor chain Ethernet jack to circuit board interface
Figure 8-12	Computer to circuit board connection
Figure 8-13	Digital I/O pin numbering on TS-5500
Figure 8-14	PC/104 Ethernet Plug
Figure 8-15	Pinout from digital I/O to ribbon cable
Table 8-3	Digital I/O to Ethernet Pinout
Figure 8-16	Cable connecting computer to circuit board
Figure 8-17	PC/104 Ethernet jack to circuit board interface
Table 8-4	Pinout from Digital I/O to Ethernet jack

Table 8-4	Multiplexer pins used by computer
Table 8-5	Thermistor converter pins used by the computer
Figure 8-18	Timing diagram of digital I/O lines
Table 8-6	10 steps in timing diagram
Table 8-7	Propagation delays in thermistor circuit
Figure 8-19	Normal 2's complement interpretation of 11-bit number
Figure 8-20	Shifted 2's complement interpretation of positive 11-bit number
Figure 8-21	Shifted 2's complement interpretation of negative 11-bit number
Table 8-8	TS-5500 memory-mapped addresses
Figure 8-22	Example directions of digital I/O lines
Table 8-9	Example digital line outputs
Figure 10-1	Overall buoy design
Table 10-1	Buoy component power requirements
Figure 10-2	Power supply circuit diagram
Table 10-2	Power supply color code
Figure 10-3	3.3V Voltage Regulator – TI UA78M33CKC
Figure 10-4	5V Voltage Regulator – TI UA7805CKTER
Table 10-3	Buoy components power consumption
Figure 10-5	Overview of thermistor chain enclosure penetration
Figure 10-6	XSJ-7-BCR contact configuration
Figure 10-7	Ethernet plug contact configuration
Table 10-4	XSJ-7-BCR to Ethernet pinout
Figure 10-8	Overview of Hydrolab enclosure penetration
Figure 10-9	VSK-6-BCL contact configuration
Figure 10-10	Female DB-9 contact configuration
Table 10-5	Hydrolab penetration bulkhead to DB-9 pinout
Figure 10-11	Overview of field link enclosure penetration
Figure 10-12	VMK-5-FS contact configuration
Table 10-6	VMK-5-FS to Female DB-9 pinout
Figure 10-13	VSK-5-BCL contact configuration
Figure 10-14	Female DB-9 contact configuration
Table 10-7	VSK-5-BCL to Female DB-9 pinout
Figure 10-15	Front and side view of Rose enclosure
Figure 11-1	Buoy shore configuration for Multi-MOOS test
Figure 11-2	Shore configuration in Multi-MOOS test
Figure 11-3	Buoy logs from Mult-MOOS test
Figure 11-4	Shore logs from Multi-MOOS test
Table 11-1	Hydrolab Profile data from UML
Table 11-2	Average thermistor readings on land
Figure 11-5	Thermistor chain spacing
Figure 11-6	Conjecture of thermistor chain underwater
Figure 11-7	Comparison of UML thermistor and Hydrolab data
Figure A1-1	PC/104 wireless configuration in /etc/pcmcia/network.opts
Figure A1-2	PC/104 wireless configuration in /etc/pcmcia/wireless.opts
Figure A1-3	PC/104 hostname resolution configuration in /etc/hosts
Figure A1-4	PC/104 initialization script for serial ports: rc.serial

- Figure A1-5 PC/104 initialization script for MOOS: runMOOS
- Figure A1-6 PC Ethernet configuration
- Figure A1-7 Wireless configuration on shore station

1 Introduction

The chemical properties of Upper Mystic Lake have been the subject of research for many years. The lake is of great interest to environmental researchers due to prior pollution of its waters. In the early 1900's, various industries dumped toxic wastes into the lake and its tributaries, and many of these wastes remain there today. The focus of much research today is to determine the fate of these pollutants by monitoring changes in the lake's chemistry.

In order to monitor chemical changes in the lake, readings and samples of the lake's chemistry are taken periodically. Although much has been learned using this approach, there are two limitations. First, the time-consuming nature of traveling to the lake and sampling precludes researchers from collecting data more often than once or twice a week. Second, since researchers must manually use instruments to sample data at a single point in the lake, the number of points they are able to collect data at is restricted. The result is a data set that is limited in both time and space.

This project establishes a new data collection method that will greatly increase the amount of data collected. A network of sensors will be deployed at Upper Mystic Lake that will autonomously collect and store data. The network will have three types of nodes: 1) stationary buoys containing sensors 2) a mobile AUV (Autonomous Underwater Vehicle) equipped with sensors 3) a computer on the lake shore. The stationary buoys will be deployed at various points in Upper Mystic Lake and report

collected data through a wireless network. The AUV will traverse the lake while collecting data and communicating with the stationary buoys acoustically. The buoys and AUV will send their collected data to the shore station where it will be logged for later analysis, and potentially real-time analysis.

This system has two main benefits. First, the buoys and AUV can be deployed at Upper Mystic Lake virtually 24 hours a day. Second, the multitude of sensors and mobility of the AUV allow us to track many more points in the lake. Therefore, the spatial and temporal resolution of the data collected will increase many times over. This additional data will give researchers more insight into how the lake's chemistry evolves.

The remainder of this document is as follows. The next section presents a more detailed description of the function of each of the nodes in the network. Chapter 3 describes the scope of this thesis project within the larger network project. Chapters 4 and 5 describe the software platform for this system and how it is used. Chapters 6, 7, 8, and 9 discuss the software written for this system including the sensor interfaces. Chapters 10 and 11 discuss the construction of a buoy prototype and the testing of the prototype.

2 System Overview

This chapter describes how the network is envisioned to function after it is completed. The functions described are not all necessarily currently being implemented. In addition, note that this document only focuses on the *collection* of data and ignores the *storage* and *distribution* of data following collection.

The Mystic Lake network is comprised of three types of nodes: 1) AUV 2) Buoy 3) Shore station. Each node is connected to other nodes through a network connection. The different nodes and connections are shown in **Figure 2-1**.

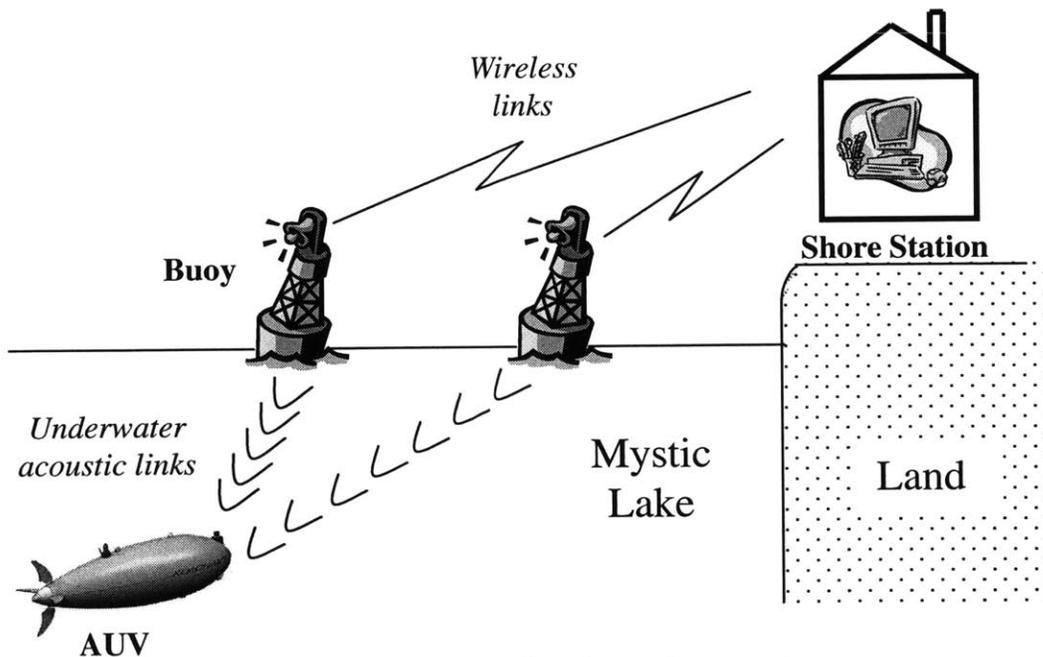


Figure 2-1: Overview of nodes and connections

The connections shown above provide the infrastructure through which data can be transported from sensors in the lake to the shore station computer. The buoy and AUV nodes will be responsible for collecting data from the lake while the shore station node

will log the collected data. The following sections further describe the network links and function of each of the three node types.

2.1 Network Connections

The network links allow all nodes to communicate with each other. Not only does this facilitate transmission of collected data, but also coordination of behavior between nodes.

2.1.1 Underwater Link

The AUV and buoys will be equipped with acoustic modems. Acoustic modems have been used in many other applications, but their range and reliability in a lake environment is uncertain at this point. Further testing is needed to evaluate their capabilities. Ideally, the acoustic modem should have a range of at least 1km (the largest diameter of the lake) such that it can communicate with any buoy in the lake.

2.1.2 Wireless Link

The wireless link will be used for communication between any of the buoys and the shore station computer. The buoys may also wish to communicate with each other through their wireless links. We have chosen to use 802.11b as our data layer protocol since it is currently the most widespread and developed standard. Thus, compatibility issues should be kept to a minimum and it gives us more flexibility in choosing appropriate hardware.

The wireless cards should have a range of approximately 1km as well. The shore station computer will be right next to the lake, so a range of 1km each for the shore station and

the buoy will ensure an adequate signal. Normally, unmodified wireless cards have a range of at most 500m. Therefore, we will likely use external antennas and/or RF power amplifiers with the wireless cards to boost their range.

2.2 AUV

The AUV will provide a mobile and dynamic element to the network. It will traverse the lake in a specific pattern while continuously collecting data. Furthermore, the AUV will also be able to dynamically respond to events in the lake. For example, a buoy may sense an abnormal amount of methane in a particular part of the lake and will summon the AUV to take a closer look. The AUV will be equipped with not only more sensors, but also sensors of higher accuracy than the buoy and may be able to collect a greater amount of useful data.

The AUV is being designed and constructed at the MIT AUV Lab with whom we are collaborating on this project. The design of the AUV is based on the Odyssey II class design. **Figure 2-2** below shows a depiction of an Odyssey-class AUV.

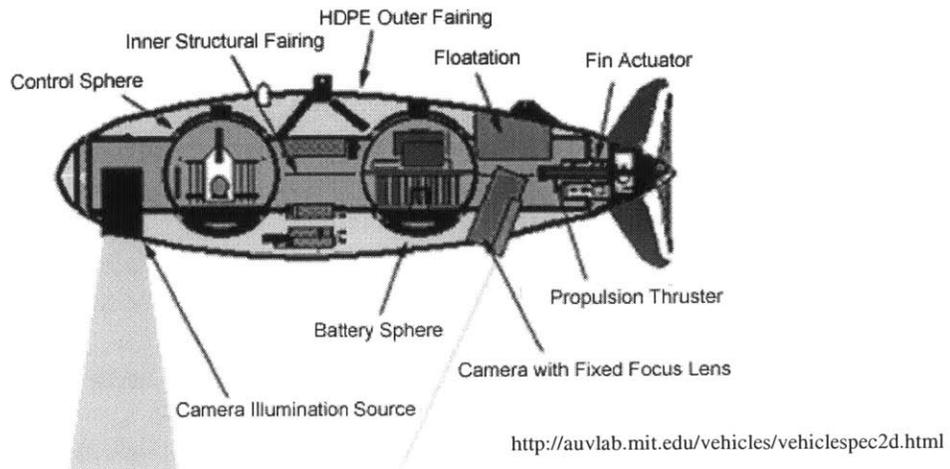


Figure 2-2: Depiction of Odyssey-class AUV

The main instrument aboard the AUV will be the NEREUS underwater mass spectrometer. NEREUS is an underwater mass spectrometer designed to measure dissolved volatile gasses in the water column [1]. The AUV will also contain an onboard computer system running the MOOS (Mission Oriented Operating Suite) system for navigation and control of the vehicle.

2.3 Buoy

The buoy will be a floating container holding a computer and sensors, moored to remain nearly stationary in the lake. A simple block diagram of the buoy is given in **Figure 2-3**.

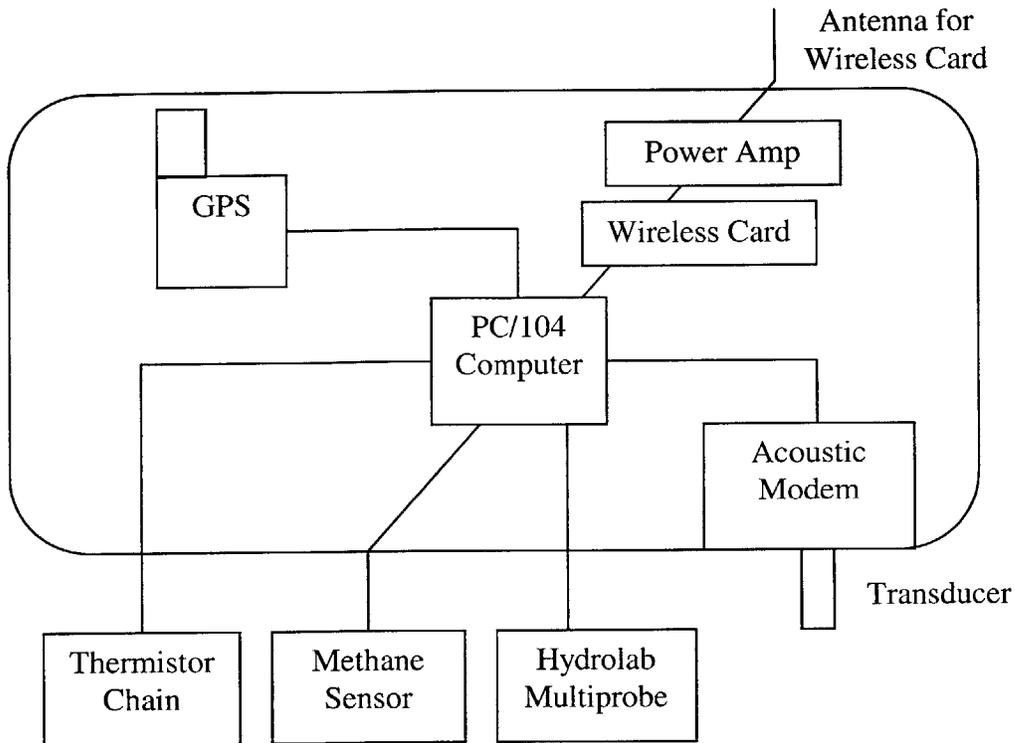


Figure 2-3: Simple block diagram of buoy components

The central component of the buoy is the PC/104 embedded computer. This computer is responsible for controlling all actions of the buoy. The term “PC/104” refers to the compact, stackable mechanical architecture of the computer. PC/104 computers generally have less processor power and memory than desktop computers, but their compact size makes them ideal for embedded computing.

The buoy has three jobs in the network. The buoy’s first job is to collect data of its own. The thermistor chain, methane sensor, and multiprobe can sense a variety of chemical attributes of the water. The GPS device can determine the exact location of the buoy.

Using these sensors, the buoy will be able to collect a large amount of data at a single point in the lake. It will send this data over the wireless link to the shore station.

The buoy's second job is to provide a link between the AUV and nodes above water. Since the AUV is underwater, it can only communicate acoustically with other nodes that are underwater. However, the buoy can use its wireless and acoustic communication links to as a relay between nodes above water and nodes below water. One application of this ability is that the buoy can relay the AUV's collected data to the shore station while the AUV is underwater. Thus, the AUV can continue its search pattern underwater while it transmits data, instead of surfacing to transmit its data.

The buoy's third job is to aid in the navigation of the AUV. Each buoy can determine its location using its GPS device. Then, each buoy can send its location information to the AUV which can then determine its own location using acoustic time-to-travel to each buoy and triangulation (assuming there are 3 or more buoys).

2.4 Shore Station

The main purposes of the shore station are to act as the master data repository and central command station for the network. The shore station will consist of a computer and a weather station. It will likely reside at a building on the shore of the lake. In the future, an Internet connection will be added to the shore station to facilitate remote communication.

All data generated in the network should end up at the shore station. It will also collect data from the weather station. All data received or collected by the shore station will be stored on its hard drive or transferred to a remote location using its Internet connection. In addition, the shore station will analyze the data in real-time to determine if any new behavior in the network should occur. If new behavior is required, the shore station automatically will send out the appropriate commands to direct such behavior.

2.5 System Design Goals

In the previous sections, I described the functions of each of the main components in the system. To produce a satisfactory network, the design and implementation of these functions must fulfill four main requirements. First, the network must be *autonomous*. Once deployed into the lake, the network must perform all of its functions without human interaction. At most, a researcher will have to travel to the lake periodically to download collected data or perform maintenance. This lack of human interaction is directly related to the second requirement: the network must be *robust*. The network must be able to identify, log, and recover from errors that occur while it is running without any human interaction. Furthermore, the network must be able to reliably collect and record data at specified intervals. The third requirement is that the network must be *scalable*. This project will lay down the network infrastructure and an initial set of buoys and sensors. In the future, it should be easy to add additional nodes and sensors to the network. The final requirement is that the network must be *flexible*. Upper Mystic Lake is being used as an initial testing site for the network, but the network should be able to function in any aquatic environment.

3 Research Activities

Implementing this sensor network is a large-scale project meant for multiple people and multiple years, so this thesis work only deals with a section of the network. The goal of this thesis was to deploy a smaller-scale sensor network consisting of one shore station and one buoy at Upper Mystic Lake as a proof-of-concept step. The buoy has three sensors: a GPS, a multiprobe device, and a thermistor chain. The buoy collects data from these sensors and transmits the data to the shore station. A diagram of the simplified buoy is given below in **Figure 3-1**:

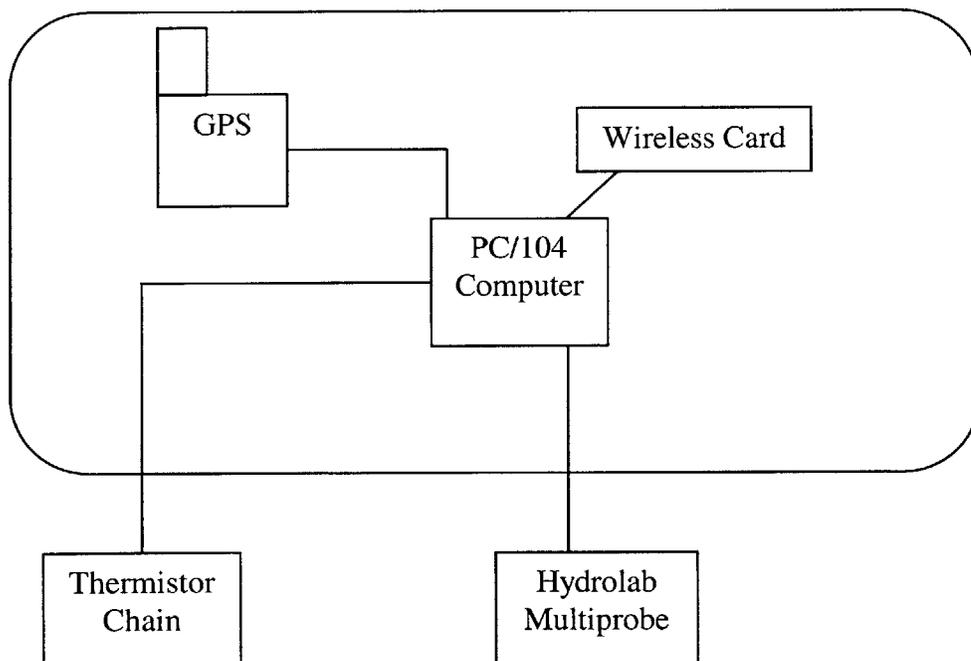


Figure 3-1: Block diagram of simplified buoy

The shore station then processes this data to determine if the behavior of the buoy should change. This network demonstrates the data collection, wireless transmission, and adaptability functionality of the final network.

Note that the following elements are missing from the network. There is no AUV node and consequently no acoustic communication. Furthermore, the lack of an AUV also means there is no NEREUS mass spectrometer. On the buoy, there is a limited set of sensors and no power amp or external antenna. The absence of a power amp circuit or external antenna will limit the range of wireless transmission in the network. Finally, the adaptability implemented in the network is extremely limited. The lack of an AUV limits the amount of dynamic behaviors the network is capable of performing.

Implementing this network involved the following tasks:

- Collaboration on design of the network architecture
- Specification of data types and flow of data within the network
- Development of protocols to communicate with the three sensors
- Development of software to collect data from these sensors
- Development of software to process collected data and to direct adaptive behavior if necessary
- Aid in design of prototype buoy
- Aid in physical construction of internal buoy structure
- Testing and evaluation of network performance and reliability

The next section covers the software platform upon which the network runs.

4 MOOS

MOOS (Mission Oriented Operating Suite) is the software platform upon which the network runs. It was originally created by Prof. Paul Newman in collaboration with the AUV Lab to coordinate all the necessary tasks on an AUV. These tasks include collecting sensor data to assist in navigation, making navigational decisions, and physically controlling the AUV. This large number of tasks requires the software of an AUV to simultaneously perform and coordinate multiple activities. MOOS provides a reliable platform on which all these activities can be executed. Although MOOS was originally designed to run an AUV, it is general enough to use in any application that requires coordination among several processes.

This chapter will give a summary of how the MOOS system works that should allow the reader to understand the rest of this document. Further information about MOOS can be obtained from Prof. Paul Newman's internal paper about MOOS located at <http://www.robots.ox.ac.uk/~pnewman/papers/MOOS.pdf>.

4.1 Why MOOS?

Why did we choose to use MOOS? The primary reason for using MOOS is to ease integration of the buoys and shore station with the AUV. The AUV Lab at MIT uses MOOS almost exclusively to run their AUVs. Therefore, it will ease communication between the network's nodes if the buoys and shore station also run on the MOOS system.

Another reason to use MOOS is its maturity as a software system. The first benefit of its maturity is that it is a well-tested and proven system. Therefore, MOOS's reliability is likely higher than any new software we could develop. The second benefit is that MOOS already provides much of the functionality we need. Instead of having to develop our own similar system, we are able to save time by building on top of the existing MOOS functionality.

4.2 MOOS Basics

MOOS is a software suite that facilitates communication between computer processes. A set of processes running on MOOS is referred to as a MOOS community. A MOOS community is arranged in a star topology. At the center of the star is a database, referred to as the MOOSDB. On the perimeter of the star are the coordinating processes, referred to as MOOSApps. **Figure 4-1** below depicts a typical MOOS community that might live on an AUV:

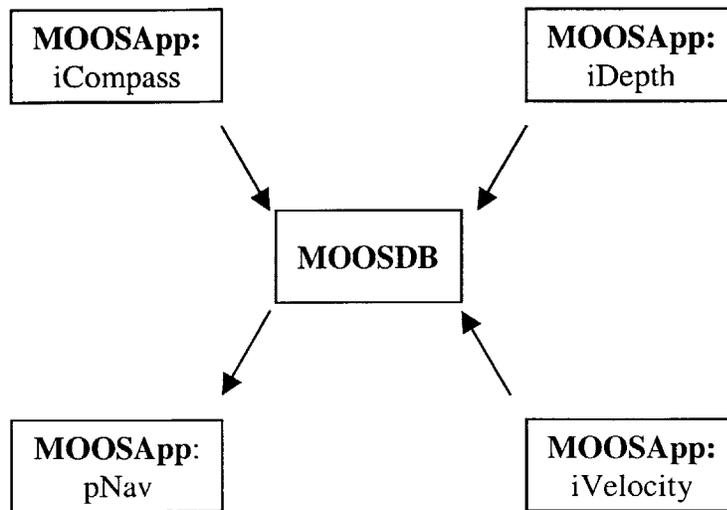


Figure 4-1: Typical MOOS Community

Each MOOSApp is responsible for a particular task. In order to perform each of their tasks properly, MOOSApps typically need to share information. Consider the example MOOS community in **Figure 4-1** above. pNav is responsible for navigating the AUV. In order to properly do so, pNav needs additional information about the direction, depth, and current velocity of the AUV. iCompass, iDepth, and iVelocity are responsible for collecting data from the corresponding sensors that reside on the AUV. MOOS allows pNav to receive the data collected by these three data-collecting processes and use that data to make better decisions.

All information is passed between MOOSApps in the form of *messages*. The contents of a MOOS message are given in **Table 4-1**:

Variable	Contents
Name	The name of the data
String Value	Data in string format
Double Value	Data in numeric double format
Source	Name of MOOSApp that sent this data to the MOOSDB
Time	Time at which data was written
Data Type	Type of data sent (STRING or DOUBLE)
Message Type	Type of Message (usually NOTIFICATION)

Table 4-1: Contents of a MOOS message

The MOOSDB is responsible for receiving and forward messages between the MOOSApps. All communication between MOOSApps flows through the MOOSDB. How communication occurs is covered more in-depth in the next section.

4.2.1 MOOS Message-Passing Protocol

Data is passed among MOOSApps using a blackboard model. Under this blackboard system, a MOOSApp can perform three actions on data:

- 1) Publish a notification on named data
- 2) Subscribe for notifications on named data
- 3) Collect notifications on named data for which the MOOSApp has subscribed

A publishing process periodically publishes notifications on a piece of data. Any process can subscribe for notifications on a piece of data. The publishing process has no knowledge of which processes have subscribed for any data it publishes. Once a piece of data is published, all processes that have subscribed for that piece of data can collect the notification.

The best way to understand this protocol is through an example. Assume that on the AUV there are two MOOSApps: iDepth and pNav. The process iDepth collects data from a depth sensor and pNav navigates the AUV. iDepth publishes one variable: DEPTH_DEPTH which represents the depth the depth sensor is measuring. pNav is interested in DEPTH_DEPTH since it would like to know the location of the AUV.

Therefore, we may see the following events occur:

- 1) pNav subscribes for notifications on DEPTH_DEPTH
- 2) iDepth collects data from the depth sensor and then publishes notifications on DEPTH_DEPTH
- 3) At a later time, pNav collects notifications for DEPTH_DEPTH

Note that all of this communication is coordinated by the MOOSDB.

4.3 Multiple MOOS Communities

While the centralized nature of the star topology eases coordination between processes, performance and reliability become issues when a large number of processes are present. First, since all communication flows through the MOOSDB, a large number of processes can overload a single MOOSDB. Second, the star topology causes all the processes to become dependent on one process: the MOOSDB.

This is exactly the case in the sensor network where there are a large number of nodes and applications attempting to share data. Therefore, instead of having one large MOOS community and database, each node is a self-contained MOOS community that has its own MOOS database. This configuration reduces the bottleneck effect at a given MOOS

database and also creates a more distributed architecture. However, now the question becomes how to share data among different MOOS communities. The answer is to use two MOOSApps that are able to bridge two MOOS communities: pMonitor and pMonitorListener.

4.3.1 pMonitor

pMonitor allows a third-party application to monitor the activity in a MOOS community. pMonitor is configured to collect notifications on a set of variables and sends text versions of those variables to a predetermined IP and port. The text version of the variable is in the format “VARIABLE_NAME=VALUE”.

4.3.2 pMonitorListener

pMonitorListener is a companion application to pMonitor. It listens on a predetermined IP and port for incoming messages. It assumes messages are of the form published by pMonitor. When it receives a message, it will parse the message to determine if the message is a notification for a variable that it is configured to listen for and if the value is of the proper type. If so, it will publish the variable to its MOOSDB.

4.3.3 Multi-MOOS example

Let’s go through an example in which an application on the shore station, pCommand, would like to monitor GPS data at one of the buoy nodes.

- 1) **Configuration** – pMonitorListener is at 192.168.0.2 and is configured to listen on port 5000. pMonitor is configured to send messages to 192.168.0.2 on port 5000.
- 2) **Startup** – The shore station node starts the pMonitorListener application. The buoy node starts the pMonitor application. pMonitor connects to the buoy's MOOSDB. pMonitor also connects to pMonitorListener at the shore station.
- 3) **Subscription** - pMonitor subscribes to collect notifications on the buoy's GPS data. pCommand on the shore station subscribes to collect notifications on the buoy's GPS data.
- 4) **Collection** – Each time GPS data is published, pMonitor collects that data and sends it to pMonitorListener on the shore station. When that data arrives at the shore station, pMonitorListener parses and publishes the data to the shore station's MOOSDB. Then, pCommand collects those notifications on the GPS data.

Note that iGPS and pCommand are completely unaware of pMonitor and pMonitorListener. Both applications only interact with their local MOOSDB and are unaware of where data goes or comes from. The advantage of hiding pMonitor and pMonitorListener in this way is that a different application could be used to share data among communities in the future. For example, one could combine pMonitor and pMonitorListener into one MOOSApp that captures both functions.

5 MOOS at Upper Mystic Lake

Let's take a more detailed look at MOOS in action in the UML network. **Figure 5-1** below gives an overview of the network architecture. A community consisting of a MOOSDB and a set of MOOSApps lives at each node. The arrows between processes indicate communication between those two processes.

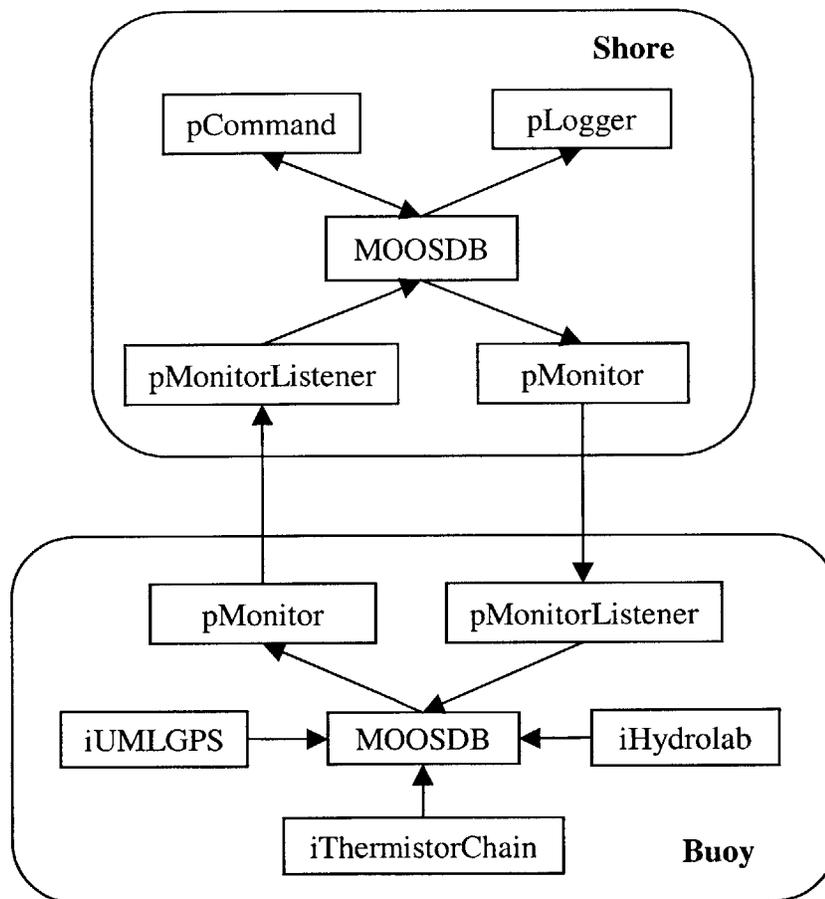


Figure 5-1: UML MOOS architecture

Let's start at the buoy node. There are three MOOSApps collecting sensor data: iUMLGPS, iHydrolab, and iThermistorChain. As their names suggest, these processes collect data from a GPS device, a Hydrolab MiniSonde, and a thermistor chain respectively. These three processes feed their collected data to the MOOSDB. The

buoy's pMonitor listens for the data collected by these MOOSApps and forwards them to the shore station's MOOSDB. There, the data is received and processed by pShoreCommand. If pShoreCommand determines that new action is needed from a MOOSApp, it will send a command to that MOOSApp through the shore station's pMonitor. A pLogger process resides at each node and records all messages passed between MOOSApps.

5.1 Variables Sources and Destinations

As described previously in Section 4.2.1, MOOS uses a blackboard model for interprocess communication. Table 5-1 below summarizes the variables that each MOOSApp publishes and the MOOSApps that subscribe to each variable.

Variable	Publisher	Subscribers
BUOY1_GPS_LONG	iUMLGPS	pShoreCommand, pLogger
BUOY1_GPS_LAT	iUMLGPS	pShoreCommand, pLogger
BUOY1_GPS_SAT	iUMLGPS	pShoreCommand, pLogger
BUOY1_THERM_1	iThermistorChain	pShoreCommand, pLogger
BUOY1_THERM_2	iThermistorChain	pShoreCommand, pLogger
BUOY1_THERM_3	iThermistorChain	pShoreCommand, pLogger
BUOY1_THERM_4	iThermistorChain	pShoreCommand, pLogger
BUOY1_THERM_5	iThermistorChain	pShoreCommand, pLogger
BUOY1_THERM_6	iThermistorChain	pShoreCommand, pLogger
BUOY1_HYDROLAB_EXTBATT	iHydrolab	pShoreCommand, pLogger
BUOY1_HYDROLAB_TEMP	iHydrolab	pShoreCommand, pLogger
BUOY1_HYDROLAB_SPCOND	iHydrolab	pShoreCommand, pLogger
BUOY1_HYDROLAB_DO%	iHydrolab	pShoreCommand, pLogger
BUOY1_HYDROLAB_PH	iHydrolab	pShoreCommand, pLogger
BUOY1_HYDROLAB_DEP25	iHydrolab	pShoreCommand, pLogger
BUOY1_HYDROLAB_ORP	iHydrolab	pShoreCommand, pLogger
BUOY1_GPS_COMMAND	pShoreCommand	iUMLGPS, pLogger
BUOY1_THERM_COMMAND	pShoreCommand	iThermistorChain, pLogger
BUOY1_HYDROLAB_COMMAND	pShoreCommand	iHydrolab, pLogger

Table 5-1: MOOS data publishers and subscribers

Note the convention used in the variable names here: LOCATION_SENSOR_DATA.

Normally, MOOS names use the convention SENSOR_DATA. However, the use of the

same sensors at multiple nodes (e.g. multiple thermistor chains) would introduce a replication of names if the location of the data is not added to the variable name. Each application publishes its variables at the frequencies shown in **Table 5-2**.

MOOSApp	Publishing Frequency
iUMLGPS	1 / minute
iHydrolab	1 / 5 minutes
iThermistorChain	1 / 5 minutes
pShoreCommand	Variable

Table 5-2: MOOSApp publishing frequencies

Note that these are the default frequencies and can be changed based on the environment.

iUMLGPS publishes its last recorded longitude and latitude along with the number of satellites it is in contact with. iThermistorChain publishes the temperature of the 6 thermistors on the thermistor chain. iHydrolab publishes a variety of chemical data measured by the Hydrolab.

pShoreCommand issues commands to iUMLGPS, iThermistorChain, and iHydrolab. Currently, pShoreCommand is only able to issue one command to a MOOSApp: it instructs the MOOSApp to change the frequency with which it collects data. The set of commands that pShoreCommand will be extended in the future as the network develops and integration with the AUV occurs. Note that pShoreCommand is the only process in the network able to issue commands to other processes. This results in a centralized command architecture in which one process is responsible for directing many, many processes.

A pLogger is located at each node. pLogger subscribes to every variable that the MOOSApps in its community publish or subscribe to. Note that in this case, this policy means that both pLoggers log every variable published in the network. The logs that pLogger creates serve as archives for all data collected in the network and can also be used later to analyze the network's behavior.

Although they are not shown, note that pMonitor and pMonitorListener are also publishing and subscribing to variables. Every piece of data that travels its home community to a partner community is subscribed to by a pMonitor within its home community and published by a pMonitorListener in the partner community.

5.2 Design Critique

This design seems feasible for the current, smaller network, but how will it behave in the future when more nodes are added? For this section, I will assume that future networks will have a single shore station, multiple buoys, and even multiple AUVs. The network will be evaluated in terms of two of the four design goals set forth in **Section 2.5**: scalability and reliability.

5.2.1 Scalability

This architecture works for 2 nodes, but how will it behave when the network size is increased to 5 or 10 nodes? The network is centered around the shore station as a central data repository and command station. These two jobs of the shore station become more difficult as the size of the network grows.

Central Data Repository

While the use of pMonitor and pMonitorListener to create multiple MOOS communities may slightly alleviate the bottleneck at the shore station, the fact remains that all data collected in the network ends up at the shore station. As the number of nodes increases, the shore station must be able to process an increasing rate of data transfers from buoy nodes. However, upon further inspection, one can see the size of data transferred is small compared to the bandwidth of the wireless link. From **Table 5-1**, it can be estimated that a buoy node may produce approximately 25 pieces of data once more sensors are added and the data from the AUV is added to the network. Note that all data currently produced in the network is numeric or text. Let's assume that each piece of data might take 1 kilobyte to capture. This is likely an overestimate, but it allows us to use round numbers. That means a buoy may produce about 25 kilobytes of data per minute if we assume the frequencies from **Table 5-2**. The bandwidth of a standard 802.11b wireless link under good conditions is approximately 100 kilobytes /sec. Thus, the shore station can transfer all the data from a buoy in .25 seconds. If there were 10 buoy nodes in the network, it could service all the nodes in the network in 2.5 seconds. Given that the shore station needs to service each node approximately once per minute, it's feasible to state that the central data repository scheme will be able to work well under the number of nodes we expect to be in the network.

Central Command Station

In addition to acting as a data repository, the shore station must also analyze incoming data and coordinate the behavior of all nodes in the network. The centralized command architecture certainly puts more strain on the shore station, but makes it simpler to manage the behavior of nodes. All of the data is already stored at the shore station, so it has access to the greatest amount of data and can make the best decisions. Furthermore, if a human were to intervene in the command loop, it is convenient for a human to access the shore station. A distributed command architecture would likely necessitate additional efforts to ensure reliable coordination among multiple command stations.

In addition to simplicity, the amount of commands sent to nodes should be considered. Commands are generally only needed when a significant event in the lake occurs. Examples of significant events include unusual concentration of gases in parts of the lake or unusual weather conditions that may call for additional sampling. These types of events will likely occur on the order of hours apart. Therefore, although the shore station is constantly processing large volumes of data, it is not sending out a large number of commands to nodes which reduces the strain on the shore station.

5.2.2 Reliability

The second goal to be considered here is reliability. How will the network react if one or more nodes fail? How will the network behave if a network connection becomes unavailable? Using multiple MOOS communities significantly increases the network's reliability in this regard.

Loss of Nodes

Let's consider how the current architecture behaves versus one in which there is only one MOOS community centered around a MOOSDB located on the shore station. One possible failure in the network may be that one of the buoy or AUV nodes may stop producing data for some period of time. In both architectures, the loss of a perimeter buoy such as a buoy or AUV would not affect the rest of the network. The only negative effect would be a gap in that node's data set.

Another possible failure in the network may be that the shore station stops functioning and the MOOSDB located on it stops executing. In the multiple community case, the loss of the shore station would prevent access to the data repository and prevent any adaptive behavior from nodes. However, the perimeter nodes would still be able to execute their default behavior and continue to collect and log data locally through pLogger. Although this is not currently implemented, one could envision the nodes using the logs created by pLogger to send the shore station any data collected during the down time once it comes back online. One could also envision the nodes recognizing the loss of the shore station and coordinating behavior among themselves using pMonitor and pMonitorListener. By making each node a self-contained community that has its own MOOSDB, there is structure within each node and with that structure comes reliability.

In the single community case, the loss of the shore station would cause greater damage. All of the nodes had been depending on the single MOOSDB at the shore station to store

data and coordinate behavior. Without the shore station, these two functions are no longer available to the network. One could certainly program MOOSApps on the nodes to store data locally and talk with other MOOSApps without the use of a MOOSDB, but this would go outside of the existing structure created by MOOS. Implementing such features outside of MOOS is risky as it is no longer as well-defined and, more importantly, would require re-implementing similar features already contained in other parts of MOOS.

Loss of Network Connection/s

Given that the network is being deployed in an outdoor environment, the loss of network connections can occur due to any number of factors that exist in the field. A fierce thunderstorm could hover over the lake and shut down the wireless links. A boat may crash into a buoy and damage the acoustic modem's transducer causing the buoy to lose contact with the AUV. Losing either network connection will affect the behavior of the nodes in the network.

There are three cases to consider in evaluating the loss of a buoy's wireless link. The buoy may lose contact with the shore station, other buoys, or both.

Case 1 – No contact with shore station: In this case, the buoy cannot communicate with the shore station, but can still communicate with other buoy nodes. Assuming the buoy can reach another node that has contact with the shore station, one can envision the buoy reconfiguring pMonitor to connect to the new

node's pMonitorListener. The data from the disconnected buoy could then be forwarded to the shore station through the new node.

Case 2 – No contact with other buoys: This will have minimal effect on the network. The buoy can still transfer all data to the shore station.

Case 3 – No contact with either shore station or buoy: The buoy will be unable to transfer data to the shore station through its own wireless link or other nodes. It's possible that it may be able to route data through the AUV, but the acoustic link has extremely low bandwidth and it may not be advisable. The worst case is that the buoy locally logs data and waits until the network links reappear.

The loss of the acoustic connection will prevent the AUV from sending data or receiving commands. Neither case is particularly damaging as the rest of the network can still function. Similar to the buoys, the AUV could locally log data during this down time.

6 GPS

The GPS is a small handheld device that provides location information among many other pieces of data. It provides a host of navigational and waypoint functions that are meant to be used in aquatic or wilderness situations in which it is difficult to determine where one is heading. For the purposes of this network, we are only interested in the latitudinal and longitudinal information that the GPS provides.



Figure 6-1: Magellan GPS 310

The interface between the GPS and the PC is rather straightforward. A data cable connects the GPS to a serial port on the PC. Once the GPS is turned on, it streams out location information over the data cable to the PC. The GPS encodes transmitted data in the NMEA 0183 message format (version 2.2). The PC can then read this data from its serial port and parse the messages for the desired data.

6.1 Hardware Interface

A Magellan PC Data cable was used to connect the GPS to the PC. This cable not only connects the GPS to a DB-9 serial port, but also provides external power to the GPS. A picture of the cable is in **Figure 6-2** below.

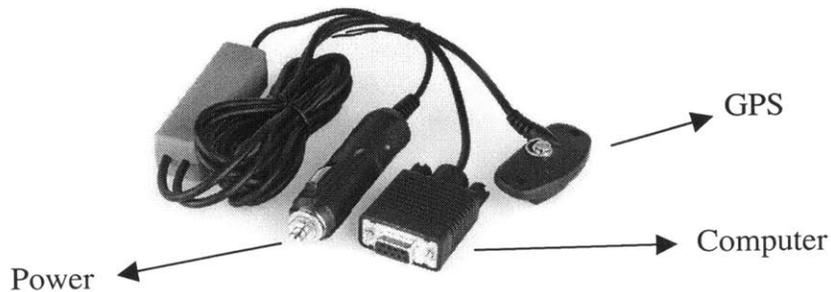


Figure 6-2: Power and data cable for GPS

The cable only has four wires: transmit, receive, power, and ground. On the GPS side of the cable, these four wires connect to the GPS through 4 metal contacts. On the power side of the cable, the power and ground wires connect to the cigarette lighter adapter. Finally, the four wires correspond to the following four pins on the DB-9 adapter shown in **Table 6-1**.

GPS	DB-9	Color
Power	4	Red
Ground	5	Black
GPS Transmit	3	Orange
GPS Receive	2	Yellow

Table 6-1: GPS wire to DB-9 pinout

Some alterations were made to the cable in order to provide power to the GPS at a lower voltage. Specifically, the gray box and cigarette lighter adapter were removed from the cable. The result is a cable that has a DB-9 connector on one side, the 4-contact GPS connector on the other side, and a breakout point in the middle where power is provided. At that point in the cable, between 3.5 and 4.0 volts is sufficient to power the GPS.

6.2 NMEA 0183 Message Format

The NMEA 0183 message format defines a set of messages that are used to encapsulate GPS information. Each message has a name and associated data. NMEA defines what data is associated with each message and the format that data is transmitted in. The Magellan GPS 310 streams out the following NMEA messages in **Table 6-2**:

Message	Content
APB	Revised autopilot message contains all of the above plus: heading to steer toward destination, bearing from the present position to the destination
GGA	GPS position, time, fix quality, number of satellites used, HDOP (Horizontal Dilution of Precision), differential reference information, and age.
GLL	GPS-derived latitude, longitude, and time of fix.
GSA	GPS receiver operating mode, satellites used in the navigation solution reported by the \$-GGA sentence and DOP (Dilution of Precision) values.
GSV	Number of satellites in view, satellite numbers, elevation, azimuth, and SNR value.
RMB	Data status, cross track error, direction to steer, origin, destination landmark, landmark location, bearing to destination, and velocity toward the destination.
RMC	Time, latitude, longitude, speed, heading, and date.

Table 6-2: Magellan GPS 310 NMEA messages

It should be noted that if the GPS cannot track at least 3 satellites, it will only output the \$GPGSV message. If it can track at least 3 satellites, it will output all the messages listed.

6.3 Software Implementation - iUMLGPS

Since the GPS constantly streams out NMEA messages, the job of the software is rather simple. It simply reads the PC's serial port and looks for the particular NMEA message that carries the desired data. Once it finds that NMEA message, it extracts the desired information from the message and records it.

The information we would like from the GPS is the following: 1) Latitude 2) Longitude 3) Number of satellites GPS is in contact with. The NMEA message that provides all of this information is \$GPGGA. The \$GPGGA message is split into 14 comma-delimited pieces of data as follows in **Table 6-3**.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
\$GPGGA	hhmmss.ss	1111.11	a	yyyyy.yy	a	x	xx	x.x	x.x	M	x.x	M	x.x	xxxx*hh

Table 6-3: NMEA 0183 \$GPGGA format

The meanings of each of the pieces of data are given in **Table 6-4** below.

ID	Meaning
1	UTC of Position; hh = hours, mm = minutes, ss = seconds
2	Latitude
3	N/S
4	Longitude
5	E/W
6	GPS Quality Indicator 0 = fix not available or invalid 1 = GPS SPS Mode, Fix valid 2 = Differential GPS, SPS Mode, fix valid 3 = GPS PPS Mode, fix valid
7	Number of satellites in use (00-12, may be different from the number in view)
8	Horizontal dilution of precision
9	Antenna altitude above/below mean sea level
10	Units of antenna altitude, meters
11	Geoidal separation - difference between the WGS-84 earth ellipsoid and mean sea level (geoid), "-" = mean sea level below ellipsoid
12	Units of geoidal separation, meters.
13	Age of Differential GPS data - Time in seconds since last SC104 Type 1 or 9 update, null field when DGPS is not used
14	Differential reference station ID, 0000-1023

Table 6-4: Meaning of \$GPGGA data

The only pieces of data that are of use in the \$GPGGA message are 2, 3, 4, 5, and 7.

These pieces are extracted and the rest are thrown away.

The convention used in this network to express latitude is that degrees north are expressed as positive and degrees south are expressed as negative. Similarly, in expressing longitude, degrees east are expressed as positive and degrees west are expressed as negative. Thus, pieces 2 and 3 are combined and published as the variable BUOY1_GPS_LAT in iUMLGPS. Pieces 4 and 5 are combined and published as BUOY1_GPS_LONG. The number of satellites is published as BUOY1_GPS_SAT.

The source code for iUMLGPS is heavily based on previous code written for iGPS by the AUV Lab, but tweaked for testing purposes of the UML sensor network. The main changes were altered variable names, publishing of latitude and longitude information, and addition of a collection frequency parameter.

7 Hydrolab MiniSonde

The Hydrolab MiniSonde is an aquatic multiprobe that measures a variety of chemical data. The MiniSonde automatically collects and sends data to many devices including: 1) Hydrolab Surveyor – a handheld device designed to communicate with the MiniSonde 2) PC – any computer with a standard serial communication program (e.g. HyperTerminal) can communicate with the MiniSonde. The problem with these two approaches is that it is difficult to retrieve the data in real-time from these devices. Therefore, software was written to collect data from the MiniSonde that could run on the buoy computer and send collected data to the shore station in real-time.

This chapter is divided into three sections. The first section describes how the MiniSonde is connected to the PC/104 computer. The second section focuses on the communication protocol used by the MiniSonde to transmit data. The final section describes the software written to read data from the MiniSonde.

7.1 Hardware Interface

The MiniSonde is pictured below in **Figure 7-1**:



Figure 7-1: Hydrolab MiniSonde

It has a 6-pin marine bulkhead connector to connect to various devices. This connector is a custom bulkhead connector (model no. RMG-6-BCP SS) made for Hydrolab instruments by Impulse Enterprises. The contact configuration of the bulkhead connector is given below in **Figure 7-2**.

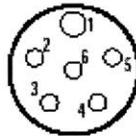


Figure 7-2: Contact configuration of MiniSonde's connector

A cable connects the Hydrolab's bulkhead to a DB-9 connector on the computer.

The contact configuration of the DB-9 connector is given below in **Figure 7-3**:

Female, 9-Pin, D-Sub
(Viewed from the front)

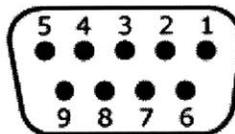


Figure 7-3: Contact configuration of DB-9 connector

A pinout of the cable connecting the MiniSonde's bulkhead to the computer's DB-9 is given below in **Table 7-1**.

MiniSonde Bulkhead	DB-9
1	4,6
2	5
3	3
4	2
5	8
6	9

Table 7-1: Pinout of MiniSonde bulkhead to DB-9

Since the MiniSonde must be deployed in the water, special cabling was made to allow it to be deployed outside of a buoy enclosure. More details about the cabling can be found in **Section 10.2** about the buoy prototype.

7.2 Communication Protocol

The MiniSonde is able to use three serial protocols to communicate with external devices: 1) Proprietary Hydrolab protocol – Hydrolab developed this protocol to communicate between its devices 2) SDI-12 – industry standard protocol that is often used to communicate with aquatic sensors 3) ANSI Terminal Emulation – standard protocol developed by the ANSI organization for remote terminal applications.

Documentation of the proprietary Hydrolab protocol could not be obtained so this option was eliminated. The SDI-12 protocol is attractive due to possible lower power consumption, but it would require additional hardware to be placed inside the buoy. The ANSI terminal emulation protocol was chosen since it is a well-documented protocol that only requires a standard DB-9 serial port. Officially, this is known as the standard “ISO/IEC 6429 – Control functions for coded character sets”, but I will refer to it as ANSI terminal emulation. Another similar protocol is VT100 terminal emulation.

7.2.1 The ANSI Terminal Emulation Protocol

The ANSI Terminal Emulation protocol allows a *host device* to control the visual state of a terminal on a *remote device*. The terminal is a window that usually consists of only text. An example application that uses the ANSI terminal emulation protocol is the serial communications program, Hyperterminal. Hyperterminal runs on the remote device, connects to the host device, and presents a terminal to the user. **Figure 7-4** below shows an example interaction between the MiniSonde and Hyperterminal. In the Hyperterminal screen shot, one can see the data that the MiniSonde is recording.

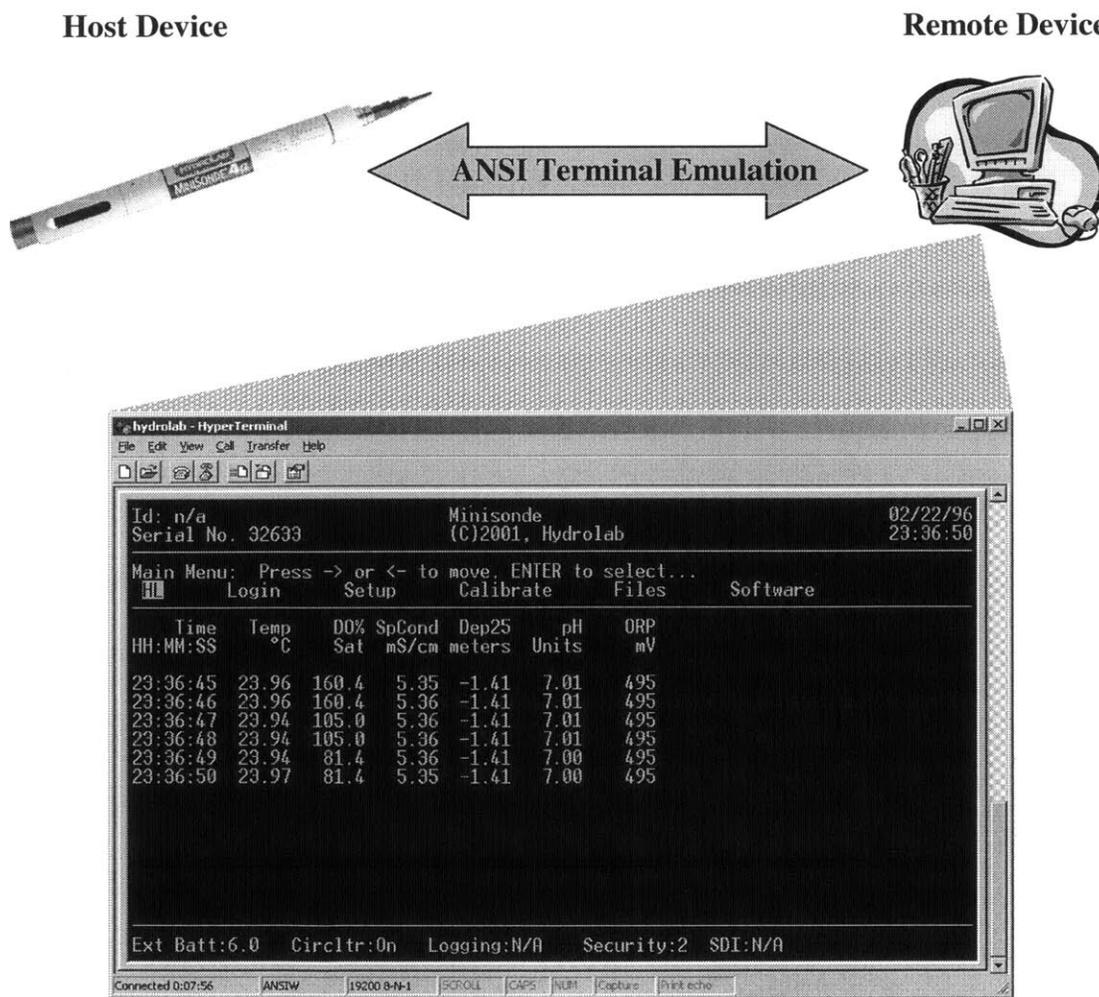


Figure 7-4: Example session between MiniSonde and Hyperterminal

The ANSI terminal emulation protocol is based on a set of commands available to the host device. These commands allow the host device to control the visual state of the screen on the remote device. Some examples of these commands are “Cursor Up” and “Cursor Down” which direct the remote device to move the cursor up or down by a specified number of lines. In addition, the host device can also query the remote device for its status. I will not be covering all the available commands and queries, but only those relevant to communicating with the MiniSonde.

7.2.2 ANSI Terminal Emulation and the MiniSonde

Communication with the MiniSonde can be broken down into two phases: 1) Initialization – MiniSonde is initialized by remote device 2) Streaming – MiniSonde streams all chemical data once per second. Once sufficient power is applied to the MiniSonde, it will enter the initialization phase. Then, once the initialization phase is completed, the MiniSonde will remain in the streaming phase as long as there is sufficient power.

7.2.2.1 Initialization Phase

Initialization consists of a query-response session between the MiniSonde and the buoy computer. This session is initiated upon power-up of the MiniSonde and will continue until the MiniSonde enters its streaming phase or is powered down. Once the session is initiated, the MiniSonde will send out an ANSI terminal query for the cursor position of the remote device. The characters for this query are:

<ESC> [6n

where <ESC> stands for the escape character (ASCII character 0x1b). After each query, the MiniSonde waits for a response. If it does not receive a response, it will send out the query again. The MiniSonde will send out a set of queries every 45 seconds. Each set of queries consists of 3 queries spaced apart by 4 seconds. For example, if you were to log the time of queries, you may see the following times (hh:mm:ss), 12:00:00, 12:00:04, 12:00:08, 12:00:53, 12:00:57, 12:01:01 and so on.

The proper response to the MiniSonde's query is to report the cursor position. The characters for this response are:

<ESC> [{ROW} ; {COLUMN} R<\r><\n>

where <ESC> is the escape character and {ROW} and {COLUMN} are the current row and column of the cursor. In addition, the characters </r> and </n> represent carriage return and newline characters respectively. Once the MiniSonde receives this response, it will enter its streaming phase. Note that the response must be sent by the remote device within a small amount of time after the MiniSonde sends out its query. Otherwise, the MiniSonde will not recognize the response as valid and will continue to send out queries.

7.2.2.2 Streaming Phase

The MiniSonde only uses the "cursor home" command once it enters streaming mode. The cursor home command sets the cursor position of the remote terminal. Any subsequent text sent by the MiniSonde will begin at that cursor position. The cursor home command syntax is the following:

<ESC> [{ROW} ; {COLUMN} H

where <ESC> is the escape character and {ROW} and {COLUMN} are the current row and column of the cursor. The MiniSonde uses the cursor home command each time a space or new line is needed in the display.

7.3 Software Implementation - iHydrolab

In order to communicate with the MiniSonde, the buoy computer acts as an ANSI terminal emulator. During the initialization phase, the buoy computer listens for a cursor position query and sends a cursor position response upon hearing the query. The MiniSonde then enters its streaming phase and the computer can then simply listen for the data. A key point to note here is that while a terminal emulation protocol is being used, we actually have no interest in displaying the data. The sole purpose here is to collect the data from the MiniSonde. Therefore, some parsing of the MiniSonde's stream of characters is needed to extract the actual data.

7.3.1 Initialization Phase

As noted before, the MiniSonde will enter the initialization phase after power-up of the MiniSonde. Our initialization protocol then is the following:

- 1) Listen for MiniSonde transmission. The transmission may be a cursor position query, chemical data, garbage, or we may not hear anything at all. If we receive any transmission we go to step 2. Otherwise, we repeat this step.
- 2) Send cursor position response. Regardless of whether the transmission is a cursor position query or chemical data, we send a cursor position response immediately after we

receive a transmission. If the transmission was a cursor position query, we obviously want to send a cursor position response. We need to send it immediately since the MiniSonde will only accept the response for a short while after it has sent the query. If the transmission was chemical data or garbage, then the sent cursor position response will be ignored so it does no harm to send it. The row and column in the response are set to 1, but their value is inconsequential since we are not actually displaying any data.

3) Determine if MiniSonde has been initialized. In step 2, we did not bother to check what the MiniSonde had sent in order to save time. In this step, we actually try to determine if the MiniSonde has been initialized based on the transmission we read in step 1. Recall that the MiniSonde sends cursor home commands in its streaming phase. Therefore, if the transmission contains cursor home commands, then it is initialized and we quit this protocol. Otherwise, the MiniSonde is not initialized and we return to step 1.

A time sequence diagram of an example initialization session is given below in **Figure 7-**

5. The protocol's steps are indicated on the right.

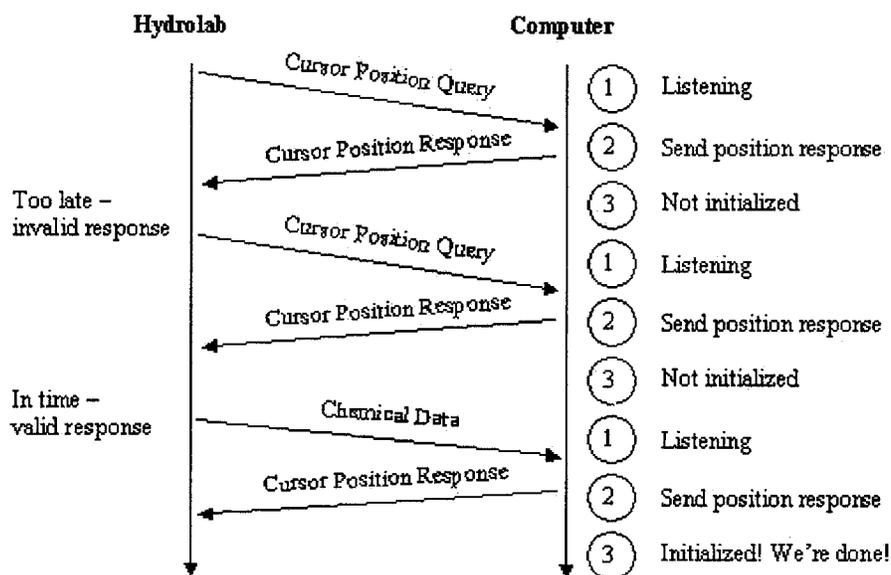


Figure 7-5: Example MiniSonde initialization session

7.3.2 Streaming Phase

In the streaming phase, the buoy computer's job is rather easy. The MiniSonde is streaming out a mix of ANSI terminal emulation commands and scientific data. The buoy computer simply has to read the characters off of its serial port and extract the data from that mix of characters.

A typical line of raw characters from the MiniSonde looks like the following:

```
[2;73H16:15:42<ESC>[24;1HExt Batt:14.1<ESC>[22;1H16:15:42<ESC> [22;10H  
23.24 <ESC>[22;17H 2.6 <ESC>[22;24H 1.099 <ESC>[22;31H -0.98 <ESC>  
[22;38H 8.68 <ESC>[22;45H 271 <ESC>[22;52H
```

Figure 7-6: Typical raw characters from MiniSonde

The gray areas represent the actual data in this string. The other characters are ANSI control commands. In order to extract the data, the following three steps are done:

- 1) Replace ANSI control commands with spaces. Only the cursor home command is used so we simply search for a pattern that matches that command. This leaves us with only the data separated by large chunks of space characters.
- 2) Check that the data is valid. We simply want to check that we have a complete line of data and are not reading from the middle of a line. Since the first four words of a line are always similar, we can use pattern matching and string comparison to ensure that the first four words of the inputted line are as expected.

3) *Read the data.* Now, each piece of data is separated by a chunk of space characters.

In addition, we know the order of data in which they appear. Therefore, we simply read each token¹ of data using a space delimiter and record the data.

There are 10 tokens of data in a typical line of characters from the MiniSonde. **Table 7-2** below shows the tokens in the order they appear and the corresponding MOOS variable the data is published under.

Token	MOOS Variable
Time	-----
“Ext” – abbreviation for external	-----
External battery voltage	BUOY1_HYDROLAB_EXTBATT
Time	-----
Temperature	BUOY1_HYDROLAB_TEMP
Specific conductance	BUOY1_HYDROLAB_SPCOND
Dissolved oxygen %	BUOY1_HYDROLAB_DO%
Depth	BUOY1_HYDROLAB_DEP25
pH	BUOY1_HYDROLAB_PH
ORP(mV)	BUOY1_HYDROLAB_ORP

Table 7-2: MiniSonde tokens and corresponding MOOS variables

¹ A token is a set of characters followed by a chosen character known as a *delimiter*.

8 Thermistor Chain

The thermistor chain is a 46-foot cable with 6 evenly spaced thermistors attached to it. Each thermistor outputs a resistance relative to the temperature of its immediate environment. In normal operation, the thermistor chain is deployed in water and connected to a metal canister through a bulkhead connector. The canister contains the proper electronics to read and log data from the thermistors. The data can then be later retrieved from the canister for analysis.

In this application, however, immediate retrieval and analysis of thermistor data is required. This requires the buoy computer to collect data directly from the thermistor chain where it can be logged and forwarded to the shore station. A small circuit board was built to interface between the thermistor chain and the PC/104 computer. The circuit board has two purposes: 1) It multiplexes the thermistor chain 2) It converts the thermistor's analog signal to a digital signal.

The basic set of steps that occur when the computer wants the temperature of a particular thermistor is:

- 1) Computer sends the thermistor's number to the circuit board
- 2) Circuit board reads the analog signal of that thermistor
- 3) Circuit board converts the analog signal to a digital output
- 4) Circuit board sends the digital output to the computer

These 4 steps are illustrated below in **Figure 8-1**.

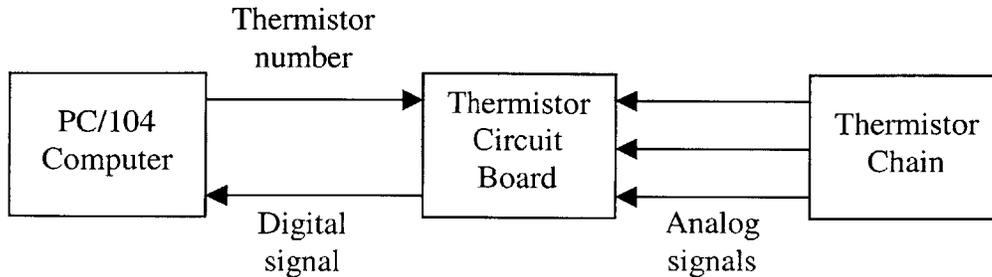


Figure 8-1: Overview of computer, circuit, and chain interfaces

Conventions

Before I go any farther, I'm going to cover some conventions and terminology about hardware interfaces that I will be using in this chapter. There are a lot of interfaces to cover so these will ease the reading and writing of them.

- For those of you who don't know, a *male* connector has pins and a *female* connector has sockets. When two connectors are connected, they are often referred to as *mated*.
- Similarly, I will use the term *plug* to refer to a male version of a connector and the term *receptacle* to refer to the female version of a connector
- A *contact configuration* shows the physical configuration of the pins/sockets on a connector and assigns a number to each pin/socket.
- A *pinout* maps the pins on one connector of a cable to the other connector of the cable

- I will only give pinouts for cables that have two different connectors. Otherwise, one can assume that the cable has a “normal” pinout (e.g. pin 1 goes to pin 1, pin 2 goes to pin 2).
- One can assume that when two connectors mate, pin 1 of the male connector goes to socket 1 of the female connector, pin 2 of the male connector goes to socket 2 of the female connector, etc.

The rest of this chapter is as follows. Section 8.1 covers all hardware components and interfaces related to the thermistor chain. Section 8.2 describes the computer’s software used to read temperatures off of the thermistor chain.

8.1 Thermistor Chain Hardware

As shown in **Figure 8-1** previously, the thermistor hardware consists of three main components: the circuit board, thermistor chain, and PC/104 computer. In addition, there are two key interfaces between the chain and circuit board and the computer and circuit board. The goal of this section is to detail the physical layout and connections of these components and interfaces. I will start with the circuit board since it is the center of the system. Then, I’ll move onto the interfaces connecting to the circuit board.

8.1.1 Thermistor Circuit Board

The thermistor circuit board has two major components: a multiplexer² and a thermistor converter³. As one would guess, they allow the circuit board to multiplex the thermistor

² Maxim 4638

³ Maxim 6682

chain and convert the chain's analog signals to digital. In addition, the circuit board has two Ethernet receptacles to connect to the thermistor chain and computer.

8.1.1.1 Front side

Connections between components are made on both sides of the board. The layout of one side of the board is given below in **Figure 8-2**.

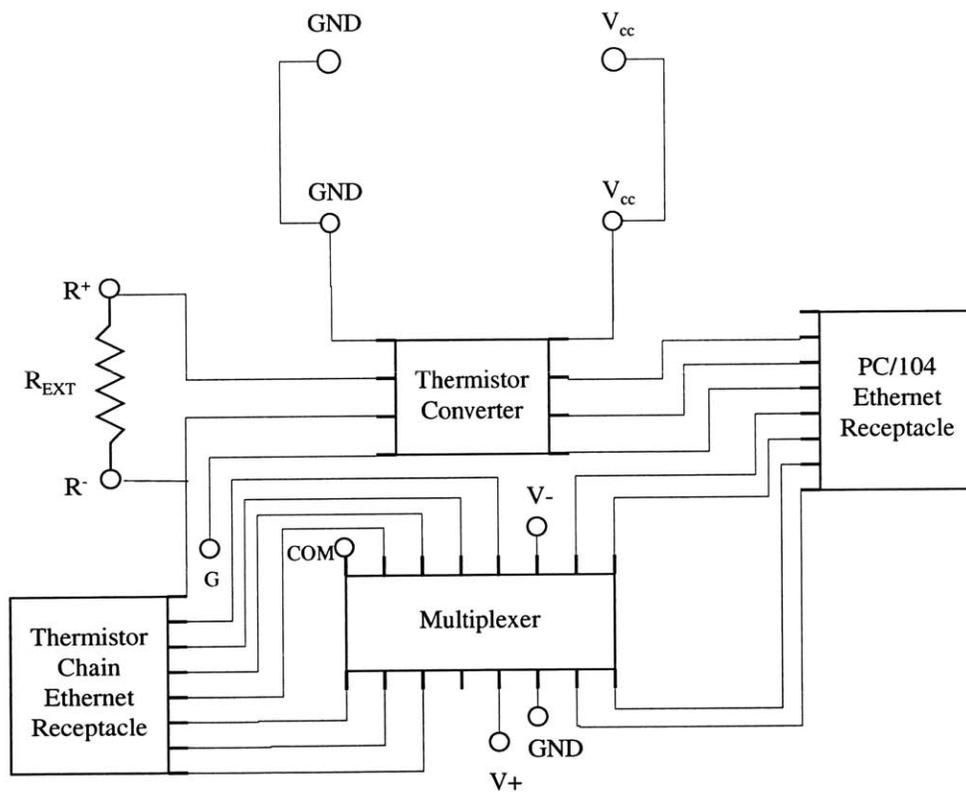


Figure 8-2: Front side of thermistor circuit board

The circles in the diagram represent holes/terminals in the board. The terminal marked “G” on the left hand side of the board connects to the ground pin of the thermistor converter. The terminal marked “COM” is the common output of the multiplexer.

Power is supplied to the thermistor converter and multiplexer from the voltage terminals shown at the top of the board. A single source 3.3V supply is used. Calculations showing the amount of voltage supply needed are given in **Appendix B**. An external resistor of 14.7 kilohms is placed across the R+ and R- terminals to calibrate input read from the thermistor chain.

The thermistor chain is interfaced with an Ethernet plug and connects to the thermistor chain Ethernet receptacle. The resistances of the thermistors are then inputted into the multiplexer. This interface is covered in more detail in **Section 8.1.2**. Similarly, the PC/104 computer is interfaced with an Ethernet plug and connects to the PC/104 Ethernet receptacle. This interface is covered in more detail in **Section 8.1.3**.

8.1.1.2 Back side of circuit board

The above figure only shows the connections on one side of the board. Additional connections between the labeled terminals are made on the opposite side of the board and are shown below in **Figure 8-3**.

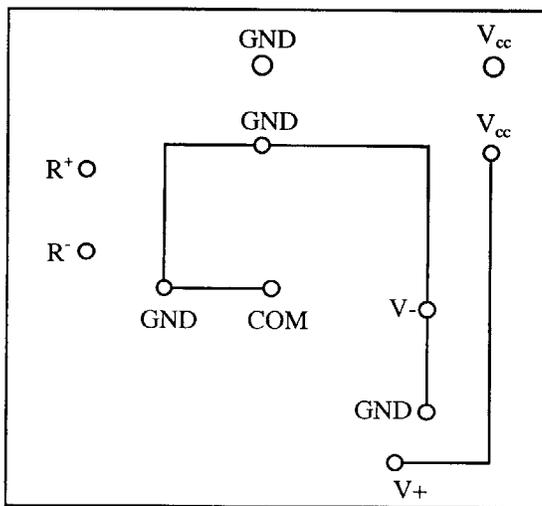


Figure 8-3: Back side of thermistor circuit board

It is unusual that the ground connections of the circuit are connected to the “COM” pin of the multiplexer. The “COM” pin is the common output of the multiplexer. This is an acceptable connection since the multiplexer is only switching thermistor inputs. Therefore, only resistances (i.e. no voltage) are output from the “COM” pin. These resistances do not interfere with the ground references.

8.1.1.3 Multiplexer Pin Configuration and Description

The pin configuration of the multiplexer is given below:

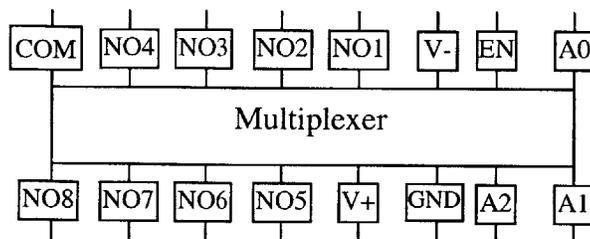


Figure 8-4: Pin configuration of Maxim 4638

The pin description is:

Name	Function
COM	Common analog output
NO1 – NO8	8 analog inputs
V+	Positive-supply voltage input
V-	Negative-supply voltage input (Connected to GND)
GND	Ground
EN	Enables the multiplexer
A0 – A2	Address inputs

Table 8-1: Pin description of Maxim 4638

A short theory of operation: The multiplexer is enabled on its EN pin. It reads the input address on pins A0 – A2. It then switches one of the 8 inputs NO1 – NO8 to its COM pin.

8.1.1.4 Thermistor Converter Pin Configuration and Description

The pin configuration of the thermistor converter is:

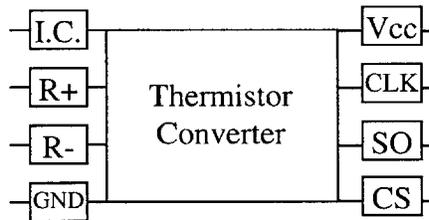


Figure 8-5: Pin configuration of Maxim 6682

The pin description is:

Abbreviation	Function
I.C.	Internally Connected. Connect to GND or leave unconnected.
R+	Reference voltage output. External resistor positive input.
R-	External resistor negative input.
GND	Ground connection for 6682 and ground return for external thermistor.
Vcc	Positive supply.
CLK	Input clock to the converter. Used in serial data exchange to drive output.
SO	Serial data output. Carries thermistor data.
CS	Chip select. Enables the serial interface.

Table 8-2: Pin description for Maxim 6682

A short theory of operation: The converter is enabled through its CS pin. It reads the voltage across its R+ and R- terminals and converts this voltage to a temperature. Then,

once it begins receiving clock pulses on CLK, it will output the temperature in a serial format on SO.

8.1.1.5 Circuit Theory of Operation

So how is this all going to work? How are we going to convert thermistor resistances to a digital output for the computer? Let's start by discussing how the thermistor converter works. The thermistor converter reads the temperature of a thermistor by measuring the voltage across the resistor R_{EXT} . Over small ranges of temperature, the voltage across R_{EXT} is approximately proportional to the temperature of the thermistor. So the question is how and why these two values are related. Let's examine the circuit again to see this relation. **Figure 8-6** below shows a portion of the circuit board with the thermistor chain connected to it. Let's assume we want to measure thermistor 6.

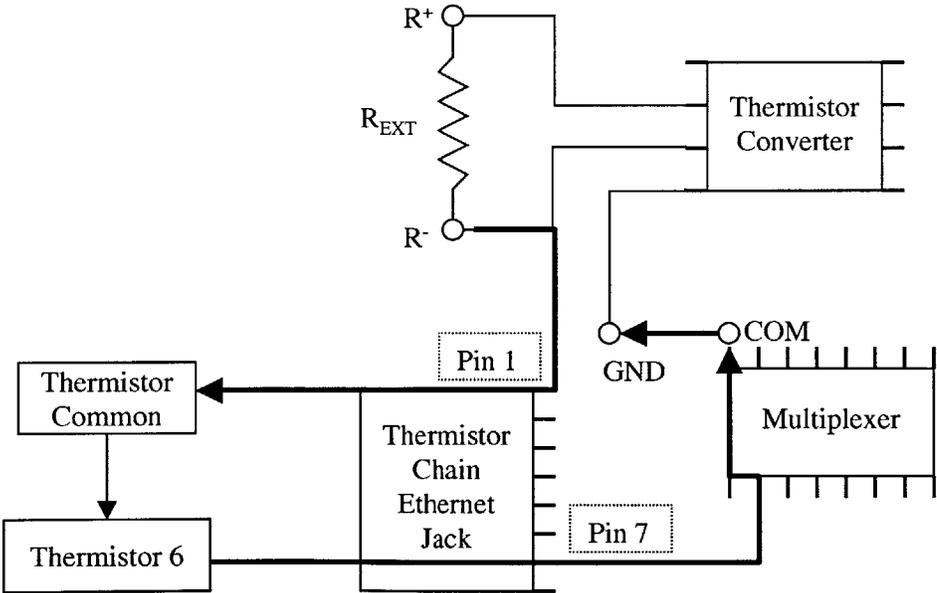


Figure 8-6: Path from R- to GND

The bolded arrows in the above figure represent a connection that runs all the way from the R- pin of the thermistor converter to the GND pin of the converter. As shown above, pin 1 and pin 7 of the Ethernet receptacle connect to the thermistor chain's common line and thermistor 6 respectively⁴. The resistance of thermistor 6 is input to the multiplexer which switches that resistance to its common output. Recall that the multiplexer common's output is connected to the converter's ground pin which completes the connection.

Since a thermistor acts as a resistor, the path shown in **Figure 8-6** can be reduced to the following circuit below in **Figure 8-7**.

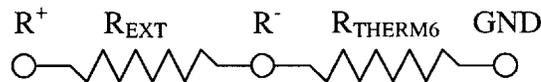


Figure 8-7: Reduced path from R+ to GND

R_{THERM6} represents the resistance of thermistor 6. This circuit of course is just two resistors in series and the voltage across R_{EXT} can be determined from a voltage divider formula as shown below in **Figure 8-8**.

$V_{EXT} = \frac{R_{EXT} * V_{R+}}{R_{EXT} + R_{THERM6}}$	where	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="padding: 5px;">V_{EXT} = Voltage across R_{EXT}</td> </tr> <tr> <td style="padding: 5px;">V_{R+} = Voltage from R_+ to GND</td> </tr> </table>	V_{EXT} = Voltage across R_{EXT}	V_{R+} = Voltage from R_+ to GND
V_{EXT} = Voltage across R_{EXT}				
V_{R+} = Voltage from R_+ to GND				

Figure 8-8: Voltage divider formula

Therefore, as the temperature of thermistor 6 changes, R_{THERM6} changes, and then V_{EXT} changes in turn. In order to measure other thermistors, the multiplexer switches their

⁴ More details are given in **Section 8.1.2**, but you'll just have to take my word for now.

input resistance to its common output. The same principles apply in measuring their temperature.

8.1.2 Thermistor Chain to Circuit Board Interface

The thermistor chain consists of 6 thermistors and one female 7-pin cable connector (XSJ-7-CCR). The chain connects to a cable with a male 7-pin bulkhead connector (XSJ-7-BCP) on one end and an Ethernet plug on the other side. Then, the Ethernet plug connects to an Ethernet receptacle on the circuit board. **Figure 8-9** below depicts these connections.

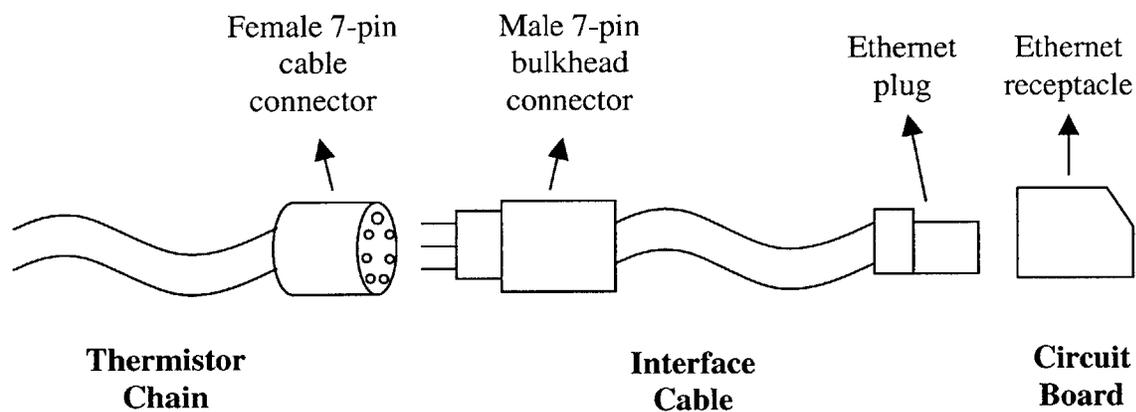


Figure 8-9: Thermistor chain to circuit board overview

I'll cover this interface in two sections. First, I'll show where and how signals from thermistors on the thermistor chain travel to the Ethernet jack on the circuit board. Then, I'll show where those signals on the Ethernet jack travel to components on the circuit board.

8.1.2.1 From Thermistor Chain to Ethernet Jack

There are two connections between the thermistors and the Ethernet jack, so there are two pinouts that need to be specified. The following diagram in **Figure 8-10** depicts the two pinouts: one from the thermistors to the cable connector pins and one from the bulkhead connector pins to the Ethernet plug pins.

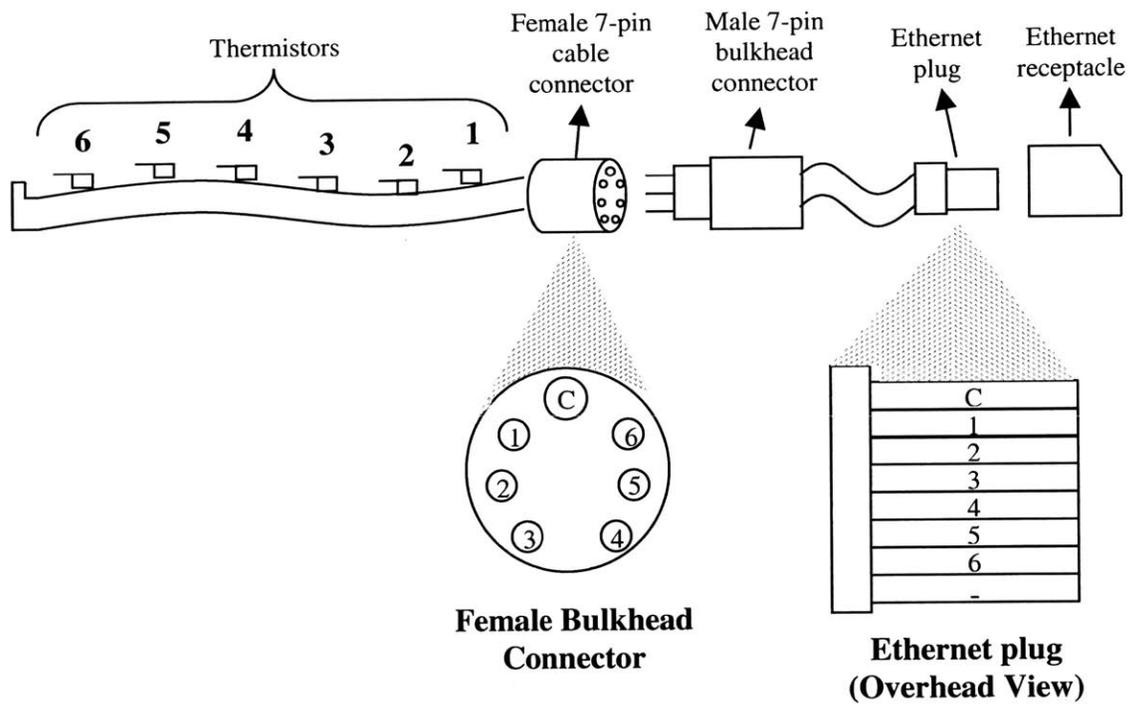


Figure 8-10: Mapping of thermistors to bulkhead connector pins and then to Ethernet pins

Let's start at the thermistors. As noted before, the thermistor chain has 6 thermistors, each of which has been assigned a number in **Figure 8-10**. Each of these thermistors is connected to a socket in the thermistor chain's cable connector. The resistance of a

thermistor can be read at its corresponding socket. The cable connector on the thermistor chain has 7 sockets: 1 for each of the 6 thermistors and 1 for a common reference.

The enlarged illustration of the cable connector shows the contact configuration of the common reference and the 6 thermistors. One can tell which pin in the bulkhead connector is the common reference pin by its slightly larger size relative to the other pins.

The cable connector on the thermistor chain then mates with the bulkhead connector on the interface cable. The bulkhead connector then connects to an Ethernet plug. Note that the bulkhead connector has 7 pins whereas the Ethernet plug has 8 pins, so one pin on the Ethernet plug is unused. The enlarged Ethernet plug shows where the cable connector's 7 pins correspond to on the plug. Finally, the Ethernet plug connects to the Ethernet receptacle.

8.1.2.2 Thermistor Chain Ethernet Jack to Circuit Components

The last area to cover in the thermistor chain to circuit interface is where each Ethernet pin connects to in the circuit board.

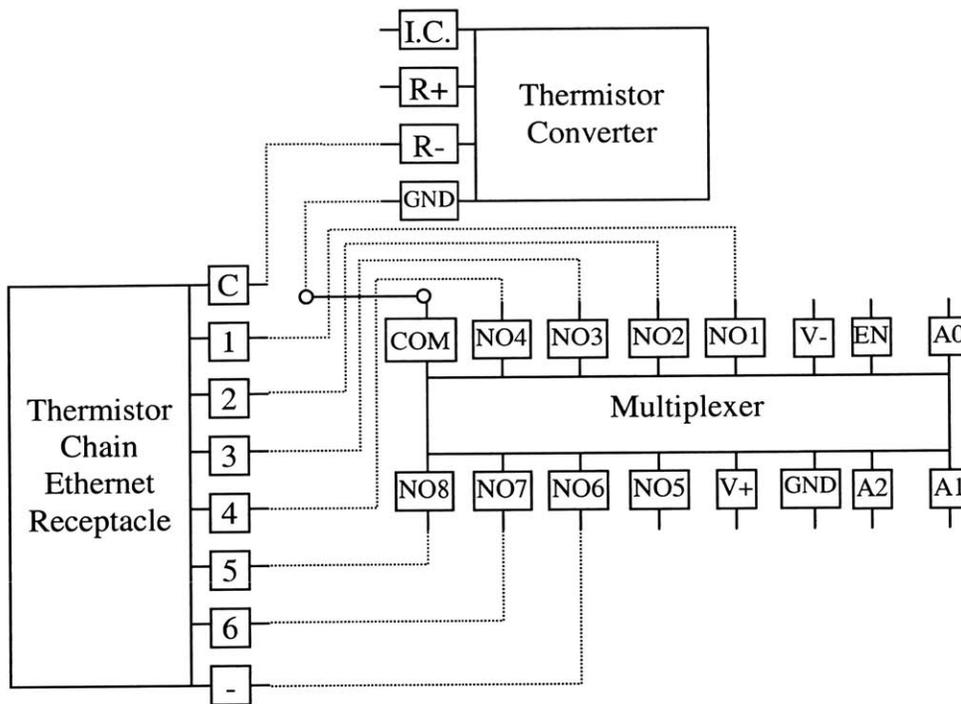


Figure 8-11: Thermistor chain Ethernet jack to circuit board interface

The thermistor common reference pin connects to the R- pin on the thermistor converter. The next 6 Ethernet pins connect to input pins on the multiplexer. It is slightly confusing that thermistor 5 corresponds to the 8th input on the multiplexer, but this association made the wiring layout of the board simpler. The same reason applies for thermistor 6 connecting to the 7th input. The lowest pin on the Ethernet jack marked “-“ connects to the 6th input on the multiplexer, but it is not attached to any thermistor.

8.1.3 Computer to Circuit Interface

While any computer could connect to the circuit board, the material in this section will reference a TS-5500 computer. The TS-5500 is the particular model of PC/104 computer that we have chosen to use in the buoy. More information on the TS-5500 is given in **Chapter 9**.

The TS-5500 uses a digital input/output interface to communicate with the circuit board. The digital I/O interface on the TS-5500 consists of 16 pins, each of which carries a low or high voltage. In order to allow the TS-5500 to communicate directly with the circuit board, a ribbon cable soldered onto an Ethernet plug is used to connect the digital I/O interface to the circuit board's Ethernet jack. An overhead view of the connection from the TS-5500 to the circuit board is given below in **Figure 8-12**.

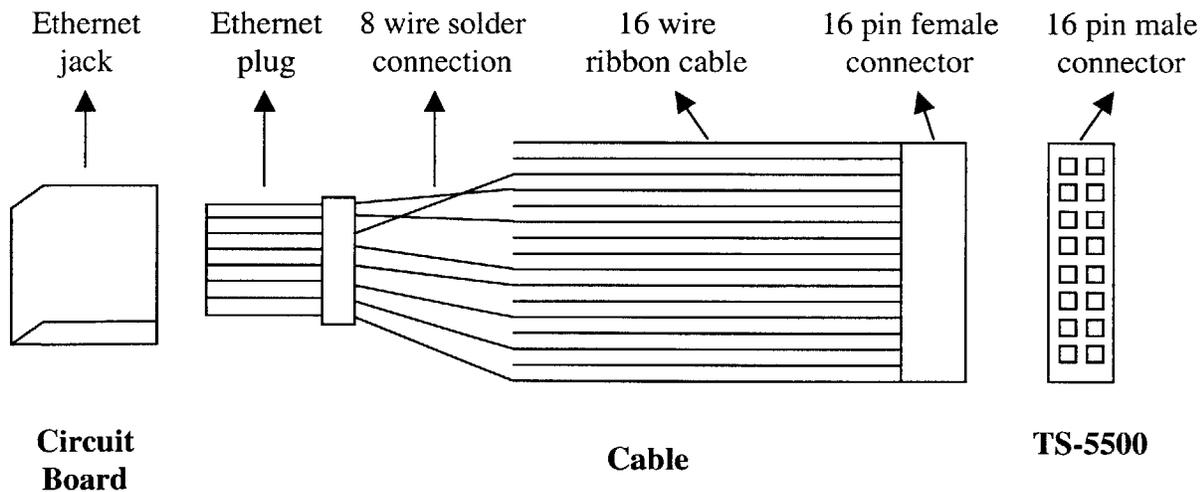


Figure 8-12: Computer to circuit board connection

I will cover this interface in two sections. The first section details the connection from the digital I/O interface of the computer to the Ethernet jack of the circuit board. The

second section shows the connections from the Ethernet jack to the components on the circuit board.

8.1.3.1 Digital I/O to Ethernet Jack

I'll cover the pin numberings of the two ends of this connection first and then conclude with the cable that connects the two sides.

Digital I/O Interface

There are two 16 pin digital input/output interfaces on the TS-5500 (named DIO1 and DIO2). We have arbitrarily chosen to use DIO1.

DIO1 has 16 pins: 14 pins for input or output, 1 pin for ground, and 1 pin for a 5V source. The pins are arranged in 2 rows of 8 (referred to as 8x2) and are numbered as follows in **Figure 8-13**:

5V	7
13	6
12	5
11	4
10	3
9	2
8	1
GND	0



Figure 8-13: Digital I/O pin numbering on TS-5500

The numbers indicate the pin number at that location. The circle above reflects the circle shown on the TS-5500 board and indicates the orientation of the above figure.

Ethernet Plug

The other side of the connection is an Ethernet plug. **Figure 8-14** below shows an overhead view of the Ethernet plug and the numbering of the Ethernet pins.

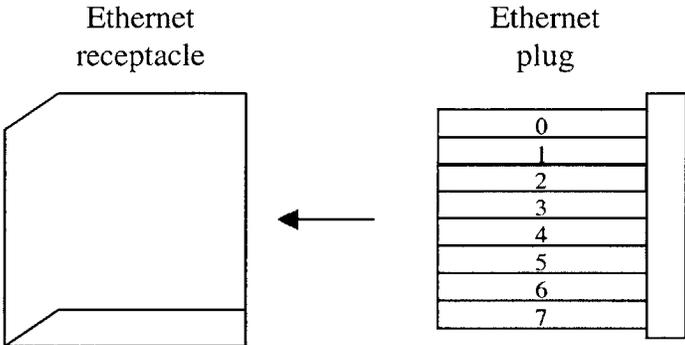


Figure 8-14: PC/104 Ethernet Plug

Now, to make the connection between the two connectors.

Digital I/O to Ethernet Cable

Since no existing cable could connect the digital I/O interface and Ethernet jack, a custom cable was made by soldering a 16-pin ribbon cable to an Ethernet plug. The ribbon cable has an 8x2 connector on each side and 16 wires between the connectors. One 8x2 connector is used to connect to the digital I/O interface. The other 8x2 connector was cut off, leaving us with 16 free wires on that end. The pinout from the 8x2 connector to the 16 wires is shown below in **Figure 8-15**:

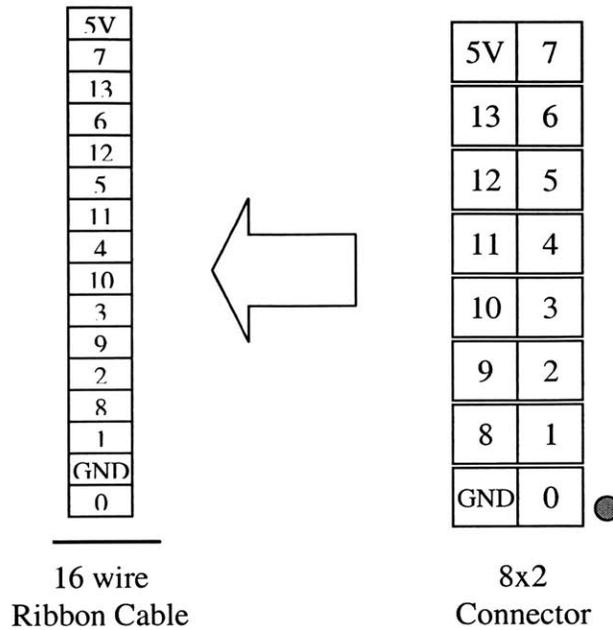


Figure 8-15: Pinout from digital I/O to ribbon cable

Note that the 8x2 connector has the same contact configuration as the digital I/O interface from **Figure 8-13**. The line above the ribbon cable specifies the orientation of the cable. It represents a gray stripe that can be found on the cable. The wires of the cable are labeled with the pin they connect to. The large arrow in the middle represents the 16 connections between the like-named components of the 8x2 connector and 16 wires.

Eight of these free wires were soldered to the Ethernet plug according to the pinout given below in **Table 8-3**:

Cable	Ethernet
GND	GND of Circuit
0	0
1	1
2	2
3	3
4	-
5	6
6	7
7	-
8	-
9	-
10	4
11	-
12	-
13	5
5V	-

Table 8-3: Digital I/O to Ethernet Pinout

Note that the GND pin of the cable is connected to the GND pin of the circuit (not to the Ethernet plug). This is to ensure that the computer and circuit board are referenced against the same ground. Refer to **Figure 8-2** in **Section 8.1** about the location of the circuit's GND pin.

Now, putting it all together, **Figure 8-16** shows the entire connection from the computer to the Ethernet jack.

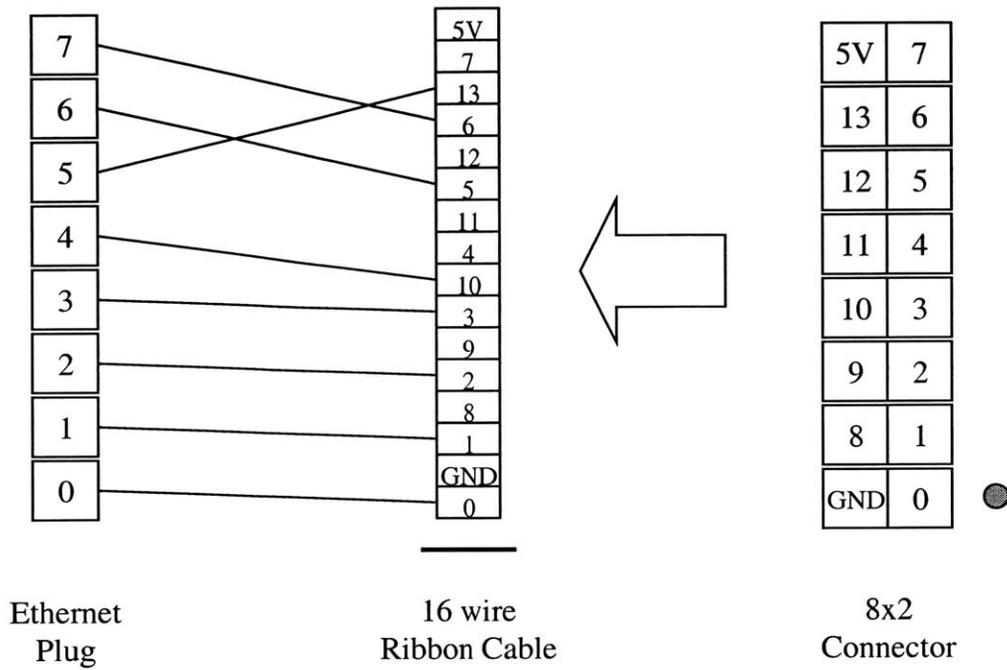


Figure 8-16: Cable connecting computer to circuit board

8.1.3.2 PC/104 Ethernet Jack to Circuit Components

The PC/104 Ethernet jack has 7 pins connecting to the thermistor-to-digital converter and also the multiplexer (shown below in **Figure 8-17**). Each of the pins on the thermistor converter and multiplexer have been labeled with their names. The pin numbers on the Ethernet jack are the same as described for the Ethernet plug in **Section 8.1.3.1**.

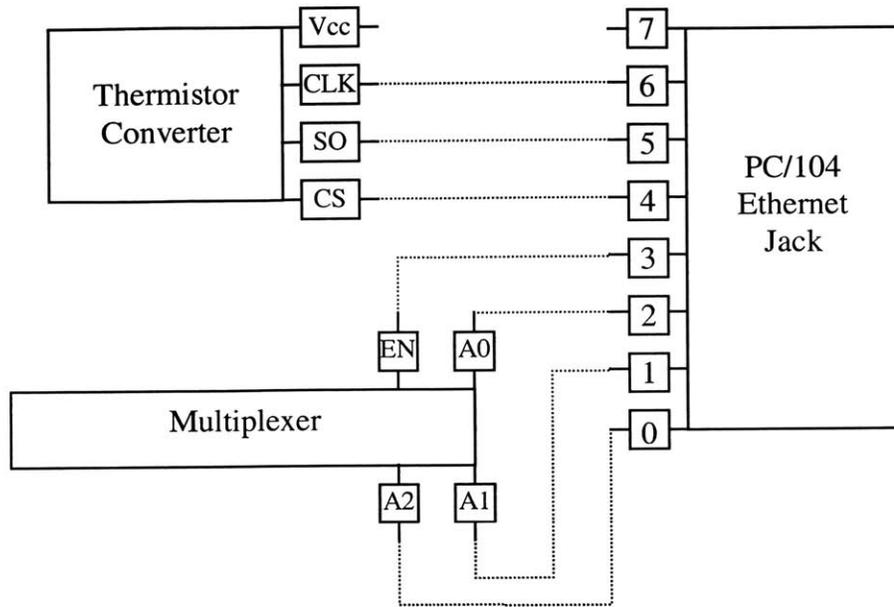


Figure 8-17: PC/104 Ethernet jack to circuit board interface

8.1.3.3 Digital I/O to Circuit Component Pinout

For convenience, I'll summarize the pinout from the digital I/O interface to the Ethernet connection to the circuit components. This will be useful in talking about the software used to drive the circuit. This pinout is shown below in **Table 8-4**.

Digital I/O	Ethernet	Circuit	Input / Output (with respect to the PC/104)
0	0	A2	output
1	1	A1	output
2	2	A0	output
3	3	EN	output
10	4	CS	output
13	5	SO	input
5	6	CLK	output
6	7	-	output

Table 8-4: Pinout from Digital I/O to Ethernet jack

The only restriction on this pinout is due to the grouping of digital I/O pins. The 14 digital I/O pins are separated into 4 groups and each group must have the same direction (input or output). Therefore, since the SO pin is the only input to the PC/104, it must be in a separate group from the other pins. The 4 groups are pins 0-3, 4-7, 8-11, and 12-13. Other than this restriction, this pinout is largely arbitrary.

8.2 Thermistor Chain Software

Now that we've covered the hardware interface, we can discuss how the computer uses this interface to read thermistor temperatures from the circuit. In essence, by raising or lowering certain digital lines at specified times, the computer can control the behavior of the multiplexer and thermistor converter. The protocol used by the computer can be broken down into two steps: 1) Computer reads an 11-bit serial output from the thermistor converter corresponding to a thermistor temperature 2) Converts that serial output to a temperature according to known conventions by the thermistor converter.

8.2.1 Circuit Pins

First, we'll cover the purpose of each of the pins that the computer uses. **Table 8-4** shows the function of the 4 multiplexer pins that are of interest to the computer:

Name	Function
EN	Enables the multiplexer
A0	1 st bit of multiplexer address
A1	2 nd bit of multiplexer address
A2	3 rd bit of multiplexer address

Table 8-4: Multiplexer pins used by computer

Table 8-5 shows the three thermistor converter pins that are of interest to the computer:

Name	Function
CLK	Input clock to the converter. Used in serial data exchange to drive output.
SO	Serial data output. Carries thermistor data.
CS	Chip select. Enables the serial interface.

Table 8-5: Thermistor converter pins used by the computer

The other pins in the circuit are used by the thermistor chain.

8.2.2 Timing Diagram

Using the 7 pins above, the TS-5500 can drive the circuit to output a thermistor's temperature onto the SO pin and read that data. To do so, the computer must output a high or low to a particular pin at a particular time. The timing diagram in **Figure 8-18** below shows the output the computer must generate:

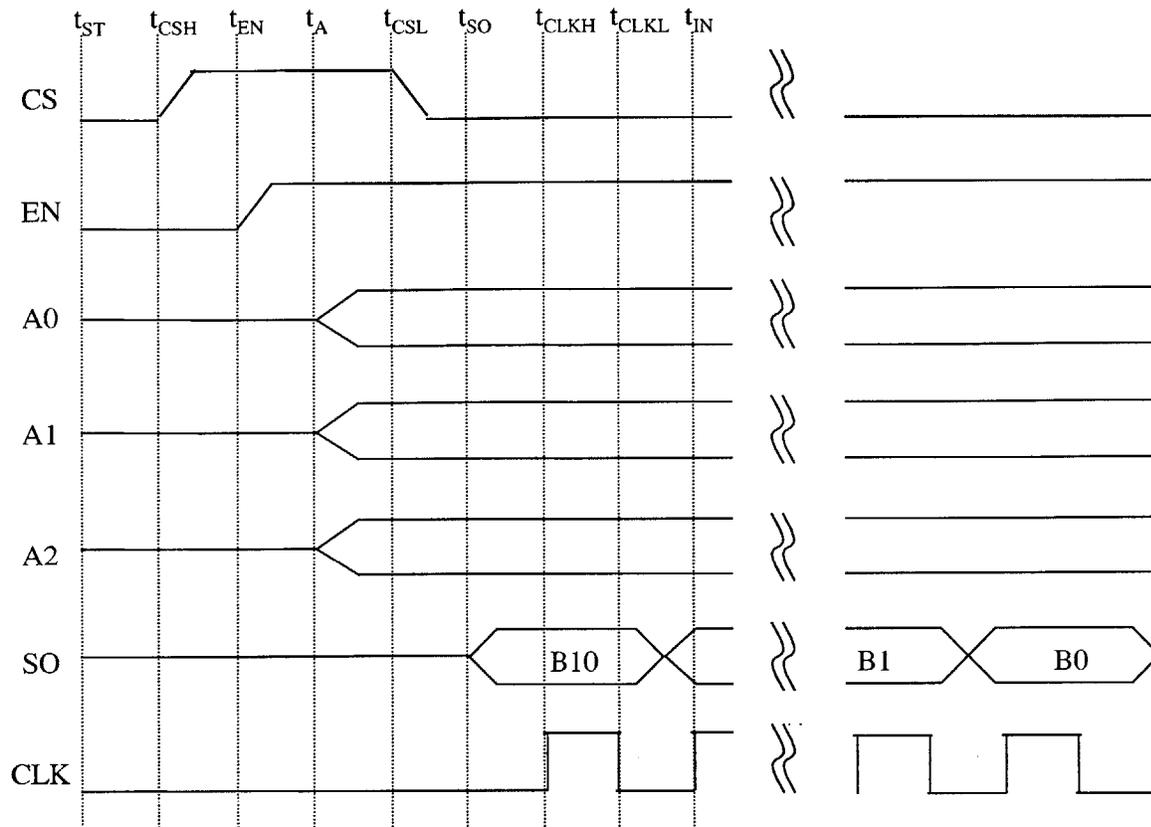


Figure 8-18: Timing diagram of digital I/O lines

Note that A0, A1, and A2 go in two directions at time t_A to indicate that A0, A1, and A2 can go up or down at that time depending on the inputted address. Similarly, SO can go up or down depending on the output of the thermistor converter.

The above timing diagram can be broken down into the 10 steps below in **Table 8-6**. The dotted lines on the diagram indicate the timing of these 10 steps:

Step	Time	Event
1	t_{ST}	Start of sequence. All lines start at 0.
2	t_{CSH}	Set CS high. Enable converter. Prevents data from appearing on SO.
3	t_{EN}	Set EN high. Enable the multiplexer.
4	t_A	Set A0, A1, A2 to desired address. Switched input appears on COM.
5	t_{CSL}	Set CS low. A/D begins. Causes first serial data bit to appear on SO.
6	t_{SO}	Thermistor converter puts first serial data bit on SO.
7	t_{CLKH}	Read in first serial data bit off SO. Set CLK high.
8	t_{CLKL}	Set CLK low. Causes next data bit to appear on SO.
9	t_{IN}	Read in next data bit off SO. Set CLK high.
10	$> t_{IN}$	Repeat steps 8 and 9 until all 11 bits are read.

Table 8-6: 10 steps in timing diagram

As shown in the timing diagram, after the TS-5500 instructs a particular line to be set high or low, it takes time for the change to occur and propagate through the circuit. The amount of time needed between each step is given below in **Table 8-7**. In addition, the amount of time actually taken between each step by the software is given as well. Note that the time given for a particular step is the amount of time that must pass between the that step and the next step. A propagation delay of 0 indicates that no time is needed to allow for propagation.

Step	Time	Propagation Delay	Actual Delay
1	$t_{ST,PD}$	0 ns	1 ms
2	$t_{CSH,PD}$	0 ns	1 ms
3	$t_{EN,PD}$	0 ns	1 ms
4	$t_{A,PD}$	80 ms	80 ms
5	$t_{CSL,PD}$	35 ns	1 ms
6	$t_{SO,PD}$	0 ns	1 ms
7	$t_{CLKH,PD}$	50 ns	1 ms
8	$t_{CLKL,PD}$	50 ns	1 ms
9	$t_{IN,PD}$	50 ns	1 ms
	Total:	80.00185 ms	88 ms

Table 8-7: Propagation delays in thermistor circuit

The only propagation delay that is longer than 50 ns is $t_{CSL,PD}$, the time between when the multiplexer address bits are set and the chip select is set low. If you recall from **Table 8-6**, when the address bits are set, a thermistor input is switched to the multiplexer's common output. When the chip select is set low, the thermistor converter begins outputting data based on the multiplexer's outputted resistance. Before the converter begins outputting data, an A/D conversion of the outputted resistance must occur. This conversion can take up to 80 ms. If the converter outputs data before 80 ms, it's possible that the conversion is not finished and the data will be invalid.

Also note that another temperature conversion may have occurred just before the start of this sequence and that some time may be needed to allow the converter to reset. This time was not specified in the specs for the Maxim 6682 chip, but a delay of 1 ms is taken after t_{ST} to ensure that the converter is ready.

8.2.3 Converting Serial Output To Temperature

The thermistor converter outputs an 11-bit signed value corresponding to the resistance it reads from the multiplexer. The most significant bit is the sign bit while the other 10 bits represent the actual value. The converter's output can be interpreted as a number in 2's complement form except that it is shifted over 3 places. In other words, the least significant bit in the thermistor output represents the 2^{-3} place instead of the 2^0 place.

To illustrate this, let's compare how we would normally interpret an 11-bit signed value versus this "shifted" interpretation. Let's say that the value is 0x1ce. The most significant bit is on the left.

Normal Interpretation											
Serial Output	0	0	1	1	1	0	0	1	1	1	0
Powers	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Binary Sum	$2^8 + 2^7 + 2^6 + 2^3 + 2^2 + 2^1$										
Decimal Sum	$256 + 128 + 64 + 8 + 4 + 2 = 452$										

Figure 8-19: Normal 2's complement interpretation of 11-bit number

Now let's look at the shifted interpretation. Recall that $2^{-x} = 1 / 2^x$.

Shifted Interpretation											
Serial Output	0	0	1	1	1	0	0	1	1	1	0
Powers	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}
Binary Sum	$2^5 + 2^4 + 2^3 + 2^0 + 2^{-1} + 2^{-2}$										
Decimal Sum	$32 + 16 + 8 + 1 + 0.5 + 0.25 = 57.75$										

Figure 8-20: Shifted 2's complement interpretation of positive 11-bit number

In the above case, the sign bit is a 0. If the sign bit is a 1 in the shifted interpretation, we treat it exactly the same as we would in a normal interpretation. Here's an example for clarity:

Shifted Interpretation											
Serial Output	1	1	1	1	1	0	0	1	1	0	1
Powers	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}
Binary Sum	$-2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^0 + 2^{-1} + 2^{-3}$										
Decimal Sum	$-128 + 64 + 32 + 16 + 8 + 1 + 0.5 + 0.125 = -6.375$										

Figure 8-21: Shifted 2's complement interpretation of negative 11-bit number

Just as a software implementation note, a simple method to convert the thermistor converter's serial output to a decimal value is the following: Hold the 11-bit output in a

16-bit variable. Shift the variable to the left 5 bits. The sign bit of the 11-bit output is now the sign bit of the 16-bit variable. Finally, simply divide the 16-bit value by 256.

8.2.4 TS-5500 Digital I/O Access

We now know what steps need to be taken, but the question remains of how to do this with actual code. The TS-5500 computer allows access to its digital I/O lines through memory-mapped addresses. In addition, the TS-5500 provides a memory-mapped location for controlling the direction (input or output) of the digital lines. **Table 8-8** below summarizes these addresses.

Address	Function
0x7a	Controls direction of digital I/O lines
0x7b	Read from or write to digital I/O lines 0 to 7
0x7c	Read from or write to digital I/O lines 8 to 13

Table 8-8: TS-5500 memory-mapped addresses

The 14 digital lines are divided into 4 groups. All I/O lines in a group must have the same direction. Bit 0 of the byte at address 0x7a controls the direction of digital I/O lines 0 to 3. Bit 1 at 0x7a controls the direction of digital I/O lines 4 to 7. Bit 5 at 0x7a controls the direction of lines 8 to 11. Digital lines 12 and 13 are always inputs. A state of “0” at one of the aforementioned bits indicates that the lines in that group are in the input direction. A “1” indicates that the lines in that group are in the output direction.

For example, let’s look at the following scenario in **Figure 8-22** that you may want to create:

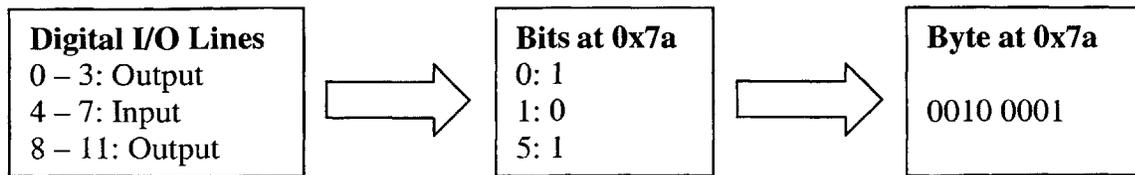


Figure 8-22: Example directions of digital I/O lines

Thus, in order to obtain the digital line state shown on the left, one would write the value shown on the right to the memory location 0x7a. The following function call in C would do the trick: `outb(0x21, 0x7a)`.

While the bits at 0x7a determine the *direction* of the digital lines, the bits at 0x7b and 0x7c determine the *values* of the digital lines. For 0x7b, bit 0 corresponds to digital line 0 and follows sequentially up to bit 7 corresponding to digital line 7. For 0x7c, only the first 6 bits are used since 0x7c only corresponds to 6 digital lines. Bit 0 of 0x7c corresponds to digital line 8 and bit 5 corresponds to digital line 13.

The best way to understand the above information is to go through an example. Let's assume that we want the following outputs in **Table 8-9** on digital lines 0 to 7:

Digital I/O Line	Output
0	0
1	1
2	1
3	0
4	0
5	0
6	1
7	0

Table 8-9: Example digital line outputs

Then, we would want to output the value 0100 0110 to 0x7b. The following function call in C would accomplish this: `outb(0x46, 0x7b)` .

We can now set the direction and values of the digital I/O lines. Using these two tools, we can implement the timing diagram given in **section 8.2.2** according to the mapping between digital I/O lines and circuit pins given in **section 8.1.3.3**. The source code for `iThermistorChain` implementing these specifications is given in **Appendix X**. The 6 thermistor temperatures are published under the variables `BUOY1_THERM_1` through `BUOY1_THERM_6`.

9 pShoreCommand

As discussed previously, the shore station is the central command station for the sensor network. A simple set of command functions is captured in the MOOSApp pShoreCommand. pShoreCommand is meant as a simple demonstration of the adaptive functionality of the network and should be extended in the future. The behaviors implemented will not necessarily be incorporated into future networks, but are designed to be triggered during testing.

pShoreCommand analyzes sensor data and sends out commands to sensors if needed. One adaptive behavior per sensor has been implemented. For the GPS, if it is detected that the GPS is tracking more than 6 satellites, the collection frequency of the GPS is increased to once per 5 seconds (the default is once per 10 seconds).

For the thermistor chain, if it is detected that the difference in adjacent thermistor temperatures has changed more than 0.5 degrees since its last record, then the collection frequency of the thermistor chain is increased to once per 5 seconds (the default is once per 10 seconds).

Finally, recall that the MiniSonde measures the voltage of its external power source. This is particularly useful since we would like to know if the batteries are low on power. If pShoreCommand detects that the batteries are supplying less than 6.3 volts, a text message is printed to the console of the shore station. In the future, this behavior could be altered to send an e-mail to a research group member (e.g. joewong@mit.edu).

10 Buoy Prototype

A low-cost prototype buoy was designed and constructed for testing of the network in a lake environment. The buoy used may not necessarily be the final implementation used, but it facilitates a realistic deployment of the sensor network. The overall design of this buoy was done by Terry Donoghue. A diagram of the buoy in water is shown below:

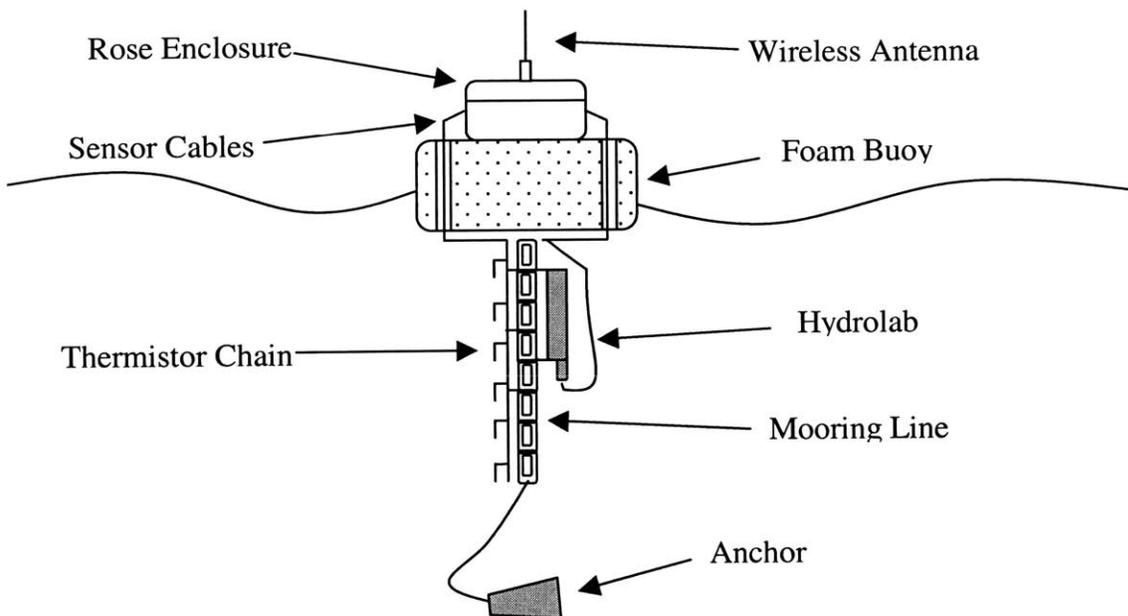


Figure 10-1: Overall buoy design

Most of the focus on implementing this system is on the enclosure at this point. The Rose enclosure is a fiberglass box that contains all the electronics of the buoy system. Penetrations are made into the Rose enclosure to allow instruments deployed in water to connect to electronics within the enclosure. I had three tasks in designing this buoy. The first task was designing how power would be supplied to the various components. The second task was specifying the penetrations needed in the Rose enclosure to allow components inside the enclosure to connect to the outside world. The final task was assisting in the physical construction of the internal enclosure packaging.

10.1 Power Supply Design

To power all the electronics, two 6V, 30 Amp-hour marine batteries were placed inside the Rose enclosure. The list of components needing power, their voltage requirements, and the current they draw is given below:

Component	Voltage Required	Current
PC/104	5V – 6V	900 mA
Hydrolab	5V – 12V	85 mA – 36 mA
GPS	3.5 – 4.0V	110 mA
Thermistor Circuit	3.3V	22 uA

Table 10-1: Buoy component power requirements

The first thing to realize is that the 6V batteries do not always provide 6 volts. They provide anywhere between 6.0 and 6.5 volts depending on how well charged they are. Therefore, we need to account for this range of voltages in designing the power supply circuitry. Below is a circuit diagram of the power supply board.

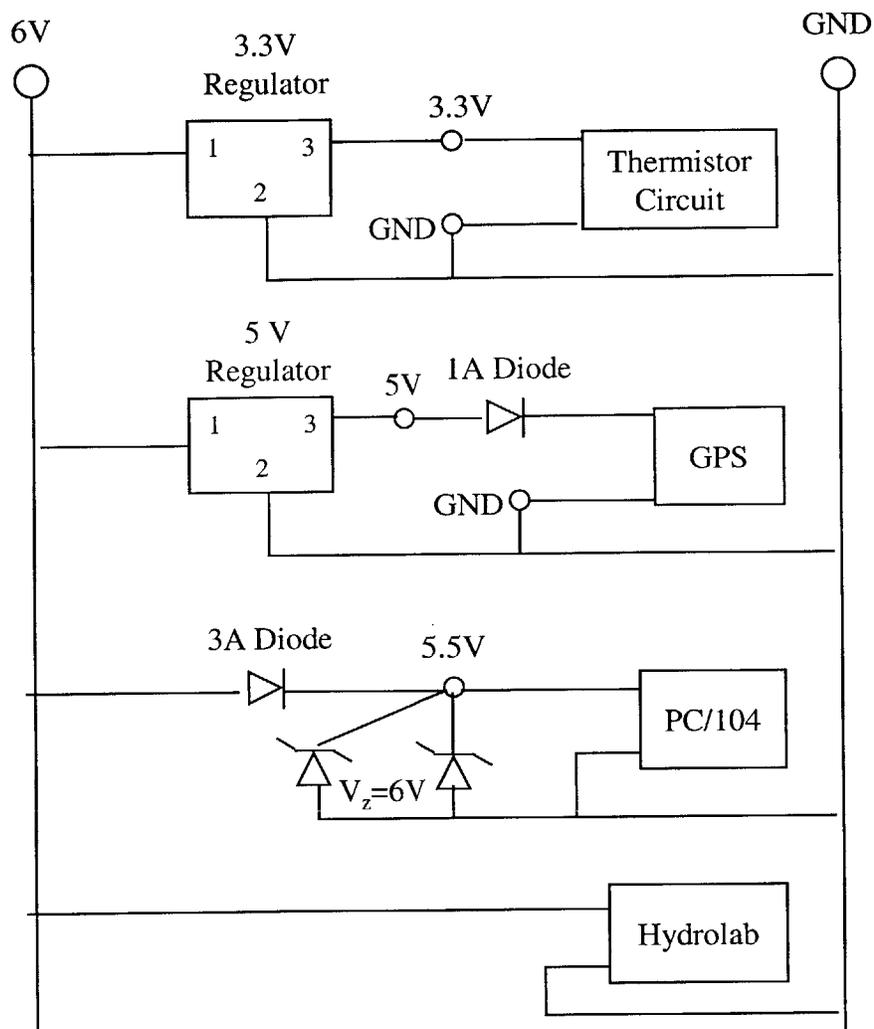


Figure 10-2: Power supply circuit diagram

The wiring was done according to the following color code:

Component	Color
Ground (All)	Black
Battery Positive	Green
Thermistor Positive	White
GPS Positive	Orange
PC/104 Positive	Brown
Hydrolab Positive	Red

Table 10-2: Power supply color code

The two batteries are connected in parallel to the 6V and GND terminals at the top of the diagram. A simple 3.3V voltage regulator is used to power the thermistor circuit. The

voltage regulator is manufactured by Texas Instruments and its model number is UA78M33CKC. A pinout of the regulator is shown below:

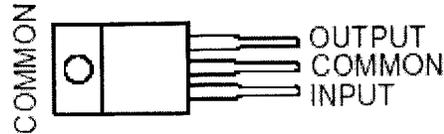


Figure 10-3: 3.3V Voltage Regulator – TI UA78M33CKC

A 5V regulator in combination with a 1A silicon diode powers the GPS. The PC/104 computer requires between 5 and 6 volts so a 3A diode is used to create a voltage drop of about 1.0 volt from the 6V power supply. Two 6V zener diodes ensure that the voltage does not go above 6V such that the computer is not damaged. The 5V regulator was manufactured by TI as well and has the model number UA7805CKTER. A view from the top of the regulator is shown below:

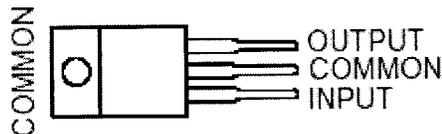


Figure 10-4: 5V Voltage Regulator – TI UA7805CKTER

Finally, the Hydrolab draws straight from the batteries since it can take between 5 and 15 volts. Note that the voltage supplied to the GPS, PC/104, and Hydrolab changes as the voltage the batteries supply changes due to the use of diodes or in the case of the Hydrolab, no circuitry. However, since each of these components can take a range of voltages, a sufficient voltage is always supplied.

So how long will this power supply last? Let's look at how much power is supplied and how power is consumed. The two 6V, 30 amp-hour batteries provide 360 watt-hours of power combined. The consumption of each component is shown below:

Component	Voltage	Current	Power
Thermistor Circuit	3.3V	22 uA	72.6 x 10 ⁻⁶ W
GPS	3.75V	110 mA	.4125 W
PC/I04	5.25V	900 mA	4.725 W
Hydrolab	6.25V	69 mA	.43125 W
Total:			5.568 W

Table 10-3: Buoy components power consumption

360-watt hours divided by 5.568 watts = **64.65 hours**. This is likely an overestimate since some power is likely lost due to the use of voltage regulators and diodes to create voltage drops. However, since the length of the buoy's deployment is not expected to last more than 1-2 days initially, this design is sufficient for now. In the future, improvements can be made on this initial design.

10.2 Rose Enclosure Penetrations

There are three external devices that require penetrations in the Rose enclosure: 1) Thermistor chain 2) Hydrolab MiniSonde 3) Laptop. The laptop will connect to the buoy computer and allow us to diagnose what's occurring on the buoy. The basic plan for each penetration is the following: A hole is made in the enclosure wall. A bulkhead connector is installed into the hole. The bulkhead facilitates watertight connections between devices inside and outside the enclosure. A nut on the inside of the enclosure secures the bulkhead in place. Finally, a cable connects the external device to the bulkhead and another cable connects the bulkhead to a device inside the enclosure. The bulkhead

connectors and external cables were obtained from Impulse Enterprises. Their model numbers are used here as reference. Internal cables will be custom made.

10.2.1 Thermistor Chain

Most of the cables and connectors needed for the thermistor chain have already been obtained. However, we may want to replace some of the parts since they may not be suitable for water deployment.

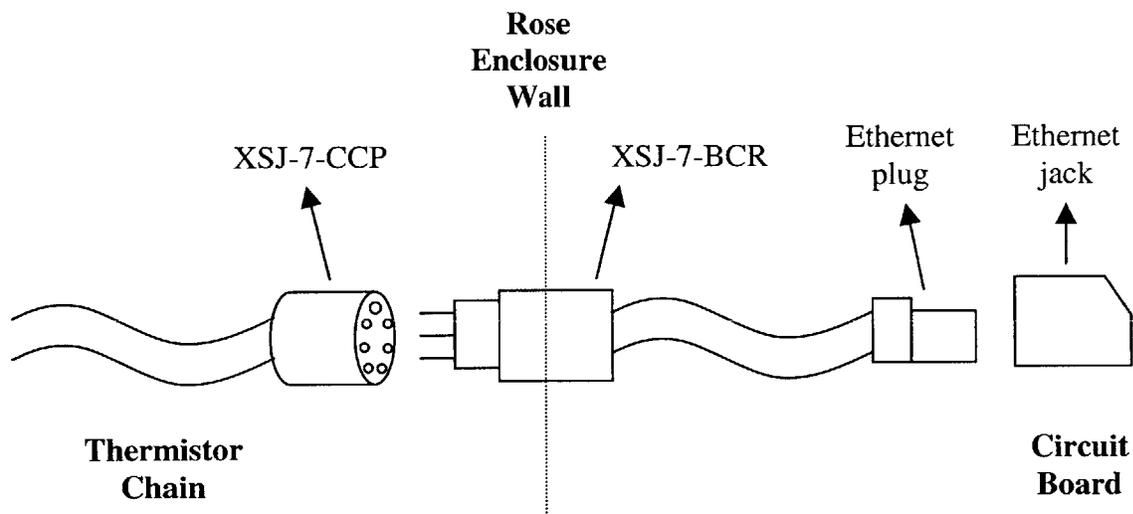


Figure 10-5: Overview of thermistor chain enclosure penetration

10.2.1.1 External Connection

The bulkhead connector is an XSJ-7-BCR connector. It attaches to an XSJ-7-CCP connector on the thermistor chain. This is a simple connection since the mating connector is directly on the instrument.

10.2.1.2 Internal Connection

The contact configuration of the XSJ-7-BCR is:

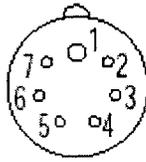


Figure 10-6: XSJ-7-BCR contact configuration

The 7 pins of the bulkhead connector are connected to an 8-pin Ethernet plug. The contact configuration of the Ethernet plug is defined to be:

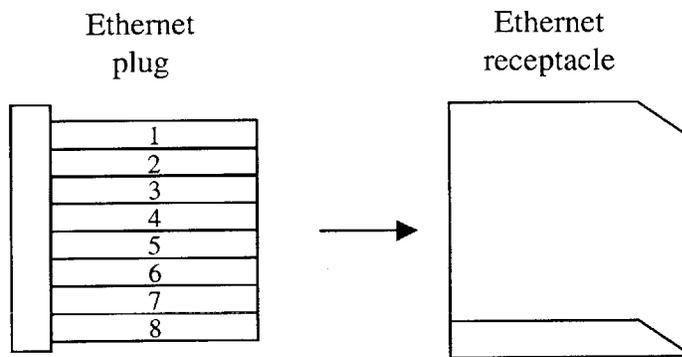


Figure 10-7: Ethernet plug contact configuration

Based on these two contact configurations, the pinout from bulkhead to Ethernet is:

XSJ-7-BCR	Ethernet
1	1
2	7
3	6
4	5
5	4
6	3
7	2

Table 10-4: XSJ-7-BCR to Ethernet pinout

10.2.2 Hydrolab

Most of the parts for the Hydrolab had to be custom made. Here is an overview of the Hydrolab setup.

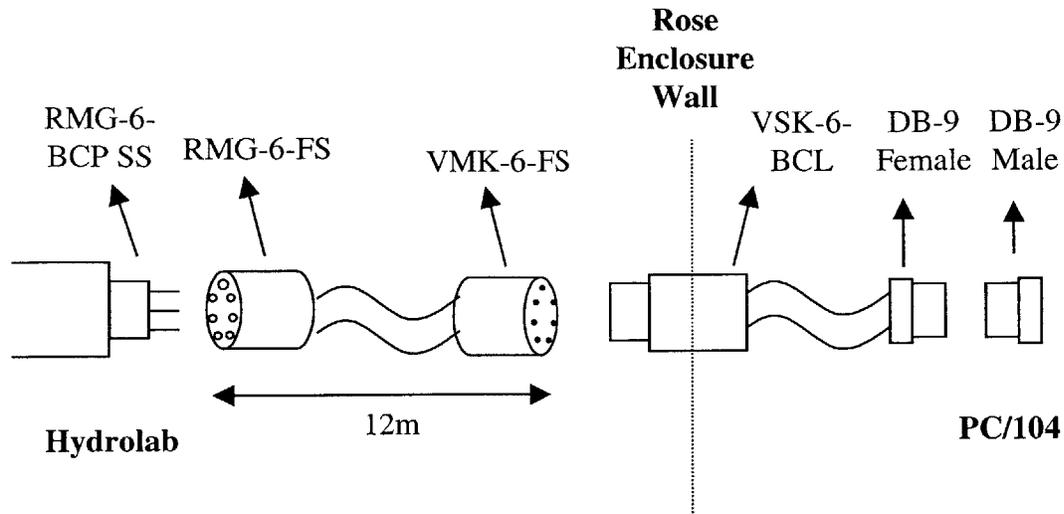


Figure 10-8: Overview of Hydrolab enclosure penetration

10.2.2.1 External Cable

The Hydrolab MiniSonde's bulkhead connector is custom made by Impulse under the model number RMG-6-BCP SS. The RMG-6-FS is a standard connector made by Impulse and will connect to the MiniSonde's special bulkhead. However, a special locking sleeve must be obtained from Hydrolab that works with the special bulkhead. On the other side of the cable, a VMK-6-FS connector mates with the VSK-6-BCL bulkhead. The actual cable is a Belden 8426 cable consisting of 6 #20 AWG. The pinout between the two connectors is a normal pinout (pin 1 to pin 1, pin 2 to pin 2, etc.).

10.2.2.2 Internal Cabling

Internal cabling will connect the 6 pins of the bulkhead connector to the DB-9 connector of the computer. We will make a custom cable by soldering 6 wires from the bulkhead

onto the appropriate pins of a female DB-9 connector. The contact configuration of the VSK-6-BCL is:

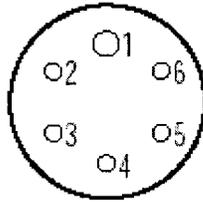


Figure 10-9: VSK-6-BCL contact configuration

The contact configuration of a female DB-9 connector is below:

Female, 9-Pin, D-Sub
(Viewed from the front)

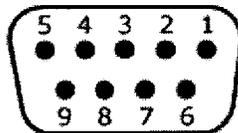


Figure 10-10: Female DB-9 contact configuration

The pinout from the bulkhead connector to the DB-9 connector is:

Bulkhead	DB-9
1	4,6
2	5
3	3
4	2
5	8
6	9

Table 10-5: Hydrolab penetration: bulkhead to DB-9 pinout

Both pins 4 and 6 are connected to pin 1 of the bulkhead connector. Power is provided on these pins.

10.2.3 Laptop

This field link takes advantage of the computer's feature that dumps its terminal information over its COM2 port to any connected computer. A typical scenario will be to connect a laptop to this penetration and use Hyperterminal to display the terminal information. The field link allows us to diagnose any problems the computer maybe having without opening the buoy enclosure. An overview of the connection is given below:

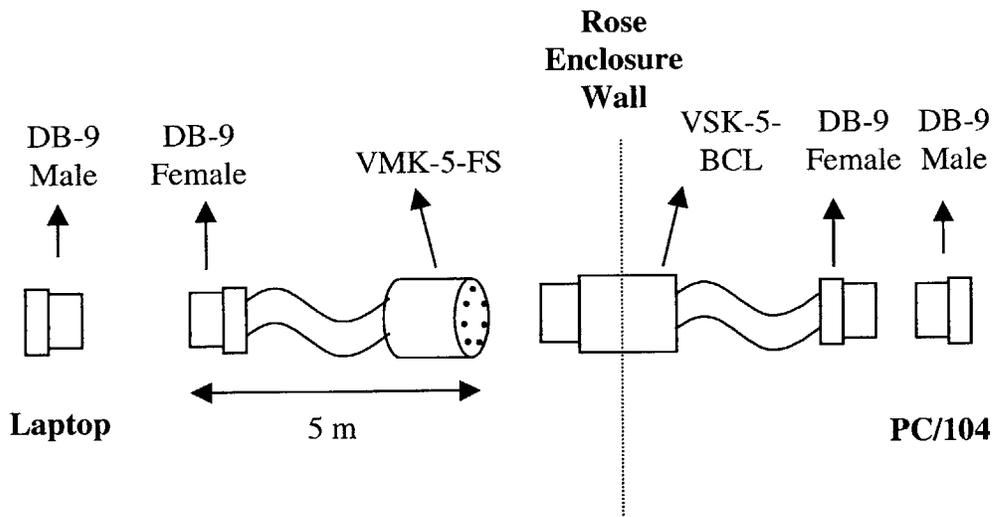


Figure 10-11: Overview of field link enclosure penetration

10.2.3.1 External Cable

The following is the contact configuration for the VMK-5-FS connector:

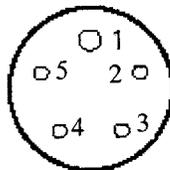


Figure 10-12: VMK-5-FS contact configuration

The contact configuration of the female DB-9 connector used in this penetration is the same as shown in **Section 10.2.2.2**. Based on these contact configurations, the pinout between the VMK-5-FS and DB-9 connectors is the following:

VMK-5-FS	Female DB-9	Purpose
1	2	Transmit (to DCE)
2	3	Receive (from DCE)
3	4	Power/DTE Indicator
4	5	Signal ground
5	6	DCE Indicator

Table 10-6: VMK-5-FS to Female DB-9 pinout

The cabling used is Belden 8425 cable with 5 #20 AWG.

10.2.3.1 Internal Cable

The contact configuration of the VSK-5-BCL is:

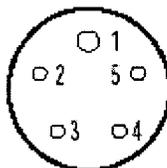


Figure 10-13: VSK-5-BCL contact configuration

The contact configuration of the female DB-9 repeated here for convenience:

Female, 9-Pin, D-Sub
(Viewed from the front)

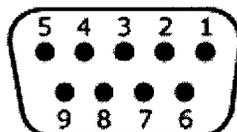


Figure 10-14: Female DB-9 contact configuration

The pinout here is the same as the external cable:

VSK-5-BCL	Female DB-9	Purpose
1	2	Transmit (to DCE)
2	3	Receive (from DCE)
3	4	Power/DTE Indicator
4	5	Signal ground
5	6	DCE Indicator

Table 10-7: VSK-5-BCL to Female DB-9 pinout

10.3 Internal Enclosure Packaging

The inside of the Rose enclosure was augmented to secure and mount the various buoy components. A sheet of metal is mounted on the bottom of the enclosure. Sheets of plastic were cut to form walls within the enclosure. These sheets of plastic are secured to the sheet of metal. The plastic walls section off the inside of the enclosure and minimize the movement of the heavy batteries. In addition, the walls provide an area to mount electronics. To create more space inside the enclosure, the PC/104 computer was mounted directly onto the wall of the enclosure. Two views of the finished enclosure are below:

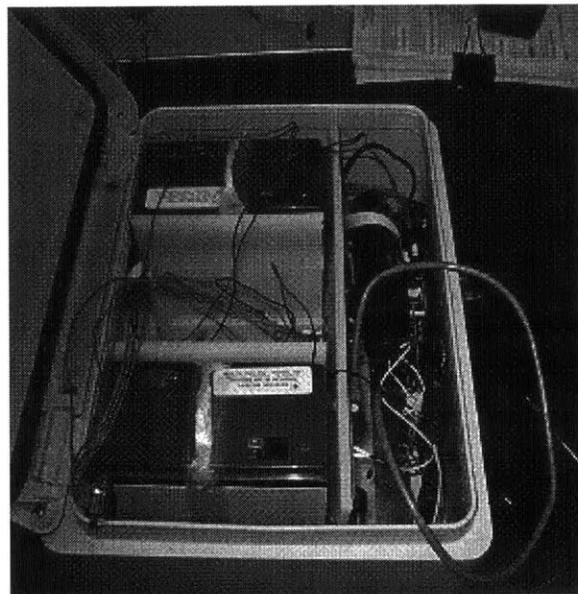
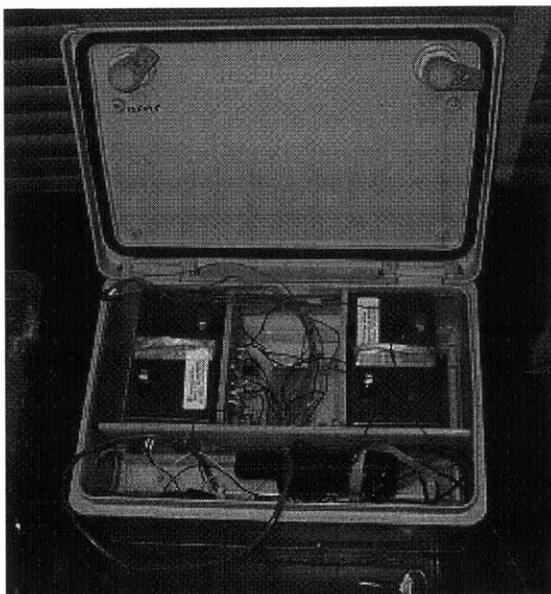


Figure 10-15: Front and side view of Rose enclosure

The two batteries are held in place by the plastic walls and also Velcro attached to the walls. Handles made of tape are attached to the battery to allow for easy removal and replacement of the batteries. All other electronics are mounted using screws and nuts to ensure they do not move. The area between the batteries is reserved for the power supply board and power amp circuitry to be added in the future.

11 Testing

Four facets of the network were tested: 1) Multi-MOOS communication 2) Adaptive behavior 3) Wireless transmission 4) Data collection.

11.1 Multi-MOOS Test

The first test done was to demonstrate communication between two MOOS communities and adaptive behavior. A MOOS community was set up on the buoy and shore station.

This test was carried out in the controlled environment of a lab. The .moos configuration file for the buoy then shore is shown below:

```
ServerHost = localhost
ServerPort = 9000

ProcessConfig = ANTLER
{
  Run = MOOSDB @ NewConsole = false
  Run = pLogger @ NewConsole = false
  Run = pMonitor @ NewConsole = false
  Run = pMonitorListener @ NewConsole = false
  Run = iThermistorChain @ NewConsole = false
}

ProcessConfig = pLogger
{
  File = UMLBuoyLog.txt
  AsyncLog = true

  Log = BUOY1_THERM_COMMAND @ 1.0
  Log = BUOY1_THERM_1 @ 1.0
  Log = BUOY1_THERM_2 @ 1.0
  Log = BUOY1_THERM_3 @ 1.0
  Log = BUOY1_THERM_4 @ 1.0
  Log = BUOY1_THERM_5 @ 1.0
  Log = BUOY1_THERM_6 @ 1.0
}

ProcessConfig = pMonitor
{
  IP = 192.168.0.1
  PORT = 5000
  MONITOR = BUOY1_THERM_1
  MONITOR = BUOY1_THERM_2
  MONITOR = BUOY1_THERM_3
  MONITOR = BUOY1_THERM_4
  MONITOR = BUOY1_THERM_5
  MONITOR = BUOY1_THERM_6
}

ProcessConfig = pMonitorListener
{
  PORT = 5000
  LISTEN = BUOY1_THERM_COMMAND
}

ProcessConfig = iThermistorChain
{
  CollectionTick = 0.1
}
```

Figure 11-1: Buoy shore configuration for Multi-MOOS test

```

ServerHost = localhost
ServerPort = 9000

ProcessConfig = ANTLER
{
  Run = MOOSDB @ NewConsole = false
  Run = pLogger @ NewConsole = false
  Run = pMonitor @ NewConsole = false
  Run = pMonitorListener @ NewConsole = false
  Run = pShoreCommand @ NewConsole = false
}

ProcessConfig = pLogger
{
  File = UMLShoreLog.txt
  AsyncLog = true

  Log = BUOY1_THERM_COMMAND @ 1.0
  Log = BUOY1_THERM_1 @ 1.0
  Log = BUOY1_THERM_2 @ 1.0
  Log = BUOY1_THERM_3 @ 1.0
  Log = BUOY1_THERM_4 @ 1.0
  Log = BUOY1_THERM_5 @ 1.0
  Log = BUOY1_THERM_6 @ 1.0
}

ProcessConfig = pMonitor
{
  IP = 192.168.0.2
  PORT = 5000
  MONITOR = BUOY1_THERM_COMMAND
}

ProcessConfig = pMonitorListener
{
  PORT = 5000
  LISTEN = BUOY1_THERM_1
  LISTEN = BUOY1_THERM_2
  LISTEN = BUOY1_THERM_3
  LISTEN = BUOY1_THERM_4
  LISTEN = BUOY1_THERM_5
  LISTEN = BUOY1_THERM_6
}

ProcessConfig = pShoreCommand
{
}

```

Figure 11-2: Shore configuration in Multi-MOOS test

The general interaction of this network is the following:

- 1) iThermistorChain collects thermistor chain data
- 2) pMonitor(Buoy) and pMonitorListener(Shore) publish that thermistor chain data on the shore which is received by pShoreCommand
- 3) pShoreCommand processes the thermistor chain data at the shore station
- 4) pShoreCommand sends a command back to iThermistorChain through pMonitor(Shore) and pMonitorListener (Buoy)

The command that pShoreCommand sends is “CollectionTick=0.2”, which will tell iThermistorChain to start collecting data every 5 seconds. Let’s look at the logs taken at each node to see this interaction take place. First, the buoy logs:

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% LOG FILE:          UMLBuoyLog.txt.alog
%% FILE OPENED ON Thu Feb 14 03:07:26 1980
%% LOGSTART          319370844.766
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
15.847      BUOY1_THERM_6      iThermistorChain 23.375
15.237      BUOY1_THERM_5      iThermistorChain 24.375
14.627      BUOY1_THERM_4      iThermistorChain 23.375
14.017      BUOY1_THERM_3      iThermistorChain 24.125
13.407      BUOY1_THERM_2      iThermistorChain 23.625
12.797      BUOY1_THERM_1      iThermistorChain 39.125
26.557      BUOY1_THERM_6      iThermistorChain 23.625
25.947      BUOY1_THERM_5      iThermistorChain 24.125
25.337      BUOY1_THERM_4      iThermistorChain 23.375
24.727      BUOY1_THERM_3      iThermistorChain 23.125
24.117      BUOY1_THERM_2      iThermistorChain 23.375
23.507      BUOY1_THERM_1      iThermistorChain 39.125
27.786      BUOY1_THERM_COMMAND pMonitorListener CollectionTick=0.2
32.267      BUOY1_THERM_6      iThermistorChain 23.125
31.657      BUOY1_THERM_5      iThermistorChain 23.375
31.047      BUOY1_THERM_4      iThermistorChain 23.750
30.437      BUOY1_THERM_3      iThermistorChain 23.375
29.827      BUOY1_THERM_2      iThermistorChain 23.375
29.217      BUOY1_THERM_1      iThermistorChain 39.125
37.977      BUOY1_THERM_6      iThermistorChain 23.375
37.367      BUOY1_THERM_5      iThermistorChain 23.125
36.757      BUOY1_THERM_4      iThermistorChain 23.375
36.147      BUOY1_THERM_3      iThermistorChain 23.375
35.537      BUOY1_THERM_2      iThermistorChain 24.125
34.927      BUOY1_THERM_1      iThermistorChain 39.125

```

Figure 11-3: Buoy logs from Mult-MOOS test

And the shore logs:

```

*****
** LOG FILE:      UMLShoreLog.txt.alog
** FILE OPENED ON Wed May 19 11:29:36 2004
** LOGSTART      1084980576.26
*****
24.956          BUOY1_THERM_1          pMonitorListener 39.125
24.954          BUOY1_THERM_2          pMonitorListener 23.625
24.953          BUOY1_THERM_3          pMonitorListener 24.125
24.951          BUOY1_THERM_4          pMonitorListener 23.375
24.949          BUOY1_THERM_5          pMonitorListener 24.375
24.947          BUOY1_THERM_6          pMonitorListener 23.375
35.046          BUOY1_THERM_1          pMonitorListener 39.125
35.044          BUOY1_THERM_2          pMonitorListener 23.375
35.042          BUOY1_THERM_3          pMonitorListener 23.125
35.040          BUOY1_THERM_4          pMonitorListener 23.375
35.039          BUOY1_THERM_5          pMonitorListener 24.125
35.037          BUOY1_THERM_6          pMonitorListener 23.625
35.687          BUOY1_THERM_COMMAND    pShoreCommand    CollectionTick=0.2

```

Figure 11-4: Shore logs from Multi-MOOS test

The first 12 entries in each log correspond with each other. The iThermistorChain process publishes 12 pieces of data and the pMonitorListener process on the shore station is publishing that data on the shore station. At that point, pShoreCommand will notice the temperature difference thermistors 1 and 2 and issue a command for iThermistorChain to speed up its collection. That command can be seen at time 35.687 in the shore logs. pMonitor(shore) forwards that data to pMonitorListener(buoy) which publishes that data at time 27.786 in the buoy logs. After that command has been issued, one can see that the readings begin to take place every 5 seconds. So, this test has shown that iThermistorChain and pShoreCommand can communicate which demonstrates that pMonitor and pMonitorListener work and two MOOS communities can communicate. Second, the test shows iThermistorChain speed up its collection frequency which demonstrates adaptive behavior.

11.2 Upper Mystic Lake

Testing culminated in a final field test at Upper Mystic Lake, the eventual site of deployment. A laptop was used as a shore station and was set up on the shore of the lake. A small boat was used to take the Rose enclosure and sensors out into the lake. The goal of this test was to evaluate the range of wireless transmission, reliability of data collection by sensors and software, and finally the validity of data collected.

The first test was the range of the wireless link. The Rose enclosure was taken as far as possible before the wireless link started to degrade in quality. The quality of the link was measured by the Orinoco Client Manager residing on the laptop. Once the quality of the link decreased below 5 (on a scale of 1 to 5), the test was stopped. The range of the wireless cards was estimated to be 30m. Considering that the necessary range of the wireless card needs to be on the order of 1km, the strength of wireless transmission needs to be greatly increased. One simple change will be the addition of an external antenna to the buoy. Currently, the wireless card does not have a clear line of sight to the shore due to its placement inside the Rose enclosure. An external antenna mounted into the lid of the enclosure will allow a clear line of sight. In addition, power amp circuitry can be added to boost the strength of transmission.

The next test was to deploy the sensors and check that they properly collected data. Each software module was configured to collect data from its sensor every 10 seconds. The GPS sensor had similar troubles to the wireless card in that its antenna seemed to be disturbed by its placement inside the enclosure. The GPS was not able to track enough

satellites to determine a fix on its location while mounted inside the enclosure. When the GPS was unmounted and held above the enclosure, it was able to determine that it was located at 42.21 degrees North and 71.08 degrees West.

Some possible sources of interference with the GPS are the other metallic objects inside the enclosure (thermistor circuit, PC/104 computer, metal mounting plate) and the wireless card. Theoretically, the wireless card and GPS should not interfere with each other since they are transmitting on different frequencies. Further testing will need to be done to determine the source of the interference. Possible solutions will be to mount the GPS such that its antenna is outside of the enclosure or obtain a GPS with an external antenna connector.

The second sensor was the Hydrolab. A profile of the lake was done by slowly lowering the Hydrolab over the edge of the boat. The results of the profile are given below in **Table 11-1**.

Depth	Time(s)	pH	Temp(C°)	Specific Conductance	ORP (mV)	Dissolved Oxygen % Sat.
0.49	0.000	8.38	20.40	0.571	440	109.9
0.98	10.110	8.37	20.25	0.568	440	110.1
1.62	20.220	8.40	20.13	0.550	439	109.2
1.97	50.450	8.41	20.04	0.426	440	109.7
2.04	40.340	8.41	20.04	0.427	440	109.4
2.05	30.330	8.42	20.04	0.429	439	109.2
2.52	60.454	8.40	19.93	0.432	440	110.2
2.90	90.590	8.04	18.62	0.436	444	106.8
2.95	80.580	8.09	18.35	0.432	444	106.8
2.97	70.470	8.17	18.39	0.430	443	106.7
4.00	100.700	7.93	15.51	0.420	448	104.0
4.01	110.810	7.80	15.27	0.421	449	96.2
4.89	131.030	7.57	12.87	0.431	454	81.6
4.90	120.920	7.63	12.99	0.427	453	88.2
5.92	141.140	7.45	10.59	0.462	459	77.6
5.92	151.251	7.40	9.23	0.466	460	72.8
6.95	161.270	7.33	7.86	0.693	464	71.3
7.02	171.380	7.32	7.91	0.690	465	70.7
7.89	181.490	7.28	6.86	0.711	468	70.2
7.95	191.600	7.27	6.84	0.716	469	70.0
10.93	232.040	7.14	5.34	0.837	479	65.9
11.60	242.150	7.11	5.37	0.845	482	62.9
11.99	252.260	7.04	4.70	0.961	485	57.9
11.99	262.370	7.04	4.72	1.014	485	47.6

Table 11-1: Hydrolab Profile data from UML

One can see the trends in the lake in relation to its depth. Also note that the depths and times recorded do not correlate⁵. This could be due to human error in profiling, possible drift of the Hydrolab in the lake during deployment, and also measurement error by the Hydrolab's depth sensor.

The final sensor was the thermistor chain. Initial testing of the thermistor chain started on land. The Hydrolab was used as a control instrument to compare the readings of the

⁵ Although this is to be expected given that the lowering of the Hydrolab was not timed.

thermistors against. A comparison of the readings is given in **Table 11-2**. The numbers shown are the averages of 6 readings taken and the absolute error compared to the Hydrolab are shown.

	Hydrolab	Thrm 1	Thrm 2	Thrm 3	Thrm 4	Thrm 5	Thrm 6
Temp.	24.56	40.79	24.83	24.04	23.79	24.67	23.17
Abs. Err.	0	16.23	0.27	-0.52	-0.77	0.11	-1.39

Table 11-2: Average thermistor readings on land

Note the large error associated with thermistor 1. The source of error is undetermined at this point, but it may either come from the chain itself or a problem in our own interface. The next test was to try the thermistor chain in water. The thermistor chain was lowered off the side of the boat. The chain is about 14 meters long and its thermistors are spaced accordingly as shown in **Figure 11-5**.

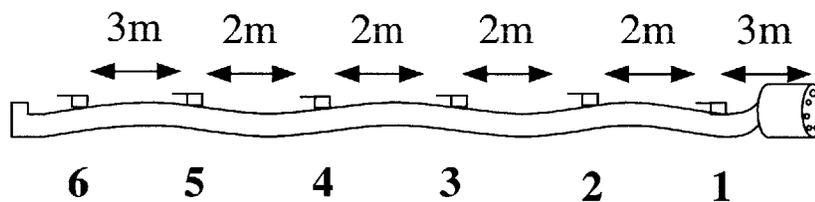


Figure 11-5: Thermistor chain spacing

Ideally, we would like to compare the temperature profile recorded by the thermistor chain versus the temperature profile recorded by the Hydrolab. However, when held under water, the thermistor chain has a low density so it tends to float and drift. Unfortunately, no mooring line or weight was available to keep the thermistor chain straight. Therefore, the depths of each thermistor during the test are unknown. However, based on the temperatures read by the thermistors and accounting for the floating of the chain, we can *conjecture* what might have happened:

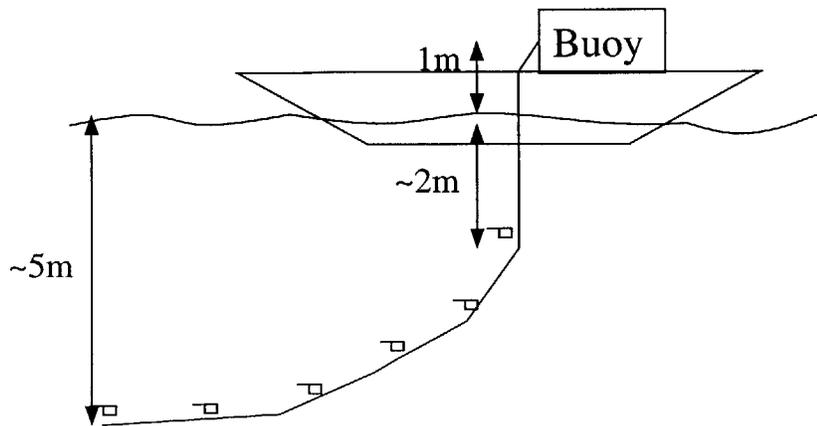


Figure 11-6: Conjecture of thermistor chain underwater

The above figure shows the boat with the buoy placed inside of it. The thermistor chain runs from the buoy and then under water. We estimated that the length from the buoy to the water was about 1 meter. Therefore, since the distance from the connector to the first thermistor is 3 meters, we can guess that the first thermistor is approximately 2 meters underwater. As the thermistor chain goes farther underwater, one would guess that it follows an asymptotic curve as shown. Based on the temperature that the final thermistor read and comparing it to data read by the Hydrolab, one could guess that the final thermistor chain was at about 5 meters. Below is a comparison of the average temperature measured by the thermistor and the pertinent range of Hydrolab data.

Thermistor	Temp(C)	Depth(m)	Temp(C)
1	36.36	0.49	20.40
2	19.78	0.98	20.25
3	16.76	1.62	20.13
4	14.04	1.97	20.04
5	12.95	2.04	20.04
6	12.05	2.05	20.04
		2.52	19.93
		2.90	18.62
		2.95	18.35
		2.97	18.39
		4.00	15.51
		4.01	15.27
		4.89	12.87
		4.90	12.99
		5.92	10.59

Figure 11-7: Comparison of UML thermistor and Hydrolab data

Although we cannot make any strong conclusions about the depth of the thermistors, one can see the trend that as the thermistor depth increases, the temperature decreases as expected.

Overall, the test was successful in demonstrating the data collection and wireless transmission functionality of the network. Improvements need to be made in increasing the range of wireless transmission and collection of GPS data.

12 Future Work

There are several major items that require attention in the near future:

- *Increasing range of wireless transmission* – As demonstrated in the field test, the strength of the wireless link needs to be increased many times over. The addition of an external antenna and/or power amp circuitry will need to be developed.
- *Integration with AUV* – Obviously, this prototype network is missing a big piece: the AUV. How the AUV will be integrated needs to be specified including defining exactly what its interaction with other nodes will be.
- *Acoustic communication* – As a corollary to integrating the AUV, acoustic communication needs to be implemented on the buoy. Simple point-to-point communication using the acoustic modems will be the obvious first step.

However, note that there are multiple buoys that can communicate with the AUV. A protocol will need to be developed that determines which buoy is responsible for communicating with the AUV at a given time.
- *Data management* – Data is currently stored in plain text files. Viewing data in such a format when there are so many points of data is difficult to say the least. A scientific data storage format should be chosen and software should be written to convert the text generated by the network into the storage format. There are a number of formats available that would suffice including NetCDF, HDF, and XML-based formats. Viewing and analysis of data will be eased by the use of these formats as there are existing associated applications designed to use them.

Appendix A Linux and Networking Configuration

This appendix covers the configuration of the shore station and PC/104 computers.

Topics covered include:

- Computer components purchased
- Setup of the Linux operating system on each computer
- Networking configuration of each that allows the computers to communicate
- How development of new software for the buoy computer can be done.

It is assumed that the reader has little knowledge of using Linux, but a general understanding of computer systems.

1 Computer Components

The buoy computer is an embedded PC/104 computer. The model chosen was a TS-5500 computer from Technologic Systems (www.embeddedx86.com). The TS-5500 contains a 133 Mhz AMD Elan 520 processor, 32 MB SDRAM, and a PCMCIA slot. The PCMCIA slot allows a wireless card to be used. In addition, the TS-5500 contains 3 serial ports and 38 digital I/O lines that provide ample room to connect sensors and other devices. A TS-SER4 board is stacked on the TS-5500 to provide 4 more serial ports. Finally, the TS-5500 contains a Compact Flash card slot which can provide ample memory for logging data, software, and an operating system.

The use of an embedded computer on the buoy meant that an embedded operating system had to be used. An embedded operating system is designed to function under the smaller amount of resources available on an embedded computer. An embedded version of

Linux was chosen due to its low cost (free) and open source policy. More specifically, we used TS-Linux which was developed by Technologic Systems specifically for its computers. Using TS-Linux was particularly easy since Technologic Systems had already tested the operating system on the exact model of computer we were using so there were minimal problems to deal with. If you've ever used Linux (especially embedded versions of Linux), you can appreciate this fact.

One caveat of using embedded Linux was that Linux had to be installed on a PC in order to develop software for the embedded computer. Debian Linux (version 3.0) was chosen as the Linux version to install on the PC. Debian is actually one of the more difficult versions of Linux to use (much to the author's chagrin), but it has worked satisfactorily without too much difficulty. A standard Dell desktop is used as a test shore station within the lab.

To provide wireless transmission, wireless cards were installed on each computer. An Orinoco Classic Gold PC Card was installed onto the buoy computer. It has a range of 500 meters and an external antenna connector. An Orinoco PCI a/b/g card was installed onto the desktop. Both cards are compatible with Linux and use 802.11b technology.

2 TS-Linux Configuration

One advantage of using TS-Linux with a TS-5500 is that all the necessary drivers are already installed in the operating system. Therefore, only configuration of drivers needs to be done and in this case, we only need to configure one driver for the wireless card.

The second item is configuring the computer to automatically start its data collection software upon boot-up. Initialization scripts are installed in the proper directories to make this happen. More information about TS-Linux and its structure can be obtained at <http://www.embeddedx86.com/support/linux/linuxhelp.php>.

2.1 Orinoco Classic Gold PC Card

The “orinoco_cs” driver maintained by David Gibson (http://www.hpl.hp.com/personal/Jean_Tourrilhes/Linux/Orinoco.html) is used for the wireless card. The orinoco_cs driver can be configured with the Wireless Tools package developed by Jean Tourrilhes (http://www.hpl.hp.com/personal/Jean_Tourrilhes/Linux/Tools.html).

I'll stop and give a brief description of these Wireless Tools as they are used to configure the wireless cards on both the PC/104 and PC. In order to ease the configuration of wireless cards, an addition was made to the Linux kernel named Wireless Extensions. The Wireless Extensions are an interface to the kernel that must be implemented by wireless card drivers. The Wireless Tools package takes advantage of the Wireless Extensions and allows a user to configure any driver that implements the Wireless Extensions. The Wireless Tools package contains a number of executables that configure or retrieve information about a driver. The main tool one will use is “iwconfig”. Iwconfig configures all of the major parameters of a wireless card driver such as the mode of the card (ad-hoc, Managed, Master), the channel it uses, and the ESSID. For more information about how to use “iwconfig”, just type “man iwconfig”.

So anyways, back to how all this wireless stuff works on the PC/104. The boot-up PCMCIA scripts will load the orinoco_cs driver according to the configurations located in /etc/pcmcia/network.opts and /etc/pcmcia/wireless.opts. In network.opts, the IP address, network address, broadcast address, and network mask of the wireless card can be set. The current configuration of these parameters is:

```
IP_ADDR = 192.168.0.2
NETMASK = 255.255.255.0
NETWORK = 192.168.0.0
BROADCAST = 192.168.0.255
```

Figure A1-1: PC/104 wireless configuration in /etc/pcmcia/network.opts

In wireless.opts, the parameters directly pertaining to the wireless card are set.

Wireless.opts allows users to set the configuration of the card based on the type of card loaded. The MAC address of the card is scanned during boot-up and matched with a configuration section in the wireless.opts file. The Orinoco PC Card is configured in the Lucent Wavelan/IEEE section and its configuration is shown below:

```
# Lucent Wavelan IEEE (+ Orinoco, RoamAbout and ELSA)
# Note : wlan_cs driver only, and version 1.0.4+ for encryption support
*,*,*,00:60:1D:*|*,*,*,00:02:2D:*)
  INFO="Wavelan IEEE example(Lucent default settings)"
  ESSID="Parsons"
  MODE="Ad-hoc"
```

Figure A1-2: PC/104 wireless configuration in /etc/pcmcia/wireless.opts

Note that the ESSID of the wireless card is set to "Parsons". The ESSID of the shore station's wireless card is also set to "Parsons". Any node in the network should set its ESSID to "Parsons" if it would like to communicate with other nodes in the network.

Another networking configuration step is also done to allow for hostname resolution within the network. Hostname resolution is the process of mapping an IP address to the hostname of a computer. Typically, this is done on the Internet through the DNS system, but since this sensor network is not connected to the Internet, hostname resolution must be done locally.

Local resolution of hostnames can be done through the `/etc/hosts` file. The `/etc/hosts` file contains a mapping of IP addresses to hostnames and aliases. Therefore, at minimum, the shore station's IP address and a hostname should be added to the `/etc/hosts` file of the PC/104 computer. The `/etc/hosts` file currently residing on the buoy computer is below:

127.0.0.1	miniepc.embeddedx86.com	miniepc
192.168.0.1	parsons-shore.mit.edu	parsons-shore
192.168.0.2	buoy1.mit.edu	buoy1

Figure A1-3: PC/104 hostname resolution configuration in `/etc/hosts`

2.2 Boot Scripts

The ideal behavior of the buoy computer once deployed into the lake is to automatically begin collecting data once power is applied. In order to do this, scripts must be run at boot time that configure the computer properly and start the data collection software.

The `/etc/init.d` directory on a Linux system contains initialization and termination scripts to be run. These files are linked to by files in the `/etc/rc.d` directory. The `/etc/rc.d` directory contains 7 subdirectories: `rc1.d`, `rc2.d`, and so on, such that for a directory

rcN.d, N is the runlevel⁶ at which the files within rcN.d should be accessed. The filenames within the rcN.d directories are of the form [SK]nn<init.d reference>. 'S' means start the job, 'K' means kill the job, and nn is a relative sequence number for starting or killing the job. The <init.d reference> is the name of the job upon which the prescribed actions are performed and a reference to a script in the /etc/init.d directory. Files in a rcN.d directory are actually symbolic links to the /etc/init.d scripts that they reference. Also note that the files in the rc3.d directory are always run during boot-up.

So to initialize the buoy computer, we need to place two scripts in the /etc/init.d directory. One script is named 'rc.serial' initializes all serial ports on the computer. The other script is named 'runMOOS' and starts the MOOS software. These two scripts are shown below:

```
#!/bin/bash

# make the /dev entries for up to 8 COM ports (first 4 are already
there).
mknod -m 666 /dev/ttyS4 c 4 68
mknod -m 666 /dev/ttyS5 c 4 69
mknod -m 666 /dev/ttyS6 c 4 70

# this is for COM3 on the TS-5500, jumper for IRQ5
setserial -v /dev/ttyS2 auto_irq autoconfig

# this is for the TS-SER4 starting at COM4 (TS-5500)
setserial -v /dev/ttyS3 port 0x2e8 auto_irq autoconfig
setserial -v /dev/ttyS4 port 0x3a8 auto_irq autoconfig
setserial -v /dev/ttyS5 port 0x2a8 auto_irq autoconfig
setserial -v /dev/ttyS6 port 0x3a0 auto_irq autoconfig
```

Figure A1-4: PC/104 initialization script for serial ports: rc.serial

⁶ Linux's processes are divided into *runlevels*. At each runlevel, a set of processes is allowed to run.

```
#!/bin/sh
#
# starts pAntler with configuration NetworkParsons.moos

export PATH=$PATH:/home/MOOSBin
/home/MOOSBin/pAntler /home/MOOSBin/NetworkParsons.moos
```

Figure A1-5: PC/104 initialization script for MOOS: runMOOS

In addition to these two scripts, we place two links to them named ‘S60serial’ and ‘S62MOOS’ in /etc/rc3.d. Note that the numbers 60 and 62 are arbitrary and could be changed if these scripts needed to be run earlier. S60serial and S62MOOS are run when the computer boots up and they automatically start the ‘rc.serial’ and ‘runMOOS’ scripts.

3 Debian Linux Configuration

This section will cover the configuration of several devices on the PC and how to develop software for the buoy.

3.1 Ethernet – 3Com 590

Configuring the Ethernet card was fairly easy. First, I registered the PC for MIT’s DHCP service and then did the following steps:

- 1) Compiled the 3c59x driver into the kernel
- 2) Inserted the following in **Figure A1-6** into the /etc/network/interfaces file

```
auto eth0
iface eth0 inet dhcp
```

Figure A1-6: PC Ethernet configuration

3.2 USB Card Reader/Writer

The USB Card reader/writer was needed to read/write to a CompactFlash card. We purchased a PNY CompactFlash reader/writer from Best Buy. Getting USB support entailed recompiling the correct drivers/options into the kernel. Use at least kernel 2.4.21. Compile the following options under the two given categories into the kernel:

- 1) SCSI support
 - a. SCSI support
 - b. SCSI disk support
 - c. SCSI generic support
- 2) USB support
 - a. USB support
 - b. Preliminary device filesystem
 - c. UHCI
 - d. Mass Storage support

The USB reader/writer can be accessed through the filename `/dev/sda`. Partitions of the flash card can be accessed by accessing `/dev/sdaX` where 'X' is the number of the partition. An entry was made in `/etc/fstab` such that `/dev/sda1` can be mounted on `/mnt/flash` with the command: `mount /mnt/flash`. Similarly, `/dev/sda2` can be mounted on `/mnt/flash2` with: `mount /mnt/flash2`.

3.3 Orinoco PCI a/b/g Combo Card

This was a wireless card that plugged into the PCI bus of the computer. Even though this card is called an Orinoco card, it is not based on the Agere chipset that normal Orinoco cards (e.g. Orinoco Classic Gold PC Card) use. Since it is an a/b/g combo card, it needs an entirely different chipset. In this case, the card was based on an Atheros chipset. The development of the Atheros driver is hosted by the MADWIFI project at <http://sourceforge.net/projects/madwifi/>. At the time of this writing, the Atheros driver is in BETA stage so there are still bugs and some portions of the driver are not well-developed. One glaring deficiency is the lack of an ad-hoc mode.

The driver is currently compiled as a module in the kernel and the wireless card's interface name is ath0 (ath0 is not a typo, it really is ath0, not eth0).

To get the combo card working, I did the following:

- 1) Downloaded the atheros driver tarball from the sourceforge site
- 2) Unpacked the tarball

In the directory where you unpacked the files, do the following commands:

- 3) Make
- 4) Make install; Steps 3 and 4 compile the necessary files and place them as modules into `/lib/modules/kernel-name/build/net` so that they can be loaded upon bootup.

5) Now we need to set up Linux so that it loads the driver modules during bootup.

Add “ath_pci” into `/etc/modules`.

6) Add into `/etc/network/interfaces`:

```
auto ath0
iface ath0 inet static
  address 192.168.0.1
  netmask 255.255.255.0
  broadcast 192.168.0.255
  network 192.168.0.0
  up /sbin/iwpriv ath0 mode 3 # sets card in 802.11g mode
  up /sbin/iwconfig ath0 mode Master essid "Parsons" Channel 1
```

Figure A1-7: Wireless configuration on shore station

You can fiddle with the wireless configuration in `/etc/network/interfaces` to suit your preferences. In particular, the “iwconfig” line contains many of the parameters used in wireless networks.

Note that similar changes to the buoy computer’s `/etc/hosts` file must be made to allow the shore station to resolve the hostnames of any buoys that want to connect to it.

3.4 Developing Software for the Buoy

The TS-5500 does not have the capability to compile code so code is usually written on the PC, compiled, and then copied onto the TS-5500. In addition, since the runtime libraries of the TS-5500 and the PC are different, special considerations must be taken to ensure that code compiled on the PC uses the TS-5500’s libraries. Fortunately, Technologic Systems has provided an easy solution to this problem (the alternative would

have been to fiddle around with gcc...ugh). They have created a Linux Development Kit which creates a virtual Linux machine inside a PC. Once you are inside the virtual Linux machine, your computer behaves as if you are running the TS-5500. Therefore, any code you compile within this environment will use the TS-5500's libraries.

Instructions for using the Linux Development Kit can be found at

<http://www.embeddedx86.com/support/linux/LoopbackHowTo.html>. The first three steps have already been completed.

Note that while compilation of code must occur on a Linux machine, development of code can be done on either Linux or Windows. Since development tends to be easier in Windows, MOOS code is already set up to be used in Visual C++ 6.0.

Appendix B Thermistor Circuit Power Supply Calculations

Each of the required logic levels for the computer, thermistor converter, and multiplexer imposes some restriction on V_{CC} . By gathering all the restrictions and comparing them, we can find a V_{CC} that satisfies all of them.

Before going into the calculations, let's go over some notation. All voltages referred to will be of the form $V_{x,y}$ where x identifies the logic level and y identifies the component. The logic level can be one of four possibilities: IL, IH, OL, OH which stand for Input Low, Input High, Output Low, and Output High respectively. The component can be one of three possibilities: PC for the computer, TC for the thermistor converter, and M for the multiplexer.

Let's first consider the logic levels for a signal that is output from the computer and input to the thermistor converter. We have the following restrictions for a logical 0:

$$V_{IL,TC} \leq 0.2 * V_{CC}$$

$$V_{OL,PC} = 0V$$

This imposes no restrictions on V_{CC} . We have the following restrictions for a logical 1:

$$V_{IH,TC} \geq 0.8 * V_{CC}$$

$$V_{OH,PC} = 3.3V$$

therefore:

$$V_{CC} \leq 4.125V$$

An additional constraint imposed by the thermistor converter is its power supply constraints. Specifically:

$$V_{CC} \geq 3V$$

$$V_{CC} \leq 5.5V$$

Now, let's consider the logic levels for a signal that is output from the thermistor converter and input to the computer. The PC uses standard TTL thresholds. We have the following restrictions for a logical 0:

$$V_{IL,PC} \leq 0.8V$$

$$V_{OL,TC} = 0.4V$$

This imposes no restrictions on V_{CC} . We have the following restrictions for a logical 1:

$$V_{IH,PC} \geq 2.0V$$

$$V_{OH,TC} = V_{CC} - 0.4$$

$$V_{CC} \geq 2.4V$$

Now let's move onto the interaction between the multiplexer and computer. The multiplexer's logic levels depend on its power supply. It can have one of three power supplies: 5V single supply, 3V single supply, 2.5V dual supplies. The voltages can swing by 10% either way. Given the two restrictions above and that it will be easier to use a single supply, we'll base the multiplexer logic levels on a 3V single supply.

We only have to worry about a signal output from the computer and input to the multiplexer. The multiplexer does not output any signals to the computer. Let's first consider a logical 0:

$$V_{IL,M} \leq 0.8$$

$$V_{OL,PC} = 0V$$

This imposes no restrictions on V_{CC} . Let's consider a logical 1:

$$V_{IH,M} \geq 2.0V$$

$$V_{OH,PC} = 3.3V$$

This also imposes no restrictions on V_{CC} . The only restriction imposed by the multiplexer is due to its power source. Its source must provide a voltage within 10% of 3V:

$$V_{CC} \geq 2.7V$$

$$V_{CC} \leq 3.3V$$

There is one final restriction that must be considered. The computer's digital output voltage is 3.3V. To avoid damaging the circuit, the supply voltage of the circuit must be greater than any incoming voltages. Therefore:

$$V_{CC} \geq 3.3V$$

Now let's look at all the restrictions on V_{CC} :

$$V_{CC} \leq 4.125V$$

$$V_{CC} \geq 2.4V$$

$$V_{CC} \geq 3V$$

$$V_{CC} \leq 5.5V$$

$$V_{CC} \geq 2.7V$$

$$V_{CC} \leq 3.3V$$

$$V_{CC} \geq 3.3V$$

These six restrictions can be reduced to the following two restrictions:

$$V_{CC} \geq 3.3V$$

$$V_{CC} \leq 3.3V$$

The only solution then is to provide $V_{CC} = 3.3V$ to the circuit board.

References

- [1] Hemond, H. F., and R. Camilli. NEREUS: Engineering concept for an underwater mass spectrometer. *Trends in Analytical Chemistry* 21(8):526-533
- [2] Camili, Rich. Kemonaut: An Odyssey Class AUV Platform for the NEREUS Underwater Mass Spectrometer. 2002.
- [3] Thomson, Elizabeth. Research on arsenic and old lakes points to nitrate pollution. Tech Talk, Vol. 47, No. 4.
- [4] Newman, Paul. MOOS – Mission Oriented Operating Suite. December 2002.
- [5] Texas Instruments. μ A7800 series Positive-Voltage Regulators. May 2003.
- [6] Texas Instruments. μ A78M00 Positive-Voltage Regulators. July 2003.
- [7] Maxim Integrated Products. Thermistor-to-Digital Converter. February 2002.
- [8] Maxim Integrated Products. 3.5Ω , Single 8:1 and Dual 4:1, Low-Voltage Analog Multiplexers. March 2002.
- [9] Magellan Corporation. GPS 310 Satellite Navigator User Manual. 2000.
- [10] Technologic Systems. TS-5500 User's Manual. January 2003.
- [11] Christopher Strangio. The RS232 Standard. 1997.
- [12] Impulse Enterprise. The Impulse Technical Manual. February 2004.
- [13] Bruce Perens. Installing Debian GNU/Linux3.0 for Intel x86. December 2002.
- [14] Osamu Aoki. Debian Reference. March 2004.
- [15] Robert M. Free. ANSI/VT100 Terminal Control. 1996.