AMP

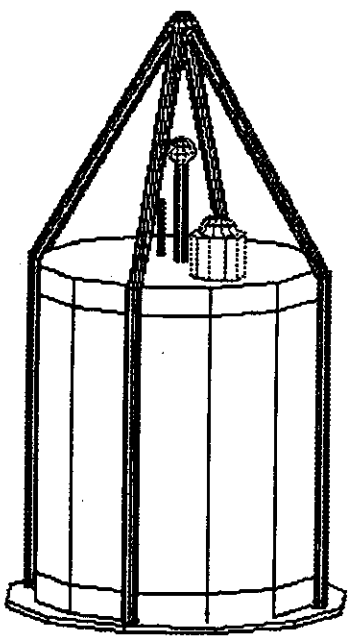
Acoustic Mapping Probe

Undergraduate Ocean Research
Tech 797

University of New Hampshire
Durham, NH

Project Advisor: Dr. Kenneth C. Baldwin

AMP Team:
Christopher N. Pacheco, ME
Jason C. Gerry, EE
James P. Inglee, EE



Sea Grant
New Hampshire/Maine

UNHMP-AR-SG-96-13

ABSTRACT

The design structure of the acoustic mapping probe (AMP) stemmed from the need to express the under water sound field from acoustic devices used in the fishing industry in three dimensions to begin to understand how the marine mammals interact with them. The system addressed this need with a measurement sampling technique involving several sensors. The design and sensor selection addressed the pertinent parameters for mapping the sound field: depth, differential global positioning system, temperature, and sound pressure level. With a unique software design created specifically for the monitoring and controlling of the sensors, AMP was capable of providing both real-time and data logged information that can be used to accurately portray the under water sound field in three dimensions. AMP's unique design features were its portability and its flexible user-friendly software.

Table of Contents

List of Figures

List of Tables

I. INTRODUCTION

The motivation for the AMP project has stemmed directly from ongoing research into the behavior of marine mammals in the presence of sound fields generated by acoustic devices used in the fishing industry. The functionality designed into AMP was driven by particular projects in the study of harbor porpoises (Phoceona phoceona) and the impact on the harbor porpoise population due to inadvertent entanglement in gill nets.

In previous research, a device was designed to be placed on the gill nets used in commercial fishing, to keep harbor porpoises both aware of and away from the gill nets. This device was designed to keep porpoises from becoming entangled in the nets that eventually drowns the porpoise. Other devices have subsequently been designed by commercial vendors and field tested in rigorous experimental conditions (Kraus etal. 1995). Questions surfaced as a result of research concerning whether or not the porpoises were responding positively to the pinging devices in such a manner as to avoid the nets, and concerning the potential for environmental noise problems and habitat exclusion. The goal of AMP was to produce a device which was capable of obtaining the sound field information produced by the

pingers, and to ultimately check for positive responses from the harbor porpoises.

AMP was designed to acquire data so the data can be used to map a given sound field in the sea within a broad frequency spectrum (currently 4kHz to 166kHz). AMP was a device designed to acquire the proper data for assessing this three dimensional sound field. These data needs included acquiring data from a hydrophone for acoustical properties at a particular location; acquiring the position of the signal using a depth transducer and a Global Positioning System; and acquiring water temperature. These data were needed to map the sound field and to provide a useful data base for future acoustic propagation modeling. All data acquired then needed to be stored for further analysis and displayed for real time "field" observations with an easy to use interface for quickly acquiring data. This system also had to be portable. This portability issue included producing a system that was manageable by one or two operators.

AMP was designed as a computer oriented instrumentation probe system which can acquire, store and display all the necessary information needed to three dimensionally plot an acoustic sound field. Along with the need to complete this project for a solution to this immediate situation, the

design team was able to design AMP with enough flexibility to be used in many other ocean acoustic applications.

II. Design Criteria and Approach

The purpose of this project was to create a functional device capable of 3-dimensionally mapping the sound field in a fluid medium. In accomplishing this task, it was necessary to address the following parameters:

I. Portability

II. Acoustic properties and characteristics
    - A. Frequency ranges
    - B. Temperature effects

III. Depth of operation
    - A. Pressure housing
    - B. Pressure transducer
        1. Cable
        2. Bio-fouling

IV. Output of data acquisition
    - A. Sound
    - B. Location
    - C. Time
    - D. Temperature
    - E. Depth
    - F. Data files
    - G. Analog/Digital conversion

To meet the project's requirements, the design team focused on the necessary measurement devices, mechanical properties, electrical properties, and software design. The governing criteria for the design of the probe was the portability issue. The issues that were first addressed were those of the proper measurement devices. The properties that were necessary to measure were those of time, temperature, depth, position in latitude and longitude, and the sound pressure level. It was then necessary to design a device capable of containing these devices along with a method of deployment and recovery. Several design alternatives were developed and evaluated for the most reliable data acquisition system. The advantages and disadvantages of each design alternative were then researched to decide on the most practical, cost effective, reliable design. These design alternatives are summarized in Table 1.

Table #1: Summary of Design Approaches

Design Alternative	Advantages	Disadvantages
I. Self Contained Unit	1. No cabling to surface 2. No equipment above the surface 3. Portable and easy to use	1. Expensive 2. No Real Time Functionality 3. Limits amount of data acquired 4. High power consumption
II. Data Transmitted Digitally	1. Real Time functionality 2. Minimize signal noise 3. Less expensive cabling	1. RS232 cable to surface 2. Reduces portability 3. High power consumption 4. Still expensive
III. Data Transmitted Analog	1. Low power consumption 2. Real time functionality	1. Signal Noise 2. Expensive cabling 3. Reduces portability

It was apparent the system would require a computer integrated program to monitor and control the instrumentation.

Because of the portability issue, the group decided to make the unit as compact as possible so its field operation was not a laborious task. This required a system that had minimal power requirements, was easily deployed and recovered, and could be operated by one person. This portability requirement governed the selection of all of the measurement devices as well as the method for data acquisition. It was decided to design a small probe to

which the necessary instrumentation was fastened. The idea was to activate the probe and lower it into the water column where it would acquire data. This data would then be analyzed back at a laboratory. The initial design was to build a self contained unit. The device contained all of the hardware necessary to acquire and store the data within the pressure vessel. This contributed to the portability criteria of the design as there was no need for a cable to transmit data to the surface. The only requirement was for a PC to initiate the profiling operation. A single person in a small boat would then have the ability to take data measurements with little difficulty.

As time progressed in AMP development, some of the project necessities changed which altered the probe's design. The portability issue remained however, there became more of an interest for real time data retrieval. The real-time functionality issue was decided on because it was of interest to the user. This design change allowed the user to monitor the data as it was recorded for its practicality and relative validity. It also allowed for more data to be collected on the surface because the data acquisition was no longer dependent on the memory in the probe. This design change resulted in the necessity for a cable to transfer the data from the probe to the user at the surface. This

reduced the amount of hardware within the probe. However, it required more equipment including another power supply on the surface. This reduced the probe's portability. The use of a cable originally resolved the low power instrumentation requirement within the probe until it was discovered that the power signal would add undesirable electrical noise to the system as it transmitted from the surface to the probe. To eliminate this noise, it was decided to place a DC power supply in the probe. The DC power supply limited the instrumentation selection to low power DC sensors.

Some of the first major issues that the group addressed were system integration and probe assembly. The original design of the acoustic mapping probe (AMP) incorporated three different hydrophones to monitor the entire frequency range of 500Hz - 180kHz. The three hydrophones monitored information for their given bandwidths. After acquiring and allocating this data in three different files, the information was stored for further data manipulation. The bandwidths of the three phones overlapped each other so it was not possible to miss any of the required bandwidth. The process of locating three hydrophones that fit within the budget constraints proved to be unsuccessful. Fortunately, this design changed when the group discovered a broad band hydrophone capable of covering a vast majority of the

required frequencies. It was decided to utilize this broad band hydrophone as this eliminated the possibility of missing data as well as reduced the number of conductors required for the cable. Reducing the number of hydrophones from three to one also simplified mounting and physical spatial problems. Finally, using only one hydrophone also reduced the power requirement.

The idea of mapping the sound field required that the probe was capable of recording its position in three dimensional Cartesian coordinates. To accomplish this requirement, the group needed to record both position in latitude and longitude as well as its height in the water column.

After researching a great deal on acoustics, the design team decided that it was important to incorporate a temperature measuring device into the system as varying temperatures greatly effect underwater acoustics. The research presented information stating some of the effects of temperature on sonar transmissions. For instance, the shadowing effects of the thermocline and the acoustic velocity variations with temperature (see Urick, Principles of Underwater Sound).

The depth issue was resolved utilizing a pressure transducer and the hydrostatic pressure equation (see Figure 1).

$$h = \frac{P}{\rho \cdot g}$$

h = height
P = pressure
ρ = density of sea water
g = gravity

Figure 1. Hydrostatic pressure equation.

Note, the pressure measurements were acquired using an approximated still water level (waves were not accounted for in the depth calculation). Issues concerning the pressure transducer consisted of pressure, sensitivity, power, size and hardware mounting. The sensitivity and depth issues for the transducer greatly effected the cost. The group found that the deeper and more sensitive the transducer, the higher its cost. The mounting issue required a transducer that could be fixed to a small probe. The power issue for the transducer was that it ran on a low voltage dc signal.

The controller issue was one of great significance to the success of the AMP design. Preliminary designs sought to implement an A/D converter inside the pressure vessel and store the data inside the probe. As the real-time requirement was developed, it was decided that the signal had to be transmitted to the surface computer for storage. However, this idea was terminated by the fact that unaided digital communication was limited to approximately 50 feet. It was then decided to transmit an amplified analog signal

to the surface. This required the location of the A/D converter to be at the surface. However, it was found that the original A/D converter was not fast enough to sample at the desired rate. This design change resulted in a faster more efficient means of data acquisition.

The cable selection was one that changed throughout the development of the probe. Originally, the only cable requirement was a mechanical strength member to lower the probe into the water column. However, after the real-time issue was addressed, the cable requirements changed. The new cable requirement was to transmit a digital signal on RS-232 cable to the surface. As the group performed the cable selection, it was discovered that the transmission along RS-232 over 500 ft of cable was impossible without the use of repeaters. The repeaters required both power sources and water sealing. The group then addressed the possibility of using fiber-optic cable. The fiber optic cable resolved the repeater dilemma however, due to the portability parameter, this idea was dropped as the fiber-optic cable required a minimum bend radius of several feet which involved a difficult deployment and recovery operation. The data acquisition portion of the design changed for its final time to a system that required the use of an analog cable.

This meant using an electro-mechanical cable for the data transmission.

The cable now required individual conductors for each sensor. The idea of multiplexing, although possible as the signals were of different frequencies, was not pursued due to its complexity. To increase noise reduction and the possibility of cross talk in the conductors, it was decided to use three twisted shielded pairs. This would provide each sensor with its own individual ground as well as individual sensor signal shielding. The mechanical aspect of the cable proved to be one of many complications. The strength member was definitely a necessity as the cable alone was not capable of supporting the weight of the probe. Ideally, it was desired to have one cable that incorporated both the required electrical characteristics as well as the mechanical requirements. This idea proved to be a costly one which forced the group to switch to an electrical cable tethered to a rope for a strength member. Finally, the group received a generous donation of electrical cable. It was then decided that the necessary strength member would be a polyethylene rope.

Time of the recorded events was not a design requirement however, the group decided that it would aid in determining

both the actual time of and the duration of each recorded event. The time record came from the internal clock of the computer.

The power supply for the system changed throughout each stage of the design. Originally, it was a small DC supply of 9 Volt batteries to power the entire compact system. Then, the option presented itself to utilize power from the surface to power the probe. This idea, as stated previously, was short lived and eventually the power supply was divided between the probe instrumentation power and the PC power. The power for the probe was supplied by a battery pack consisting of 9 Volt batteries connected in a series circuit. The PC power came from a 12 Volt battery and an AC rectifier.

III. SENSOR SPECIFICATION AND SELECTION

The sensor selection was a direct result of the final design approach of the AMP system. The specifications for the selection were set by the design criteria. The sensor selection was greatly influenced by the availability of, and salvageability of, functional hardware.

Hydrophones

The broad band hydrophone was originally salvaged from a former project. The hydrophone was performing adequately considering its physical appearance until an accident occurred in a tank test during which the hydrophone pre-amplifier combination was soaked. To resolve this problem the group looked to purchasing a new hydrophone. This accident occurred late in the development of the system which greatly limited the purchasing process. Fortunately, the group located and ordered a new hydrophone from Spartan Electronics, which is the hydrophone presently used in the AMP system. The hydrophone has a pre-amplifier for impedance matching and driving the cable. The frequency range of the hydrophone/pre-amplifier device utilized by the AMP system covers the broad band of 4 kHz - 166 kHz. The sensitivity of the device was -155 dB and +/- 3 dB across the entire range with a resonance frequency of approximately 140 kHz (see Appendix D). The pre-amplifier required a 9 Volt power supply.

Temperature

The issue of temperature measurements was addressed by implementing a thermistor into the system. Because the fluid for which the system was designed was ocean water, the

necessary temperature range varied from 0°C to 30°C. The thermistor used by the probe was salvaged from a damaged Endeco current meter. The device operates on changing resistance due to temperature changes.

Depth/Position

The depth measurement was taken using a pressure transducer and the hydrostatic pressure equation. The pressure transducer was purchased from Keller PSI. The device had a static accuracy of .25% (see Appendix E). The device consisted of a piezoelectric transducer that generated a current corresponding to a certain pressure while the crystal was displaced. The pressure transducer was powered by a 9-30 Volt power supply and had a 5 Volt DC output. The transducer was also equipped with a vent filter for water absorption. The pressure transducer's operating range was from 0 to 250 psi which was the operating range set by the design specifications.

The latitude/longitude positioning system used by AMP was a Differential Global Positioning System (ΔGPS). The ΔGPS provided accurate 2-dimensional positioning to within 5 - 10 meters of the actual site. Raw data was taken from the device and incorporated into the software of the AMP system.

IV. Data Acquisition

The method of acquiring data in the AMP system used measurement devices that were monitored by a computer. Software in the computer was used for the data storage and data displaying of the recorded information.

Software Design

The software portion of AMP presented many challenges in design and implementation. The software in AMP can be considered an information "manager." AMP's main purpose was to acquire information from the hydrophone, depth and temperature sensors as well as the GPS system, and convert this information into useable data. The software was responsible for the storing of this data as well as displaying this information to the user. AMP was designed to accept user input for the configuration of AMP's data acquiring tasks. Figure 2 presents the general processes involved in the AMP code, in an overall top- down view of the AMP functionality. The functionality can be broken into two main sections.

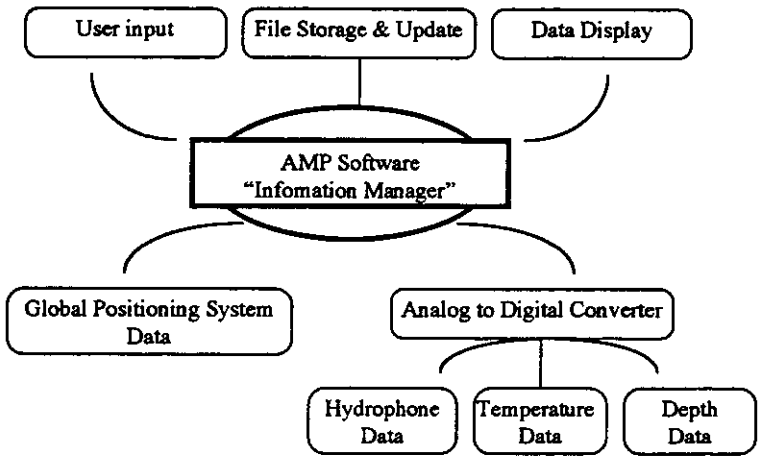


Figure 2. Top down view of AMP functionality.

The pseudo processes on the top of the diagram was manipulation of the data for use in analysis. This use, included obtaining data from the user to set up operations and specifications of the software. (User specifications are discussed in Appendix A, the AMP users guide). File storage and File update took translated data and stored the data into files for future use and analysis. The Data display gave a real-time display of the data for "in field" observations and for data checking and assurance.

The data collection portion of the software included all the operations necessary to obtain data from the external systems used. These systems included the Global Positioning System (GPS) and the sensors connected to the Analog to Digital (A/D) converter.

Acquisition

Acquisition was completed in a linear process. Temperature and depth data were acquired initially and then GPS data were obtained. These data were then filtered and translated by the AMP software for analysis. The hydrophone was then sampled. The hydrophone was handled independently due to the large amount of samples. Therefore, a large amount of data needed to be handled by AMP.

AMP was set up to handle millions of samples at a time. The ability to handle large sums of sampled data also created the need for this sampling process and data translation to take place separate to all other procedures to maximize efficiency. The actual functionality of the acquisition of data can be seen in the following pages in the flow charts provided. The main procedure which handled the acquisition functionality was the "GET DATA" function. It was here that the necessary functions were called to accomplish all of the above tasks.

Conversion

Once the necessary data were acquired the AMP software converted the data into a recognizable format. The GPS data were acquired through the Communications port of the computer and were then stripped of unwanted data. There was a large amount of excess data that were received from the GPS system, these data were unnecessary for AMP's purpose and were parsed so as to leave the latitude and longitudinal coordinates of the system. These data were then stored into character strings until there was a need for it to be accessed. The depth and temperature sampled readings at this time were converted from a sampled voltage signal to a recognizable format. Temperature was converted to Farenheight and depth was converted to meters. The voltage

voltage to temp and voltage to depth were calibrated and the specific conversion code was placed in the software.

The hydrophone data remained as voltage values. These data were used to create a sound pressure level (SPL) value. This SPL value and the independent sampled values were then stored and displayed accordingly.

User Level

The user level included all the AMP procedures that were not transparent to the user. Here the user was presented with the options to configure the AMP software. These options were generally set before any actual data Acquisition took place. Once the user configured all of the options and the acquisition process occurred, further user level procedures occurred.

After data passed through the conversion level, two main processes occurred. The first involved the real time display of data. One window displayed the latitude and longitude received from the GPS, the depth of the probe, the temperature of the water at probe depth, as well as time, and the sound pressure level of the sampled hydrophone data. The second window displayed the first 1000 samples that the hydrophone acquired. The second window display was

simply used as a check for the end user. This gave the user a view of the data the hydrophone was receiving. The group chose not to display the entire sampled data for several reasons. For one, the computing time and memory used to display the data were an inefficient use of resources, and observing the samples in a real-time fashion was not useful in any data analysis situation.

In the file storage portion of the code, two files were created. One file contained the GPS, depth, temperature, time & SPL. The other file contained solely the sampled hydrophone data. The data were stored from the user input prefix file. For example, if the input was "test", the AMP program created one file named "test.txt" which contained the GPS data etc. AMP also created a set of incrementing files starting at "test_hyd.1" and incrementing the file name for each group of hydrophone samples taken.

Code Implementation

AMP was created in the Windows ver3.1 environment. This environment was beneficial for programming. The resources existed to implement the code in Windows. It provided for an intuitive feel and use of a computer. Most importantly, the Keithley A/D board was designed to work well in this environment. The board has memory managers and drivers

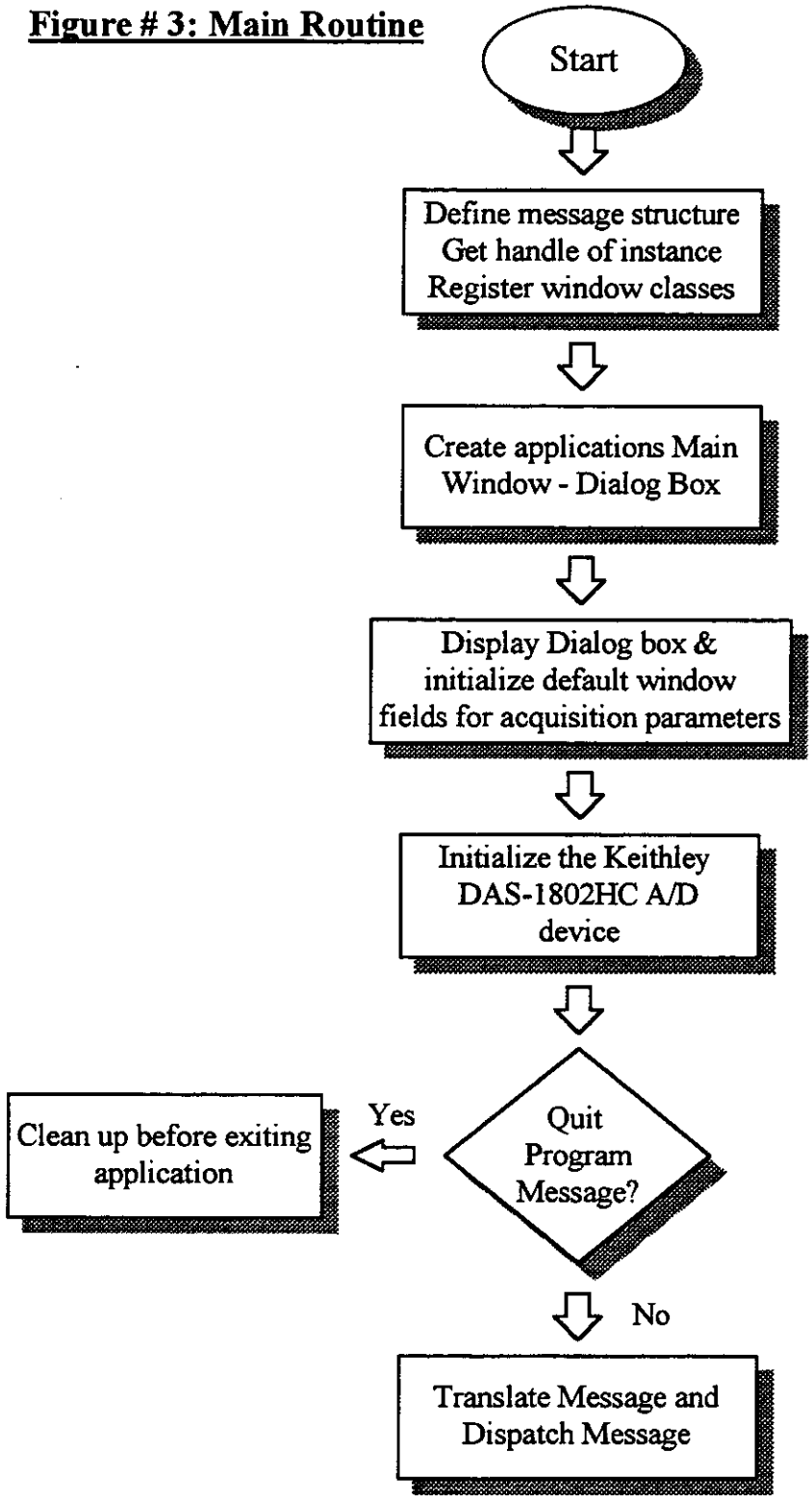
designed for the Windows environment. The functionality of the code can best be seen in the flow chart on the following pages (see Figures 3-14). These flow charts give an excellent in detail view of the procedures described above, without going into the windows coding overhead. There was a large amount of Window's background processing occurring, that the code handled, but was not necessary to show in order to get the feel for the code. (If it is necessary to view this information, see the code in Appendix B.)

General Program Flow

Throughout the design, the group visualized the general program flow of the Acoustic Mapping Probe software by using the flow charts seen in Figures 3-14. It was easier to analyze the various routines separately. This was accomplished by tracing through the program beginning with the initial execution and then following the execution of the various subroutines throughout the completion of the programs task. This helped to better understand the general program flow of the AMP software.

Upon execution of the initial stages of the software, the operating system followed the *Main Routine* procedure outlined in Figure 3. The purpose of this routine was to set up the user interface, and initialize the program such

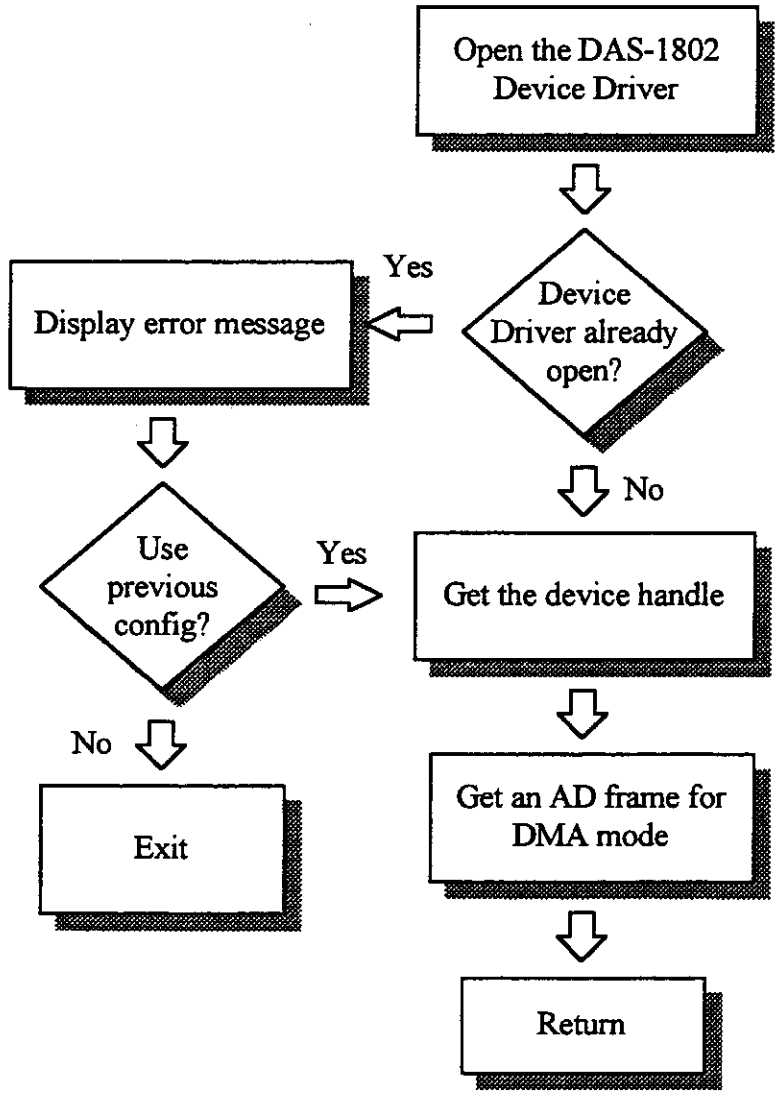
Figure # 3: Main Routine



that it integrated itself into the Windows Operating System. It started with taking care of general Window procedures which were not specific to the AMP software (any program that operates in Windows must follow these initial steps). These procedures included defining a message structure, retrieving the handle of the instance, registering the Window classes, creating the applications main window, and initializing the displayed dialog box.

The *Main routine* then initialized the Keithley DAS-1802HC A/D device by calling the routine seen in Figure 4. In this subroutine, the program loaded the DAS-1802 device driver and checked to see if the driver was already loaded. If the device driver was already loaded, then it displayed an appropriate error message to the user and asked if it was okay to use the previous configuration of the device. If the user did not wish to use the previous configuration of the device, the driver would not be loaded, and the program terminated normally. However, if the previous configuration was implemented, the program proceeded to get a handle for the device, and the A/D frame handle for the Direct Memory Access (DMA) mode operation (Refer to Keithley Manual). The DAS initialization subroutine then returned to the *Main Routine*.

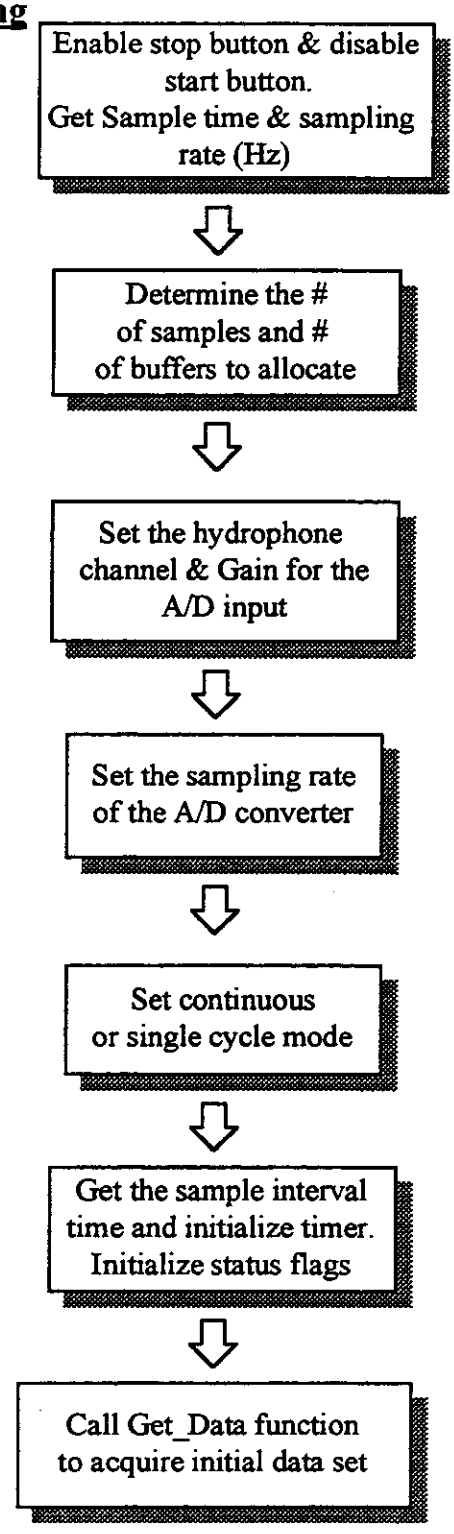
Figure #4: Initialize DAS Device:



After the initialization was complete, the *Main Routine* took care of the message handling for the program instance. All messages were translated by the operating system and dispatched to the *Main Windows Procedure*. This was seen as a series of "switch" statements in the attached code (see Appendix B). This procedure provided service routines for the Windows events as well as the user initiated events within the program. It continued to translate and dispatch commands until a "quit" message was received and the program was terminated.

The AMP software was now initiated and awaited the interaction of the user. Once the user entered the appropriate parameters to start the acquisition process (see User's Manual for details in Appendix B), the *Start Acquiring* function was initiated by clicking on the Start Button (the *Start Acquiring* subroutine can be seen in Figure 5). The purpose of this routine was to configure the A/D device with the parameters defined by the user. Initially, the *start time* was retrieved, the stop button was enabled, and the start button was disabled. This ensured that the

Figure #5: Start Acquiring



user did not attempt to begin another *Start Acquiring* subroutine until after the stop button was pressed. This would have caused an error to occur.

Next, the Sample Time Parameters - *Sample Interval*, *Sample Time* and *Sample Rate*, were retrieved from the user interface. The *Sample Interval* was used to set the value of *Timer Procedure 2* (note: this will be discussed later). The *Sample Time* and *Sample Rate* allowed the program to determine the total number of samples, and, in turn, determine the number of buffers to allocate in memory for the DMA mode operation. Allocating multiple buffers allowed the A/D device to sample continuously over the Sample Time. However, there were a few limitations, such as the maximum buffer size and the maximum number of buffers that could be allocated. These defined the limits on the user defined input for the Sample Time Parameters. (See Keithley manual for more details). The A/D channel, gain, and sampling rate for the hydrophone input were retrieved and set to the user defined parameters. This allowed the user to connect the hydrophone, thermistor, and pressure transducer to any channel, as well as to determine the gain used for each channel making the program more versatile.

The software then determined from the user interface whether the A/D mode should be set to Continuous or Single Cycle Mode. In Continuous Mode, the A/D continuously sampled the hydrophone channel. It filled the allocated memory space and then overwrote it until the user pressed the stop button. This was useful for random sampling, in which one did not want the Sample Time Parameters to have any affect. The Single Cycle Mode filled the allocated buffers and then stopped automatically when the buffer was full. This was useful for the user to set the Sample Time Parameters to take multiple timed samples at fixed intervals for an extended period of time.

At the end of the *Start Acquiring* subroutine, the *Process Timer 2 Procedure* was initialized using the *Sample Interval* parameter defined by the user. The timer placed a message on the message queue each time the interval time elapsed. The message was then handled by the *Main Window Procedure* which was instructed to call the *Process Timer #2 Procedure* seen in Figure 6. Each time the *Process Timer #2 Procedure* was called, it determined the total elapsed recording time. If the elapsed time was less than the total desired sample time, then the *Get Data* function was called to get another data set (this procedure can be seen in Figure 7). If the elapsed sample time exceeded the total desired sample time,

Figure #6: Process Timer #2 Procedure

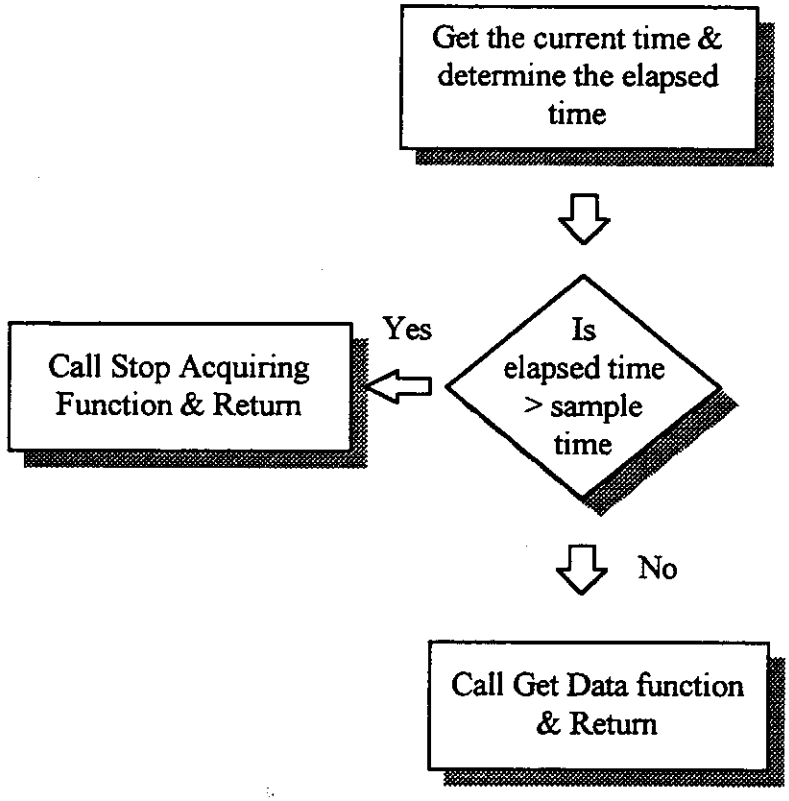
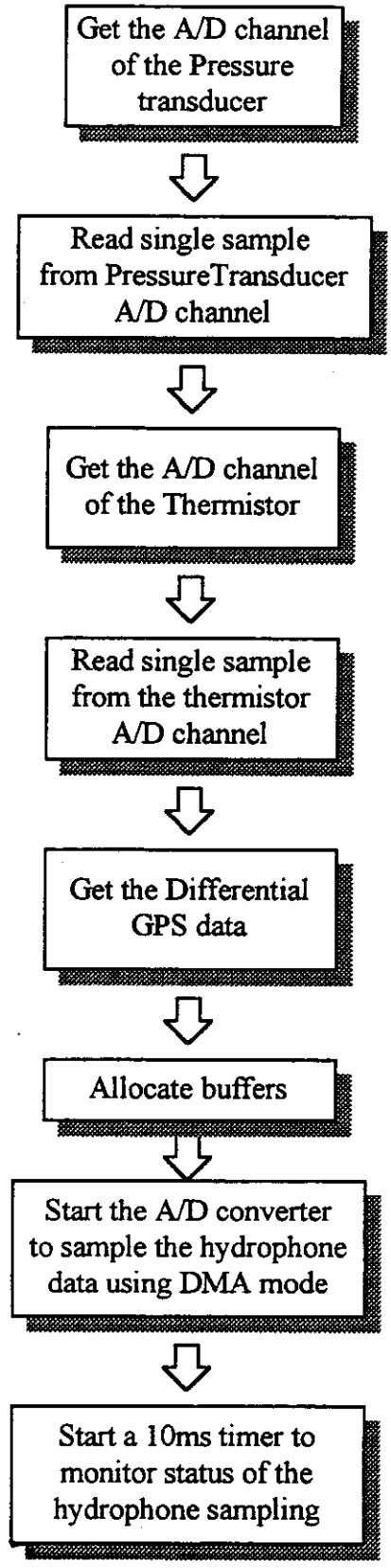
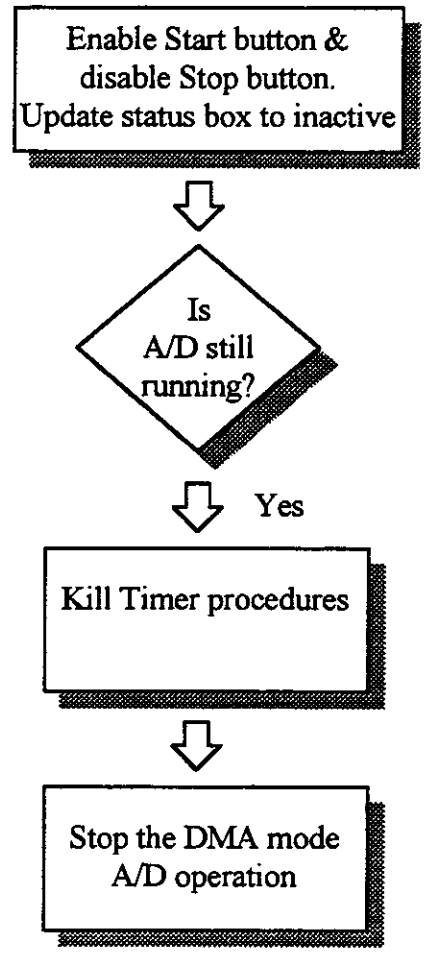


Figure #7: GetData Function



the *Stop Acquiring* function was called (see Figure 8). This function allowed the user to define a total desired sample time, and take samples at fixed intervals.

Figure #8: Stop Acquiring Function



When the *Get Data* function was called, the program proceeded to acquire another data set. Initially, the A/D channel of the pressure transducer was retrieved from the user interface. The channel was sampled once to acquire the

current pressure data from the pressure transducer which was then stored in memory. The A/D channel of the thermistor was then retrieved from the user interface. A single A/D sample then acquired the current temperature data from the thermistor channel and stored it in memory. The GPS data was then read from the COM port by calling the functions *Open Connection*, *Process COM Notification*, and *LatLong* (these can be seen in Figures 9-11). This allowed the GPS data to be read, and the appropriate data to be truncated so that the latitude and longitude were acquired.

The next step in the *Get Data* function was to allocate the appropriate number of buffers for the DMA mode operation. This was defined by taking the total number of samples and dividing it by the maximum buffer size. The buffers were then initialized with the maximum number of samples in each buffer. The remaining samples left over were allocated to the last buffer. The amount of memory allocated was dynamic, in that it only acquired the amount needed for the given number of samples. The program used the minimum amount of memory needed each time it acquired data. The buffer list on the A/D device was updated, and the frame was informed of the multiple buffers and number of samples. Now that the memory was allocated, the *Get Data* function signaled the A/D converter to begin sampling the hydrophone

Figure #9:OpenConnection:

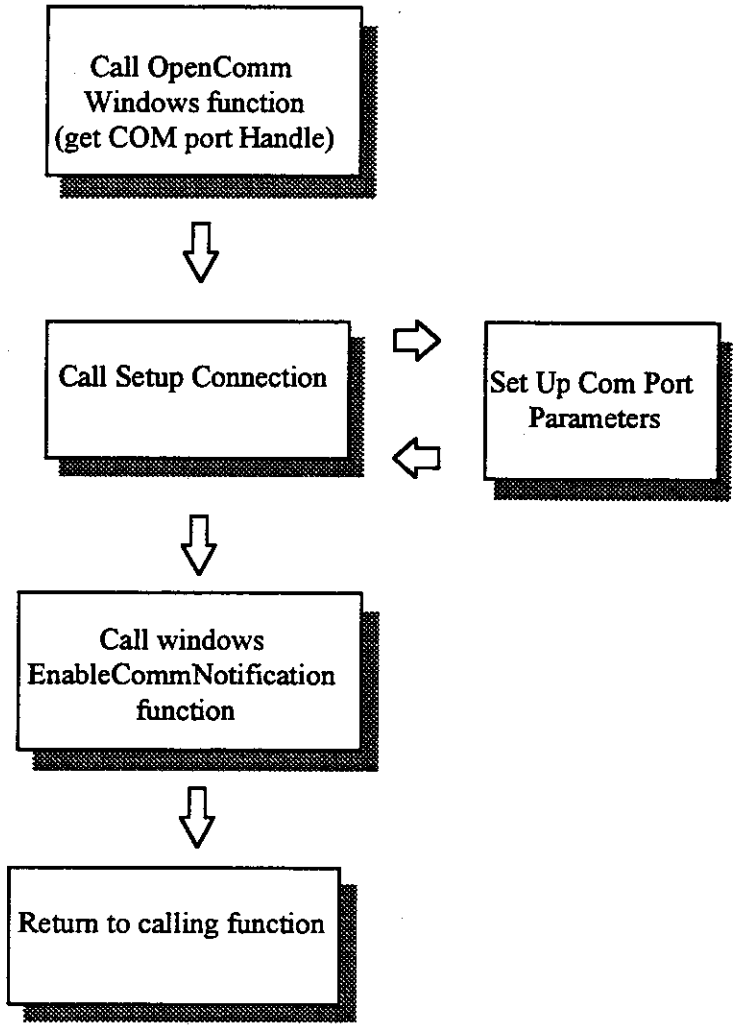


Figure #10: ProcessCommNotification:

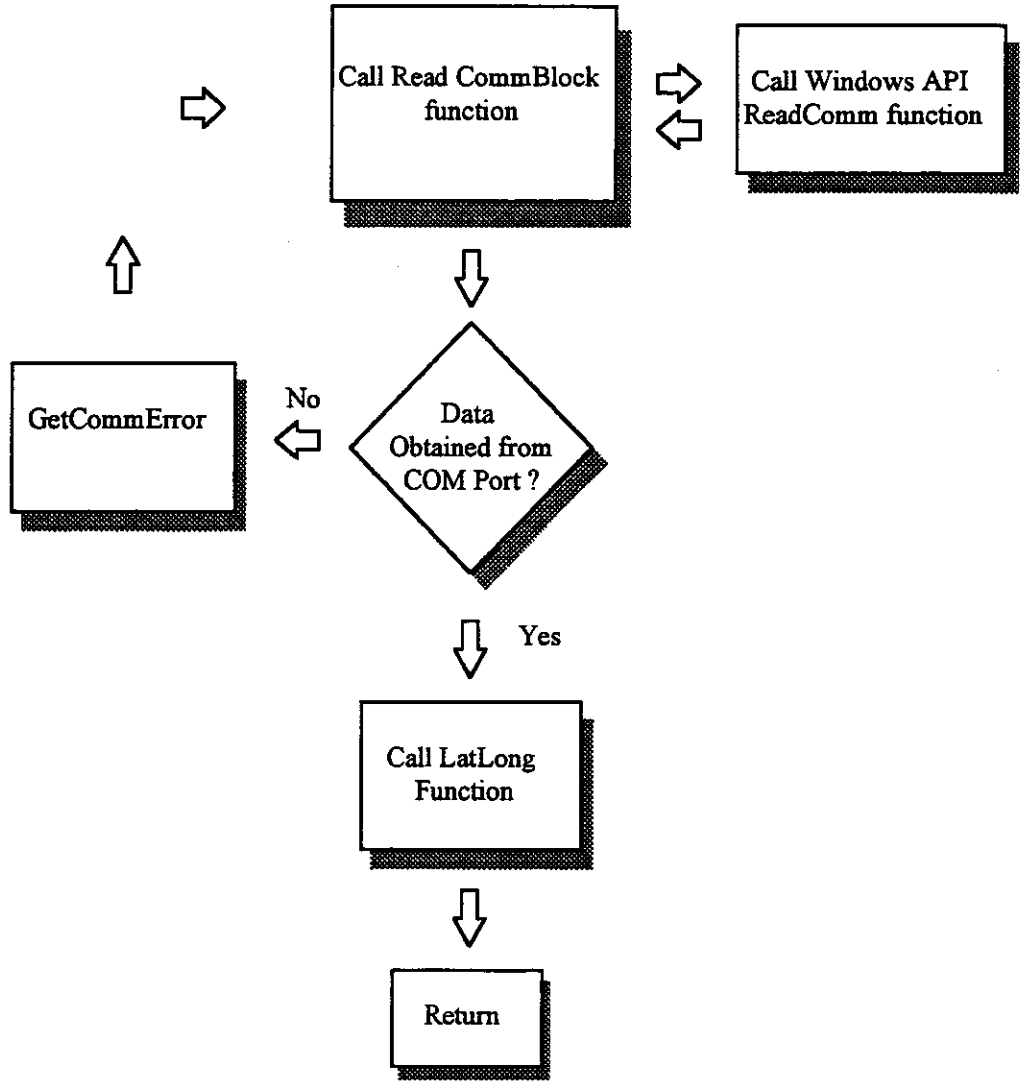
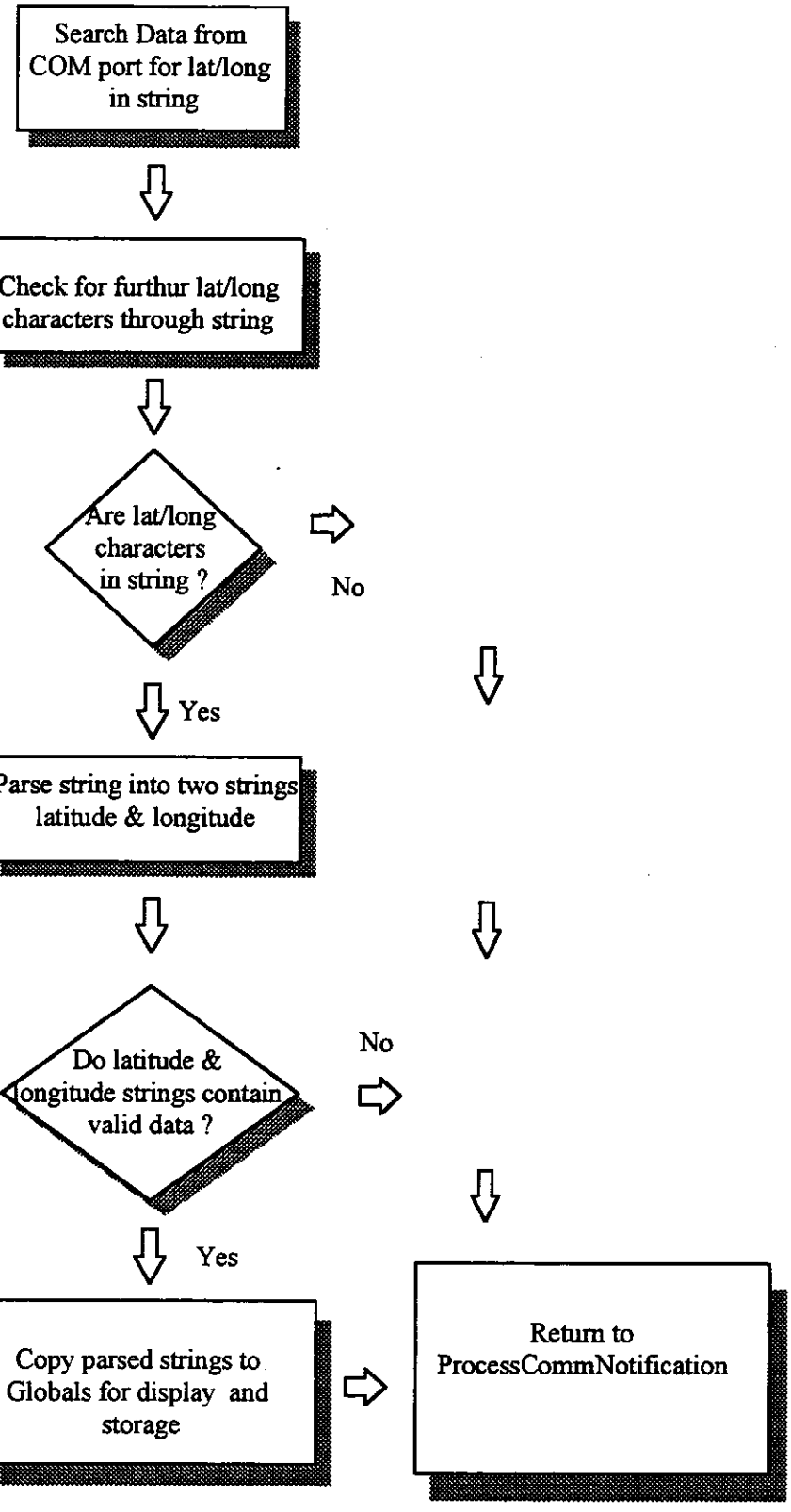


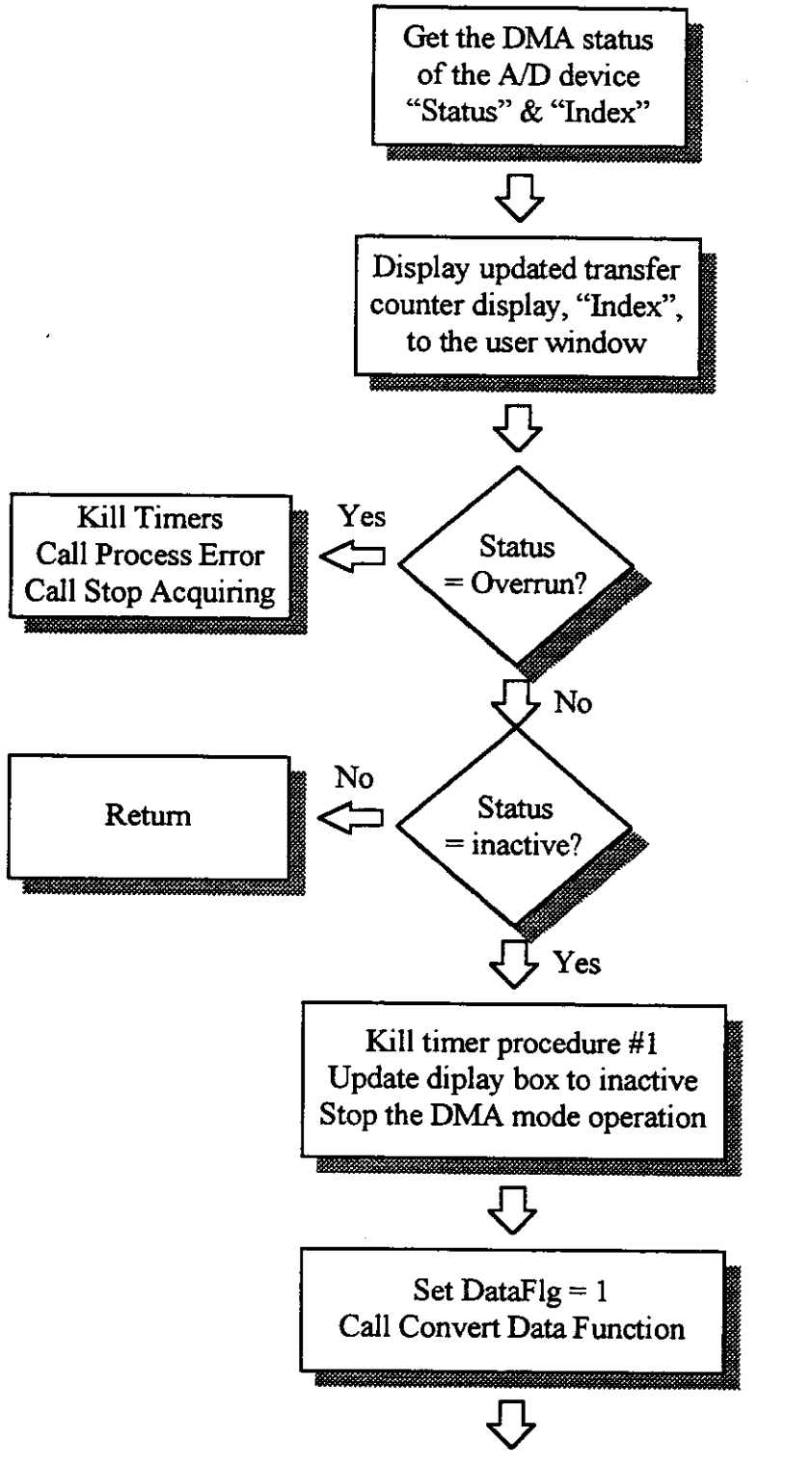
Figure #11: LatLong



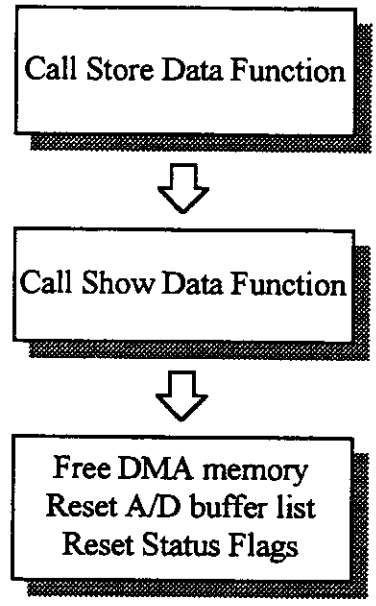
data using DMA mode. This meant that the sampled data was stored directly into the memory of the computer without interrupting the CPU. This allowed for fast, and virtually continuous sampling of hydrophone data until the frame of multiple buffers was full (depending on either continuous or single cycle mode). The number of buffers was defined by the sampling time and sampling rate. This allowed the user to acquire the desired amount of data with exception to the limitation of buffer size and maximum number of buffers.

At the end of the *Get Data* function, the *Process Timer 1 Procedure* was initiated. The timer worked the same as the previous timer by adding a message to the message queue when the cycle time elapsed. However, the subroutine that was called was directed to the *Process Timer #1 Procedure* (see Figure 12). This procedure monitored the status of the A/D device while it sampled the hydrophone data. The time set for the timer was set to update every ten milliseconds. Then every ten milliseconds the *Process Timer #1 Procedure* retrieved the DMA status of the A/D device, checked the status, and read the index (the index is the actual number of samples taken by the device). The status was checked for an overrun error as well as if the device was active or inactive. When an overrun error was encountered, the timers were reset, the *Stop Acquiring* function was called, and an

Figure #12: Process Timer #1 Procedure



Process Timer #1 Procedure (cont'd):

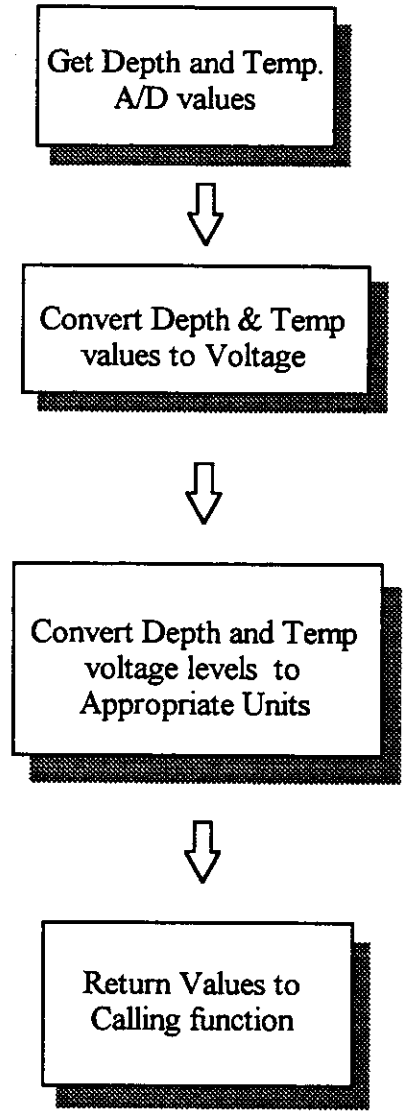


error message was displayed. Otherwise, the timer continued to check if the device was still active. If the device was active, the index was updated to the count display on the user interface and the function returned. Note that the counter incremented only to the maximum number of samples in the buffer. When one buffer was full, the device began to fill another buffer; the count started at zero and counted to the maximum buffer size. Once the device was inactive, the *Process Timer 1 Procedure* was reset and the DMA mode operation was stopped.

Now that the data was acquired, the next step was to convert and manipulate the data. At this point a Data Flag was set to exclude any other function from altering the data (i.e. the *Process Timer 2 Procedure* from trying to get another set of data). This included converting the pressure transducer data to a depth, the thermistor data to a temperature, and calculating the sound pressure level from the hydrophone data. This was completed in the *Data Translation* function seen in Figure 13.

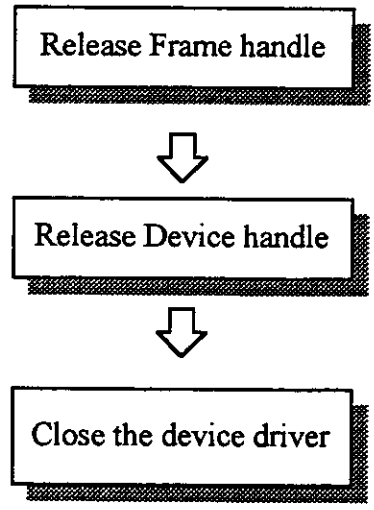
The next step was to store the acquired data on disk. By doing this, the program created several files. The hydrophone data were stored to a separate file for future analysis. After the depth, temperature, and sound pressure level were converted, the values were placed in a string and stored in another file. For each set of acquired data, a hydrophone data file was created and a second file was updated with the temperature, depth, time, latitude, longitude, sound pressure level and the filename of the hydrophone data. After the data were stored the user interface was updated. The allocated memory was freed and the A/D buffer list and status flags were reset. The program then waited for the *Get Data* function to be called by the *Process Timer 2 Procedure*, or for the user to terminate the program by pressing the stop button.

Figure #13: DataTranslation:



The *Stop Acquiring* Function, seen in Figure 8, was called when either the user clicked on the stop button, or the total sample time defined by the user elapsed. This function enabled the start button, disabled the stop button, and updated the status box to inactive. It then checked the A/D device to see if it was still active. When the device was not active the function returned from the subroutine. When the device was active, both *Process Timer 1* and *Process Timer 2* procedures were cleared and the DMA mode A/D operation was stopped. It then reset the status flags and returned. This allowed the user to stop the data acquisition, and reconfigure the acquisition parameters to their original state. The user could then restart or close and exit the program. If the user decided to exit the program, the *Release Driver* Function was called and the program terminated. The *Release Driver Function* can be seen in Figure 14. If the user started the program again, the program continued to acquire data until the user finished taking data.

Figure #14: Release Driver



Signal Conditioning

The device signals required some initial amplification to drive the long cable length. Each signal was amplified through a separate amplifier, and then transmitted along the corresponding twisted shielded pair within the cable. The signal was then received at the opposite end and connected to the A/D converter input, where it was then sampled. The amplifier specifications varied for different signals. For the thermistor and pressure transducer, the signals were virtually DC level signals. Thus the amplifier did not require much in the way of bandwidth. However, the hydrophone signal was conditioned using a pre-amplifier which was purchased with the hydrophone from Spartan

Electronics. This amplifier was specifically designed for AMP due to the sensitivity of the hydrophone and the characteristics of the cable. This was to ensure that the full range of the hydrophone was utilized. The amplifier schematic can be seen in Appendix D along with the hydrophone specifications.

Hardware

Several hardware components were necessary to implement the data acquisition aspect of the project. These included LM741 operational amplifiers used as a base in amplifying the signals created by each device. The amplifiers were designed such that the output impedance of the amplifier matched the impedance of the communication cable. This was done to minimize the reflection coefficients in the signal, as well as to reduce the attenuation and corruption of the data signal. At the receiving end of the cable, it was necessary to ensure that the input impedance to the A/D converter matched that of the communication cable.

Another hardware device which played an important role in the data acquisition was the A/D converter. By the Nyquist Theorem, it was necessary for the A/D converter to be able to sample at least twice the maximum frequency of the sampled data signal. In this case, it was twice the maximum

frequency of the hydrophone signal which was approximately 170 kHz. This required the A/D converter to sample at 340 kHz. Initially, it was planned to use the Motorola MC68HC11 Evaluation Board (EVBU) to sample the data. The EVBU was equipped with an on board A/D converter which was capable of sampling up to four different inputs. It could be programmed using microcode for specific applications and was also fairly inexpensive. However, it was found that the A/D converter on the 68HC11 was not fast enough to sample at the desired sampling rate. Thus, the move from the 68HC11 to the Keithley Metrabyte-18002HC was a necessary design change for the creation of a faster more efficient means of data acquisition.

The Keithley Data Acquisition System was a high performance A/D board for the IBM PC and compatible computers. The DAS-1802HC featured 64 single ended inputs or 32 differential inputs, as well as continuous, high speed, gap-free data acquisition under Windows programming environments. The onboard FIFO(first-in-first-out) buffer and dual channel Direct Memory Addressing (DMA) mode allowed for continuous acquisition of data at a maximum sampling rate of 333k Samples/second (see Keithley manual for more details).

This benefited the AMP system in many ways. The utilization of DMA allowed the system to sample a much larger and continuous time frame segment before storing or displaying the data. It also allowed the data to be stored in CPU memory without interrupting the CPU, and therefore acquisition throughput was virtually unaffected by program flow. Note however, the program flow did effect the amount of continuous acquisition time between samples. The continuous acquisition time between samples was the speed at which the program stored the data and updated the display. This effected the minimum sample interval and therefore, the continuity of the acquired data. Once the allocated buffers were full, the A/D could not sample data again until the previously sampled data were manipulated by the program. This effected the Real-time design aspects of the software.

The use of the Keithley Data Acquisition System also provided an easier way to integrate the Differential Global Positioning System (ΔGPS) into the AMP system, due to the newly vacant serial port previously needed by the MC68HC11 EVBU. The GPS was connected directly to the PC COM port, and read by the AMP software to acquire the necessary latitude and longitude data. Therefore, the hardware of the AMP data acquisition system was fully integrated within the PC, which made it compact and easy to use.

V. INSTRUMENT HOUSING

The instrumentation was mounted to a pressure housing for deployment. The housing contained all of the necessary parameters to meet the requirements of the sensors and ultimately the probe's design.

Pressure Vessel

The pressure housing was borrowed from a former instrumentation project. It was decided that only new endcaps to accommodate the sensors and a new internal pc board & battery pack were all that were needed to allow for the pressure vessel to be interchangeable. End caps were machined for this final housing in the UNH machine shop. All of the measurement devices were mounted to the pressure vessel via the proper through hull penetrations. The pressure vessel was designed to mount the three sensors and the cable connection as well as contain in a water tight compartment the controlling devices for each sensor, a power supply, and several signal amplifiers. The pressure vessel was designed to the required depth of 500 ft corresponding to a pressure of approximately 250 psi. The pressure vessel was fitted with clamps to access the power supply and perform modifications to the measurement devices as needed.

The material used for the housing was Aluminum as it was the least costly material to use. Aluminum was also the least corrosive material considering the marine environment for which AMP was designed. For the complete pressure vessel analysis see Appendix C. For a design layout see Figure 15.

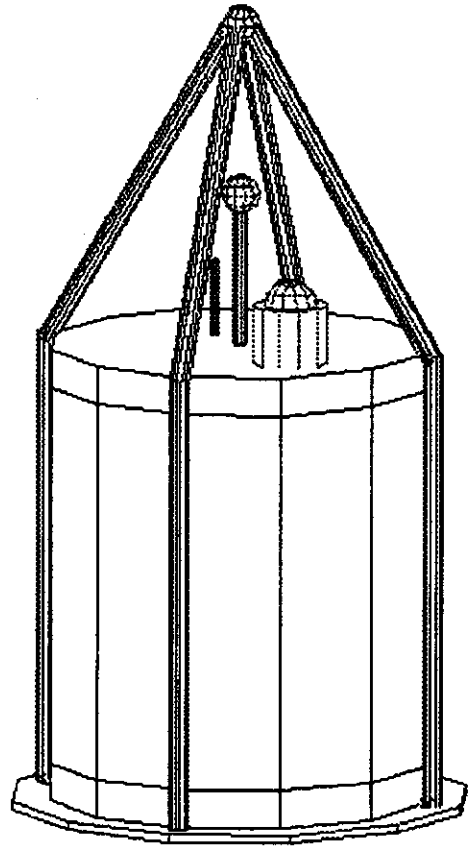


Figure 15 AMP housing and cage.

Sensor Location

The issue of mounting the hydrophone was one of serious concern. Due to the acoustic properties of the omni-directional hydrophone element, it was necessary to mount the hydrophone as far as possible from the housing so as to minimize reflections. This was done by mounting and potting the hydrophone to an eight inch rod which protruded from the pressure vessel. The pressure transducer and thermistor were fixed with tapered pipe threads and mounted to the housing using Teflon tape.

The pressure housing was encased in a cage to protect it from damage that could occur both in transit and insitu, i.e. fish bite. The cage consists of four 5/16" rods formed into an apex at one end (see Figure 15). The cage was also fitted with four nuts for assembly/disassembly in order to provide access to the pressure vessel and ultimately the instrumentation.

Cable

The measured signals were transferred to the surface via an electro-mechanical cable. The electrical cable consisted of three twisted shielded pairs of 22 gage conductors. The 500 foot cable was terminated with water proof six pin cable connectors supplied by D.G. O'Brien. The impedance and

capacitance of the cable were 50Ω / 1000 feet and 30pf/ foot respectively. The mechanical aspects of the cable included a low-stretch double braided, torque balanced polyethylene rope. The electrical cable was tethered to the rope via a plastic adhesive. Markings were placed every five feet for a manual depth reference. Five inches of cable between each attachment point was provided for initial cable stretch.

VI. BUDGET CONSIDERATIONS

The AMP project throughout its duration had accessible to it an operating budget of $2,500. This budget acquired graciously through Sea-Grant project, provided the design team with a means to design and build the AMP system. Due to the complexity of the project and costs of many of the important features in the project, the cost of AMP was actually considerably greater than what the budget allocated (refer to Table 2). The group was fortunate to be assisted by a few key businesses and organizations to help defer the cost of the project.

Item	Approximate Cost	AMP's Cost
Hydrophone	$1200	$1200
Pressure Transducer	$600	$600
Torque Balanced Rope	$250	$250
Misc. Hardware	$100	$100
Misc. Group resources	$100	$100
Probe Cage	$30	$30
Ocean Spec. Cable Conn.	$1500	$0
Cable	$500	$0
A/D Converter	$2900	$0*
Differential GPS	$600	$0*
Pressure Vessel	$200	$0*
Thermistor	$50	$0
Computer	N/A	$0*
Total	$8030	$2280

Table 2 - AMP Cost Analysis

(* = resources available through the Ocean Engineering Department)

The group was able to acquire much of the needed equipment for free or for a very inexpensive cost from a few different resources. The group acquired 1000' of Olflex Cable from Heilind Electronics for no cost. This saved AMP upwards of $500. The group was also able to acquire free underwater connections for this cable as well as the testing of the cable from D.G. O'Brien, an under-sea connection specialist. This saved amp approximately $1500. The team also obtained the appropriate Analog to Digital converter from the Ocean Engineering Department for the project. The board alone cost approximately $2900 which would have consumed the

entire operating budget, (note the other * components which the group acquired from the Ocean Engineering Dept.)

Much of the hardware was rejuvenated from previous projects. The pressure vessel was used on an earlier project and converted quite easily to suit AMP's needs. This was a savings of $200. The temperature probe and the Differential GPS system were also obtained from the Ocean Engineering Department at no cost.

As one can see much of the $2,500 was placed in only a few needed resources. The hydrophone system alone was purchased for $1,200 which was 48% of the budget. The purchase of the pressure transducer was also costly at $600, 24% of the budget . This consumed a considerable portion of the budget with the remainder going to various hardware and accessories.

VII. TESTING AND EVALUATION

Cable: The electrical cable was tested at D.G. Obrien. For the results see Appendix F.

Pressure Sensor: The pressure transducer was tested at Keller PSI. For test results see Appendix E.

Thermistor: The thermistor was tested by varying the temperature that the thermistor was exposed to and monitoring the change in electrical output.

Pressure Vessel: The pressure vessel was tested on two separate occasions. The first test to a depth of 200 feet and 15 minutes of exposure resulted in a great deal of leaking. It was believed that the vessel leaked due to a machining problem which was resolved for further testing. The second test to the same depth and similar duration proved successful with no water leaks.

SOFTWARE: Software testing of the acoustic mapping probe was an on going process throughout the project. The programming evolved using a step by step building process in which each function developed was tested before proceeding. This was to ensure that the overall functionality was correct, and the software functioned as designed.

1. The software interaction with the Keithley A/D device was tested and worked with the given Keithley library functions. The device was configured by the user and functioned according to these configurations. This included taking a single sample of thermistor, and

pressure transducer channels as well as continuously storing the hydrophone data into CPU memory via DMA mode.

2. The software interface with the differential GPS was tested and functions as designed. The latitude and longitude were obtained and then updated to the screen successfully.
3. The software user interface was also tested. The user was able to input the desired sampling time requirements, and the program operated successfully utilizing these parameters.
4. The data validation of the program was also tested. A voltage was input into the A/D port and the program displayed the correct voltage in the display. Thus, the data being received was valid. The acquired data was converted, updated to the screen, and stored to the appropriate files successfully.

VIII. Summary

The AMP system was developed to aid in the necessity to map the sound field in the harbor porpoise environment. AMP was developed in a portable, user friendly, cost effective fashion. The system, when operated, provided the user with the necessary information to accomplish the sound field mapping task. The user friendly software of the AMP system provided the user with a file containing lat/long, depth, temperature, and sound pressure level measurements at a particular time for a given location. This was the necessary information for the proper mapping of the under water sound field.

The AMP system included the probe, electro-mechanical cable, a PC and PC power supply. These system components when assembled allowed for operation including deployment, data logging, and recovery by one user.

The software was written in a style that provides future programmers with the options for relatively simple coding advancements. The pressure vessel used was also designed with room for future expansion.

IX. References

Kraus, Scott, A. Reed, Kenneth C. Baldwin, E. Anderson,A. Solom, T. Spradlin, J. Williamson. "A Field Test of the Use of Acoustic Alarms to Reduce Incidental Mortality of Harbor Porpoise in Gill Nets." April, 1995.

Urick, Robert, "Principles of Underwater Sound." 3rd edition, 1983, McGraw-Hill.

Keithly Data Acquisition, "DAS-1800 Series Function Call Driver User's Guide." 1994.

Version 1.0

AMP Software User's Guide

WELCOME TO AMP

AMP SOFTWARE CAPABILITIES

1. Acquire & Store Sampled data at various sampling rates
2. Update Global Position of probe
3. Acquire Depth & Temperature levels from probe
4. Store and Display Acquired Data
5. Compute Sound Pressure Levels from Sampled Hydrophone Data

AMP is a data acquisition program for under sea acoustic data acquisition. The AMP team has attempted to create a software package that allows for a large amount of flexibility in data acquisition as well as an easy to use interface to aid in the necessary tasks associated with data acquisition.

AMP can be considered an information "manager." AMP, applying user settings, performs tasks required to acquire information necessary for data analysis (See figure A).

Configuring the Software

Due to AMP's functionality and its dependency on proper running external equipment, it is important to correctly connect and setup the Global Positioning System, A/D, and probe systems. . For further detailed information be sure to review the Keithly A/D and Magnavox GPS manuals. For further probe system setup, refer to AMP project report.

Configuring the software for your data acquisition session involves only a few simple steps. The first step before loading the program is to decide upon the amount of data to sample and the Sample Rate to sample upon, (note: currently the sample rate will have to be the maximum of 333333 kHz, this is due to the lack of a low pass filter currently in the design, sampling below this rate without a low pass filter will result in aliasing. Refer to any signal processing book for further information.) Once these values are chosen, select the AMP icon to run the program. The main window should appear see fig "AMP Main Window."

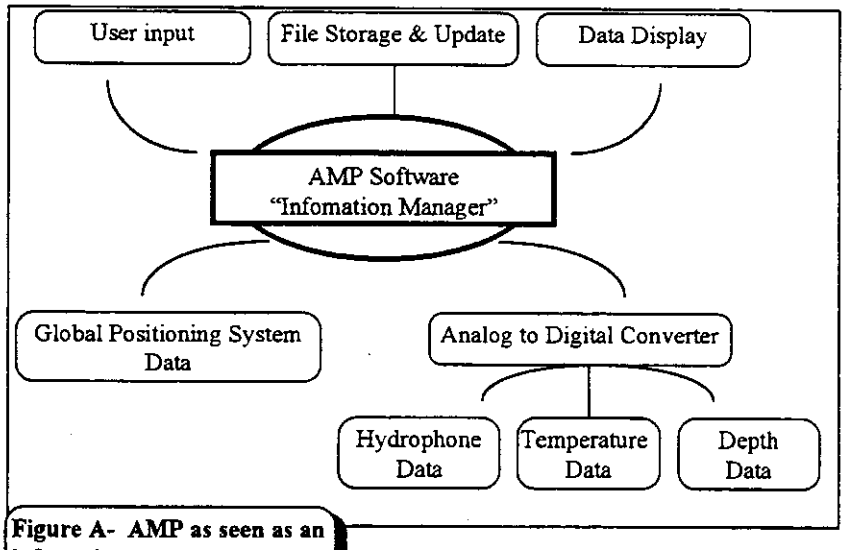
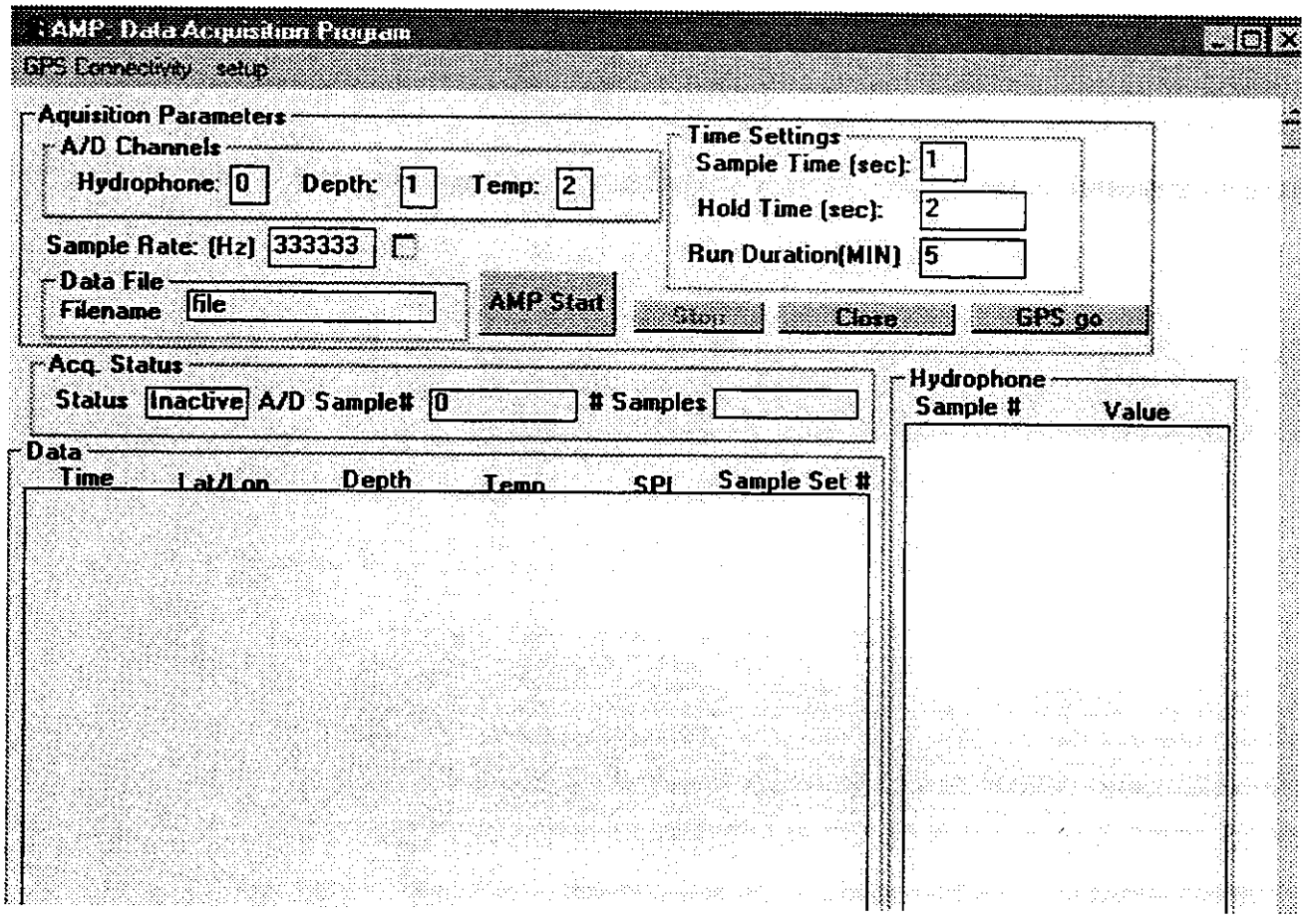


Figure A- AMP as seen as an infomation manager



The top left of the main screen contains the A/D channel configuration. The channels default to 0,1,2 for hydrophone, depth and temperature respectively. If there is a need to relocate the connections to these sensors on the jumper board do so before running AMP and then change values in AMP accordingly. The channels can be seen on the jumper board.

Below "A/D Channels" is the sampling rate. As mentioned, this rate will default to 333,333 kHz, and should remain there unless a low pass filter is used on the hydrophone connector.

The "Data File" box is where the prefix you will be using to store the data is entered. The prefix is the name of the file you will be storing to without the three character extension associated with most DOS files. When AMP is started, the file name you give here will be used to store one file which contains the Time, Latitude, Longitude, Depth, Temperature and Sound Pressure Level. This file will be created and named with your prefix and the extension ".txt" (in our case the file stored would be "file.txt") The other files created is a group of files containing hydrophone data.

The hydrophone data is sampled and each grouping of data is stored into an individual file. i.e. the first sample group of hydrophone data acquired will be stored in "file_hyd.1" The next in "file_hyd.2" , "file_hyd.3" etc. This will allow more flexibility and better analysis of the data. It also gives us a good way to break up files since they can become quite large (files may reach upwards of 2.5 Megabytes !).

Once a File name is selected, the time settings also must be tailored. These settings are crucial for proper acquisition. The first value to set is the Sample Time. This value in seconds is the amount of time the A/D board will take samples from the hydrophone, currently this can be anywhere from 1 to 7 seconds (7 seconds is approximately 2.3 Million samples @ 333,333 Hz sample rate !)

The next value "Hold Time" is the value in seconds between samples. Note this value is actually not permanent. Due to variations in CPU speed and drive access speed, if a value you enter is too low the software automatically replaces your value with the minimum possible time between samples. The extra time may be needed for GPS data acquisition of storing samples to disk, or a multitude of other operations occurring (Note this doesn't affect sample rate)

The Run duration is the amount of time that you wish the program to acquire data. This is the overall time of the sampling session. Once all of the above options are set, it is then time to setup the GPS for use.

The GPS has only two options both located in the GPS Connectivity menu. The first configuration is the "COM" port selection of the GPS. The options "COM1" and "COM2" appear, the default is "COM2." Once you select the "COM" port simply re-enter the menu and Select "Connect GPS". The GPS is now connected and ready for satellite reading (refer to Magnavox GPS manual for further information) . You do not have to use the GPS in your acquisition program if you do not need this added information. If you do not wish to run the GPS in your data session, do not follow above steps.

You are now ready to start the program. Click on AMP Start and the program will automatically start acquiring all the data you have requested. You can check and verify data in the "Acq. Status" area. The status of the A/D board will be displayed in the Status window as either *Active* or *Inactive*, the "A/D Sample#" window is updated with the number of samples received so far, and the "#Samples" box displays how many samples are being acquired by the A/D board from the hydrophone. Below this window is where data from the sensors are displayed and two the right in the Hydrophone window is where the first 1000 of each hydrophone sample group is displayed for signal integrity.

To stop AMP you can wait for the Run Duration to time out or you can stop the program manually by clicking on the stop button.

To exit Simply Click on the "Close" button. This will exit you from AMP.

If you have any further questions configuring AMP please refer to the AMP project report for further detailed explanation of use.

Appendix B

AMP Program: C Code

```
/*************************************************************/
/*      AMP CODE                                             */
/*      Universtiy Of New Hampshire                          */
/*      Tech 797 Ocean Projects                              */
/*      AMP - Acoustic Mapping Probe                         */
/*      Project Advisor:  Dr. Kenneth Baldwin                */
/*      Team Members:          Chris Pacheco,  ME            */
/*                                  Jason Gerry,   EE        */
/*                                  James Inglee,  EE        */
/*                                                           */
/*************************************************************/

#include <io.h>

#include <fcntl.h>
#include <sys\types.h>
#include <sys\stat.h>


#include "gps.h"
#include <time.h>
#include "dasdecl.h"
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include "Amp1.h"
#include "Amp2.h"
#include "resource.h"

#define UsedStringSize  70

// Global Variables Below used in GPS Acqusition, Manipulation and Display

int             COMMID;                         // Value for com device opened for GPs
int             GPS_FLAG=0;         // flag to determine if GPS will be used
int             COMMPORT_FLAG=0;    // flag to determine if COM port has been selected
char            which_comm[4];      // string for com port selcted
int             NOTIFY_FLAG=1;      // flag to determine data correctly parsed
char            WEST[10];           // GPS long coord
char            NORTH[9];           // GPS latitude coord
char far *north;            // pointers to GPS data strings
char far *west;

// Global Variables Below used in A/D board initialization & acquisition

DWORD       hDrv1800;           // Driver Handle
DDH    hDev1800;            // Device Handle
FRAMEH      hAD;                // Frame Handle
void            *pDMABuf[125];      // Pointer for Buffers used in DMA
WORD            hMem[125];          // Handle of the  DMA pointer
char            NumberOfBoards;     // Number of boards to configure
long            NumSamp;            // number of samples to take
int                 NumBuf=0;           // number of buffers used
long int  holder;           // used in allocation of buffers
```

```
long            BufCount;          // count of buffers used

// Various Globals involving timers, files, display strings, and notify flags

char far szTimeString[10]={0};  // time holder
short           ADOP = 0;          // Mode Flag
short           Done  = 0;         // Done Flag
int             DataFlg = 0;       // flag to keep program out of critica data
short           Status;            // Status variable
short           Err;               // Return value from the functions
char            Temp[80] = {0};    // temporary gloabal array
char            szErr[20];         // Error display buffer
unsigned long Index;               // Index variable for A/D
int             nIsIcon=0 ;        // windows iconize flag
long            InitTime;          // initialize time

float tempvolt;
float presvolt;
char  vtemp[5];
char  vpres[5];

// unknown variables
short           DevOpen;
short     nFocus;

// Beginning of Code

int PASCAL WinMain(HANDLE hInstance, HANDLE hPrevInstance, LPSTR lpszCmdLine, int
nCmdShow)
{
/*********************************************************************/
/* HANDLE hInstance;     handle for this instance                     */
/* HANDLE hPrevInstance;  handle for possible previous instances     */
/* LPSTR  lpszCmdLine;    long pointer to exec command line          */
/* int    nCmdShow;      Show code for main window display           */
/*********************************************************************/

MSG     msg;        /* MSG structure to store your messages */
int     nRc;        /* return value from Register Classes   */

strcpy(szAppName, "CWEX1");
hInst = hInstance;



if(!hPrevInstance)
 {
   /* register window classes if first instance of application  */
   if ((nRc = nCwRegisterClasses()) == -1)
   {
    /* registering one of the windows failed                */
    LoadString(hInst, IDS_ERR_REGISTER_CLASS, szString, sizeof(szString));
    MessageBox(NULL, szString, NULL, MB_ICONEXCLAMATION);
    return nRc;
```

```
    }
  }

/* create application's Main window, actually a dialog box!      */
hWndMain = CreateDialog( hInstance, szAppName, 0 , NULL) ;

//get handle to window


/* display dialog box*/
ShowWindow(hWndMain, nCmdShow);

/* setup default fields for acquisition parms, etc.          */
InitWindowFields(hWndMain) ;

/* initialize the DAS device */
InitDASDevice() ;

while(GetMessage(&msg, NULL, 0, 0))    /* Until WM_QUIT message */
  {
   TranslateMessage(&msg);
   DispatchMessage(&msg);
  }

/* Do clean up before exiting from the application           */
CwUnRegisterClasses();
return msg.wParam;

} /*  End of WinMain                                  */

/*********************************************************************/
/*                                                                   */
/* Main Window Procedure                                             */
/*                                                                   */
/* This procedure provides service routines for the Windows events   */
/* (messages) that Windows sends to the window, as well as the user  */
/* initiated events (messages) that are generated when the user      */
/*selects the action bar and pulldown menu controls or the corresponding */
/* keyboard accelerators.                                            */
/*                                                                   */
/*********************************************************************/

LONG FAR PASCAL WndProc(HWND hWnd, UINT Message, UINT wParam, LONG lParam)
{
 int      nRc=0;          /* return code               */

 switch (Message)
  {
   case WM_COMMAND:
    {
      // this is a notification that a menuitem has been selected
      // or a button has been pressed
        switch (wParam)
```

```
{
case StartBtn:
            // start button pressed
    StartAcquiring( hWnd) ;
    break;

case StopBtn:           // stop button pressed
    StopAcquiring( hWnd) ;
    break;

case CloseBtn:          // close button pressed
case IDCANCEL:
    StopAcquiring( hWnd) ;
    ReleaseDriver();
    PostQuitMessage(0);         // quit application
    break;

case COMM_1:
        COMMPORT_FLAG=1;

        strcpy(which_comm, "COM1" );
        break;

case COMM_2:
        COMMPORT_FLAG=1;

   strcpy(which_comm, "COM2" );
        break;


case gps_connect:
    CreateGPSInfo(hWnd);
    GPS_FLAG=1;
    if(!(COMMPORT_FLAG)){
    MessageBox( GetFocus(), "Comm Port Not Selected, Please Select from Menu",
        "GPS SETUP",MB_OK ) ;

     break;
     }
        OpenConnection( hWnd);
        break;

case gps_disconnect:
    GPS_FLAG=0;
        CloseGps( hWnd);
        break;

case gpsgo:
ProcessCOMMNotification( hWnd, (WORD) wParam, (LONG) lParam ) ;
    break;

}
return (0);
}
```

```
    case WM_TIMER:
        {
          switch(wParam)
          {
         case ID_TIMER1:

          ProcessTimer1(hWnd);
          break;

         case ID_TIMER2:

         ProcessTimer2(hWnd);
         break;
         }

        }
    case WM_SIZE:
       if (wParam == SIZE_MINIMIZED)
          {                    // app is being iconized
          nIsIcon = 1 ;
          if (ADOP && !Done)
            SetWindowText(hWnd, "DAS-1800 [Active]") ;
          else
            SetWindowText(hWnd, "DAS-1800 [Inactive]") ;
          }
       else                    // app is being restored
          {
          nIsIcon = 0 ;
          SetWindowText(hWnd, "AMP: Data Acquisition Program") ;
          }
       break;

    case WM_CLOSE:  /* close the window                    */
       StopAcquiring( hWnd) ;
       ReleaseDriver();
       DestroyWindow(hWnd);
       if (hWnd == hWndMain)
        PostQuitMessage(0);  /* Quit the application       */
       break;


    default:
    /* For any message for which you don't specifically provide a  */
    /* service routine, you should return the message to Windows   */
    /* for default message processing.                     */
       return DefWindowProc(hWnd, Message, wParam, lParam);
   }
  return 0L;
}    /* End of WndProc                          */
```

```
//****************************************
// CloseGps Function
// note currently no error checking
//****************************************

void CloseGps( HWND hWnd)
        {
//      CloseComm(idComDev);
        }

/*************************************************************************/
/*                                                                       */
/* nCwRegisterClasses Function                                           */
/*                                                                       */
/* The following function registers all the classes of all the windows  associated   */
/* with this application. The function returns an error code  if unsuccessful,        */
/* otherwise it returns 0.                                               */
/*                                                                       */
/*************************************************************************/

int nCwRegisterClasses(void)
{
 WNDCLASS  wndclass;   /* struct to define a window class          */
 memset(&wndclass, 0x00, sizeof(WNDCLASS));


 /* load WNDCLASS with window's characteristics                  */
 wndclass.style = CS_HREDRAW | CS_VREDRAW | CS_BYTEALIGNWINDOW;
 wndclass.lpfnWndProc = WndProc;
 /* Extra storage for Class and Window objects                   */
 wndclass.cbClsExtra = 0;
 wndclass.cbWndExtra = DLGWINDOWEXTRA;
 wndclass.hInstance = hInst;
 wndclass.hIcon = LoadIcon(hInst, "ampicon");
 wndclass.hCursor = LoadCursor(NULL, IDC_ARROW);
 /* Create brush for erasing background                          */
 wndclass.hbrBackground = (HBRUSH)(COLOR_WINDOW+1);
 wndclass.lpszMenuName = NULL;
 wndclass.lpszClassName = szAppName; /* Class Name is App Name */
 if(!RegisterClass(&wndclass))
   return -1;


 return(0);
} /* End of nCwRegisterClasses                                   */

/*************************************************************************/
/*  CwUnRegisterClasses Function                                         */
/*                                                                       */
/*  Deletes any references to windows resources created for this         */
/*  application, frees memory, deletes instance, handles and does        */
/*  clean up prior to exiting the window                                 */
/*                                                                       */
/*************************************************************************/
```

```
void CwUnRegisterClasses(void)
{
 WNDCLASS  wndclass;  /* struct to define a window class          */
 memset(&wndclass, 0x00, sizeof(WNDCLASS));

 UnregisterClass(szAppName, hInst);
}  /* End of CwUnRegisterClasses                              */


/*************************************************************************/
/*  InitWindowFields Function                                            */
/*                                                                       */
/*  This function initializes the state of certain buttons, and          */
/*  the sets the default text for all input boxes.                       */
/*                                                                       */
/*************************************************************************/
void InitWindowFields( HWND hWnd )
{
      /* initialize some input boxes */
      SetDlgItemText( hWnd, TempChanBox, "2" ) ;
      SetDlgItemText( hWnd, PressChanBox, "1" ) ;
      SetDlgItemText( hWnd, Hydrophone, "0" ) ;
      SetDlgItemText( hWnd, NumSamplesBox, "1" ) ;
      SetDlgItemText( hWnd, SampleRateBox, "333333" ) ;
      SetDlgItemText( hWnd, FileNameBox, NULL);
      SetDlgItemText( hWnd, WriteStatusBox, "0");
      SetDlgItemText( hWnd, SampIntBox, "2");
      SetDlgItemText( hWnd, SampTimeBox, "5");
      /* these are static input boxes, not for user input */
      SetDlgItemText( hWnd, INTStatusBox, "Inactive" ) ;
      SetDlgItemText( hWnd, INTTransferBox, "0" ) ;

      /* disable some buttons and input boxes */
      EnableWindow (GetDlgItem(hWnd, StopBtn), FALSE);

}

/*************************************************************************/
/*  InitDASDevice Function                                               */
/*                                                                       */
/*  This function initializes the board and driver according to the      */
/*  settings of the configuration file.                                  */
/*                                                                       */
/*************************************************************************/
void InitDASDevice(void)
{
  // open our DAS-1800 device
  if((Err=K_OpenDriver("DAS1800","das1800.cfg",&hDrv1800)) != 0)
  {
    if (Err==0x6035)  // driver already opened
    {
      Err=MessageBox( GetFocus(), "DAS-1800 driver has already been opened.  Continue using
previous configuration?",
```

```
                "DAS-1800",MB_YESNO ) ;
        if (Err==6)   // yes
          Err=K_OpenDriver("DAS1800","",&hDrv1800);
        else
          exit(1);
      }
      else
      {
        ProcessError(Err);
        exit(1);
      }
    }

    // now get a device handle
    if((Err=K_GetDevHandle(hDrv1800,0,&hDev1800)) != 0)
    {
      ProcessError(Err);
      exit(1);
    }

    //  now get an AD Frame
    if((Err = K_GetADFrame(hDev1800, &hAD )) !=0)
    {
      ProcessError(Err);
      exit(1);
    }
} // end of InitDASDevice

/*************************************************************************/
/* ReleaseDriver Function                                                */
/*                                                                       */
/* This function releases driver resources such as frame and             */
/* driver handles.                                                       */
/*                                                                       */
/*************************************************************************/
void ReleaseDriver(void)
{
    // release frame
    K_FreeFrame(hAD);

    // release device
    K_FreeDevHandle(hDev1800);

    // close driver
    K_CloseDriver(hDrv1800) ;
}
```

```
/*************************************************************************/
/* StartAcquiring Function                                               */
/*                                                                       */
/* This function signals the board to start the A/D acquisition based    */
/* on the user specified input parameters.                               */
/*                                                                       */
/*************************************************************************/
void StartAcquiring(HWND hWnd)
{
 static long     Rate;                                          // Rate Clock Divisor
 static long     Samples, Interval, SR, ST;      // Sample Interval
 static int      nGain, nChan;
 char                    TempBuf[80] = {0};
 char                    SampInt[80] = {0};

 // get the number of samples and allocate a buffer

    InitTime = GetTickCount();

    GetDlgItemText(hWnd, NumSamplesBox, (LPSTR)TempBuf, 10);  // 10 digit max
    ST=atol(TempBuf);

    GetDlgItemText(hWnd, SampleRateBox, (LPSTR)TempBuf,10);
                SR  = atol(TempBuf);
          Samples = SR*ST;

    NumSamp = Samples ;
    NumBuf = Samples/Max_Samp;
                                                    // save the sample number for later
    if(Samples%Max_Samp != 0)                   // if there is a remainder then allocate
another buffer
          NumBuf=NumBuf+1;

          BufCount = Samples;                   //set buffcount equal to sample number to
keep track of
                                                    //how many samples are
needed in the last buffer

                                                                    //set the
hydrophone channel

       GetDlgItemText(hWnd, Hydrophone , (LPSTR)TempBuf, 7);
         nChan = atol(TempBuf);

      if((Err=K_SetChn(hAD,nChan))!=0)
         {
                ProcessError(Err);
                return;
         }

      nGain=3;
      //set sample range to +- 1.25 volts
      if((Err=K_SetG(hAD,nGain))!=0)
         {
```

```
                ProcessError(Err);
                return;
            }

        // set the internal clock sample rate
        GetDlgItemText(hWnd, SampleRateBox , (LPSTR)TempBuf, 7); // 7 digits max

        Rate  = atol(TempBuf);              // convert to a clock divisor
        Rate  = (long)(1/(Rate  * .0000002)); // 5MHz clock!

        if((Err = K_SetClkRate(hAD, Rate)) != 0)
        {
           ProcessError(Err);
           return;
        }

    // Set continuous mode if user checks the box;
    // Otherwise set to single-cycle mode
        if ( IsDlgButtonChecked( hWnd, ContBox) )
        {
           if((Err = K_SetContRun(hAD)) != 0)
           {
              ProcessError(Err);
              return;
           }
        }
        else
        {
           if((Err = K_ClrContRun(hAD)) != 0)
           {
              ProcessError(Err);
              return;
           }
        }

                // enable stop button & disable start
        EnableWindow (GetDlgItem(hWnd, StopBtn), TRUE);
        EnableWindow (GetDlgItem(hWnd, StartBtn), FALSE);

        Get_Data(hWnd,hDev1800);

        // Start a timer which handles the timer interval set by user
        GetDlgItemText(hWnd,SampIntBox,(LPSTR)SampInt,2);
        Interval = atol(SampInt);
        Interval = Interval*1000;  //change seconds to milliseconds
        if(!SetTimer(hWndMain, ID_TIMER2, Interval, NULL))
        {
            wsprintf(szErr,"Timer2 Error!");
            MessageBox(NULL, szErr," Error ", MB_OK | MB_ICONEXCLAMATION);
            exit (1);
        }

} // end of StartAcquiring
```

```
/*************************************************************************/
/*  ShowData Function                                                    */
/*                                                                       */
/*  This function displays the acquired data in the ListBox.  The data   */
/*  displayed is limited to the first 100 samples, or less.              */
/*                                                                       */
/*************************************************************************/

void ShowData(HWND hWnd)
{
 char           szDataString[10];
 char           szIndexString[10];
 char           szString[128];
 short          SampData ;
 int            i;
 short far      *pBuffer;

 long           ShowSamp;
 DWORD                  j;
 float vval;
        ShowSamp=NumSamp;


   if (ShowSamp >1000) ShowSamp = 1000 ;   // limit to first 1000 samples
                                                                    // added
to test continuity

   pBuffer=( short far *)pDMABuf[1];

        // limit to first 1000 samples
        // Clear the data box first
   SendMessage (GetDlgItem(hWnd, DataListBox),LB_RESETCONTENT, 0, (LONG) (LPSTR)
szString);

        for (i=0 ; i<50 ; i++)
   {
         SampData = pBuffer[i];
         vval = (SampData * 2.5)/4096;
         //ltoa( vval , szDataString , 10) ;     // base 10
         _gcvt( vval, 3, szDataString );
    itoa( i , szIndexString , 10 ) ;          // base 10
    j++;
    strcpy( szString, "                                          ") ;
    strncpy( szString+5, szIndexString, strlen(szIndexString)) ;  // add index
    strncpy( szString+15, szDataString, strlen(szDataString)) ;   // add data
        SendMessage (GetDlgItem(hWnd, DataListBox),LB_ADDSTRING, i, (LONG) (LPSTR)
szString);
        }
}
```

```
/*************************************************************************/
/* StopAcquiring Function                                                */
/*                                                                       */
/* This function signals the board to stop the A/D process.              */
/*                                                                       */
/*************************************************************************/
void StopAcquiring(HWND hWnd)
{
                        // enable start button & disable stop
        EnableWindow (GetDlgItem(hWnd, StopBtn), FALSE);
        EnableWindow (GetDlgItem(hWnd, StartBtn), TRUE);

                // update status box
  SetDlgItemText( hWnd, INTStatusBox, "Inactive" ) ;

    if (nIsIcon)  // if app is icon, update display
      SetWindowText(hWnd, "DAS-1800 [Inactive]") ;

    if(ADOP == 1)  // only allow if started
    {
      KillTimer(hWnd, ID_TIMER1);
      KillTimer(hWnd, ID_TIMER2);
      SetDlgItemText( hWnd, INTStatusBox, "Inactive" ) ;

      if((Err = K_DMAStop(hAD, &Status, &Index)) != 0)
       {      // Stop Operation
              ProcessError(Err);
        return;
       }


      Done = 1;
      ADOP = 0;
  }

} // end of StopAcquiring

/*************************************************************************/
/*      GetData Function:                                                */
/*                                                                       */
/*      This function will get the appropriate A/D channels for the      */
/*      thermistor, and pressure transducer and read the data.  It then  */
/*      starts the A/D converter to collect the hydrophone data and starts */
/*      a 10 ms timer to monitor the status of the A/D board.            */
/*                                                                       */
/*************************************************************************/
void Get_Data(HWND hWnd,DDH James)
{
 long int              SR , ST;
 int                   PressData;
 int                   TempData;
 char                  Pchan[80] = {0};
 char                  Tchan[80] = {0};
 char                  Hchan[80] = {0};
```

```
int                     PressChan;
int                     TempChan;
int                     i;
static long     Samples;
char                    TempBuf[80] = {0};
short                   temp;

short                   pres;


char                    PCTime[9];
char                    szString[70];
char    Xstring[UsedStringSize];
char                    Xmark[]="                       ";

if(GPS_FLAG==1)
        {
        do
                {
          ProcessCOMMNotification( hWnd, NULL, NULL ) ;
    }
  while(NOTIFY_FLAG==0);
  }

        //Get the pressure transducer A/D channel and read PressData
GetDlgItemText(hWnd, PressChanBox, (LPSTR)Pchan, 10);
PressChan = atol(Pchan);

if((Err=K_ADRead(hDev1800,PressChan,3,&PressData))!=0)
        {
        ProcessError(Err);
        return;
        }

pres=PressData;
// itoa(pres,Pres,10);

                                //Get the Thermistor A/D channel and read TempData
GetDlgItemText(hWnd, TempChanBox, (LPSTR)Tchan, 10);
TempChan = atol(Tchan);

if((Err=K_ADRead(hDev1800,TempChan,3,&TempData))!=0)
        {
        ProcessError(Err);
        return;
        }

_strtime(PCTime);
temp=TempData;
//itoa(temp,Temp,10);

DataTranslation(temp , pres);
```

```
strcpy( szString, "
                          ") ;

        strncpy(szString+8,PCTime, strlen(PCTime)); //add current time


        strncpy( szString+20, vtemp, strlen(vtemp)) ;  // add index
    strncpy( szString+31, vpres, strlen(vpres)) ;   // add data
    strncpy( szString+39, Xmark, strlen(Xmark));
        if(GPS_FLAG==1)
        {

            strncpy( szString+40, NORTH, strlen(NORTH)) ;  // add index
      strncpy( szString+52, WEST, strlen(WEST)) ;
                  }
                  strncpy(szString+65,szTimeString,strlen(szTimeString));
    strncpy(Xstring,szString,68);

   SendMessage (GetDlgItem(hWnd, DataUpdateBox),LB_INSERTSTRING, 0, (LONG) (LPSTR)
Xstring);

   Store2(hWnd,Xstring);


                          //InitTime = GetTickCount();
     GetDlgItemText(hWnd, NumSamplesBox, (LPSTR)TempBuf, 10);  // 10 digit max
          ST=atol(TempBuf);
      GetDlgItemText(hWnd, SampleRateBox, (LPSTR)TempBuf,10);

                  SR  = atol(TempBuf);
          Samples = SR*ST;
       ltoa(Samples, TempBuf, 10) ;
      SetDlgItemText( hWnd, SamplesPerRead, TempBuf ) ;

     NumSamp = Samples ;
     NumBuf = Samples/Max_Samp;
          // save the sample number for later
     if(Samples%Max_Samp != 0)// if there is a remainder then allocate another buffer
        NumBuf=NumBuf+1;
        BufCount = Samples;       //set bufcount equal to sample number to keep track of
                                                  //how many samples are needed in the last buffer


        for(i=1;i<=NumBuf;i++)
     {

                          if((BufCount-Max_Samp) >0)
        {
                  holder = Max_Samp;
                              BufCount=BufCount-Max_Samp;
                          }
        else
        holder = BufCount;
```

```
        if((Err = K_DMAAlloc(hAD,holder , (void far * far *)&pDMABuf[i], &hMem[i])) != 0)     //
(void far * far * )
        {
        ProcessError(Err);
        return;
        }

        // tell the frame about the buffer and number of samples
        if((Err = K_BufListAdd(hAD, pDMABuf[i],holder )) != 0)
        {
        ProcessError(Err);
        return;
        }

        }//end of for loop, i++

        ADOP = 1;          // Set Operation Flag
        Done = 0;          // Clear Done Flag
        Status = 0;        // Clear  Status Flag
        DataFlg = 1;

  //  reset status/count display
     SetDlgItemText( hWnd, INTTransferBox, "0" ) ;
     SetDlgItemText( hWnd, INTStatusBox, "Active" ) ;

        // Start the A/D and get the hydrophone data

        // Start A/D MODE
     if((Err = K_DMAStart(hAD)) != 0)
     {
       ProcessError(Err);
       return;
     }

        // Start a 10ms timer to monitor status
     if(!SetTimer(hWndMain, ID_TIMER1, 10, NULL))
     {
       wsprintf(szErr,"Timer1 Error!");
       MessageBox(NULL, szErr," Error ", MB_OK | MB_ICONEXCLAMATION);
       exit (1);
     }

 return;
}//end of Get_Data function

//****************************************
//        Store2
//
// Function to store Time Depth GPS Temp String
//
// input string
// stores file with .txt extension
//****************************************
```

```
void Store2(HWND hWnd , LPSTR GenString)
{
  unsigned char NameBuf[10];
 unsigned char Fsuffx[]=".txt";
 FILE *outfile;


 int i,j;
 int fh;
 unsigned byteswritten;
int p;

 int amount;


        GetDlgItemText(hWnd,FileNameBox,(LPSTR)NameBuf,8);


        strncat(NameBuf,Fsuffx,10);



 if ((fh = _open( NameBuf , _O_RDWR | _O_TEXT | _O_APPEND | _O_CREAT, _S_IREAD |
_S_IWRITE))!=-1 );
    {


     byteswritten = _write( fh, GenString , UsedStringSize);
    }


   _close( fh );
    }
//************************************************************
// DataTranslation Function
//
// Translates Data from A/D units to Voltage and to Depth & Temp
//
//************************************************************

void DataTranslation(int Temperature , int Pressure)
{


 tempvolt = (Temperature * 2.5)/4096;
 presvolt = (Pressure * 2.5)/4096;

  _gcvt( tempvolt, 3, vtemp );
  _gcvt( presvolt, 3, vpres );


return;
}
```

```
// GLOBAL filename suffix
int       hydfilesufx=0;

/*************************************************************************/
/* ProcessTimer1 Function                                                */
/*                                                                       */
/* This function processes all ID_TIMER1 events. The acquisition task    */
/* is monitored here and the transfer count is updated.                  */
/*                                                                       */
/*************************************************************************/
void ProcessTimer1(HWND hWnd)
{
char TempBuf[20];

int k;
 long realcount;
 long realstatus;
     if((Err = K_DMAStatus(hAD, &Status, &Index)) != 0)
     {
        KillTimer(hWnd, ID_TIMER1);
        KillTimer(hWnd, ID_TIMER2);
        ProcessError(Err);
        StopAcquiring( hWnd) ;
        return;
     }

     // update the transfer counter display
      realstatus = Status/256;
      realcount = (realstatus-1)*Max_Samp+Index;
     ltoa(realcount, TempBuf, 10) ; // base 10
     SetDlgItemText( hWnd, INTTransferBox, TempBuf ) ;
     ltoa(Status, TempBuf, 10) ;
      SetDlgItemText( hWnd, INTStatusBox, TempBuf ) ;

     if((Status & 4)==4)    // overrun
     {
        KillTimer(hWnd, ID_TIMER1);
        KillTimer(hWnd, ID_TIMER2);
        wsprintf(szErr,"DAS-1800 Overrun Error");
        MessageBox(NULL, szErr," Error ", MB_OK | MB_ICONEXCLAMATION);

        StopAcquiring(hWnd) ;
        return;
     }

     if((Status & 1)==0)    // finished ?
     {
         KillTimer(hWnd, ID_TIMER1);
        //KillTimer(hWnd, ID_TIMER2);
        //StopAcquiring(hWnd) ;
        SetDlgItemText( hWnd, INTStatusBox, "Inactive" ) ;

        if((Err = K_DMAStop(hAD, &Status, &Index)) != 0)
```

```
        {       // Stop Operation
          ProcessError(Err);
          return;
        }
        //set data manipulation in process flag
        DataFlg = 1;
        //convert the data

        ShowData(hWnd);//call show data function


        StoreData(hWnd);

        for(k=1;k<=NumBuf;k++)
        {
        if((Err = K_DMAFree(hMem[k])) != 0)
        {     // Free memory
          ProcessError(Err);
          return;
        }
        if((Err=K_BufListReset(hAD))!=0)
        {
         ProcessError(Err);
         return;
        }
        }//end of for loop(i++)
        //Done = 1;
        //ADOP = 0;
        DataFlg = 0;
        return;
      }

} // end of ProcessTimer



//*************************************************
// StoreData function
//
//
// Procedure to store hydrophone data
//*************************************************


void StoreData(HWND hWnd)
{
 unsigned char NameBuf[10];
 unsigned char Fsuffx[4];
 FILE *outfile;
 void *pData;

 int i,j;
 int fh;
```

```
unsigned byteswritten;
short sdata;
int p;
int *PING;
int amount;
        i=0;
        i++;
        hydfilesufx++;

        GetDlgItemText(hWnd,FileNameBox,(LPSTR)NameBuf,8);
        itoa(hydfilesufx,Fsuffx,10);
        strncat(NameBuf,"_hyd.",10);
        strncat(NameBuf,Fsuffx,10);


 if ((fh = _open( NameBuf , _O_RDWR | _O_BINARY | _O_CREAT, _S_IREAD | _S_IWRITE))!=-1 );
   {
   amount = Max_Samp;
    for(p=1;p<=NumBuf;p++)
    {
    if(p==NumBuf)
      amount = holder;

    pData=   (void *)pDMABuf[p];


     byteswritten = _write( fh, pData , amount);
   }
   }

  _close( fh );

 }//end of StoreData function

/*************************************************************************/
/*        ProcessTimer2 Function                                         */
/*                                                                       */
/* This function processes timer events defined by the timer             */
/* interval.  It checks to see if the sample time has been met.          */
/* If it has, it will stop acquiring data. Otherwise will will           */
/* call the sample data function to continue data acquisition.           */
/*                                                                       */
/*************************************************************************/

void ProcessTimer2(HWND hWnd)
{
 long CurrentTime;
 long ElapsedTime;
 long StopTime;
 int NumSamples=0;

 char SampTime[80] = {0};
```

```
        CurrentTime = GetTickCount();
        ElapsedTime = (CurrentTime - InitTime);
        ElapsedTime = ElapsedTime/1000; // convert to seconds

        GetDlgItemText(hWnd,SampTimeBox,(LPSTR)SampTime,2);
        StopTime = atol(SampTime);
        StopTime = StopTime*60;//convert from minutes to seconds
        if(ElapsedTime > StopTime)
                {
                        StopAcquiring(hWnd);
                        return;
                }

        if(DataFlg == 0)
                {

                        Get_Data(hWnd,hDev1800);
                }
   //ShowData(hWnd);
        //++NumSamples;
        ltoa(ElapsedTime,szTimeString,10);

                return;
}

/************************************************************************/
/* ProcessError Function                                                */
/*                                                                      */
/* This function displays an error message and the argument passed      */
/* as error number.                                                     */
/*                                                                      */
/************************************************************************/
void ProcessError(short ErrNum)
{
   wsprintf(szErr,"DAS-1800 Error = %4x", ErrNum);
   MessageBox(NULL, szErr," Error ", MB_OK | MB_ICONEXCLAMATION);
}


//*****************************************
// LatLong
//
//  Function for parsing latitude and Longitude
//*****************************************

void NEAR LatLong(HWND hWnd, LPSTR strip , int nLength)
 {
 int     i;
 char    string1[]="xxxxxxxxx";
 char    string2[]="xxxxxxxxxx";
 char    str[] =   "$PMVXG,021";
 float   CurrentTime;
 float   ElapsedTime;
 char    *pdest;
```

```
int    result;

north = string1;
west = string2;
pdest = strstr( strip, str );
result = pdest - strip + 1;

        if(pdest[31]='N')
        {
      if( pdest[44]='W')
        {
         if(pdest[38]='.')
                {
         if(pdest[50]='.')
                {
                 if(pdest[25]='.')
                        {
                                for(i=0;i<10;i++)
                                        {
                                if(isdigit(pdest[i+33])|| ispunct(pdest[i+33]))
                                        {
                                                                west[i]=pdest[i+33];
                                                                NOTIFY_FLAG=1;
                                                               }
                                else
                                {
                                         NOTIFY_FLAG=0;
                                         return;
                                        }
                                }

                                for(i=0;i<9;i++)
                                        {
                                if(isdigit(pdest[i+33])|| ispunct(pdest[i+33]))
                                        {
                                         north[i]=pdest[i+21];
                                           NOTIFY_FLAG=1;
                                        }
                                else
                                        {
                                         NOTIFY_FLAG=0;
                                         return;
                                        }
                                }
                         }
                         }
                }
        }
        }

strcpy(WEST,west);
strcpy(NORTH,north);
 return;
}
```

```
 //*********************************
 // OpenGps function
 // function to open port for gps data retrieval
 //
 /*********************************/

BOOL NEAR OpenConnection( HWND hWnd )
{
  char     szTemp[ 10 ] ;
  BOOL     fRetVal ;
  HMENU    hMenu ;
  NPTTYINFO  npTTYInfo ;
// open COMM device

  if ((COMMID = OpenComm( which_comm , RXQUEUE, TXQUEUE )) < 0)
                return ( FALSE ) ;
    COMDEV(npTTYInfo)=COMMID;
  fRetVal = SetupConnection( hWnd ) ;

  if (fRetVal)
  {
    CONNECTED( npTTYInfo ) = TRUE ;

    // Enable notification for CN_RECEIVE events.

   //    EnableCommNotification( COMDEV( npTTYInfo ), hWnd, MAXBLOCK, -1 ) ;

    // assert DTR

    EscapeCommFunction( COMDEV( npTTYInfo ), SETDTR ) ;

        }
  else
  {
    CONNECTED( npTTYInfo ) = FALSE ;
    CloseComm( COMDEV( npTTYInfo ) ) ;
  }

  return ( fRetVal ) ;

} // end of OpenConnection()

//-------------------------------------------------------------------
// LRESULT NEAR CreateGPSInfo( HWND hWnd )
//
// Description:
//    Creates the  information structure used for GPS setup
//
// Parameters:
//    HWND  hWnd
//       Handle to main window.
//
//-------------------------------------------------------------------
```

```
LRESULT NEAR CreateGPSInfo( HWND hWnd )
{

  NPTTYINFO   npTTYInfo ;

  if (NULL == (npTTYInfo =
            (NPTTYINFO) LocalAlloc( LPTR, sizeof( TTYINFO ) )))
    return ( (LRESULT) -1 ) ;

  // initialize TTY info structure

  COMDEV( npTTYInfo )       = 0 ;
  CONNECTED( npTTYInfo )    = FALSE ;
  CURSORSTATE( npTTYInfo )  = CS_HIDE ;
  LOCALECHO( npTTYInfo )    = FALSE ;
  AUTOWRAP( npTTYInfo )     = TRUE ;
  PORT( npTTYInfo )         = 2 ;
  BAUDRATE( npTTYInfo )     = CBR_4800 ;
  BYTESIZE( npTTYInfo )     = 8 ;
  FLOWCTRL( npTTYInfo )     = FC_RTSCTS ;
  PARITY( npTTYInfo )       = NOPARITY ;
  STOPBITS( npTTYInfo )     = ONESTOPBIT ;
  XONXOFF( npTTYInfo )      = FALSE ;
  XSIZE( npTTYInfo )        = 0 ;
  YSIZE( npTTYInfo )        = 0 ;
  XSCROLL( npTTYInfo )      = 0 ;
  YSCROLL( npTTYInfo )      = 0 ;
  XOFFSET( npTTYInfo )      = 0 ;
  YOFFSET( npTTYInfo )      = 0 ;
  COLUMN( npTTYInfo )       = 0 ;
  ROW( npTTYInfo )          = 0 ;
  HTTYFONT( npTTYInfo )     = NULL ;
  FGCOLOR( npTTYInfo )      = RGB( 0, 0, 0 ) ;
  USECNRECEIVE( npTTYInfo ) = TRUE ;
  DISPLAYERRORS( npTTYInfo ) = TRUE ;

   SetWindowWord( hWnd, GWW_NPTTYINFO, (WPARAM) npTTYInfo ) ;


  return ( (LRESULT) TRUE ) ; }


//--------------------------------------------------------------------
// BOOL NEAR SetupConnection( HWND hWnd )
//
// Description:
//   This routines sets up the DCB based on settings in the
//   GPS info structure and performs a SetCommState().
//
// Parameters:
//   HWND hWnd
```

```
//
//
//-------------------------------------------------------------

BOOL NEAR SetupConnection( HWND hWnd )
{
  BOOL      fRetVal ;
  BYTE      bSet ;
  DCB       dcb ;
  NPTTYINFO  npTTYInfo ;

  if (NULL == (npTTYInfo = (NPTTYINFO) GetWindowWord( hWnd, GWW_NPTTYINFO )))
    return ( FALSE ) ;

  GetCommState(COMMID , &dcb ) ;
            // COMDEV( npTTYInfo )
  dcb.BaudRate = CBR_4800;   // BAUDRATE( npTTYInfo ) ;
  dcb.ByteSize = BYTESIZE( npTTYInfo ) ;
  dcb.Parity = PARITY( npTTYInfo ) ;
  dcb.StopBits = STOPBITS( npTTYInfo ) ;

  // setup hardware flow control

  bSet = (BYTE) ((FLOWCTRL( npTTYInfo ) & FC_DTRDSR) != 0) ;
  dcb.fOutxDsrFlow = dcb.fDtrflow = bSet ;
  dcb.DsrTimeout = (bSet) ? 30 : 0 ;

  bSet = (BYTE) ((FLOWCTRL( npTTYInfo ) & FC_RTSCTS) != 0) ;
  dcb.fOutxCtsFlow = dcb.fRtsflow = bSet ;
  dcb.CtsTimeout = (bSet) ? 30 : 0 ;

  // setup software flow control

  bSet = (BYTE) ((FLOWCTRL( npTTYInfo ) & FC_XONXOFF) != 0) ;

  dcb.fInX = dcb.fOutX = bSet ;
  dcb.XonChar = ASCII_XON ;
  dcb.XoffChar = ASCII_XOFF ;
  dcb.XonLim = 100 ;
  dcb.XoffLim = 100 ;

  // other various settings

  dcb.fBinary = TRUE ;
  dcb.fParity = TRUE ;
  dcb.fRtsDisable = FALSE ;
  dcb.fDtrDisable = FALSE ;

  fRetVal = !(SetCommState( &dcb ) < 0) ;

  return ( fRetVal ) ;

} // end of SetupConnection()
//-------------------------------------------------------------
```

```
// BOOL NEAR ProcessCOMMNotification( HWND hWnd, WORD wParam, LONG lParam )
//
// Description:
//   Processes the WM_COMMNOTIFY message from the COMM.DRV.
//
// Parameters:
//   HWND hWnd
//
//
//   WORD wParam
//     specifes the device (nCid)
//
//   LONG lParam
//     LOWORD contains event trigger
//     HIWORD is NULL
//
//---------------------------------------------------------------

BOOL NEAR ProcessCOMMNotification( HWND hWnd, WORD wParam, LONG lParam )
{
  char     szError[ 10 ] ;
  int      nError, nLength ;
  BYTE     abIn[ MAXBLOCK + 1] ;
  COMSTAT  ComStat ;
  NPTTYINFO  npTTYInfo ;
  MSG      msg ;
  int i=1;

 do
 {
  if(nLength = ReadCommBlock( hWnd, (LPSTR) abIn, MAXBLOCK ))
    {

    LatLong(hWnd , (LPSTR) abIn, nLength);

       }
     if (nError = GetCommError( COMMID, &ComStat ))
     {
     }
    // {

      // if (DISPLAYERRORS( npTTYInfo ))
      // {
        // MessageBox( GetFocus(), "GPS Problem",
         //         "GPS Comm Error",MB_OK ) ;
          // wsprintf( szError, "<CE-%d>", nError ) ;
    // }
    // }
    }
    while ((!PeekMessage( &msg, NULL, 0, 0, PM_NOREMOVE )) ||
        (ComStat.cbInQue >= MAXBLOCK)) ;

  return ( TRUE ) ;
} // end of ProcessCOMMNotification()
```

```
//--------------------------------------------------------------------
// int NEAR ReadCommBlock( HWND hWnd, LPSTR lpszBlock, int nMaxLength )
//
// Description:
//   Reads a block from the COM port and stuffs it into
//   the provided block.
//
// Parameters:
//   HWND hWnd
//
//
//   LPSTR lpszBlock
//     block used for storage
//
//   int nMaxLength
//     max length of block to read
//
//--------------------------------------------------------------------

int NEAR ReadCommBlock( HWND hWnd, LPSTR lpszBlock, int nMaxLength )
{
  char     szError[ 10 ] ;
  int      nLength, nError ;
  NPTTYINFO  npTTYInfo ;

 nLength = ReadComm( COMMID , lpszBlock, nMaxLength ) ;

  return ( nLength ) ;

} // end of ReadCommBlock()
```

AMP1.h Include File

```
#include <windows.h>
#include <string.h>
#include <stdlib.h>

#define IDS_ERR_REGISTER_CLASS     1
#define IDS_ERR_CREATE_WINDOW      2
#define StartBtn                   101
#define StopBtn                    102
#define CloseBtn                   118
#define ContBox                    124
#define StartChanBox               104
#define StopChanBox                105
#define NumSamplesBox              108
#define SampleRateBox              110
#define INTStatusBox               113
#define INTTransferBox             114
#define INTTransferBox             114
#define DataListBox                119
#define ID_TIMER1                  1
#define ID_TIMER2                  2
#define FileNameBox                131
```

```
#define WriteStatusBox          132
#define DataUpdateBox           145
#define SampIntBox              146
#define SampTimeBox             147
#define gps_connect             200
#define gps_disconnect          201
#define COMM_1                  888
#define COMM_2                  999
#define Max_Buf                 72
#define Max_Samp                65000
```

Amp2.h Include File

```
char szString[128];  /* variable to load resource strings      */
char szAppName[20];  /* class name for the window              */
HWND hInst;
HWND hWndMain;

LONG FAR PASCAL WndProc(HWND, UINT, UINT, LONG);
int nCwRegisterClasses(void);
void CwUnRegisterClasses(void);
void InitWindowFields( HWND hwnd );
void InitDASDevice(void) ;
void ReleaseDriver(void) ;
void StartAcquiring(HWND hWnd) ;
void StopAcquiring(HWND hWnd) ;
void ProcessTimer1(HWND hWnd);
void ShowData(HWND hWnd) ;
void ProcessError(short ErrNum) ;
void OpenGps(HWND hWnd);        // Functions for GPS info
void ReadGps(HWND hWnd);
void DisplayGps(HWND hWnd);
void CloseGps(HWND hWnd);
void ProcessTimer2(HWND hWnd);
void Get_Data(HWND hWnd, DDH James);
void StoreData(HWND hWnd);
void DataTranslation(int Temperature , int Pressure);
void Store2(HWND hWnd , LPSTR GenString);
```

gps.h Include File

```
//--------------------------------------------------------------------
//
// Module: gps.h
//
// Purpose:
//    Includes prototypes for GPS procedures
//
//--------------------------------------------------------------------
//
// Written by Microsoft Product Support Services, Windows Developer Support.
// Copyright (c) 1991 Microsoft Corporation.  All Rights Reserved.
```

```
//
//--------------------------------------------------------

#define WIN31       // this is a Windows 3.1 application
#define USECOMM     // yes, we need the COMM API
#define STRICT      // be bold!

#include <windows.h>
#include <commdlg.h>
#include <string.h>

#include "version.h"
#include "resource.h"

// constant definitions

#define GWW_NPTTYINFO     0
#define ABOUTDLG_USEBITMAP  1

#define ATOM_TTYINFO      0x100

// terminal size

#define MAXROWS       25
#define MAXCOLS       80

#define MAXBLOCK      3000  // change  by Jay

#define MAXLEN_TEMPSTR  100

#define RXQUEUE       4096
#define TXQUEUE       100

// cursor states

#define CS_HIDE       0x00
#define CS_SHOW       0x01

// Flow control flags

#define FC_DTRDSR     0x01
#define FC_RTSCTS     0x02
#define FC_XONXOFF    0x04

// ascii definitions

#define ASCII_BEL     0x07
#define ASCII_BS      0x08
#define ASCII_LF      0x0A
#define ASCII_CR      0x0D
#define ASCII_XON     0x11
#define ASCII_XOFF    0x13

// data structures
```

```
typedef struct tagTTYINFO
{
  int    idComDev ;
  BYTE   bPort, abScreen[ MAXROWS * MAXCOLS ] ;
  BOOL   fConnected, fXonXoff, fLocalEcho, fNewLine, fAutoWrap,
         fUseCNReceive, fDisplayErrors;
  BYTE   bByteSize, bFlowCtrl, bParity, bStopBits ;
  WORD   wBaudRate, wCursorState ;
  HFONT  hTTYFont ;
  LOGFONT lfTTYFont ;
  DWORD  rgbFGColor ;
  int    xSize, ySize, xScroll, yScroll, xOffset, yOffset,
         nColumn, nRow, xChar, yChar ;

} TTYINFO, NEAR *NPTTYINFO ;

// macros ( for easier readability )

#define GETHINST( hWnd ) ((HINSTANCE) GetWindowWord( hWnd, GWW_HINSTANCE ))

#define COMDEV( x ) (x -> idComDev)
#define PORT( x )  (x -> bPort)
#define SCREEN( x ) (x -> abScreen)
#define CONNECTED( x ) (x -> fConnected)
#define XONXOFF( x ) (x -> fXonXoff)
#define LOCALECHO( x ) (x -> fLocalEcho)
#define NEWLINE( x ) (x -> fNewLine)
#define AUTOWRAP( x ) (x -> fAutoWrap)
#define BYTESIZE( x ) (x -> bByteSize)
#define FLOWCTRL( x ) (x -> bFlowCtrl)
#define PARITY( x ) (x -> bParity)
#define STOPBITS( x ) (x -> bStopBits)
#define BAUDRATE( x ) (x -> wBaudRate)
#define CURSORSTATE( x ) (x -> wCursorState)
#define HTTYFONT( x ) (x -> hTTYFont)
#define LFTTYFONT( x ) (x -> lfTTYFont)
#define FGCOLOR( x ) (x -> rgbFGColor)
#define XSIZE( x ) (x -> xSize)
#define YSIZE( x ) (x -> ySize)
#define XSCROLL( x ) (x -> xScroll)
#define YSCROLL( x ) (x -> yScroll)
#define XOFFSET( x ) (x -> xOffset)
#define YOFFSET( x ) (x -> yOffset)
#define COLUMN( x ) (x -> nColumn)
#define ROW( x ) (x -> nRow)
#define XCHAR( x ) (x -> xChar)
#define YCHAR( x ) (x -> yChar )
#define USECNRECEIVE( x ) (x -> fUseCNReceive)
#define DISPLAYERRORS( x ) (x -> fDisplayErrors)

#define SET_PROP( x, y, z )  SetProp( x, MAKEINTATOM( y ), z )
#define GET_PROP( x, y )     GetProp( x, MAKEINTATOM( y ) )
#define REMOVE_PROP( x, y )  RemoveProp( x, MAKEINTATOM( y ) )
```

```
// global stuff
char    gszTTYClass[] = "TTYWndClass" ;
char    gszAppName[] = "TTY" ;
HANDLE  ghAccel ;
WORD    gawBaudTable[] = { CBR_110,
                           CBR_300,
                           CBR_600,
                           CBR_1200,
                           CBR_2400,
                           CBR_4800,
                           CBR_9600,
                           CBR_14400,
                           CBR_19200,
                           CBR_38400,
                           CBR_56000,
                           CBR_128000,
                           CBR_256000   } ;

WORD    gawParityTable[] = { NOPARITY,
                             EVENPARITY,
                             ODDPARITY,
                             MARKPARITY,
                             SPACEPARITY } ;

WORD    gawStopBitsTable[] = { ONESTOPBIT,
                               ONE5STOPBITS,
                               TWOSTOPBITS } ;
// function prototypes (private)

BOOL NEAR InitApplication( HANDLE ) ;
HWND NEAR InitInstance( HANDLE, int ) ;
LRESULT NEAR CreateGPSInfo( HWND ) ;
BOOL NEAR DestroyTTYInfo( HWND ) ;
BOOL NEAR ResetTTYScreen( HWND, NPTTYINFO ) ;
BOOL NEAR KillTTYFocus( HWND ) ;
BOOL NEAR PaintTTY( HWND ) ;
BOOL NEAR SetTTYFocus( HWND ) ;
BOOL NEAR ScrollTTYHorz( HWND, WORD, WORD ) ;
BOOL NEAR ScrollTTYVert( HWND, WORD, WORD ) ;
BOOL NEAR SizeTTY( HWND, WORD, WORD ) ;
BOOL NEAR ProcessTTYCharacter( HWND, BYTE ) ;
int NEAR ReadCommBlock( HWND, LPSTR, int ) ;
BOOL NEAR WriteCommByte( HWND, BYTE ) ;
BOOL NEAR MoveTTYCursor( HWND ) ;
BOOL NEAR OpenConnection( HWND ) ;
BOOL NEAR SetupConnection( HWND ) ;
BOOL NEAR CloseConnection( HWND ) ;
BOOL NEAR ProcessCOMMNotification( HWND, WORD, LONG ) ;
void NEAR LatLong(HWND hWnd, LPSTR strip , int nLength);
VOID NEAR GoModalDialogBoxParam( HINSTANCE, LPCSTR, HWND, DLGPROC, LPARAM ) ;
//-----------------------------------------------------------
// End of File: gps.h
//-----------------------------------------------------------
```

Appendix C

Calculations

APPENDIX C

Calculations.

Pressure Housing Analysis

$\rho_{sea\_water} := 64 \cdot \frac{lb}{ft^3}$ Density of sea water.

$h := 500 \cdot ft$ Design depth.

$P := \rho_{sea\_water} \cdot g \cdot h$ Acting pressure at design depth.

$P = 222.222 \cdot psi$

$P := 250 \cdot psi$ Design pressure requirement.

Stresses experienced by the pressure housing.

Modes of failure: 1. Yielding
2. Instability
3. Buckling

Design stresses for yielding:

$P_o := 250 \cdot psi$ External pressure.

$P_i := 14.7 \cdot psi$ Internal pressure.

$r_o := 4.5 \cdot in$ Outside radius of the housing.

$r_i := 4 \cdot in$ Inside radius of the housing.

$r := r_o$ Radius under investigation.

$$\sigma_t := \frac{P_i \cdot r_i^2 - P_o \cdot r_o^2 - r_i^2 \cdot r_o^2 \cdot \frac{P_o - P_i}{r^2}}{r_o^2 - r_i^2}$$ Tangential stress experienced by the housing.

$\sigma_t = -2.022 \cdot 10^3 \cdot psi$

Reference Shigley, Machine Design 1989.

$$\sigma_r := \frac{P_i \cdot r_i^2 - P_o \cdot r_o^2 + r_i^2 \cdot r_o^2 \cdot \frac{P_o - P_i}{r^2}}{r_o^2 - r_i^2}$$

Radial stress experienced by the housing.

$\sigma_r = -250 \cdot psi$

Design stresses for Buckling:

$E := 10.4 \cdot 10^6 \cdot psi$ Young's Modulous.

$\upsilon := .29$ Poisson's Ratio.

$t := .5 \cdot in$ Shell thickness.

$R := 4 \cdot in$ Radius.

$L_f := 14 \cdot in$ Length between frames.

$$P_b := \frac{2.42 \cdot E}{\left(1 - \upsilon^2\right)^{\frac{3}{4}}} \cdot \frac{\left(\frac{t}{2 \cdot R}\right)^{\frac{5}{2}}}{\left[\frac{L_f}{2 \cdot R} - .45 \cdot \left(\frac{t}{2 \cdot R}\right)^{\frac{1}{2}}\right]}$$

Pressure necessary for failure by buckling.

$P_b = 1.603 \cdot 10^4 \cdot psi$

Reference Sedor,
Class notes.
OE 754/854

Maximum pressure for General Instability:

$L := L_f$ The distance between frames or endcaps.

$m := \frac{\pi \cdot R}{L}$

$n := 2$

$I := 5 \cdot in^4$ Moment of inertia.

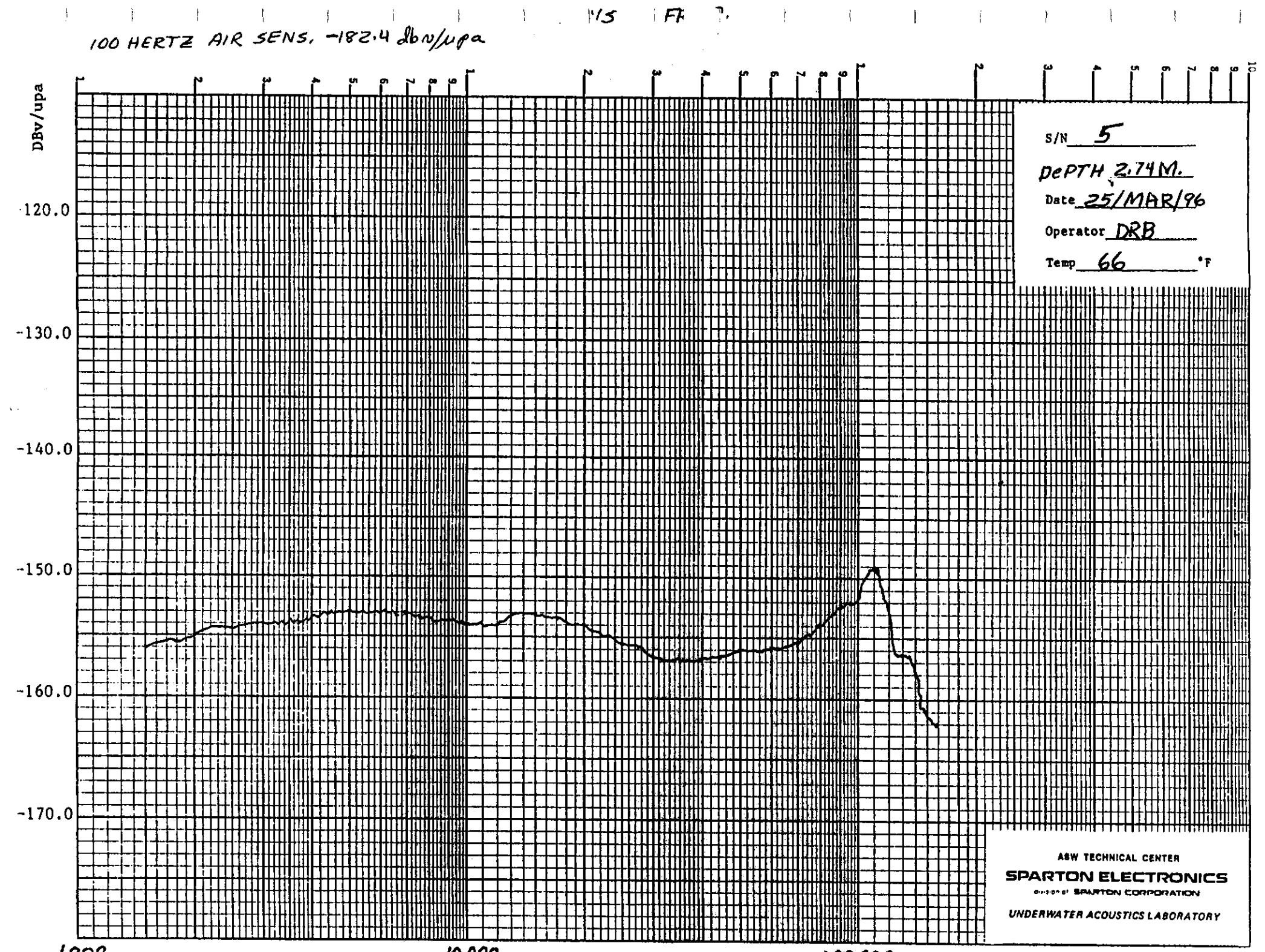
$$P_{cr} := \frac{E \cdot t}{R} \cdot \left[\frac{m^4}{\left(n^2 + \frac{m^2}{2} - 1\right) \cdot \left(n^2 + m^2\right)} \right] + \frac{\left(n^2 - 1\right) \cdot E \cdot I}{R^3 \cdot L_f}$$

$P_{cr} = 2.257 \cdot 10^5 \cdot psi$

Reference Sedor,
Class notes.
OE 754/854

Appendix D

Hydrophone Specification Sheets


100 HERTZ AIR SENS. -182.4 dbv/µpa
DBv/upa
-120.0
-130.0
-140.0
-150.0
-160.0
-170.0
S/N 5
DEPTH 2.74M.
Date 25/MAR/96
Operator DRB
Temp 66 °F
ASW TECHNICAL CENTER
SPARTON ELECTRONICS
SPARTON CORPORATION
UNDERWATER ACOUSTICS LABORATORY

RVS VS FREQ.

S/N 5 DATE 25/MAR/96

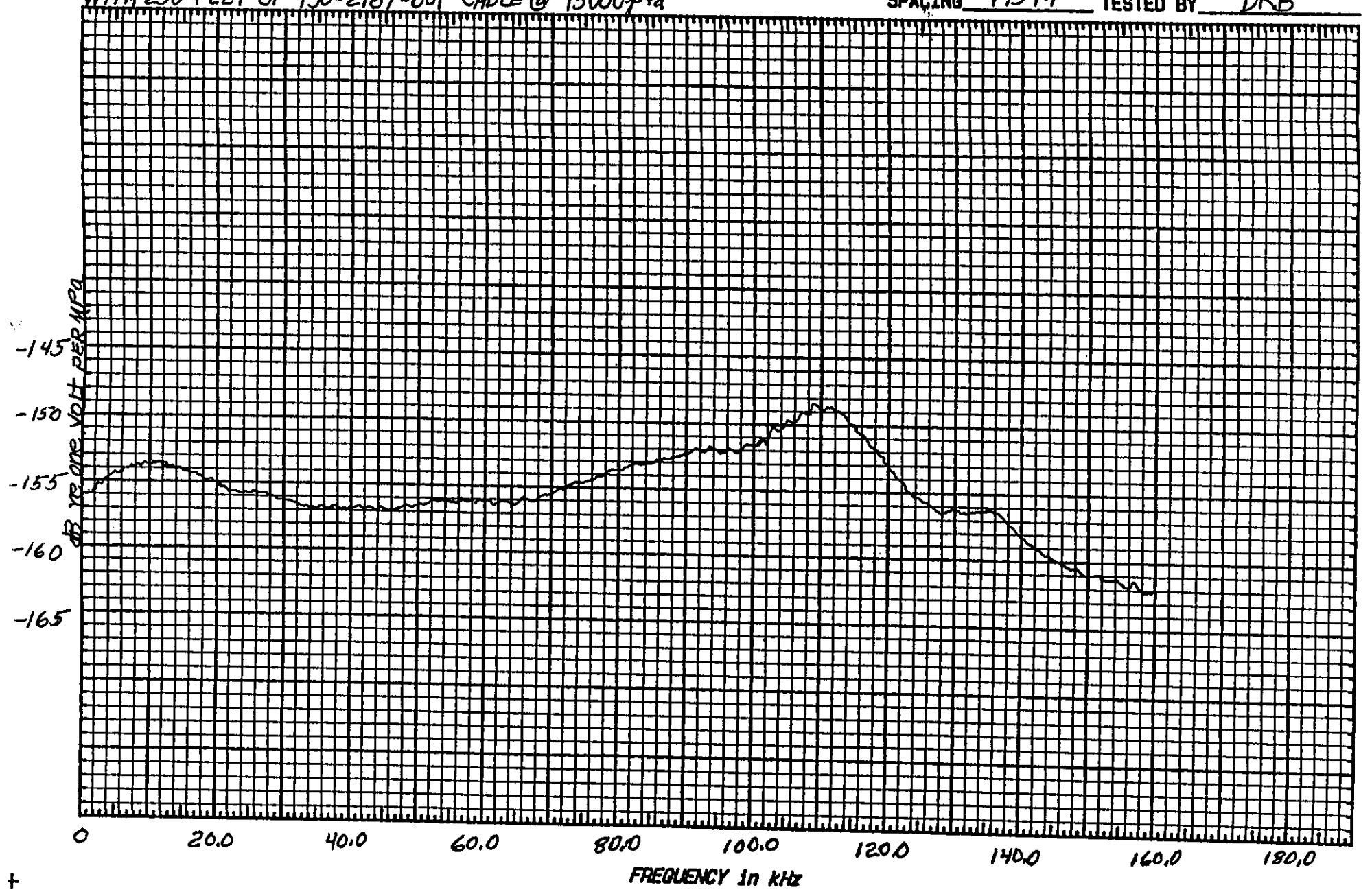
SAMPLE GATE 1.0mS RECEIVE DELAY 1.5mS

WATER TEMP 66 F DEPTH 2.74M

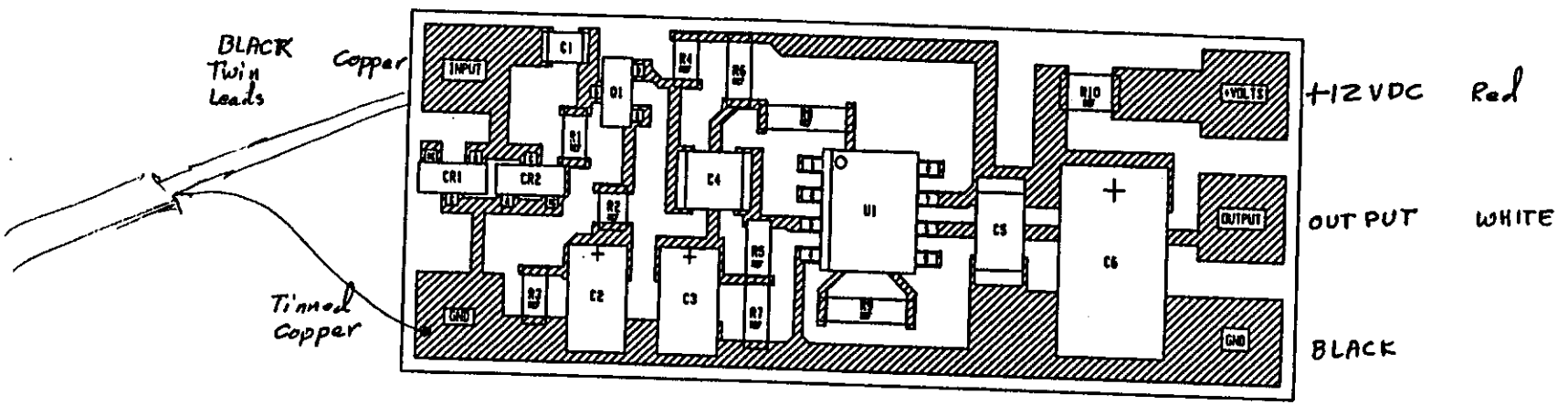
SPACING 1.5M TESTED BY DRB

100 HERTZ AIR SENS, -182.5 dbv/mpa

WITH 250 FEET OF 130-2187-001 CABLE @ 15000pfd



SPARTON ELECTRONICS



SIZE	FSCM NO	DRAWING NO	REV
D	92268	130-0098	
SCALE 10/1		SHEET 2 OF 5	

MULTI CIRCUIT

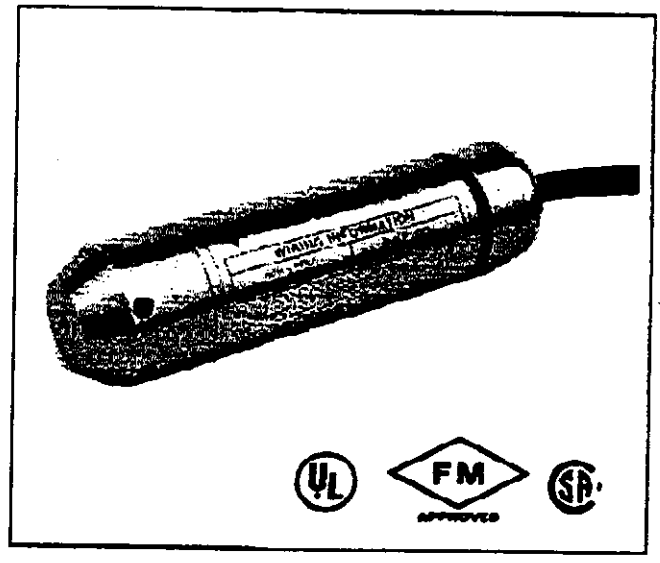
Appendix E

Pressure Specification Sheets

URM6EC120

KPSI

SERIES 200 S and 210 S SUBMERSIBLE PRESSURE TRANSDUCER



APPLICATIONS

- Stormwater
- Well Monitoring
- Dewatering
- Dams
- Pump Control
- Lift Stations
- Drydocks
- Water Towers
- Slug Tests
- Reservoirs
- Soil Remediation
- Irrigation Ponds
- Tank Level
- Oceanographic Research

FEATURES & BENEFITS

HIGH STATIC ACCURACY & REPEATABILITY:
Guarantees reproducible measurements.

WELDED 316 SS CONSTRUCTION:
Trouble-free operation for demanding applications.

SMALL RUGGED PACKAGE:
Withstands severe environmental conditions.

CALIBRATED & SERIALIZED:
Insures performance and NIST traceability.

BROAD SELECTION OF PRESSURE RANGES:
A standard range for your specific requirements.

UNIQUE CABLE SEAL SYSTEM:
Ensures water-tight integrity.

FULLY TEMPERATURE COMPENSATED:
Accurate data over extreme temperature excursions.

INSTRUMENTATION SIGNAL COMPATIBLE:
Operates with popular dataloggers, displays/controllers, SCADA and computer data acquisition systems.

OPERATION

Series 200 S and 210 S transducers are specifically designed to meet the rigorous environments encountered in liquid level measurement applications. These transmitters provide repeatable, precision depth measurements under the most hostile conditions. These units are designed for installation in a Class I, Division 1, Groups A, B, C, and D, Class II, Division 1, Groups E, F and G, Class III, Division 1 hazardous location when connected to appropriate Stahl apparatus.

All KPSI transducers incorporate our isolated diaphragm sensors which are specifically designed for use with hostile fluids and gases. These sensors utilize a silicon pressure cell that has been fitted into a stainless steel package with an integral, compliant stainless steel barrier diaphragm. This sensor assembly is housed in a rugged 316 SS case which provides for a variety of pressure inputs as well as electrical output connections.

Series 200 S and 210 S transducers feature high performance internal signal conditioning. They are available in both 4 to 20 mA and 0 to 5 VDC output versions. Units are identical except for static accuracy.

Each KPSI transmitter is shipped with a calibration card specific to that transmitter. The card specifies I/O conditions as well as actual data reflecting the unit's static accuracy and thermal characteristics. Custom calibration is available for those applications where more extensive data is required.

01

KPSI

Carola

Fax Fax Fax

To : Chris Pacheco

From : Kenneth D. Lenz KL

Company: University of New Hampshire
Ocean Engineering Department
Durham, NH 03824
Phone : (603)862-4482
Fax : (603)862-0241

Date : 02/20/96 11:20 AM

Pages: 3
(Including Cover)

Subject : Quote ; Series 200S Submersible Pressure Transducer

Dear Mr. Pacheco:

Thanks for your call.

I have listed below pricing and delivery information and included the engineering specifications for our series 200S submersible pressure transducer. If more information is needed, please call me at (800) 328-3665.

Item	Qty	Description	Unit $	Total $
1	1	Series 200S submersible pressure transducer Range : 0-250 psis ~~Output: 4-20mAdc~~ 0-5V DC 3wire Specifications per data sheet	$595.00	$595.00
2	3 ft	Polyurethane cable	$1.65/ft	$4.95

Delivery : 4 weeks ARO. Expediting services as follows :
10 business days : $30.00 each transducer
5 business days : $60.00 each transducer

FOB : Hampton, VA 23666

Terms : Net 30 days upon approved credit, Master Card / Visa.
Quotations are in U.S. funds, valid for 60 days and subject to KPSI's standard terms and conditions.

KDL 2002

KPSI

CHANGE ORDER

BILL TO ADDRESS
CO: U OF NEW HAMPSHIRE
DV: PURCHASING
A1: ELIZABETH DEMERITT HOUSE
A2:
CY: DURHAM
ST: NH CT: USA
ZP: 03824-3561
SHIPPING / TERMS / PO# / BUYER
SD: 3/13/96
FB: HAMPTON
BYR: CHRIS PACHECO

PH: 603-862-4256
FX: 603-862-0241
FN: CHRIS
LN: PACHECO
DT: 2/28/96
FU: 3/1/96
RP: HOUSE
SV: UPS GROUND
TERMS: NET 30
PN: 603-862-4256

SHIP TO ADDRESS
CMP: U OF NEW HAMPSHIRE
DIV:
AD1: ATTN: CHRIS PACHECO
AD2: 24 COLOVOS RD
CTY: DURHAM
STA: NH CNT: USA
ZIP: 03824
PO: URM6EC120
SO:
FAX: 603-862-0241

SHIP UPS GROUND, PREPAY AND ADD.

****PLEASE NOTE: EPEDITED SHIPMENT - 10 WORKING DAYS

Order Acknowledgement
Please Review, Sign & Fax to 619-967-0563

Christopher N Pacheco 29FEB96

THANKS,

CAROLA

Itm	Qty	Unt	Series	Pressure	Output	Unit $	TOTAL $
1	1		SUB 200S, 0-250 PSIG, 0-5 VDC			$595.00	$595.
2	3		POLY CABLE, 3 FT ON 1 EA OF ITEM 1			$1.65	$4.
3	1		SERIES 810, VENT FILTER			$0.00	$0.
4	1		2 WEEKS EXPEDITED DELIVERY			$30.00	$30.
							$0.
							$0.
							$0.
							$0.
							$0.

Post-it® Fax Note 7671 | Date 29FEB96 | # of pages ▶ 1
To KPSI | From Chris Pacheco
Co./Dept. | Co. U N H
Phone # | Phone # (603) 862-4482
Fax # 619-967-0563 | Fax # (603) 862-0241

PD: 200S KP: KVB AN: EAST CG: W Oc/Hm: O TOTAL: $629.
UN: 1 RD: T: S: R

SERIES 200 S and 210 S SUBMERSIBLE PRESSURE TRANSDUCER

PERFORMANCE

Pressure Range..... 0-5 through 0-300 PSIG, PSIS
Static Accuracy* ±0.25% FSO BFSL (200 S)
±0.1% FSO BFSL (210 S)
Thermal Error**..... 0.05% FSO /° C worst case
Proof Pressure..... 1.5 X rated pressure
Burst Pressure..... 2.0 X rated pressure
Resolution..... Infinitesimal

*Static accuracy includes the combined errors due to nonlinearity, hysteresis and nonrepeatability on a Best Fit Straight Line (BFSL) basis, at 25°C per ISA S51.1.

**Thermal error is the maximum allowable deviation from the Best Fit Straight Line due to a change in temperature, per ISA S51.1

ENVIRONMENTAL

Comp. temp. range..... 0° C to 50° C
Operating temp. range....-10°C to 60°C

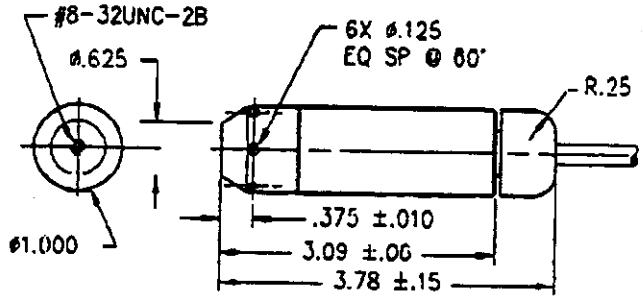
ELECTRICAL

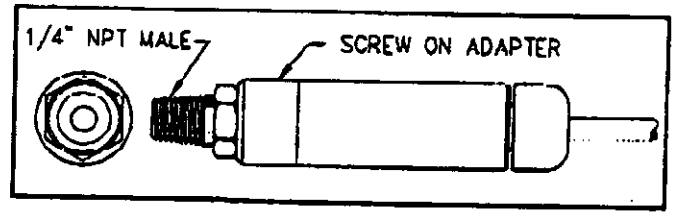
Excitation..... 9 to 30 VDC
Input Current..... 20 mA maximum
Output..... 0-5 VDC (3 wire)
Wiring Red = +, Black = -
White = Signal
4-20 mA (2 wire)
Wiring Red = +, Black = -
Zero offset, max..... 0-5 VDC: ± 60 mV
4-20 mA: ± .12mA
Output impedance..... <10 ohms
Insulation resistance..... 100 megohms at 50 VDC
Circuit protection..... Polarity, surge, shorted output

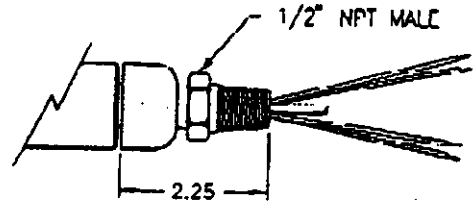
PHYSICAL

Weight..... 7 oz (not including cable)
Cable..... Polyurethane jacketed shielded cable with polyethylene vent tube and Kevlar tension members. 200 lbs pull strength. Conductors are 22 AWG. Approximate weight is 0.04 lb/ft. Tefzel jacket optional.
Pressure connection..... Ported nosepiece with provision for attaching weights. Optional field removable 1/4" Male NPT screw-on adapter.
Wetted materials..... 316 SS, fluorocarbon

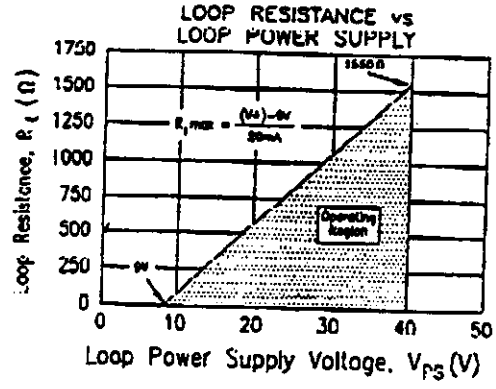
Note: Consult factory for highly corrosive media, tighter tolerances on environmental specifications and special low/high pressure applications.







OPTIONAL 1/2" CONDUIT FITTING SHOWN INSTALLED



(Dimension in Inches)

California: 800-328-3665 / 619-967-6066 / FAX: 619-967-0563
503 Vista Bella #11, Oceanside 92057
Fax-on-Demand Service: 619-967-8363

Virginia: 800-678-7226 / 804-865-1243 / FAX: 804-865-8744
34 Research Drive, Hampton 23666
Internet: kpsi@crash.cts.com

Rev. C (12/94)

KPSI
Keller PSI, a Pressure Systems Inc. Subsidiary

KELLER-PSI CALIBRATION REPORT

Customer: UNIVERSITY OF NEW HAMPSHIRE
Model No: 2005-130-00250
Serial No: 961710
Pressure Range: 0 to 250 PSIG
Excitation: 9-30 VDC
Output: 0-5 VDC

Test Date: 03-01-96
Test Excitation: 15 VDC
Test Temperatures: Room = 25 C
Cold = 0 C
HOT = 50 C
PCU SERIAL# = 0400

Test Pressure PSIG	BFSL Rm Temp Outputs	Run #1 Rm Temp Outputs	Run #1 Error %FSO	Run #2 Rm Temp Outputs	Run #2 Error %FSO	Run #3 Cd Temp Outputs	Run #3 Error %FSO	Run #4 Ht Temp Outputs	Run #4 Error %FSO
-0.0010	0.031	0.0289	-0.050	0.0288	-0.052	0.0130	-0.015	0.0413	0.009
50.0053	1.031	1.0315	0.002	1.0318	0.008	1.0228	-0.007	1.0436	0.010
100.0138	2.031	2.0328	0.028	2.0330	0.032	2.0317	0.000	2.0443	0.010
150.0024	3.031	3.0322	0.024	3.0324	0.028	3.0377	0.005	3.0435	0.010
199.9998	4.031	4.0300	-0.016	4.0304	-0.008	4.0419	0.009	4.0413	0.008
249.9973	5.031	5.0263	-0.086	5.0267	-0.078	5.0440	0.011	5.0368	0.005
199.9872	4.031	4.0318	0.025	4.0321	0.031	4.0420	0.009	4.0417	0.009
149.9994	3.031	3.0328	0.037	3.0326	0.033	3.0375	0.005	3.0426	0.009
100.0070	2.031	2.0333	0.041	2.0329	0.033	2.0314	0.000	2.0440	0.010
49.9775	1.031	1.0321	0.026	1.0325	0.034	1.0235	-0.006	1.0426	0.009
0.0000	0.031	0.0290	-0.048	0.0294	-0.040	0.0131	-0.015	0.0409	0.008

Maximum Static Error: -.086 %FSO

Maximum Non-Repeatability: 0.008 %FSO

Maximum Thermal Error @ Cold: -.015 %FSO/C

Maximum Thermal Error @ Hot: 0.010 %FSO/C

Electrical Termination: Red +Input
White +Output
Black Common

D. G. O'Brien, Inc.

QUALITY CONTROL TEST REPORT

CUST. P.O. _____
CUST. ITEM NO. _____
CUST. DWG. NO. _____
CUST. SPEC NO. _____

DGO JOB NO 6201
DGO PART NO. _____
DGO ITEM NO. 001 & 002
DGO PC SHT NO. _____
DGO PART NO. Services

DGO TEST PROCEDURE PER G. SEDOR

A) HYDROSTATIC PRESSURE TPLK-105 REV B PASS ✓ FAIL ___
HYDRO AT 500 PSIG FOR 30 MINUTES CLOSED FACED. THERE WILL BE NO LEAKAGE, MECHANICAL DAMAGE OR IMPAIRED ELECTRICAL CHARACTERISTICS

EQUIPMENT USED	CALIBRATED	CAL DUE
PRESSURE GAUGE T- 305	4-4-95	4-4-96

TESTER Mark Francenchini DATE 3-27-96

B) CONTINUITY TPCR-105 REV D PASS ✓ FAIL ___
PERFORM CONTINUITY THROUGH EACH CONDUCTOR USING A BEEPER.

C) WITHSTANDING VOLTAGE TPHP-105 REV F PASS ✓ FAIL ___
PIN TO PIN 350 VAC OR 500 VDC, PIN TO SHIELD AND PIN TO BODY 350 VAC OR 500 VDC, SHIELD TO SHIELD AND SHIELD TO BODY 350 VAC OR 500 VDC. HOLD EACH TEST FOR 1 MINUTE. AC OR DC POTENTIAL CAN BE USED WHEN TESTING CABLES. THERE WILL BE NO BREAKDOWN OR FLASHOVER DURING THE TEST. WHEN PERFORMING HYPOT TEST AND THE CABLE HAS NO SHIELDS DISREGARD ALL TESTING PERTAINING TO SHIELDS.

D) INSULATION RESISTANCE TPIN-105 REV C PASS ✓ FAIL ___
IR > 1000 MEGOHMES AT 100 VDC,2 MINUTE HOLD MAXIMUM. TEST EACH PIN, CONDUCTOR OR SHIELD TO ALL OTHER PINS, CONDUCTORS OR SHIELDS AND BODIES.

EQUIPMENT USED	CALIBRATED	CAL DUE
B) BEEPER	NOT REQUIRED	NOT REQUIRED
C) HYPOT T- 220	5-16-95	5-16-96
D) MEGOHMMETER T- 221	16-2-95	4-2-96

TESTER Mark Francenchini DATE 3-27-96

QUANTITY TESTED 3 Items

SERIAL NUMBERS N/A _____ _____ _____ _____

_____ _____ _____ _____ _____ _____

_____ _____ _____ _____ _____ _____

QTR54FMOD

X. Acknowledgments

This work was the result of research sponsored in part, by the National Sea Grant College Program, NOAA, Department of Commerce, under grant #NA56R60159 through the University of New Hampshire/University of Maine Sea Grant College Program.

The AMP team would like to thank the following who contributed to the success of this project:

Kenneth C. Baldwin Phd. Project advisor, UNH Durham, NH

The AMP team would like to express thanks to Dr. Baldwin for his assistance in the completion of this project. Dr. Baldwin on several occasions met with the design team in the initial stages of the AMP design and through his efforts and knowledge of the project requirements was able to direct the group to the proper solution. The team also greatly appreciated the confidence Dr. Baldwin showed in the team throughout the design and fabrication of AMP.

Thomas Krasuski OE Graduate student, UNH Durham, NH

Tom donated a great deal of time to working with the group in the electrical and computer software development aspects of the project.

Gerald Sedor Phd.,PE UNH Professor, D.G. O'Brien connection

The group would like to express our appreciation to Dr. Sedor and the employees of D.G. O'Brien for their generous donation of time and hardware. D.G. O'Brien provided the group with underwater cable connectors which were crucial to the operation of the AMP deployment, use, and recovery.

David E Dunfee General Manager D.G. O'Brien

Mr. Dunfee provided support for the project with his generous donation of both materials and time. With his consent, the group was able to obtain the proper equipment to make AMP appear as a professional system.

Steve Christenson AIRMAR Technologies

The group would like to thank Mr. Christenson for his time and generous efforts in sealing and potting of the hydrophone mount used by the AMP system. The services provided by AIRMAR were greatly appreciated.

Heilind Electronics

Heiland Electronics donated 1000 feet of six conductor-3 twisted shielded pair cable. This cable donation saved the group a great deal of money.

Jon Scott ME Graduate student; UNH Durham, NH

John assisted the group in the purchasing of equipment. The AMP team would like to thank him for his support. John Scott's quick turn over time helped expedite the purchasing process.

Paul Lavoie Marine Diver Safety, UNH Durham, NH

Paul Lavoie provided the group with advice and also provided pressure tests on the probe using the University's hyperbaric chamber. His support enabled the group to test the system in-house in a controlled environment.

Robert Champlin, Supervisor, UNH Machine Shop, Durham, NH

Mr. Champlin provided the skilled machining required in this design. His support and experience provided the group with both professionally machined hardware and skilled advice.

Marine Programs & Ocean Engineering Department, UNH Durham,NH

The AMP team would like to thank these departments for the use of their equipment and facilities throughout the duration of this project.

Adam Perkins, EE department technician, UNH Durham, NH

The AMP team appreciates Adam's patience and help in use of test equipment and in hardware recommendations.