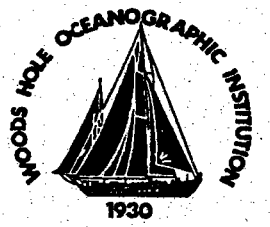
WHOI-91-01

Copy 1

Woods Hole Oceanographic Institution



The High Resolution Profiler (HRP)

User's Guide and Software Modifications Documentation

by

Ellyn T. Montgomery

January 1991

Technical Report

Funding was provided by the Office of Naval Research
under Contract No. N00014-89-J-1073.

WHOI-91-01

The High Resolution Profiler (HRP)

User's Guide and Software Modifications Documentation

by

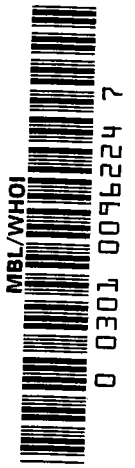
Ellyn T. Montgomery

Woods Hole Oceanographic Institution
Woods Hole, Massachusetts 02543

January 1991

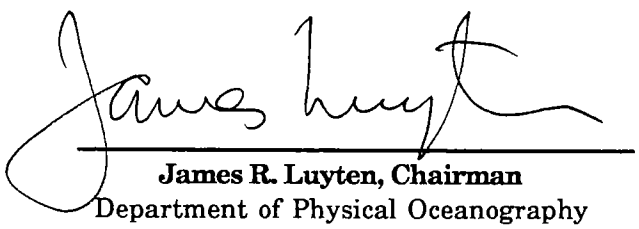
Technical Report

Funding was provided by the Office of Naval Research
under Contract No. N00014-89-J-1073.



Approved for public release; distribution unlimited.

Approved for Distribution:



James R. Luyten, Chairman
Department of Physical Oceanography

Table of Contents

List of Figures

Abstract

This report provides a User's Guide for operation of the High Resolution Profiler (HRP) and documentation of the software structure and recent modifications.

The HRP is an instrument that acquires and stores up to 16 types of physical oceanographic data. A profile is logged as the HRP falls through the water column during each deployment. It controls its dives based on user-specified trigger levels input during a pre-cast setup. Communications, the setup process, and how to check out and run the profiler are described fully. Also included are the current sensor configuration and summaries of assembly, mechanical preparation, tracking, data transfer and processing.

During 1990, the software controlling the HRP was almost completely reworked in order to port VRTX (Versatile Real Time eXecutive) to the HRP. This was accomplished to facilitate use of a fast data link that was being developed. Other modifications were made to the software to improve the user interface, to allow use of up to 16 Mbytes of Random Access Memory, to speed up the serial interface, and to fix previously undetected problems. In addition, the use of an altimeter to determine height above bottom was added to the dive control logic of the profiler.

Overview:

The High Resolution Profiler (HRP) is a free-fall oceanographic microstructure measurement device. It was designed as a vehicle to make high quality fine and microstructure measurements using the interface bus computer (IBC), developed at WHOI by Ed Mellinger. The HRP was first used in FASINEX (Frontal Air-Sea Interaction Experiment) in 1986. A schematic of the HRP is shown in Figure 1. For additional information on the development of the HRP, see the paper by Schmitt *et al.*, 1987.

A HRP drop consists of: setup, which provides dive controlling parameters; deployment; tracking during the descent; weight drop and ascent; recovery; and data transfer to another machine. Data acquisition can occur in two modes: fine and micro, with the transition between them triggered by the CTD's (conductivity-temperature-depth sensor's) pressure reaching user-defined threshold values. The sampling in both modes is driven by a 200 Hz interrupt: in fine mode (sampling at 10 Hz or every twenty 200 Hz ticks), CTD data is collected on cycle 18, analog data on 19 and both stored on cycle 20, while in micro mode, micro structure data is collected every cycle, along with continuing the fine scale measurements. All the data collected is stored internally in a 16 Mb RAM (random access memory) mass storage area. Communications with the HRP are accomplished using SAIL (Serial ASCII Instrumentation Loop).

The IBC is the HRP's controller, handling everything from software setup to data acquisition and storage. A suite of sensors interfaced to the IBC provide data on the physical properties of the water column through which the HRP descends. The IBC itself is based on the Intel 8086 chip, has Multibus architecture, and was designed to fit on a series of six inch diameter cards, in order to fit in a pressure housing for underwater applications (Mellinger *et al.*, 1986). The HRP was the prototype application for the IBC when it was developed in 1985. At that time a monitor and other software were written to control the HRP's operation (there is no real operating system, so interface with the computer is achieved through the monitor). The original code was written by Ken Prada and Ed Mellinger, using C and some assembly language routines. The compiler used was Computer Innovations, and the original software will be referred to in this report as the CI code.

One of the problems with operating the HRP at sea was that transferring the data from memory (RAM) to an archival and processing machine (a DEC MicroVAX, originally) took too long. A 1000 m cast took about one hour to complete, then transferring the data took nearly two hours at 4800 baud which was the best rate obtainable during development. For the next use of the HRP, as part of the 1990 Warm Ring Inertial Critical Layer experiment (WRINCLE), a fast data link using Proteon Pronet hardware was planned.

The work on the fast data transfer was done in conjunction with the Deep Submergence Laboratory (DSL) at WHOI and Electronic Data Systems (EDS) in Houston. The DSL uses IBCs in their equipment, but use newer software to operate them. The plan chosen was to have EDS make hardware for the IBC based on the Proteon Pronet system, and software to drive it would be developed on the DSL system.

The DSL IBC software evolved from the HRP prototype, but uses VRTX (Versatile Real Time eXecutive) to allow near real time processing. The VRTX system is fairly complex, based on

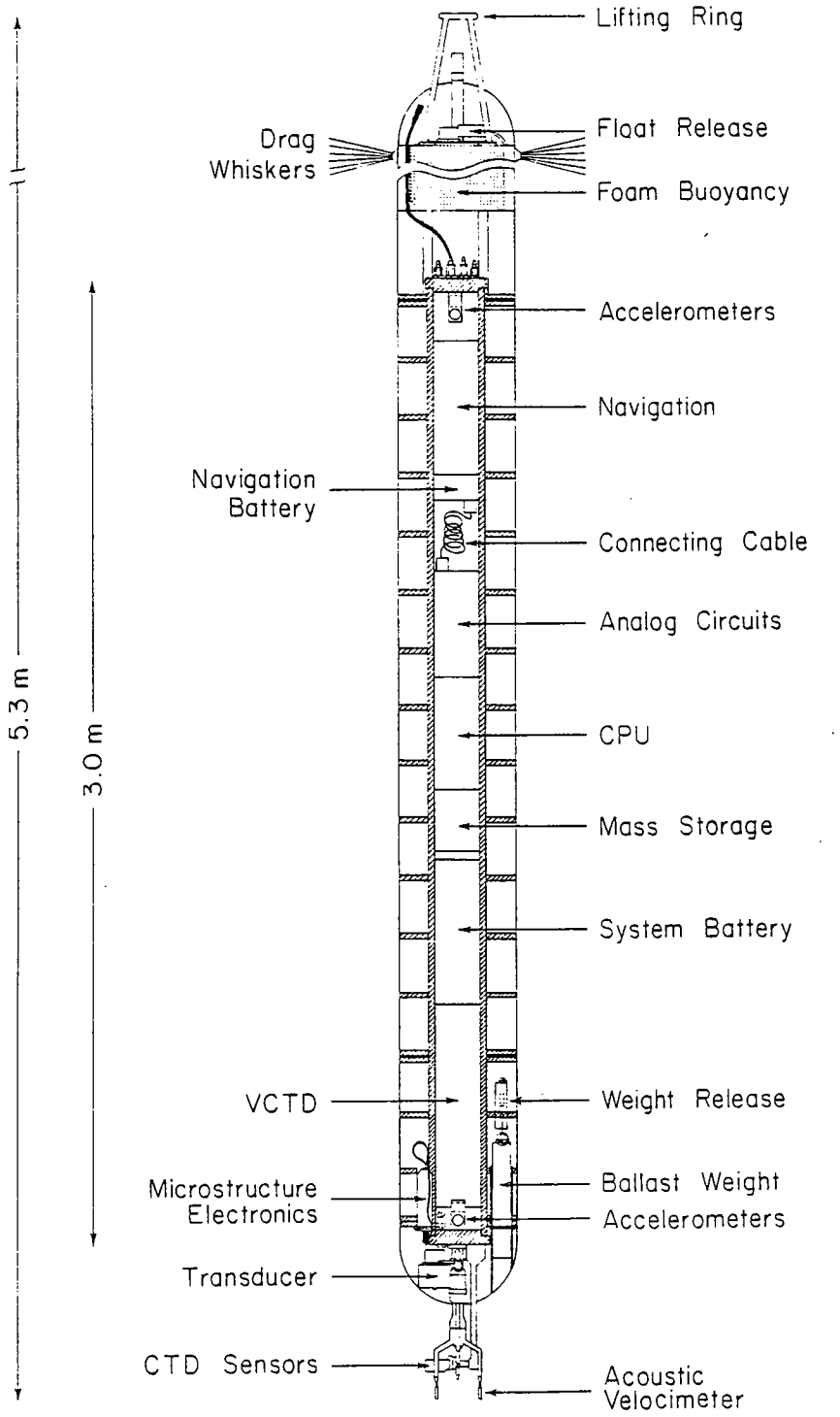


Figure 1: Schematic of High Resolution Profiler

swapping tasks very quickly to approximate a multitasking environment. For this application, the key things to know are that VRTX operation is based on interrupts occurring 10 times a second (10 Hz), and that there are libraries of VRTX functions available for use in software development. The software for the IBC using VRTX will be called the VRTX code in the rest of this report to differentiate it from the original CI code. This is not intended to refer to the code that makes up the VRTX environment, it only indicates an application written for the IBC using VRTX. Beyond this, no extensive understanding of how VRTX works is required to understand this report, so readers who are interested in the details of VRTX are referred to the VRTX documentation supplied with the software by Ready Systems (Hunter and Ready, 1984).

In order for the HRP to use the fast link system being developed, the software controlling the profiler's operation had to be converted to VRTX and the new 6 inch Pronet card and software installed. Then a Pronet card could be installed in the MicroVAX, and the data transfer could occur at greater speeds. Other changes implemented include expanded memory and the addition of a bottom finder. The purpose of this report is to document how to operate the instrument, the tools used to modify the software, the changes made and the reasons behind it all. Part 1 is the user's guide, and Part 2 contains the software development documentation.

PART 1 : USERS GUIDE

Introduction:

This part of the report includes information needed to operate the HRP. The mechanical as well as software set up will be presented, along with some discussion of the launch and recovery methods used.

Sensor Configuration:

In order for the HRP to run correctly, it has to have its sensor configuration entered during the setup process. Usually 12 fine and 4 micro sensors are used. The sensors' configuration, corresponding A/D channels and offsets currently used are the following:

	A/D channel	offset	multiplier	divisor
fine sensors				
pressure	0	0	1	10
temperature	0	0	1	2000
conductivity	0	0	1	1000
accelerometer, top X	0	0	1	1
accelerometer, top Y	1	0	1	1
accelerometer, bottom X	2	0	1	1
accelerometer, bottom Y	3	0	1	1
acoustic current meter, X velocity	4	0	1	1
acoustic current meter, Y velocity	5	0	1	1
X magnetometer	6	0	1	1
Y magnetometer	7	0	1	1
A/D ground	14	0	1	1
micro sensors				
micro conductivity	10	0	1	1
micro temperature	11	0	1	1
shear X	12	0	1	1
shear Y	13	0	1	1

The CTD has its own A/D converter so does not have channels assigned to it. The CTD and Acoustic Current Meter (ACM) are both made by Neil Brown Instrument Systems, the accelerometers are made by Sunstrand, the micro temperature and conductivity probes are made by Seabird, and the shear sensors made by Edward Scheimer, of Baltimore, Maryland, according to Rolf Lueck's design. Some of these sensors are mounted directly on the endcap, which is attached to one of the electronics racks. The rest are attached separately to the instrument and plugged into the endcap.

Assembly Summary:

The assembly of the HRP is a complicated process that will not be discussed in detail in this report. The general procedure is as follows:

1. The upper electronics rack and connecting cable are loaded into the pressure case and sealed.
2. The top module is secured, lifting bail and flow hood attached.
3. The plastic skin covering the upper area is adjusted, and cables run between pressure case and skin.
4. The upper and lower electronics racks are plugged together, and the lower rack slid in, with the air in the case displaced with freon before tightening the end cap.
5. The sensors not attached to the end cap are mounted, and each is plugged into its specific connector.
6. The lower skin is adjusted and the lower flow disturbance reducing hood is secured.

Operational Details

Shipboard deployment/recovery

The HRP is handled from a rolling cart or cradle. Figure 2 shows the HRP's cradle and handling rig. A set of angle aluminum tracks bolted to the deck controls the HRP's movements from its maintenance position to its launch position on the stern. A hydraulic lifter is used to raise the HRP from horizontal to the vertical position so that it dangles off the stern. A winch attached to the hydraulic rig lowers the instrument into the water, where a quick release completes the deployment. The HRP is tracked in the water as it descends, using a Line Scan Recorder. The HRP pings every 20 seconds during descent, once a second for 20 seconds at weight release, then changes to every 5 seconds during ascent. A Northstar radio tracking system helps to locate the HRP on the surface. Despite the bright yellow cap, it is difficult to spot in rough weather. Once spotted, the ship moves to the HRP's position and drives slowly by so that the profiler passes along near the starboard rail and can be hooked by lines with snap hooks attached to long poles. The HRP is guided to a position where the line from the winch can be attached. The HRP is pulled out of the water, settled in its cradle, and lowered back to horizontal.

Mechanical preparations

Before each cast, weights have to be loaded into the two releases. The release mechanisms should be cleaned every 10 dives. If a float is to be used on deployment, it must be installed with bungie cord on the top release as part of the preparations. The light and radio on the HRP should be checked. Also the drag whiskers should be replaced when sheared off, and adjusted periodically. After the software setup is complete, the cable must be unplugged and both ends capped with dummy plugs. The plug on the HRP activates a run timer that will trigger release 80 minutes after it is plugged in. This provides a backup mechanism for weight release in case the software-driven

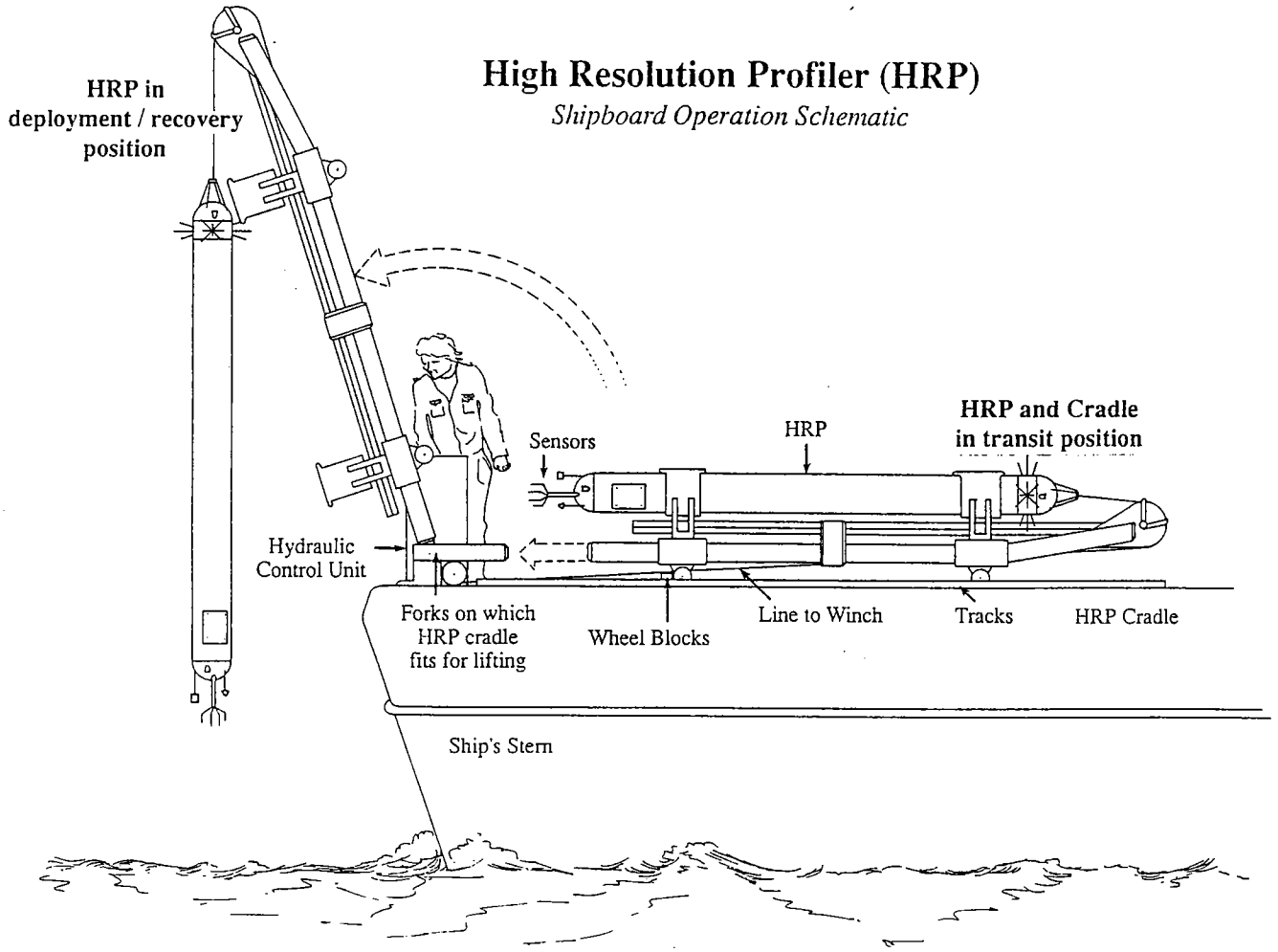


Figure 2: Diagram of HRP Handling Rig

release and shear pins malfunction. The protective sensor covers should be removed before the instrument goes in the water.

Software setup-

The operation of the instrument is determined by the parameters supplied in the software setup. A step-by-step set of instructions on running the HRP is supplied in Appendix 1. The following will be a brief description of the process.

Communication with the HRP is achieved through its monitor which uses serial communications via a SAIL current loop. Any communications software can be run on the PC provided it can be set for 9600 baud, 7 data bits, even parity, and 1 stop bit. During program development, when new code is downloaded and run in RAM, use IBT.EXE to interface with the HRP's monitor. <Esc T> allows an Intel hex file to be downloaded. PROCOMM is most often used when doing ordinary HRP runs when download capability is not needed.

The software setup process is designed to ascertain that the sensors are still functioning; and to enter the dive specific information. Before setting up for a new dive, make sure that any data in the mass storage RAM has been removed, as some parts of setup overwrite previous data. Checking that the sensor configuration is correct is a good idea at this point (option 6 in setup menu). All the A/D channels (0-14) should have offset 0, slope multiplier and divisor 1. The CTD is the same except the pressure divisor should be 10, the temperature divisor 2000, and the conductivity divisor 1000.

The next step would be to go into the tests menu and examine several cycles of CTD data and look at several channels of A/D. This information is usually logged to the printer for the cast notebook. The information obtained above is used on the log sheets (battery reading and CTD deck pressure especially). The actual dive parameters are set in setup option 2. The position, cast number, start pressure for the dive, pressure for starting micro, stop pressures, depthfinder start pressure and stop range, and comments are all entered here. Unfortunately, if a key entry error is not corrected on the line made, the whole of option 2 must be redone. There is no way to choose and fix a single item. Option 6 can be used to view the header. When the setup is completed successfully, exit the setup menu and choose profiler option 2 to start the cast.

All that needs to be entered to start a cast is the number of minutes to wait to start, and the number of minutes to let the cast run before overtime termination. The countdown starts when the second carriage return is entered. At that point, SAIL should be powered off, the cable disconnected, and the deployment commenced. A value between 3 and 5 minutes is about right to allow the last minute preparations to be done and get it in the water. For 1000 meter dives, the descent time was about 30 minutes, so a value of 30 to 45 minutes for duration is good. Be sure that the duration is not so large that the HRP would hit the seafloor before it timed out, in the case that the pressure trigger did not stop the dive.

Post dive data transfer-

Once the HRP is on deck after a dive, the data must be removed from memory and stored elsewhere. Profiler option 3 handles the HRP end of the transfer, and rcv4k.exe handles the PC

end. Rcv4k has to be run once for each file transferred. If the previous file transfer terminated normally, the HRP should not be hung, and rcv4k can be restarted immediately. Otherwise, the HRP end of the transfer has to be restarted first. The program hangs more often with a full or much fragmented disk, so the PC should be backed-up and unfragmented regularly.

Once the files are on the PC they should be transferred to the MicroVAX for final archival and processing. The next cast may be started before the files are transferred to the VAX, but the transfer should be done as soon as possible thereafter. The details of the transfer to PC and subsequent transfer to VAX are presented at the end of Appendix 1.

The shipboard processing was done with CTD group software modified for the HRP. Plots of the fine and microstructure variables are made, and stored in the cast notebook. The plots allow a quality control check, and suggest areas of interest for heavier sampling. The shipboard processing will not be discussed in depth, but the general procedure is outlined below.

1. Run pro2ctd_n.exe on the raw fine data file to convert to CTD78 format files (also converts to scientific units).
2. Edit finequal.com (a plot5 command set) and use it to create the fine plot
3. Run pro2ctd_n.exe on the raw micro data file
4. Run plot5 with "do micplt4" to create the micro plot
5. Make hardcopies of both on Zeta plotters with the "zplot" command.

The preceding paragraphs supply information on the operation of the HRP. Appendix 1 supplies a more step-by-step description of the user interface of the HRP. There may be ambiguities left that only the author can help with, so queries are welcomed.

PART 2: SOFTWARE DOCUMENTATION

Project Goals:

The main goal of the project was to implement VRTX as the operational environment for the HRP while maintaining all the original interface and functional components. The change in software was driven by the need for the fast data link being developed by EDS and DSL for use under VRTX. Other tasks were to modify software to allow access to 16 Mbytes of mass storage RAM instead of 4 Mbytes, to add a Datasonics depthfinder to the HRP and integrate its data into the dive control logic, and to achieve greater speeds for the serial data transfer.

Method:

Since the VRTX software for IBC was developed using the CI code, there were many programs in common. The approach taken was to start with a working VRTX IBC application, and append to it the routines from the HRP software that are specific to the operation of the HRP, and make other changes as necessary.

The software used was from the Tethermoor buoy. The first step was to remove the routines specific to Tethermoor, leaving the startup code and the monitor. The next step was to identify and modify items in the CI code needed for operation of the HRP for use with the VRTX code. Then the .mak files had to be redone to accommodate the new files and directory structure.

The development environment with the VRTX code included the Microsoft C compiler version 4.0, the Microsoft assembler, the Systems and Software Link and Locate method of code generation, and the Multimake utility to automate the compilation and linking of all the programs and functions to create the final Intel hex format file. Different make files were used to handle making ROMable (read only memory) or RAM downloadable files. The WHOI technical memorandum by Ann Martin (1989) covers the procedural details for working with VRTX on IBC's at WHOI.

The CI software generation was managed by DOS batch files; the Computer Innovations C compiler was used; the Microsoft assembler and a program called ROMGEN were used to create Intel hex format files. ROMGEN modified the Intel hex format file to make it usable from either ROM or RAM. This information is included because the CI code remained in ROM to allow downloads throughout development, and had to be updated occasionally to agree with changes made in the VRTX code.

Software Structure:

Both sets of software are based on a three part structure: 1) initialization of devices, interrupts etc., which is invisible to the user, 2) the monitor, which provides a user interface to the computer, and 3) application specific programs. Part 1 is vastly different between CI and VRTX, mostly due to overhead associated with having VRTX involved. Part 2 is pretty similar in structure and function between the two sets of code, and part 3 is the application specific code which is not expected to be similar. Figure 3 shows a schematic representation of how the software is now structured for the HRP under VRTX.

Schematic of HRP Software Organization under VRTX

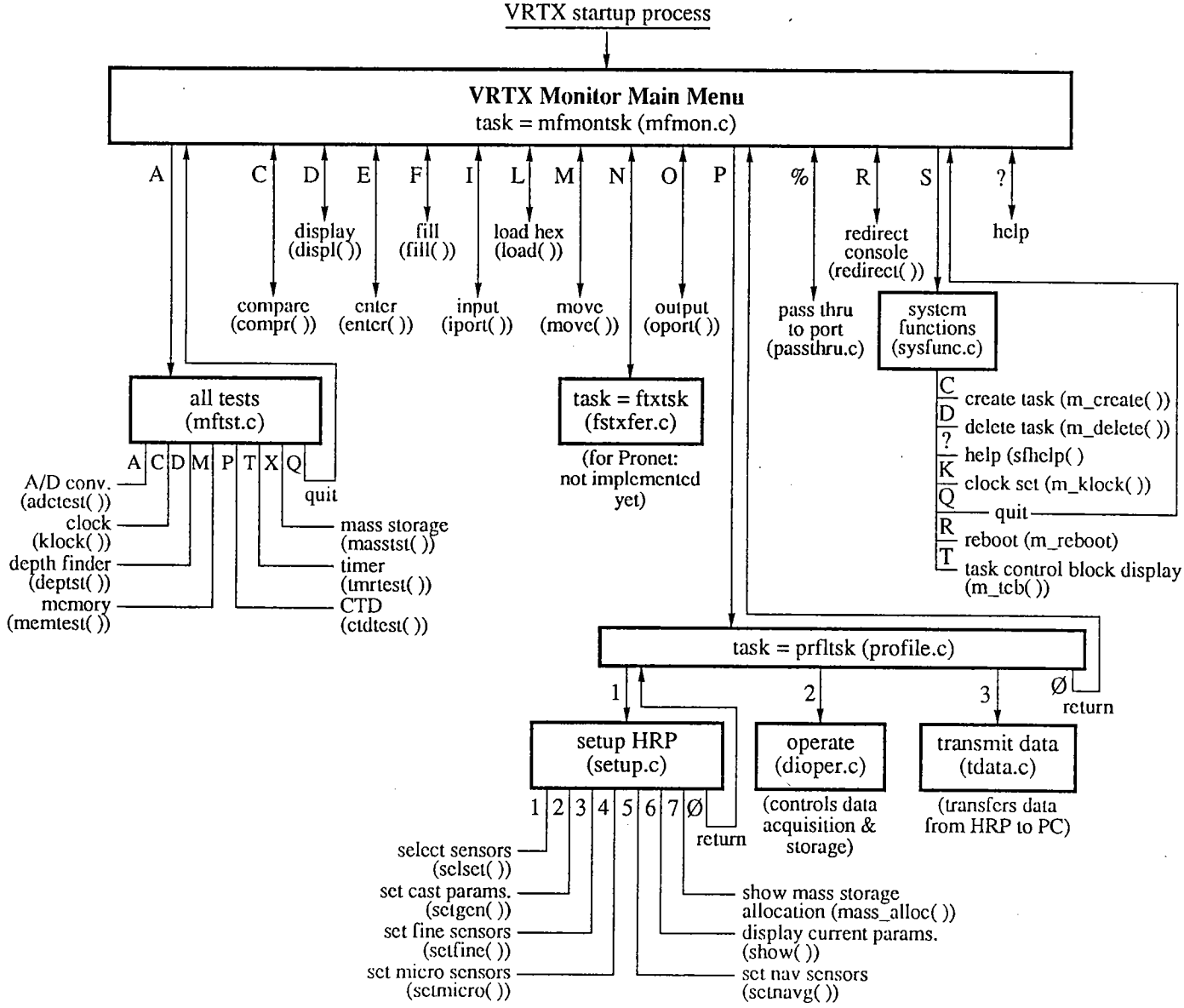


Figure 3: Chart of HRP Sofware Organization under VRTX

A list of the steps taken in order to accomplish the conversion is presented in Appendix 2. The general process will be described below with the major problems encountered and their solutions. The order will be generally temporal, starting with getting the monitor up and progressing to speeding up the data transfer.

Before starting the software modification description, some more specific details about VRTX in this application and about the HRP are needed. The IBC VRTX code assumes the following interrupt assignments:

1 – VRTX
2 – CL_DEV (SAIL loop, for communications)
3 – FSK_DEV
4 – OC_DEV
5 – UND_DEV

That left interrupt 6 available in the interrupt table for use with the 200 Hz interrupt which controls data sampling. The CI code used interrupt 1 for the 200 Hz interrupt, so the original profiler references to interrupt 1 had to be changed. Also, the 200 Hz interrupt that controls the sampling had to have the highest priority, so that the chip had to be set up to make interrupt 6 highest followed by 7,0,1,2.... Since interrupts 7 and 0 are not assigned in this application, the VRTX interrupts had the second highest priority.

The startup portion of the VRTX code is shown in Figure 4. This is provided primarily as a roadmap, if an application gets into the monitor successfully, this level of detail can be ignored. In the HRP, the multitasking capability of VRTX is not used. Only one task (the monitor) exists at startup; when the P(rofile) option is chosen a separate task is created, and the monitor task deleted. When the profile task is complete, the monitor task is restarted and the profile task terminated. Multitasking may be implemented in the future for the data transfer task, so that the transfer can go on in the background, while the next profile is set up.

The structure of the HRP's memory is also important to understand. The HRP has three kinds of memory: RAM, ROM, and bank switched mass storage RAM. ROM contains the program executed at power up or reset. Conventional RAM contains the interrupt vector table, the data used by the program, some VRTX code, and downloaded programs. The HRP's data is stored in and retrieved from the mass storage RAM. Figure 5 shows the structure of the memory as typically used for the HRP and how the mass storage is allocated for the four data types. Martin (1989) and Mellinger *et al.* (1986) shed more light on IBC memory maps in general and the IBC's memory cards respectively.

Software Modifications Synopsis:

All of this development was done with consistency in mind, so that anyone at WHOI who works with IBCs could work on any application without having to figure out where everything is and how a given application is structured (all other IBC at WHOI already use VRTX).

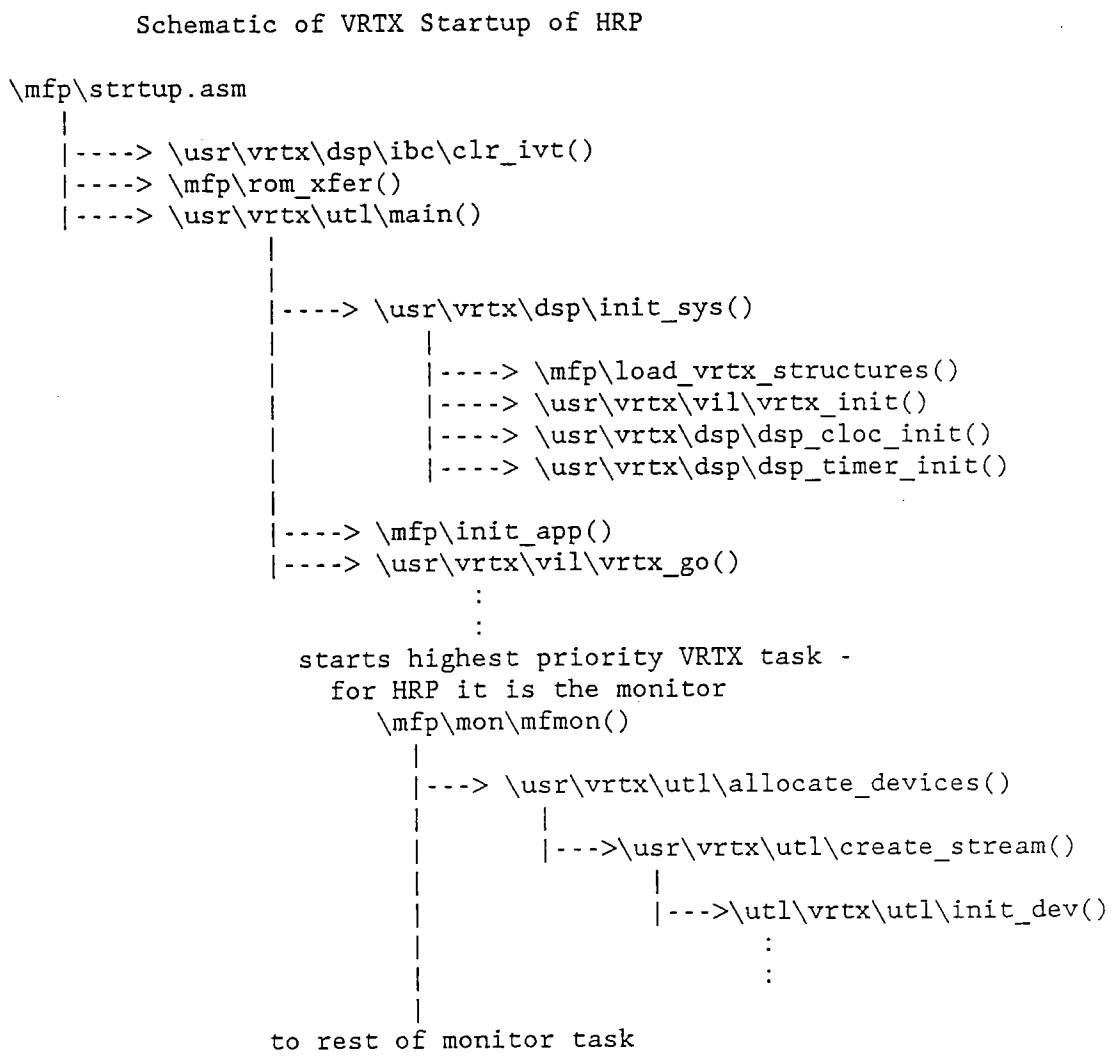
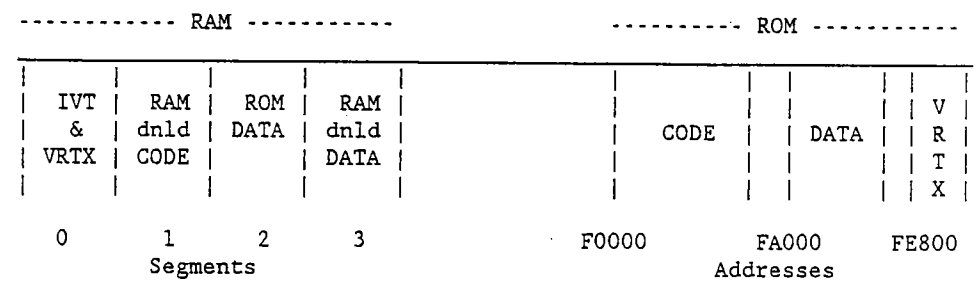


Figure 4: VRTX Startup Function Sequence

** addresses given in the form where FA000 is equivalent to F000:A000 **

RAM addresses		(size)	ROM addresses	
code:	10000	8C00	code:	F0000
data:	30000	3700	data:	20000
rom_xfer:	1A000		rom_xfer:	FA000
VRTX:	1E800		VRTX:	FE800
mass stor:	40000		mass stor:	40000

HRP conventional memory -



mass storage start
address = 40000

HRP extended memory
(bank switched)

uses 4 boards with 4 Mbytes each
to store dive data

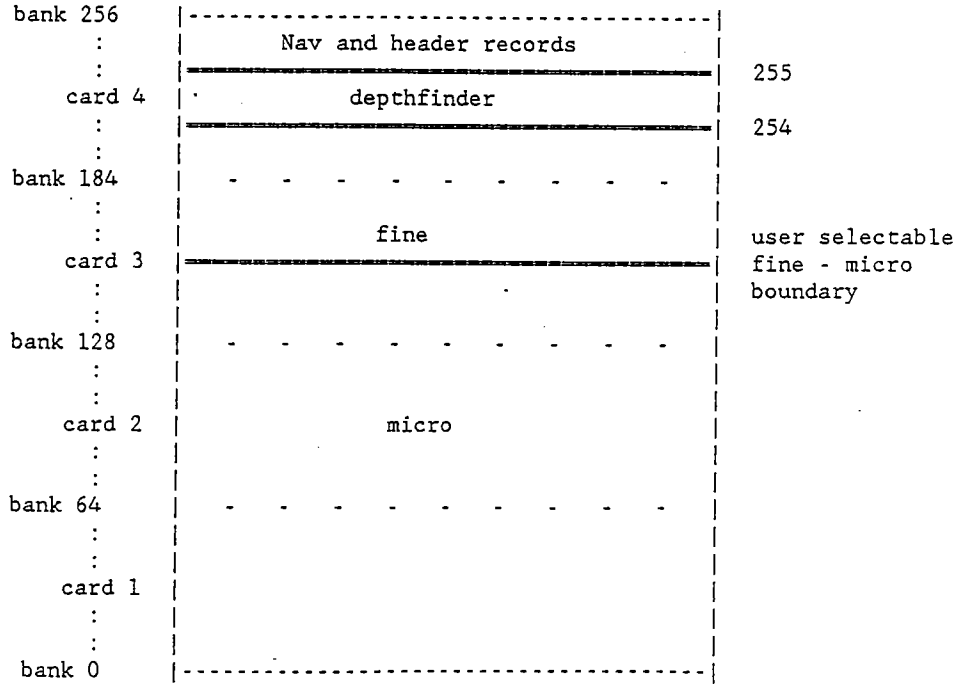


Figure 5: Diagram of Memory Allocation for the HRP

The initial task was to make the HRP wake up running VRTX. This involved removing the three assembly language programs that got the CI code to the monitor, and assuring that all the component routines were where they were expected to be, and that all the referenced variables were declared. The Microsoft C compiler requires different methods of variable declarations and initialization than CI, so a fair amount of time was spent putting the HRP specific variables into the correct places in the VRTX code. This process is detailed in items 1–22 of Appendix 2.

The most interesting part of getting VRTX started had nothing to do with VRTX. The HRP running CI used UART 1 addresses 20–23 for SAIL. The applications using VRTX have SAIL running off UART 2, addresses 24–27. Since the application in ROM isn't seen by the RAM application, it was assumed that SAIL could be run from either UART depending on which application was controlling the HRP. It turned out that without major changes (hardware and software), using both UARTs for SAIL was impossible. It was easier to convert the CI code to using addresses 24–27 than changing all the references in the VRTX code. Once this was done, the monitor came up successfully.

Next came testing to see where incompatibilities came up. Many of the problems were related to the difference in how the old and new PCs handled screen display. Displays that would overwrite existing information no longer worked, so some method of scrolling had to be implemented. Another problem discovered in this phase was the stack increment value used in CI assembly modules was for use with Lattice. The Microsoft Large model is used with the DSL VRTX code, so those references had to be changed from 4 to 6, which cleared up problems with the stack pointer indicating the wrong place. Items 23–31 detail the work done as part of the initial testing.

The modifications required to allow varying amounts of memory to be used were implemented at about this point. The user now specifies the maximum number of banks to use (up to 256) and the other locations are derived from this. Some flexibility in hardware configuration is gained from these changes also.

Assuring that the actual data sampling and storage worked correctly was the next task. Initially, the sampling would hang up shortly after starting. This was because the sensors had not been set up, and therefore the number of bytes per record was fantasy, which caused the program to attempt continuous writing to mass storage, which finally hung the program. Simply running through the set-up menus before starting a profile allowed the sampling to run longer. Then as longer runs were attempted, it started hanging up again. It took quite a while to discover the circumstances when the interrupts interacted and hung the machine. Changing the source of the VRTX interrupt was the solution used, and since then the interrupts have worked fine. Other problems related to changing the startup timing were encountered, and fixed, as were things associated with changing to a 386 PC for the development environment (some of the marginal CI modules failed because of the different BIOSs). Items 35-49 of Appendix 2 cover the operational testing and debugging.

The final thing to work on before the WRINCLE cruise was finding a way to get data out of the profiler. The Pronet card was not going to be ready, so serial data transfer across the SAIL loop became the method to use on the cruise. The original method using the MicroVAX was tested for use as the backup, but because of the slow speed (9600 baud maximum), it was more of a backup

option than a first choice. Transfer to a PC at 19200 or 38400 baud seemed like a reasonable goal. Jim Doutt wrote a program to handle the PC end of the transfer. The HRP data transfer routine also had to be modified to remove BIOS calls that had controlled the transfer. The two programs were debugged in time for use on the cruise. The serial transfer speed improvement work is described in items 52–57 of Appendix 2.

Pre-cruise in-water tests were done at the WHOI pier once VRTX Proms were made. The HRP team went to sea for 21 days in March and April 1990, and completed 78 1000-meter HRP drops. A WHOI Technical Report by Schmitt, 1990, (in prep.) will give more details of the cruise. The HRP worked well, though several bugs were found that slipped through the testing. The worst of these was a problem writing to certain parts of the mass storage memory. Doing the 1000 meter drops was ok, but when deeper ones were attempted, it appeared that some micro data wrote over the beginning of the fine data, and sometimes the header. This turned out to be due to improperly placed jumpers on the mass storage cards, and once replaced, the memory addressing worked correctly. Items 58–64 describe the pre-cruise in-water testing, and 65–71 detail the post-cruise modifications.

Finally, in preparation for the next cruise for the Abrupt Topography initiative, a Datasonics 900 depthfinder was incorporated into the HRP. This device determines the range in meters from the bottom acoustically. Having this data as an added termination condition will enable sampling nearer the bottom without risking the sensors, than pressure termination alone would allow. Implementing the depthfinder as a second task of equal priority to the profile task was the original approach. It didn't work as hoped, so a second approach, that of using interrupts was tried. Interrupt 3 was assigned to the depthfinder, and the software to control the interrupt was successfully integrated into the HRP's operating code. Now the HRP software gives range from the bottom and pressure equal importance in terminating a dive: whichever threshold is passed first will cause weight release. The trigger values have to be set carefully in order to get close without crashing. As with the profiler set up, the depthfinder data is converted to tenths of meters, since the IBC has no floating point capabilities.

To summarize, the HRP is now fully functional under VRTX, and compatible with the other IBC-Based instruments at the Institution. It is a platform able to use the Pronet card for a very fast transfer, when it is finally completed. In the interim, enhancements such as the enlarged memory, faster serial data rates, and use of the bottom finder make the HRP a more useful instrument than it was originally.

Acknowledgements:

The help of Steve Merriam and Ed Hobart of the Advanced Engineering Laboratory was invaluable in getting all the kinks worked out of this IBC application. Their good humor and technical insights were greatly appreciated. Richard Koehler's assistance with working out the hardware problems, Raymond Schmitt's editorial comments on this report, and Veta Green's help in preparing this document are all appreciated. This project was funded by ONR grant # N00014-89-J-1073.

References

Hunter and Ready, Inc., 1984. VRTX/86, Users Guide Document #591113001. Ready Systems, 445 Sherman Ave., P. O. Box 60803, Palo Alto, CA. (415) 326–2950.

Martin, A., 1989. A software guide to using an IBC with VRTX. WHOI Technical Memorandum # 2–89, 47 pp.

Mellinger, E. C., K. E. Prada, R. L. Koehler, R. W. Doherty, 1986. Instrument Bus: an Electronic System Architecture for Oceanographic Instrumentation. WHOI Technical Report 86–30, 86 pp.

Schmitt, R. W., J. M. Toole, R. L. Koehler, E. C. Mellinger, K. W. Doherty, 1987. The developement of a Fine- and Microstructure Profiler, *J. Atmos. and Ocean. Tech.*, **5**, 484–500.

Schmitt, R. W., Oceanus 218 - Cruise Report. WHOI Technical Report (in prep.)

Appendix 1

How To Set Up And Run The HRP

SET UP:

1. Plug the cable at the top end of the profiler into the long four-conductor cable with the SAIL connection and two loose wires on the other end.
2. Plug the other end of the cable into the SAIL box (it doesn't matter which way), and the white wire to the black terminal of the reset box, and the black wire to the yellow terminal of the reset box.
3. Make sure either the battery is connected, or there is 30V (−15V to +15V) power supplied from some external source.
4. The white with blue and purple striped wire coming out of the IBC backplane should be connected to the plug on top of the serial card.

COMMUNICATIONS:

1. Use Procomm software to talk to the IBC (invoke from e:\hrpdata on Ellyn's 386) and use <Alt P> to set communications parameters to 9600 baud, 7 data bits, even parity, one stop bit, COM1. IBT or any of several other communications packages can also be used.
2. Press the reset button on the grey box and a "SAIL address?" prompt should appear. Enter "#MFP01".
3. At this point you will be in the monitor.
4. The next query is how many banks to use. When in doubt, answer "256". This number can be adjusted to allow missing memory to be avoided, or to enlarge or shrink the total memory, depending on the hardware configuration. When all 4 cards are installed with 64Kb on each, 256 is the correct answer.

POWER UP SOFTWARE INSTRUCTIONS:

At power up (after changing the battery, of if power is otherwise lost) it is necessary to set the System Clock and copy it to the Real Time Clock. Also if the time displayed looks weird, and the C(lock) option under A(ll tests) doesn't fix it, do the following.

1. From the "mon>" prompt, enter "S" for system.
2. Enter "K" for Klock menu.
3. Enter "V" to view.

4. If the time shown is correct, use 2 "Q"s to exit this menu and skip to # 9 on this list.
5. Enter "D 01 12 1990" to set date (dd mm yyyy format). DO DATE FIRST, as a reset happens at the end of the Time set option.
6. Enter "T 08 00 15@" to set time, (use hh mm ss@), then don't hit the CR until the actual seconds agree with the time entered. Then a reset will happen and the "SAIL address?" prompt will appear
7. Enter "# mfp01" to start the monitor.
8. Enter "256 <cr>" in response to the query unless the hardware configuration is different from that described under communications #4.
9. Enter "A" to chose All tests.
10. Enter "C" to choose Clock options.
11. Enter "S" to copy the newly set system clock to the real time clock (confirm with "y").
12. Enter "V" to view and confirm the real time clock is set correctly.
13. Now the setup can proceed as described below, starting at #5.

NORMAL OPERATION - Setup and Start Profile:

1. After entering "256 <CR>, the "mon>" prompt will be displayed.
2. Enter "A" for All tests.
3. Enter "C" for Clock.
4. Enter "R" for copy Real time clock to system clock, and enter "Y" if you really want to do it.
5. Enter "Q" for exit clock options.
6. Enter "A" for A/D tests.
7. Enter the digit of the desired channel number to be displayed.
8. Enter "S" repetitively for single readings— the first value will be No Good.
9. Enter "C" for continuous display, but you have to reset to get out of this option.
10. Enter "Q" to exit the channel specified.
11. If you want to see more channels, enter the channel's number, otherwise enter "-1" to exit.
12. Enter "P" to choose display of the CTD sensor data.
13. Enter "S" to display single scans, the first cycle's data is never good.

14. If desired, enter "C" to display continuously. Again a reset is required to get out of the continuous display.

15. Enter "Q" to return to the test menu.

16. Other options may be chosen from the test menu, such as mass storage memory test, RAM, etc., but they are not part of the normal startup procedure.

17. Enter "Q" to exit the test menu and return to the monitor menu.

18. Now that the instrument's sensors are checked out, the dive can be started.

19. Enter "P" to go to the Profile menu.

20. Enter "1" to start the Setup menu.

21. Enter "6" then "F" to examine the fine header and sensor setup.

22. If the channels are not set up, the Setup options 3, 4 and 5 must be used to set the parameters for the fine, micro and nav sensors. These must be set up for the acquisition and logging to work correctly. See User's Guide text for the HRP's normal sensor configuration.

23. If the sensors are set up and there is no data to be dumped in the mass storage banks, the next dive's header can be set up. To do that enter "2", and answer the questions. The only question that is not pretty obvious is the one about # blocks micro file. This refers to how much memory to set aside in mass storage at the beginning for the micro data. In the HRP's normal configuration there is about 5 times more micro than fine data collected. Therefore, if there are 256 available banks, and the profile runs fine and micro for its complete length, # blocks micro should be about 210.

    To compute the actual # of blocks of memory needed for a cast, assume a .6 m/sec fall rate for the HRP. For micro, you get 8 bytes 200 times/second for 1600 bytes/second. If micro data is to be collected for 1000 m: (1000 m/.6 m/sec) x 1600 bytes/second = 2,666,666 bytes/1000 m. Then divide by 65,535 bytes/bank to get 40.6 banks or 41 banks per 1000 m micro cast. For fine, you get 240 bytes/sec; so [240 bytes/sec x (1000 m/.6 m/sec)]/65,535 bytes/bank gives you about 7 banks required for 1000 m of fine data. So to accommodate this sample cast, you could set # of micro blocks as low as 41 and have all the data fit.

24. Enter "0" to exit the setup menu and return to the profile menu when ready.

25. To start a cast, enter "2" at the profile menu.

26. Enter the number of minutes to wait before releasing the float and the number of minutes to allow for the dive. The countdown actually starts the moment the <cr> is hit after the second question.

This is the time to disconnect the SAIL cable, dummy up the plug on the profiler, and deploy the HRP.

Data Dump to PC:

Once the profiler is secured on deck after a cast, the following procedure should be used to remove the data from RAM.

1. Plug the SAIL cable into the cable at the top of the profiler.
2. Restart Procomm on the PC (in E:\ hrpdata).
3. Hit the reset button on the grey box, and the "SAIL address?" prompt should appear.
4. Enter as above:

```
#mfp01
  256      <CR>
   a
   c
   r
   y
   q
   q
   p
```

5. At the profiler menu, enter "3" to start the data transfer.
6. Enter "38400" for baud rate to use.
7. Enter <cr> to signal to continue.
8. Enter "alt F4" to go to the Procomm DOS shell.
9. Enter "rcv4k" to invoke the PC end of the transfer.
10. Enter "38400" for baud rate query (or whatever was entered at the HRP's prompt earlier).
11. Enter "f", "m", "d" or "n" for fine, micro depth or nav data, depending which you want. (DO NOT enter a <CR> here — that will make it use the default file name.)
12. Enter the desired file name for the data to be transferred.
13. If the name entered already exists, rcv4k will ask whether to overwrite, append, or re-enter. If the transfer hangs, the block it hung on is displayed by rcv4k. If append is chosen, this number should be entered at the next query.
14. Correct functioning of the program is indicated by the number at the lower left of the screen changing about every 20 seconds, and being a number like 33000. The middle light on the SAIL box will light up and blink green and red as the data is being passed. If the light is not blinking, something is wrong. Try again, starting from a reset.

15. When the first file is transferred, enter "rcv4k" again to start the transfer of the next data file.
16. Repeat for the Depth and Nav files.
17. Occasionally when the transfer hangs, you have to return to the HRP and restart the data transfer option there.
18. After the data is transferred to the PC, it should be transferred to the MicroVAX for processing.

Data transfer to VAX using telnet and FTP:

Two examples are shown here for 386 PCs with different configurations

1)	On Ray's PC:	On Ellyn's PC
	>c:	>c:
	>cd pcdc	>ncsa
	>init	
	>telnet 128.128.16.156	>telnet 128.128.16.156
	(for puce)	
2)	Hit return, and login at the prompt.	
3)	so now you're at a PC, but logged on to the VAX.	
	Enter "Alt e" to start a DOS shell.	
4)	Change directory to where the data actually is:	
	>d:	>e:
	>cd \pccomm	>ncd hrp
5)	Enter "exit" to get out of the DOS	"
	shell and return to VMS.	
6)	Start an ftp session: (this connects from the VAX	
	to the PC:) ftp twins	ftp zafod
	<CR>	<CR>
7)	Set to transfer binary data by entering: "set type image"	"
8)	Start transfer by entering: "get filename"	"
	If you transfer the fine file first, the processing	
	can be started while micro, nav and depth are transferred.	
9)	Repeat #8 for each of the four files from the cast.	"
10)	Enter "exit" to get out of ftp and return to VMS	"
11)	Use Pro2ctd_n to convert the files to ctd format, and go from	"
	there with the processing.	

Appendix 2:

List of Software Modifications Associated With Conversion to VRTX

1. Compared contents of same-named files between CI and VRTX, and noted differences, especially for cgen.c and cutils.c.

2. The HRP uses a 9516 A/D converter, and Tethermoor used a 7107. Therefore ibcports.h had incorrect definitions for the HRP A/D. #ifdef AD9516 was added the file to differentiate between the tw0 sets of definitions. The corresponding #define AD9516 was added to mftst.c.

3. The HRP was originally configured to use UART address 20 as the base to use with SAIL. To work with the VRTX software, the hardware was changed to use 24 as the base, and to agree with the .h files, ubase=24 is used instead of base=20 from CI.

4. Changed hardware and software to allow the VRTX interrupts on 1, and the 200 Hz interrupts on 6. In the CI version, the 200Hz interrupts were interrupt 1...

5. Peek (a Prada routine) was used in the get_ functions. Changed to using Microsoft's peekb.

6. Added vstruct.h, vrtxcgen.h, and msc_ok.h to #include lists of HRP specific programs to provide needed parameters.

7. For the change from CI to Microsoft compiler, added a type declaration to all the functions. (i.e. disphed() changed to void disphed() in setup.c)

8. Created prfl.h and mftst.h to contain external program declarations, and added these to the appropriate programs' #include lists.

9. Created data_ mfp.c to contain variable and structure initializations. (the declarations were made in header.h & prfl.h). The #define AD9516 and #define MFP also went in here.

10. Since a mfpagen.asm existed under VRTX (a subset of CI agen.asm), moved the modules used of those remaining to vautilsb.asm (the VRTX version of autils.) As part of making the CI assembly language code work under VRTX, had to add setstk, pbegin, cseg, pend and some other macro commands to the CI code. The changes made are lower case, while the original CI assembly language commands are in uppercase.

11. In TMRSRV, the routine vectored to on 200 Hz interrupts, had to use "CEXTERN intsrv", and "Call FAR PTR intsrv", to get it to find intsrv successfully, a C function in profile.c. This is due to the Microsoft rules of mixed language programming.

12. Added HRP specific options to the main monitor menu, and to the tests sub-menu. Also changed some of the letter choices so that HRP users could continue to use original letters. In main menu, P = Profile (from CI), in VRTX, P = passthru — now % = passthru, and P = Profile.

13. Removed the gofor() and trace() options brought from CI. They are not tenable under VRTX.

14. Had to initialize the 200 Hz interrupt during setup. Added ivt_install() line to load_vrt.c. This is not the elegant way to do it, but making it elegant would have entailed re-making the VRTX libraries, which would only be done if absolutely necessary, because of multiple versions of some of the programs being available.

15. In operate.c, the 200 Hz interrupts are "turned on" by changing a mask value to let the desired ones go to the CPU. Anded the mask value with 0xbf to make sure interrupt 6 would get through.

16. Make files for each subdirectory of mfp were made, appropriately referencing the programs there. Compilation occurs in the subdirectories, and linking everything is controlled from mfp1ram.mak in \mfp.

17. Once the make worked well enough to produce .hex and .mp2 files, the code was downloaded to RAM for testing. The CI code was in ROM, and a program called term was used for communications. Term had an option to download to RAM, <esc T>, which puts the intel hex file in the locations specified in the file.

18. After the download, the data segment must be manually moved from RAM to imaginary ROM so that rom_xfer.c can move it back. This procedure is strange but driven by the needs of the program when running from ROM, for simplicity of making, rom_xfer is left in for RAM testing, but the data segment has to be relocated. So after download, M2000:0000,3200,1000:A000 has to be entered, which takes 3200 bytes from 2000:0000 and copies them to 1000:A000. It took a while to get the hang of this. Rom_xfer, mfp1ram.loc, and the Move command all have to agree, and there has to be space available for the move without writing over other data.

19. If the above two steps worked, entering G1000:0 should start the program in RAM. The "SAIL address" prompt is displayed if the VRTX monitor starts successfully.

20. The program that actually runs VRTX was not in the libraries originally linked (the Tethermoor application pulled VRTX in from elsewhere). The libraries were remade to include the real VRTX code. The procedure for doing this under Link and Locate is described below:

```
>masm realvrtx                                  ! create a .obj file
>rename realvrtx.obj _vrtx_code.obj
>lib                                            ! put the .obj file in a library
        libname: tmp1.lib
        operation: + _vrtx_code.obj
        listfile: tmp1lib.lst
>cnvlib tmp1.lib realv1.llb                     ! convert to L&L form
>xlib86                                         ! invoke L&L library utility
        pclib> d ibcvdsp.llb(vrtxcode)          ! remove dummy code
        pclib> a realv.llb(realvrtx) to ibcvdsp.llb  ! add the real stuff
        pclib> list ibcvdsp.llb p               ! verify new module
pclib> exit
        realvrtx
        _vrtx_code
>cnvlib ibcvdsp.lib ibcvdsp.llb                 ! convert again
```

21. Re-made mfp1ram with the new ibcvdsp.llb, and got the 1st successful entry in the VRTX monitor, but the display never came up. The program hung in the code diagrammed in Figure 4.

22. With help of the logic analyser, determined that the CI code using SAIL base address 20 and the VRTX version using SAIL base 24 were incompatible, partially due to jumpers on the board. Went back to the CI code and changed the SAIL address to 24, recompiled and relinked, made new proms, changed the jumpers, downloaded the VRTX version to RAM, and it worked!

23. Played with the sequence of task startup/switching. Determined that anything that deals with the display had to happen when only one task was active, so decided to switch tasks rather than use more than one at a time. Monitor, Profile and Fastxfer are the tasks implemented, and when one is created, the calling one is suspended.

24. Discovered that the use of get_time() when 200 Hz interrupts are active messes everything up. Changed all get_time() calls to get_clock(), a VRTX routine, and changed associated display utilities. Timing was better but not perfect.

25. Changed the initialization sequence used on the 82c59 Programmable interrupt controller to give the 200 Hz interrupt top priority, and use edge triggering.

26. Still had some problems with the interrupts clobbering each other. Changed the delay for start of dive code significantly to take advantage of VRTX utilities instead of CI ones. First it has to wait till 0, 20, or 40 seconds to start pings, then has to wait for the actual start time.

27. Changed the general interrupt termination command in TMRSRV to a specific end of interrupt, so that terminating a 200 Hz interrupt wouldn't affect the VRTX interrupts.

28. The program appeared to start sampling appropriately, but never got anything in the files. Discovered that sensors have to be set up (at least as dummies) in order for the routine that writes to mass storage to compute the record size. Otherwise record size is huge, and program tries to write continuously, which fouls up VRTX.

29. Changed tdata() to display a known ascii string and called it etdata(). Used this program to test new faster UARTs, and they are noticeably faster so the monitor's default baud can be changed to 9600.

30. Changed the CI memtst() routine to perform a given number of passes correctly and named the new one xmemtst(). The memtst() in VRTX tests either RAM or ROM, but not the bank switched memory, which xmemtst() covers.

31. The data displayed by ctdtst() under VRTX was different than when run under CI. The stack increment in mirror() (called by ctdin(), called by ctdtst()) turned out to be wrong. It should have been 6 for use with the large model in the compiler.

32. All the data storage and retrieval programs were modified to handle up to 16 Mbytes of total mass storage RAM.

33. Set up a query to find out how many banks of mass storage will be used.

34. The new mass storage boards were tested with xmemtst() and all the possible memory locations were written to and read from successfully.

35. To have the 200 Hz interrupt have absolute highest priority, tried it as interrupt 0. It ran fine initially, but when a hard reset was attempted, it wouldn't work. Changed the interrupt # back to 6 and had to rely on the priority at the 82c59.

36. Masked off the 200 Hz interrupts at the termination of sampling which allows a successful return to the monitor (only possible when testing and the pinger is not needed). Discovered that screen display always got fouled up when attempted with 200 Hz interrupts running. At sea, this is not a problem, as the HRP is not plugged in while profiling, and 200 Hz interrupts are only active during data collection.

37. During sampling, the while loop in operate() cycles fast enough to foul up the VRTX interrupts. Put a call to wait_for() (a VRTX utility in VASL) in the loop that causes the cycling of the loop the slow to about once per second, which is more than adequate for the resolution needed for changing modes of terminating the dive.

38. Lessened the rate of checking dive termination condition from once a cycle to once every 10 cycles (.75 seconds), because get_clock() caused problems when executed in rapid sequence.

39. Changed many of the variables associated with addressing fine and micro files to global so that all the arguments of the functions would not have to be changed.

40. Problems with profile timing seem to be connected primarily with conflict between the 10 and 200 Hz interrupts that occur several minutes into a run.

41. A new AT-386 machine took over the function of the old NEC and AT that were together handling display, compile, link, and download. It turns out that a fair amount of the old CI code is written specifically for the old NEC and the display drivers are different from the current ones.

42. Modified all references to clear(), cursor() and spaces() which seem to be the routines most affected by using the new PC.

43. Started using IBT or PROCOMM instead of TERM to interface with the monitor. TERM uses code that is incompatible with the 386. With the new PC, and IBT, <Esc T> still invokes the download, and the m(ove) command has to follow it immediately, and then G1000:0 to start, as the PROMS are still the CI version.

44. After much work with the oscilloscope and logic analyser, decided to get the 10 Hz signal from the same source as the 200 Hz. Initially each interrupt was generated by a different timer. Setbase() was where another frequency divider was inserted to allow obtaining both from the same source. Some jumpers also had to be changed on the counter timer boards to accomplish this.

45. Also associated with change 44 were several changes to the VRTX definitions in the .h files, and so the programs that were changed and are part of a library had to be replaced in its library with the new version. Dsp_pic_init which belongs in ibcvdsp.llb was the main one that required special treatment. The process used was the same as described in # 20.

46. Changing the source of the 10 Hz interrupts made the profile duration longer though there were still some timing bugs to be worked out.

47. Changed the wait_for() in the operate() while loop to a count loop using register integer numbers so that less of the CPU's attention is used. This improved the reliability of correctly stopping the profile at the specified time.

48. Used the logic analyser to check the timing of the 200 Hz interrupts under the CI code to compare to the current timing. The current interrupts vary by +/−.003 miliseconds, which turned out to be slightly better than the variability under CI.

49. With the operational details going appropriately, it was time to start working on getting the data out of the IBC. The Pronet card for the IBC that drove the conversion to VRTX was not going to be ready for our March cruise date, so the fall back position was to go with serial RS-232 data transfer. The original transfer was designed to interface to a MicroVAX computer at 4800 baud. Either the original software could be used at 9600 baud (the best possible speed on our MicroVAXes) or modify the IBC end to work with a transfer program running on the 386 PC (which should have rates of 38400 or better possible).

50. That the original CI transfer software (IBC to MicroVAX) still worked under VRTX was tested. This also allowed use of some of the Ethernet connection system we planned to use on the cruise. Once this process was ironed out, implementation of the faster PC method was attempted.

51. Jim Doutt was contacted to write the PC data reception and logging software, and I rewrote the IBC end of the transfer program. No restructuring of the way the programs interfaced was done at this point, the original method of sending a block number and having the IBC convert it into an address and send the 1024 bytes of data located there was kept.

52. The CI tdata() program used DOS interrupts to drive the transfer. Some time was spent trying to implement these under VRTX, but there were lots of problems, and the time got so short that conin() and conout() were tried in place of the bdos() calls.

53. Also associated with doing faster transfers, the crystal on the serial board had to be replaced. The fastest the old one could go was 19200, and the newer one could go up to 115000 baud.

54. The divisor values for the various bauds had to be changed to accommodate the change in hardware. These are located in ibcbios.h. At the same time the VRTX defined baud rate specifiers were put into baud() which is invoked in tdata().

55. The UART status tested for by conout() had to be changed to make the program work, but the implementation went pretty smoothly otherwise.

56. Finally the two new programs were tested and debugged to determine that the data was transferred correctly. 19200 and 38400 baud were shown to work with the 10 and 15 minute data acquisition runs done in the lab. Jim's program was well done and the debugging was relatively quick.

57. Next the code that had been downloaded had to be converted into a ROMable form, and PROMs burnt. This entailed changing the addresses used in rom_xfer(), making a mfp2rom from mfp2ram, and making the mfp2rom.loc addresses agree with the rom_xfer() ones.

58. Once a .hex file was successfully created, it had to be edited to do the same thing the m2000:0,2500,1000:a000 command did. The Norton Editor handled this task easily using the global search-for-and-replace function. For the ROM code, the data was put at 20000, and needed to be moved to FA000 (code segment starts at F000:0, and ends F000:Cxxx, and the VRTX code doesn't start till FE000). So with Intel hex, the search string would be: <alt F>:020000020<alt F>:0200000FA, then 21 would be replaced by FB, 22 by FC and 23 by FD, which completely moves the data segment.

59. The proms were then burnt using a BP microsystems EP-1 (the documentation is good on how to use these so the procedure will not be detailed here).

60. The HRP was then assembled for in-water tests. One of the first things discovered was that the RAM download did not work with the VRTX proms. This was not a drastic problem, since the CI proms could be swapped back in for the rest of the pre-cruise modifications and tests.

61. The other problem encountered when assembling the HRP was that the top row of memory SIPs on each mass storage board did not fit in the pressure case. All the SIPs were placed a little off center, and the top ones were the only ones to overhang the cards.

62. This left the mass storage RAM missing banks 56–63, 120–127, 184–192, and the top 8 banks: 248–255. There were several emergency solutions possible, and the most feasible-seeming one was to use 3 of the 4 cards with the lowest addressed board having its 8th sip attached with a flexible connection so that it could hang down in front of the board. This worked out to allow banks 0–119 (on the first 2 cards) to be contiguous to hold micro data, and fine and nav would have the space between 128 and 183 (the third card).

63. After a few other minor glitches, the HRP was lowered into the water, pingers pinging; the float release was heard to unlatch on time for startup, and both pressure and time were observed to terminate the dive successfully.

64. The work of the cruise made some other errors show up. Despite this and foul weather, 78 1000-meter dives were completed before we ran out of expendable spares.

65. After the cruise there were several things to fix. First was to find out why only some of the mass storage RAM addressed correctly. It turned out that the jumpers had been put in the wrong position on each mass storage board, and it remained undiscovered because the lower memory that we used most was accessed correctly, while the higher was not.

66. The rate of pinging on the ascent was changed from once every 20 seconds to once every 5 seconds.

67. The user interface at startup was changed to allow input of # of minutes to wait and # of minutes to allow for the dive instead of entering the absolute times.

68. Because the data transfer process hung up so often at sea, restarting the transfer where it left off would have been helpful and saved time. Consequently, recvr() and getdat() (renamed getdat2()) were modified to time out when a transfer had done nothing for 10 seconds and announce what record number it was on. Then the front end was changed to allow files to be opened for append if the file name matches an existing file, and is confirmed by the operator as the correct action. This also will prevent overwriting files because of typos, as the operator must enter append, overwrite or re-enter.

69. Implemented a mass storage allocation display, so the user can confirm that the memory is allocated as expected.

70. Corrected the problem causing the RAM download not to work with VRTX code proms. Load(), in mfmon.c, is called when <esc T> is entered, and and after much examination with the logic analyser, the problem turned out to be the change made to conout() to make tdata() work fouls things up in load(). The conout()'s were all replaced by outp(..), conin() pairs, which are exactly what the old conout() was.

71. Under VRTX, to start the downloaded code by entering "S" (for system) then C1000:0,1,1 (for Create task at 1000:0, priority and i.d.1). At this point the SAIL address? prompt should appear.

72. With the fixes completed, integration of the Datasonics 900 depthfinder into the system was the next task. The depthfinder was to use the other available UART (One is already used by SAIL for monitor communications and data transfer), so the first project was to set up the UART at address 20.

73. The depthfinder is "programmed" by dipswitches, and can operate by sending at specified time intervals, or by responding to queries. Unfortunately the time taken for the response is not constant (a measurement made far off the bottom will have a longer return time than one close to the seafloor), which complicates the data acquisition.

74. Questions were added to setup() to obtain depth to start the depthfinder, and how far from the bottom to terminate the dive. Operate() (now called dioper()) was modified to set a flag when the trigger pressure was reached, and turn on the depthfinder power at this point.

75. The first approach was to run the depthfinder as an equal priority task with profile, and allow VRTX to handle the timing complexities. Using VRTX tasks to handle the acquisition of the depthfinder data proved to be very difficult and code intensive, so it was dropped.

76. Having the data present at the UART generate an interrupt whenever it was ready was tried next. Interrupt 3 was allocated to this routine, and a program modeled on TMRSRV for the 200 Hz interrupts called DEPSRV was added to vautilsb.asm (renamed dautils.asm).

77. DEPSRV then points to depth(), which actually processes the data from the depthfinder. Depth() waits till an "R" is received, then acquires the next 5 characters, stuffs them into a string, stores the string, and converts the string to an integer for use in dioper().

78. In order to store and access data, new routines putd() and getd() were added to cutils.c (renamed dutils.c), and bank max-2 was assigned to the depthfinder data (max-1 is where the nav and header data is stored). So the routines that access fine data had to be changed so that the end of fine does not write over the depth data, or that it is not retrieved as fine data.

79. Modifications also had to be made to the header structure to allow the size of the depth data to be stored. Tdata() had to be changed to allow another type of data to be retrieved (the nav header is used for general dive information, but the file size is gotten from the general header).

80. Recvr, the PC end transfer program, was modified to allow for the fourth data type and deal with it appropriately.

81. An option was added to the tests menu to allow the depthfinder data to be displayed to the monitor for a general operational test. In order to have the display part work, the 200 Hz interrputs had to be masked out during test operation of the depthfinder (they have to be there during actual operation).

82. In-water testing of the depthfinder and the dive termination logic will be completed soon.

83. Timing tests of the open collector mode of SAIL transfer were done after the second serial board was repaired. The open collector allows the transfer to run at 57600 nominally, but the actual speed is only about 35000 baud. Initially the data transferred this way was garbage, but the addition of resistors to the SAIL box, and a well-placed capacitor on the serial card made the data quality improve. The rate was still not great enough to make using open collector SAIL more desirable than normal SAIL.

84. The rate was improved to nearly 39000 baud at 57600 nominal by transferring 4096 bytes at a time instead of 1024, but more drastic structural changes will have to occur before it improves much more (recvr was renamed rcv4k at this point).

DOCUMENT LIBRARY

January 17, 1990

Distribution List for Technical Report Exchange

Attn: Stella Sanchez-Wade
Documents Section
Scripps Institution of Oceanography
Library, Mail Code C-075C
La Jolla, CA 92093

Hancock Library of Biology &
Oceanography
Alan Hancock Laboratory
University of Southern California
University Park
Los Angeles, CA 90089-0371

Gifts & Exchanges
Library
Bedford Institute of Oceanography
P.O. Box 1006
Dartmouth, NS, B2Y 4A2, CANADA

Office of the International
Ice Patrol
c/o Coast Guard R & D Center
Avery Point
Groton, CT 06340

NOAA/EDIS Miami Library Center
4301 Rickenbacker Causeway
Miami, FL 33149

Library
Skidaway Institute of Oceanography
P.O. Box 13687
Savannah, GA 31416

Institute of Geophysics
University of Hawaii
Library Room 252
2525 Correa Road
Honolulu, HI 96822

Marine Resources Information Center
Building E38-320
MIT
Cambridge, MA 02139

Library
Lamont-Doherty Geological
Observatory
Colombia University
Palisades, NY 10964

Library
Serials Department
Oregon State University
Corvallis, OR 97331

Pell Marine Science Library
University of Rhode Island
Narragansett Bay Campus
Narragansett, RI 02882

Working Collection
Texas A&M University
Dept. of Oceanography
College Station, TX 77843

Library
Virginia Institute of Marine Science
Gloucester Point, VA 23062

Fisheries-Oceanography Library
151 Oceanography Teaching Bldg.
University of Washington
Seattle, WA 98195

Library
R.S.M.A.S.
University of Miami
4600 Rickenbacker Causeway
Miami, FL 33149

Maury Oceanographic Library
Naval Oceanographic Office
Bay St. Louis
NSTL, MS 39522-5001

Marine Sciences Collection
Mayaguez Campus Library
University of Puerto Rico
Mayagues, Puerto Rico 00708

Library
Institute of Oceanographic Sciences
Deacon Laboratory
Wormley, Godalming
Surrey GU8 5UB
UNITED KINGDOM

The Librarian
CSIRO Marine Laboratories
G.P.O. Box 1538
Hobart, Tasmania
AUSTRALIA 7001

Library
Proudman Oceanographic Laboratory
Bidston Observatory
Birkenhead
Merseyside L43 7 RA
UNITED KINGDOM

50272-101

REPORT DOCUMENTATION PAGE	1. REPORT NO. WHOI-91-01	2.	3. Recipient's Accession No.

4. Title and Subtitle
The High Resolution Profiler (HRP)
User's Guide and Software Modifications Documentation

5. Report Date
January 1991

6.

7. Author(s)
Ellyn T. Montgomery

8. Performing Organization Rept. No.
WHOI-91-01

9. Performing Organization Name and Address
Woods Hole Oceanographic Institution
Woods Hole, Massachusetts 02543

10. Project/Task/Work Unit No.

11. Contract(C) or Grant(G) No.
(C) N00014-89-J-1073
(G)

12. Sponsoring Organization Name and Address
Office of Naval Research

13. Type of Report & Period Covered
Technical Report

14.

15. Supplementary Notes
This report should be cited as: Woods Hole Oceanog. Inst. Tech. Rept., WHOI-91-01

16. Abstract (Limit: 200 words)

This report provides a User's Guide for operation of the High Resolution Profiler (HRP) and documentation of the software structure and recent modifications.
The HRP is an instrument that acquires and stores up to 16 types of physical oceanographic data. A profile is logged as the HRP falls through the water column during each deployment. It controls its dives based on user-specified trigger levels input during a pre-cast setup. Communications, the setup process, and how to check out and run the profiler are described fully. Also included are the current sensor configuration and summaries of assembly, mechanical preparation, tracking, data transfer and processing.
During 1990, the software controlling the HRP was almost completely reworked in order to port VRTX (Versatile Real Time eXecutive) to the HRP. This was accomplished to facilitate use of a fast data link that was being developed. Other modifications were made to the software to improve the user interface, to allow use of up to 16Mbytes of Random Access Memory, to speed up the serial interface, and to fix previously undetected problems. In addition, the use of an altimeter to determine height above bottom was added to the dive control logic of the profiler.

17. Document Analysis **a. Descriptors**
high resolution profiler
microstructure measurement
software upgrade

b. Identifiers/Open-Ended Terms

c. COSATI Field/Group

18. Availability Statement
Approved for public release; distribution unlimited.

19. Security Class (This Report)

21. No. of Pages
32

20. Security Class (This Page)

22. Price

(See ANSI-Z39.18) *See Instructions on Reverse*

OPTIONAL FORM 272 (4-77)
(Formerly NTIS-35)
Department of Commerce