Development of a Real Time Radar Acquisition System

Prepared by:

Stephen K. Modise
MSc. Electrical Engineering Student
University of Cape Town

9th January 2003

File: thesis.lyx
Document No:

Title: Development of a Real Time Radar Acquisition System

by

Stephen Karabo Modise

A dissertation submitted in fulfilment of the requirements for the degree of

M.Sc.(Eng)

Declaration

I declare that this dissertation is my own, unaided work. It is being submitted for the degree of Master of Science in Engineering in the University of Cape Town. It has not been submitted before for any degree or examination in any other university.

Signature of Author .....

Cape Town
9th January 2003

Abstract

The Geosonde radar system, developed for use in bore holes, includes a data acquisition system. Development is currently being conducted by the Radar Remote Sensing Group at the University of Cape Town and Stellenbosch University.

This thesis describes the development of a real time operating system and an overall upgrade of basic interfaces to the Geosonde system. The current system employs the use of an embedded MS-DOS operating system and supports basic user control and data exporting over serial line.

The objectives of this thesis is to perform major upgrades on the system by introducing an XML based form of network control and NTP synchronization of the processing board. As a result, an investigation into the adequacy of MS-DOS as a target operating system bearing in mind the intended upgrades was carried out.

Taking into consideration the failings of MS-DOS as far as the system requirements are concerned, an investigation into available real time executives was conducted and a decision based on the requirements was made. Embedded GNU/Linux was chosen as the target software platform.

The software design of the application shows all the necessary design issues considered. The implementation phase of the thesis describes all the tools necessary to implement the embedded Linux system and all the components necessary to meet the needs of the Geosonde system. The network and serial interfaces were tested and shown to be fully functional. The XML based control in particular offers a more flexible and more platform independent solution than the serial interface.

Acknowledgements

Thanks to my supervisor Professor Mike Inggs for his time and dedication, Alan Langman for his invaluable advice and my parents and family for their love and support.

Thanks go to:
M.L. Barry and the people at CSIR for the financial assitance.

Thanks also to:
Everyone on the uClibc mailing list
Everyone on the Busybox mailing list

Contents

List of Figures

List of Tables

Nomenclature

DTD A document type definition (DTD) is a specific definition that follows the rules of the Standard Generalized Markup Language (SGML).

ecos™ Embedded Configurable Operating System, a complete, open-source run-time environment.

GNU Recursive acronym for GNU's Not Unix

i386 A common name for the 32-bit members of the Intel x86 family. Members include 386, 486, Pentium ("i586"), and Pentium Pro ("i686").

NTP Network Time Protocol is an Internet protocol that provides the mechanisms to synchronize time and coordinate time distribution in a large, diverse Internet operating at rates from mundane to lightwave.

NTS Network Time Server

PC/104 PC/104 gets its name from the popular desktop personal computers initially designed by IBM called the PC, and from the number of pins used to connect the cards together (104). PC/104 cards are much smaller than ISA-bus cards found in PC's and stack together which eliminates the need for a motherboard, backplane, and/or card cage. Power requirements and signal drive are reduced to meet the needs of an embedded system.

POSIX Portable Operating System Interface. A set of standards that grew out of the UNIX operating system.

RFC A Request for Comments (RFC) is a formal document from the Internet Engineering Task Force (IETF) that is the result of committee drafting and subsequent review by interested parties. These are essentially standards of the Internet.

RTOS Real-Time Operating System.

TCP/IP Transmission Control Protocol based on IP. This is an Internet protocol that provides for the reliable delivery of streams of data across the web.

UTC Coordinated Universal Time

x86 A generic name for the Intel 8086 architecture family.

XML Extensible Markup Language

Chapter 1

Introduction

This report sets out the results of the thesis project undertaken. The thesis project involves upgrading of the data acquisition system used to capture and record data from the Geosonde Radar System. The Geosonde Radar system was developed primarily for usage in mining bore-holes, where it is used in the extraction of geo-physical data.

The basic components of the Geosonde Radar system are the radar probes that transmit and receive signals to and from the surrounding rock and soil formation. These probes are connected to a PC/104 computer on the surface via an optical link. The PC/104 computer is housed in a drum casing and the radar probes are lowered into the bore-holes by a winch attached to the drum.

The PC/104 board forms the core of the Geosonde system. This is where the received data is digitized by using an A/D digitizing expansion card connected to the main PC/104 module via its PC expansion bus. Following the digitization phase is the stacking of the acquired data, which will ultimately be exported to a console over an on-board RS232 port.

This project will focus on upgrading of the communication interfaces as per system-requirement in section 1.1 of the document and implement local time-stamping of data. The scope of this thesis is on embedded systems programming and real-time operating systems.

1.1 Problem Requirement

The problem statement for the project and the desired functionality to be added to the current system is briefly summarized in the following points:

- The Geosonde subsystem time-stamping[1] shall form part of the Sensor Based Recording and Processing system. The Sensor Based Recording and Processing system is briefly outlined in [1].
- Look into adding more functionality to the current DOS-based system. This includes:
  - Implement TCP/IP on the current DOS system.
  - Implement the NTP [RFC 1305] Protocol client-side synchronizing software for DOS and time-stamping capabilities.
- As a more permanent alternative to the current DOS system, look into a number of possible embedded RTOSs that will be able to meet the real-time requirements of the application.
- If necessary rewrite the software that resides on the PC/104 board to conform to real-time requirements of the chosen RTOS. This might include conforming to a certain API as most RTOSs have unique programming interfaces.
- Improve basic interfaces to the Geosonde subsystem. This consists of the following sub-points:
  - The manner in which the current software interfaces to the RS232 port for issuing of commands for control purposes.
  - Implement an XML format of control and define data formats as the basic manner of interfacing to the Geosonde subsystem over the Internet.
- Implement NTP on the board as a way of synchronizing the Geosonde to an external source of time.
- Implement time-stamping of acquired data using UTC.

An Ethernet interface to the Geosonde subsystem is essential for connecting to other Sensor Units on the Logging and Control system. One sensor in-particular being the *Cable Odometer* which will be used for calibrate the distance traveled by the radar probes into the bore-hole. This will enable all data to be time-stamped for storage in memory (or a common network database as the case maybe) whereby an NTS will provide a time base for network synchronization of all processor based units on

[1] From now on the Geosonde system shall be referred to as a *subsystem*. This is because it is merely one of the Sensor Units that forms part of the whole system.

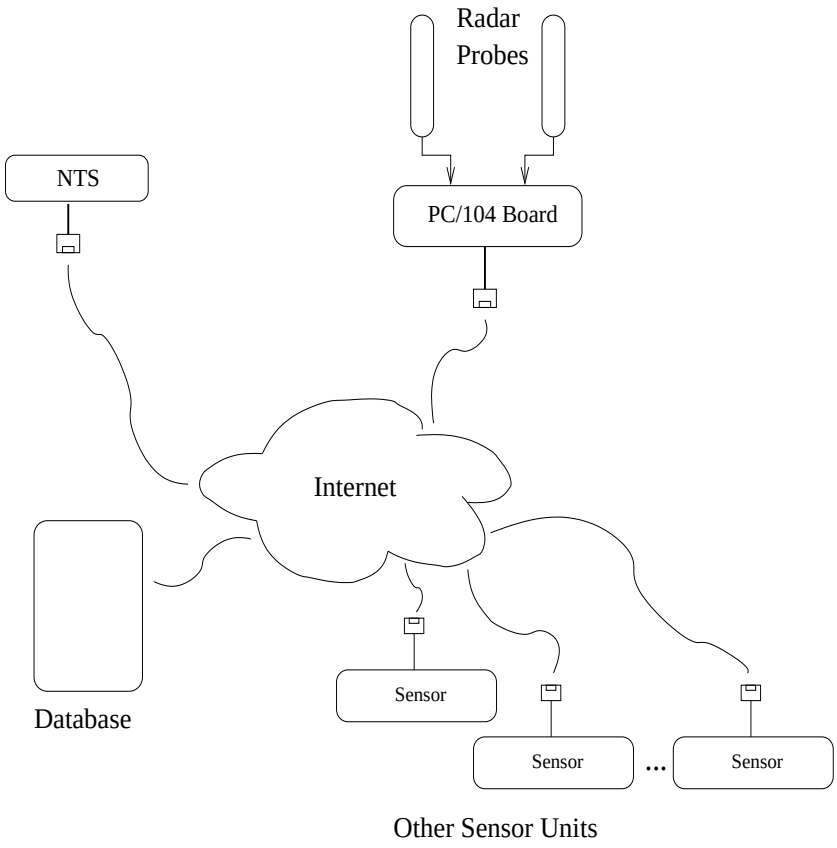

Figure 1.1: Overview of proposed system

the network cloud as seen in figure 1.1. Time-stamping of data provides for a better alternative to the current method of distance calibration whereby "blank lines" are inserted in the required data to mark a certain length of distance covered.

1.2 Steps taken in completion of the project

1.2.1 Concept study

A thorough pre-study of the topic was undertaken before commencing with the design and implementation phases of the project. The pre-study involved investigating the system requirements and a subsequent review of the requirements. The current state of the Geosonde subsystem was investigated and familiarity with the on-board software and workings of the hardware was achieved.

1.2.2 Investigation of a suitable RTOS

A number of real-time operating systems were researched as part of an investigation of a suitable operating system for the subsystem. A further study of the chosen operating system, GNU/Linux was conducted with the aim of finding the best method for achieving *real-time* behavior under Linux.

1.2.3 System design and software architecture

A software design document was drawn up to provide both a high-level design of the overall system and detailed design of the firmware. System block diagrams were drawn up to illustrate the major sub-components of the system. The necessary data structures employed for temporary on-board data storage are also illustrated as well as flow diagrams depicting the flow of data in the system. The design model adopted for designing real-time application software was also detailed in the system design document.

1.2.4 Software implementation

The implementation phase of the project was carried out in two stages. Firstly a highly stripped down GNU/Linux system was built with only the bare necessities included to support a functional Linux system with real time capabilities. Secondly the application software was developed entirely in *C* in accordance with the software architecture document. Further sub-divisions inherent in the system were written to conform to a special API and compiled separately as kernel modules. These however were written to interact with Linux user space programs. The governing focus of development is to minimize resource usage. This is typical when implementing a real time solution to a given problem.

1.3 Scope of thesis project

This thesis project involves real-time operating systems and embedded systems software development.

The software for the PC/104 acquisition board requires a firm understanding of real-time operating systems, *C/C++* programming languages, debugging tools, various communication protocols and GNU/Linux scripting languages. The operating

system and embedded software required a thorough knowledge of the GNU development tool-chain and the various GNU/Linux support tools for developing embedded systems. To provide real-time capabilities under GNU/Linux required knowledge of the unique API used by the chosen Real Time Linux implementation.

With regard to the radar aspects of the project, the scope is confined to the understanding of the very limited processing requirements of the acquired data.

A basic understanding of XML and related Internet technologies like DTD and Schema is also essential for implementing the control software for the host PC.

1.4 Plan of development of thesis project

The report states the requirements of the system and expresses in detail these requirements. A detailed specification of the system to be developed is given.

A concept study is given where results of the investigation into available open-source technologies suitable for implementing the system are set out. The present status of both the hardware and software of the system is also given in the concept study. This is followed by both the system and software design and the development process followed. The implementation of the overall system is also given as well as a discussion on the test results of the software and system.

Finally the various goals set out at the beginning of the project are discussed. Conclusions on the failures and successes of the project are drawn and suggested future improvements to the system are discussed.

Chapter 2

Concept Study

As part of the concept study for this thesis project, and with the aim of gaining familiarity with the workings of the Geosonde subsystem, a fair amount of work was carried out in upgrading the MS-DOS version of the system. The limitations imposed by the MS-DOS operating system as a platform for development of the project bearing in mind the desired functionality to be added to the system are given as part of the study.

The concept study includes an investigation into the various aspects of the Geosonde subsystem.

The following points in question are covered in the concept study:

- An overview of the present state of the subsystem
- A summary of the work done on upgrading the MS-DOS system
- Geosonde subsystem requirements

2.1 Present state of the Geosonde subsystem

Most if not all the work to be done for this thesis will focus more on the software side than on the hardware. The hardware system specification is however included for completeness.

2.1.1 Hardware specification

The Geosonde subsystem is build around the PC/104 platform. The specifications of the hardware on the TB486 board consists of the following peripherals :

Command	Explanation	Return status string
N	Change stacking	Yes
S	Stop, aborts any acquisition in progress	Yes
R	Run , starts or continues data acquisition	Yes
CTRL-D	Quit, aborts any acquisition in progress	No acknowledgment
T	Test, starts acquisition for a single shot	No
G	Set Gain	Yes

Table 2.1: Available user commands

- 16MB of RAM, with an absolute limit of 64M
- A standard 2MB flash memory[1] (with a limit of 4MB)
- An AMD 33MHz core processor (with a limit of 100MHz).

The hardware features stated above are essentially system-wide constraints on the hardware side. This effectively limits the amount of flexibility present on the actual hardware. A number of on-board peripherals and add-on cards are also available. An overview of the board and all the features are covered in the PC/104 board manual [2].

2.1.2 Software specification and basic system interfaces

The current system runs on an embeddable version of MS-DOS. Only a single means of interfacing with the outside world is supported. All communication for data exchange and control takes place through the RS-232 port.

Below are some of the features the current system has to offer:

- All acquired data is exported in 16-bit binary format over RS-23 port.
- In support of user interaction the software must understand the six commands tabulated in table 2.1. Some of the commands return a 9 byte ASCII status string as acknowledgment.

[1] The system BIOS occupies the first 128Kb of the flash memory, effectively reducing the amount of flash available for deployment to around 1.8MB.

2.2 Upgrading the MS-DOS system

The main thrust in upgrading the MS-DOS system is to implement TCP/IP under MS-DOS. In order to achieve this goal a number of tools had to be used.

Listed below are the tools used:

- DJGPP a collection of GNU tools for MS-DOS
- WATT-32 a 32 bit implementation of the WatTCP (Waterloo TCP)

The CS-8900 Ethernet controller was successfully tested using the aforementioned TCP/IP libraries under the DJGPP enhanced DOS system.

A small server-side program for the PC/104 board and a client-side program for Linux were written to test the Ethernet controller. The aim was to have the two programs interchange small messages to insure adequate communication was taking place.

2.2.1 DJGPP

DJGPP is a complete 32-bit C/C++ development system for Intel 80386 and higher PCs running MS-DOS [3].

Below are some of the benefits of using DJGPP under MS-DOS:

- Makes it possible to write protected mode software under MS-DOS
- Includes a wealth of existing tools (mostly UNIX) already ported to DOS

The GNU tools were installed on a 343MB IDE device and connected to the PC/104 board for testing. The software written for the Geosonde subsystem had to be ported to the DJGPP environment. This is mainly because the code was written for a 16-bit operating system (MS-DOS) and to a lesser extent it was written for the Borland compiler suite and as a result it was not very portable.

2.2.2 Watt-32

Watt-32 is essentially a library for writing networked TCP/IP programs using C/C++[4]. It is an optimized 32-bit port to the Waterloo TCP suite of libraries. WatTCP itself requires a network driver that implements the PC/TCP Packet Driver Specification. These were provided by *Cirrus Logic, Inc.* on their website for download.

2.2.3 Shortcomings of DOS as a development platform

A number of issues were noted during the development under MS-DOS and the effectiveness of using MS-DOS as the operating system of choice for the project. These are summarized below:

- The operating system and the supporting DJGPP tools had expanded to such a size as to render the deployment of the software to the 2MB flash impossible.
- Attempts to implement the NTP protocol under MS-DOS had proved to be a time consuming and difficult exercise. Attempts were also made to port some of the existing NTP client software written for GNU/Linux.
- Implementing a real time operating is one of the project goals. MS-DOS and DJGPP at the time of development had no support for real time preemptivity of tasks. The development tools also did not support threads or multitasking.

In light of the problems encountered, the only viable solution was to look for an alternate operating system that would meet the requirements of the project goals. This investigation into an alternate operating system is the subject of a later chapter, chapter 3.

2.3 Geosonde subsystem requirements

2.3.1 Basic interfaces to be supported[2]

The subsystem will support two main protocols for communication and control. These are listed below:

- Interface to the external systems is established by utilizing an on-board *Cirrus Logic CS8900A* Ethernet controller.
- An alternative means of interfacing to the Geosonde sub-system is by using a serial RS-232 link.

As a result the Geosonde subsystem must be endowed with TCP/IP capabilities. The *RRSG Seth*[3] board offers an alternate option for network connectivity via RS-232.

[2]The interface standards and protocols to the NTS and local database server will be established at a later stage of development. As far as the NTS and related protocols are concerned, the project is limited to testing using existing public network time servers.

[3]A Serial-to-Ethernet board build by members of the RRSG (UCT) group.

2.4 A High Level Software Specification

Using the software that currently runs on the PC/104 as the basis for the project, a high level specification of the software to be designed was drawn-up.

- The software shall provide functions or modules for reading and interpreting commands from user and exporting data over RS-232 to a console workstation.
- Efficient exporting of data shall be performed in 16bit Binary format over RS-232.
- The current system is configured for a data-rate of 19200 bps, this must be maintained for the sake of conformity.
- In order to limit power consumption and complexity of the data acquisition system, a number of constraints must be imposed on the Geosonde sub-system software. The following must be performed on the console side:
  - All permanent storage of data
  - Floating-point processing
  - Displaying of data
- Stacking of multiple waveforms to achieve $\sqrt{n}$SNR gain must be accomplished by routines on the board.
- A TCP/IP protocol stack shall be implemented. This will enable the Geosonde subsystem to connect to an existing IP network.
- Most of the user commands and configurable options tabulated in table 2.1 must be supported.

Time-stamping formats and capabilities to be included in software and the NTS protocols are currently not defined for the Logging and Control system. This might impose constraints on the development process. Provisions in software shall be made until such a time when the entire protocols are properly defined. This also applies to the system-wide database in figure 1.1.

2.5 Conclusions

This chapter has described the current state of both the hardware and the software of the Geosonde subsystem. A software specification was drawn up based on the software that runs on the Geosonde subsystem. MS-DOS was shown to be inappropriate as a target embedded operating system for the project because of its lack of support for preempting tasks. Implementing TCP/IP and NTP protocols under MS-DOS also proved to be difficult and time consuming.

The failings of MS-DOS as a target operating system implied that an alternate operating system that could meet the real-time demands of the project had to be researched.

Chapter 3

Investigation of a suitable RTOS

The focus of this chapter is on describing all the issues involved in choosing a real-time operating system. The various real-time environments that were considered for the project will be mentioned as well as their appropriateness or inadequacies for them to be chosen as the operating environment for the development of a real-time data acquisition system. The decision shall be based on the various requirements imposed by the application.

3.1 What is real-time?

This question is a subject of ongoing debate on the Internet, with whole newsgroups dedicated to defining just what constitutes a real-time system and what does not. The term real-time is a term with effectively no fixed meaning as different applications place different demands on defining what exactly a real-time system is.

An article on Embedded System Programming [5] magazine defines real-time as follows:

> "A real-time system is one in which the correctness of the computations not only depends on their logical correctness, but also on the time at which the result is produced. In other words a late answer is a wrong answer".

The article goes on further and describes two important terms that are used to describe real time systems, *predictability* and *deterministic*. A predictable system is defined as a system whose timing behaviour will always be within an acceptable range. A deterministic system is then described as a special case of a predictable

system whose timing behaviour can be predetermined. However it should be noted that to achieve predictability in a system, the system does not necessarily have to be deterministic.

The definitions of the above two terms is underpinned by the existence of an "operating environment" as defined in Embedded Linux Journal [6] as follows:

> " The term *operating environment* means the operating system as well as the collection of processes running, the interrupt activity and the activity of hardware devices (like disks)".

A further point made by the article is that real-time applications require a deterministic operating environment and only a real-time operating system can truly provide a deterministic environment. A real-time operating system therefore comes with the assurance that a response time to an event will always be within bounds to meet the application's deadline.

The precise definition of a real-time system therefore demands that one should peruse the system requirements document and state all the external system deadlines as well as how the behaviour of the system is affected by missed deadlines. Adopting an application-centric approach should essentially help in deciding whether the application at hand justifies the need for a real-time operating system. A system which demands relatively (on the time scale) fast responses does not necessarily have to be considered real-time, the inverse is also true.

3.2 Real-time candidates

As part of the project aims, a number of operating systems were looked into and judged among others on the following criteria:

- Price - The real-time operating system of choice must be an open source, royalty-free operating system.
- Ease of portability - The x86 architecture should be easily target-able
- Developers familiarity - An operating system (this includes among others the native development tools like compiler suites) which is very much familiar to the developer would be ideal as this would slash the development time overhead by a sizable amount.

- Support - A solution offering an ample amount of support for future improvements to the implementation is very much desirable.

Several other criteria had to be met in view of the requirements the application demands of the operating environment as stated in the system requirements section of chapter 2.3. The different operating systems researched vary from a full embedded real-time operating system like eCos [7] to what can be considered by many as a using a hammer to crack a nut approach of the use of a Linux system as an embedded real-time operating system. Emphasis was mainly put on finding a solution build around open-source technologies as the best solution for the problem posed by the application.

3.2.1 Red Hat eCos

> "eCos is an open source, highly configurable, portable and royalty-free embedded real-time operating system. eCos is designed to offer developers maximum control and overall flexibility over their real-time solution. This gives the developers the ability to configure the final real-time system around their application-specific needs. Features that are not needed can be removed from the system, ensuring a strict usage of the limited on-board hardware resources like flash memory" [7, pg. 9].

eCos supports the following features [8, pg. 2]:

- eCos supports applications with real-time requirements and provides features such as:
  - Interrupt handling mechanisms and minimal interrupt latencies
  - Full kernel preempt-ability
  - Synchronization primitives and a number of scheduling policies
- Full TCP/IP support
- The RedBoot bootstrap and debug firmware
- A highly portable Hardware Abstraction Layer(HAL)
- A real-time kernel that provides thread support, system timers, alarms and counters

The eCos real-time operating system offers a lot of enticing features like high configurability, a small memory footprint and full TCP/IP support with optional SNMP support, however on further investigation it was discovered that for the i386 PC architecture eCos does not yet provide the capability of running the operating system from a flash memory device. The overhead of learning a new operating environment given the time frame of an MSc. also weighed-in heavily against choosing eCos for development of the system.

A popular question embedded developers usually ask themselves regarding choosing a royalty-free embedded environment is, Can I use eCos? If the answer is no for some reason only than can the developer resort to Linux. Naturally the next sub-section focuses on Linux for embedded development.

3.2.2 Real-time GNU/Linux variants

The GNU/Linux operating system is well known in the desktop and server arena and is gradually making inroads in the embedded side of things. The Linux kernel has evolved over the years and currently includes a lot of the POSIX real-time functions.

There are however certain key problems that impede the use of Linux as a hard real-time operating system. It must be noted that Linux was designed with system throughput in mind as a result it does not offer deterministic response. Determinism is however central to all real-time applications.

The major problems with Linux being a real-time operating system involves the inability of the kernel to preempt itself regardless of whether a high priority thread is ready to execute or not. This leads to huge latencies which can run into the 100ms range. The kernel also has a tendency of disabling interrupts for unbounded periods of time, thus delaying its response to system events like interrupts.

The second issue is that of the Linux scheduler. The Linux scheduler is designed in a such a way that given a number of threads, before a single thread can be dispatched all the dispatch-able threads are examined to determine which one should execute first. It is quite simple to see that this delay will increase linearly with the increase in the number of threads [9].

There is a lot of work being done currently to make the Linux kernel more responsive. The efforts to make Linux suitable for real-time applications fall into roughly two categories, the preemption improvement approach and the sub-kernel approach. These two approaches are discussed in the following two sections, focusing on their strong points as well as their intrinsic limitations for adequate support of real-time applications.

3.3 The Preemption Improvement approach

This section shall give a brief overview of the two most popular approaches of solving the Linux kernel preemption problem. The first approach is that developed by MontaVista called "the Preemption patch" and the second is called "the Low Latency patch" pioneered by Ingo Molnar.

The preemption patch approach adopted by MontaVista, involves utilizing SMP (symmetrical multiprocessing) macros already available in the kernel and have the system run as if it were running on a multiprocessor machine even though it is still a single processor machine. The result is a system that will utilize the kernel's preemption capabilities in providing SMP support to enhance the preempt-ability of the Linux kernel on a uniprocessor system [10].

MontaVista also introduced a new real-time scheduler in the kernel, resulting in reduced interrupt latency and improved kernel handling of real-time threads [9].

Further information on these techniques can be found on the MontaVista web site [11]. MontaVista have implemented these techniques in their latest version of Hard Hat Linux called Journeyman Edition.

The Low latency patches developed by Ingo Molnar [12] offer a different approach in reducing the amount of scheduler latency inherent in the monolithic Linux kernel. This approach adopts a more simpler approach to reducing latency by identifying high latency blocks of code in the kernel and trying to safely introduce points where the scheduler can be called, effectively reducing latent periods that may affect external system events like interrupts. Implementation of low-latency patches is by no means a simple feat and can prove to demanding as far as maintenance is concerned [13].

This overall Preemption (both the Preemption patches by MontaVista and the Latency patches originally developed by Ingo Molnar) approach might prove adequate for an application with real-time specification in the 100ms range and they offer and added advantage of developing software entirely based around the Linux API, but for truly hard real-time application it is often the case that the system demands responses that lie in the sub-millisecond range and for such system specification we have no choice but to look beyond solutions based around Linux. This brings us to solutions that employ an external kernel in addition to the native Linux kernel.

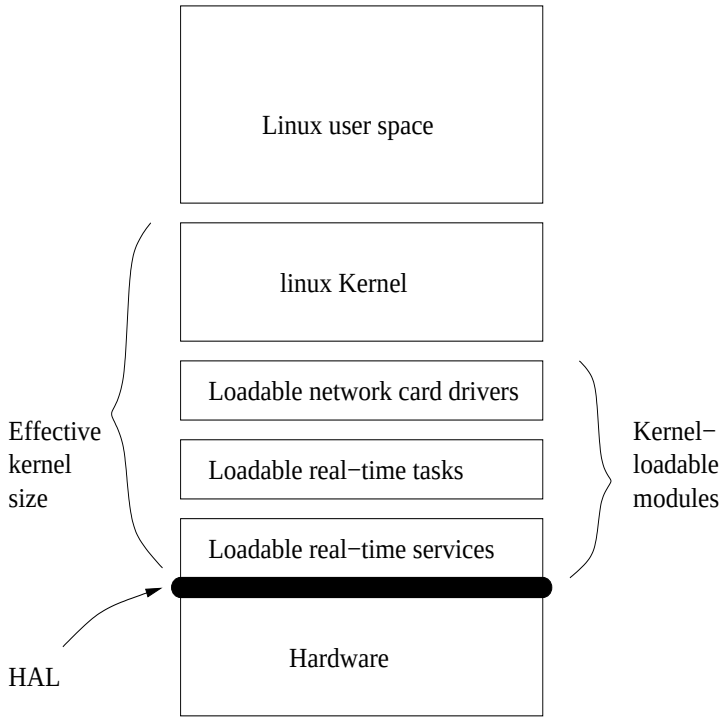

Figure 3.1: Sub-kernel integration into Linux system [15, pg. 77]

3.4 The Real-Time Sub-kernel approach

The sub-kernel approach to real-time for Linux involves introducing a separate, small, real-time kernel between the hardware and Linux [14]. The sub-kernel is not a full blown operating system and it can not exist solely by itself. It is merely an extension to Linux. At the moment the two most popular projects that employ the sub-kernel approach are the Linux **R**eal-**T**ime **A**pplication **I**nterface (`http://www.rtai.org`) and Real-Time Linux (`http://www.fsmlabs.com`). Both these projects are governed by the GPL (or LGPL in the case of Linux RTAI) license although the inventor of RT-Linux has a patent for the special technique used in RT-Linux which is a subject of major discussion on both their mailing lists. RTAI Linux is however still fully maintained in the open-source realm.

RT-Linux and RTAI Linux provide a specially designed API for handling interrupts and for writing real-time tasks. In addition to this unique API, POSIX functions are still supported. Figure 3.1 provides a picture of how the sub-kernels fit into the overall Linux system.

It is clearly visible that RT-Linux and RTAI Linux adopt a kernel space approach to real-time, this is almost similar to writing device drivers modules where these modules run in kernel-space without typical user-space comforts such as memory

protection. Both these implementations offer a vast array of mechanisms to communicate with user-space tasks. This is useful in taking full advantage of the underlying Linux functionality provided by the whole Linux system. These mechanisms include *FIFOs*, *shared memory* and *mailboxes* among others. RTAI provides a module called LXRT (Linux Real-Time) which is an alternative to kernel-space programming that makes it possible for a developer to write hard (nearly) real-time applications in user-space using the same RTAI Linux API. Only a slight performance penalty is incurred by such applications. RT-Linux has also started including user-space capability as of RT-Linux V3.0pre-7.

3.5 Conclusions

The investigation of a suitable RTOS chapter has discussed the major issues involved in deciding on a real-time target for the Geosonde subsystem. A number of real-time solutions were looked at, including a full real-time operating system, eCos and various embedded Linux implementations. Various forms of introducing determinism into the Linux kernel were discussed as well as highlighting their weaknesses. The sub-kernel approach to real-time under Linux and in particular RTAI Linux was chosen for development over the preemption patch approach.

The design (to a lesser extent) and more importantly the implementation phases of the project are centered around the use of an RTAI Linux enhanced Linux system as the target operating system.

Chapter 4

Software Design

This chapter describes the detailed design of the firmware for the PC/104 processor board. All abstract specifications of the services provided by the Geosonde subsystem are produced in this chapter. This chapter will also describe how the Geosonde subsystem was partitioned into different sub-components to handle different system services. All the necessary component interface designs are described in this chapter, including Geosonde subsystem interfaces to the outside world. All the data structures employed by the subsystem for temporary data storage are also detailed in this document.

The scope of this document is limited to only software design issues. All implementation specific issues will be dealt with in a separate chapter, chapter 5. This is to insure that all the design decisions made are independent of any implementation specifics such as the RTOS chosen for the implementation.

The software running on the current system is *Version 0.90*. From now on this shall be used as the system identifier and all modifications and additions noted as belonging to this particular version.

This chapter focuses on the following issues:

- *Algorithm design* The procedures for all the actions in the system are detailed and specified.
- *Component design* The different services the system provides are identified and allocated separate components.
- *Data structure design* Data structures employed by the system for data storage are designed.

4.1 Design Considerations

This section outlines all the prime issues that must be tackled in order to implement a complete and appropriate design solution.

4.1.1 Assumptions and Dependencies

To be able to implement proper time-stamping of data, it is assumed that all the necessary networking capabilities are functional. This includes the NTS protocols and interfacing to a common Network Database for all subsystems on the Data Logging and Control system.

On the hardware side, A RTC is essential for accurate time-stamping and network synchronization of the PC/104 board to a Network Time Server or some other external source of time. It shall thus be assumed that the hardware platform (PC/104 board) provides a RTC.

The language of choice for the software implementation is C, however it must be pointed out that assembly language might be used if such a need arises during the implementation phase. The choice of C as the main language of development is partly based on the fact that the current system was written and runs entirely C code. The C language does not provide inherent support for concurrency, as a result to be able to implement a real-time application using C as a development language relies heavily on the fact that the operating system (RTOS or the real-time executive) must be able to provide facilities for process creation, synchronization and adequate process and resources management.

A workaround to this problem of interdependency between C design specifications and the underlying environment provided by the RTOS in the design phase, is to use ADA as a PDL (Program Description Language) for *design description*. This will ensure a flexible design without any dependence on a particular real-time executive.

4.1.2 General Constraints

Outlined below are the principal constraints that have a significant impact on the software design:

- *Memory limitations* Given the size of the on-board flash memory as described in the system requirements in section 2.3, it is essential that the software should not have over sized memory requirements.

- *Performance requirements* The real-time nature of the application imposes strict constraints on the servicing of events. Software modules written to handle things such as external interrupts have to be simple, short and have fast execution times.

4.1.3 Goals and Guidelines

These are the principles that will dominate the design phase of the software.

- A lot of emphasis will be placed on speed (system responsiveness) in the "real-time" sense of the application.
- The basic functionality of the current system will be preserved as much as possible. This for instance includes retaining the same user control commands employed by the current system for the limited user interfacing with the Geosonde acquisition board.

4.1.4 Development Methods

The method adopted in the software design follows the approach of implementing real-time systems for data acquisition systems as detailed in [16, sec. 16.5]. This approach shall be followed entirely as it is with adequate changes where applicable. Obvious requirements imposed by the unique Geosonde subsystem application means adequate changes being made to the approach in [16, sec. 16.5].

4.2 Geosonde Subsystem Architecture

The system architecture aims to provide a top-level overview of the functionality provided by the Geosonde subsystem and how the major responsibilities of the software are partitioned and ultimately assigned to subcomponents.

The Geosonde subsystem is classified as a data acquisition system. It collects data from radar probes and digitizes the acquired data. The data is then placed in a buffer from which it can be extracted and the required processing performed on it. Subsequent exporting of data to the console follows the processing step. Figure 4.1 shows the system model of the Geosonde data acquisition subsystem.

One way of looking at this system is as a number of stimuli that the system must process and the relevant responses produced. Effectively all the component partitioning of the software will be based on the stimuli-response model. Each stimuli

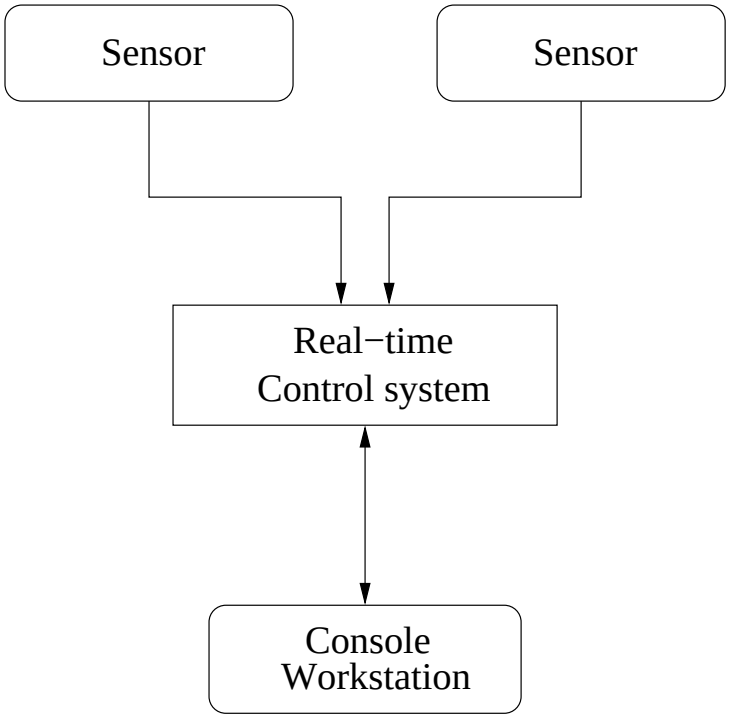

Figure 4.1: Geosonde Real-time System Model [16, Pg. 299, Fig. 16]

can be classified as either periodic or non-periodic. Periodic stimuli are produced by the sensors and pulses are generated at known periodic intervals. Aperiodic stimuli on the other hand occur at irregular intervals. These can be handled by interrupt handling mechanisms specifically written for the application or the board's own handling mechanisms.

The control software expects instances of each type of stimuli. Certain stimuli require responses to acknowledge received input and some do not.

Periodic stimuli expected:

- Returned (pulse) waveform from radar probes or sensors
- No response required

The system has only periodic stimuli generated by the sensors at predictable intervals.

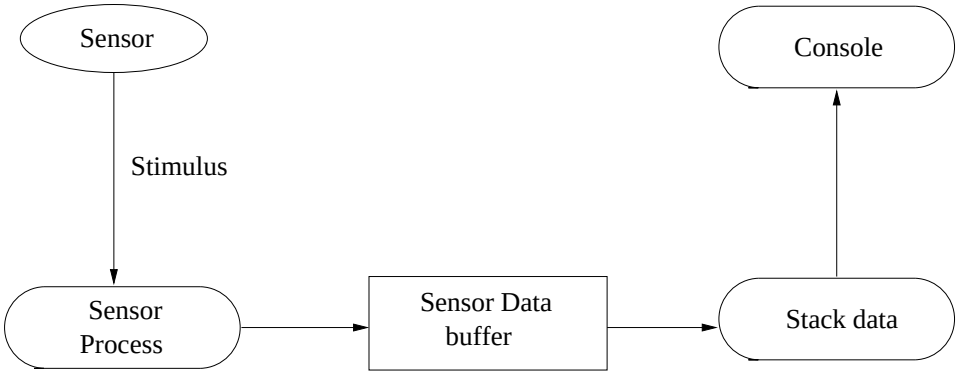

Figure 4.2: Sensor to console control process [16, Pg. 299, Fig. 16.2]

4.3 Detailed Subsystem Design

This part of the design chapter provides a more detailed discussion of the periodic components of the software as partitioned in section 4.2 of the document.

Design of real-time software dictates that all identified stimulus and response processing be grouped into a number of concurrent processes. The approach that will be followed here is to associate a process with each class of stimulus and response.

4.3.1 Returned pulse from sensors

The sampled data coming in from the radar probes is classified as one particular class of stimulus. Figure 4.2 shows the data flow of each sensor value from data acquisition by a probe to console.

The functionality and various system activities provided by this class of stimulus-response processes is listed below:

- The system collects data from a number of radar probes
- The sensor data is placed in a buffer from which it is extracted and the necessary stacking routine is performed. The stacked data is exported over a communication channel[1] to the console for any further processing (floating point processing) and displaying.
- Each sensor must have an associated process which will be responsible for converting analog input data into a digital signal.

[1] This can either be RS-232 or TCP/IP depending on which communication channel was chosen.

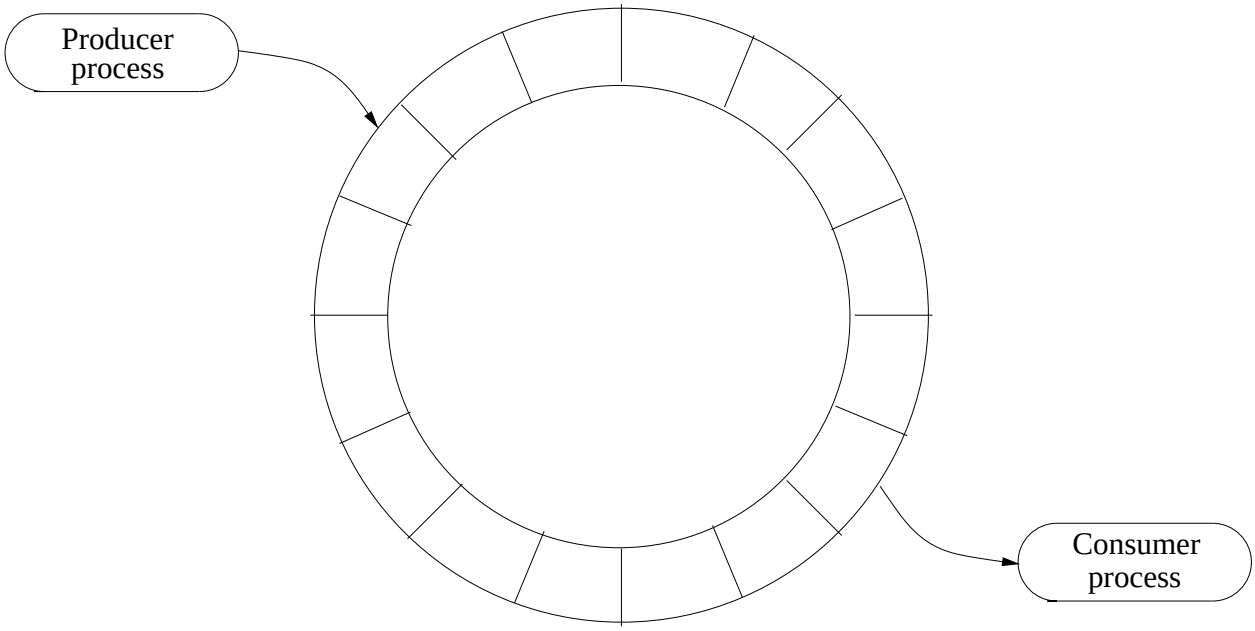

Figure 4.3: A ring buffer for data acquisition [16, pg. 313, fig 16..14]

- The data is then tagged with a sensor identifier (if more than one probe is used) and passed to the sensor data buffer.
- The process responsible for data stacking extracts the data from the buffer, stacks the data and passes it to the exporting process for exporting to the console.

A typical practical problem for real-time system implementation of a data acquisition system with subsequent processing is the speed differences between the processing process and the execution speeds and time taken by an acquisition process [16, Pg 313]. One solution to this problem is to use a ring buffer for temporary data storage.

A ring buffer consists of two processes. A *producer* process is in-charge of inserting data into the buffer and another process called a *consumer* that is responsible for reading the data from the buffer. A ring buffer is shown in figure 4.3.

A number of issues have to be taken into consideration when implementing a ring buffer:

- A producer process must not access the same element in the ring buffer as the consumer process at the same time.
- A producer process must not try to add data into a full buffer.
- A consumer process must not try and take data from an empty buffer.

```
task Sensor_data_buffer is        --specification
      -- Get and Put are parallel procedures.
      -- They are the only interface to the buffer.
      entry Put ( Val: SENSOR_RECORD ) ;
      entry Get ( Val: in out SENSOR_RECORD ) ;
end Sensor_data_buffer;
```

Figure 4.4: Task specification of Sensor_data_buffer [16, pg. 314]

A plausible solution to the first item is to use a mutual exclusion concurrency strategy. The last two points can be solved by implementing the buffer as an abstract data type with the *Producer* and *Consumer* operations implemented as concurrent operations [16, pg. 313].

An EDA design description listing of the sensor data buffer process is shown in figure 4.4. The notion of a *task* can be used in describing parallel procedures, in this instances there are only two such parallel processes (*Put* and *Get*) allowed to manipulate the data buffer directly. The *Sensor_ data_ buffer* task specification describes the interface part of the task. This is essentially what all the other tasks defined in the system will see. The body of the sensor data buffer is shown in figure 4.5. It describes the overall dynamic behavior of the buffer task.

4.3.2 Real-time Scheduling

After designing appropriate algorithms for carrying out the required computations it is essential that a proper scheduling scheme is decided on.

The user commands do not have any timing restriction as opposed to the periodic data coming in from the sensors. The sensor data process should have the highest priority in the scheme, this is because of the frequency at which these processes are to be executed. The RTAI scheduler offers a default scheduling policy of FirstIn-FirstOut (FIFO), on a per task basis. This shall be the preferred scheduling policy for all tasks unless stated otherwise.

4.4 Operational modes of Geosonde subsystem

The Geosonde subsystem has two operational modes as shown in figure 4.6. This is in adherence to the system requirements that the software support RS-232 and Internet operational mode of control[2].

[2]Either of the two modes should be operational at any point in time

```
task body Sensor_data_buffer is --body
        Size: constant NATURAL := 5000 ;
        type BUFSIZE is range 0..Size ;
        Store: array (BUFSIZE) of SENSOR_RECORD ;
        Entries: NATURAL := 0 ;
        Front, Back: BUFSIZE := 1 ;
begin
loop
        --Put will only execute when there is space in the buffer,
        --Get will only extract items from the buffer when it is not empty
        select
                when Entries < Size =>
                        accept Put (Val: SENSOR_RECORD) do
                                Store (BACK) := Val ;
                        end Put ;
                Back := Back rem BUFSIZE'LAST + 1 ;
                Entries := Entries + 1 ;
        or
                  when Entries < 0 =>
                        accept Get (Val: in out SENSOR_RECORD) do
                                Val := Store (Front) ;
                End Get ;
                Front := Front rem BUFSIZE'LAST + 1 ;
                Entries := Entries - 1 ;
        end select ;
end loop ;
end Sensor_data_buffer ;
```

Figure 4.5: Body describing behaviour of Sensor_data_buffer [16, pg. 314]

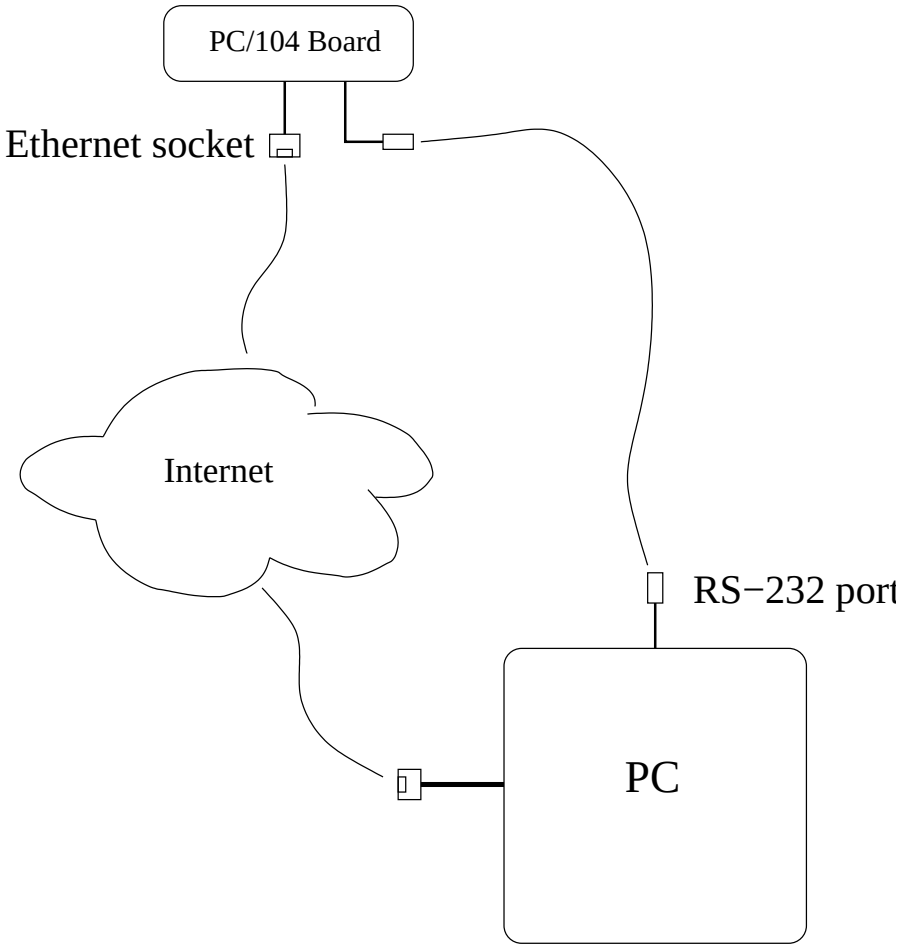

Figure 4.6: Operational modes of subsystem

4.4.1 RS-232 Operational mode

The current system supports a crude and inefficient way of interfacing with a console PC over the RS-232. Both hardware and software handshaking are not supported. As an improvement to the subsystem, the implementation shall be controlled by a Text-terminal under Linux.

4.4.2 Internet Operational mode

The Internet mode is an upgrade to the subsystem and shall use XML tools to define both command and data interchange between the Geosonde subsystem and a PC running a network control software.

4.5 Real-time application software design under RTAI Linux

The software design for the Geosonde sub-system follows the design model similar to that of designing drivers for a new piece of hardware in a Linux system. A split-architecture approach was followed where the different system activities were partitioned into time-critical and non time-critical activities. All the time-critical activities are to be written as real-time tasks that run in kernel space. Non time-critical activities of the software will be written as tasks that run under the memory-protection umbrella of the Linux system.

A form of inter-process communication support has to be facilitated whereby the RTAI task modules can communicate and interchange data with the User-space tasks, this is provided by the RTAI Linux environment which supports various forms of inter-process communication that includes among others **FIFOs** and **Mailboxes**.

- FIFOs are used for the development phase of the software.

The manner in which the system modules are defined is in line with the development model imposed by the usage of sub-kernels, in this instance RTAI Linux. The system modules are conveniently divided as follows:

- Real-time design - This is the core of the acquisition system where tasks are defined that operate in kernel space.
- Linux functionality design - These are the supporting tasks that interface to the real-time tasks.

The real-time application design phase of the Geosonde sub-system shall adopt where appropriate the "*Five Real-time Application Development Steps*" outlined in the Hard real-time Linux article in Electronic design [15].

The following design steps were extracted from the aforementioned article:

1. Partition the system into time-critical and non time-critical elements. This is illustrated by the separate kernel-space and user-space regions in figure 4.7.
2. Provide both data communication and control structures between the real-time activities and the support elements running in user-space. This is achieved by using independent FIFOs for Data and Command. This is illustrated by A and C in figure 4.7.

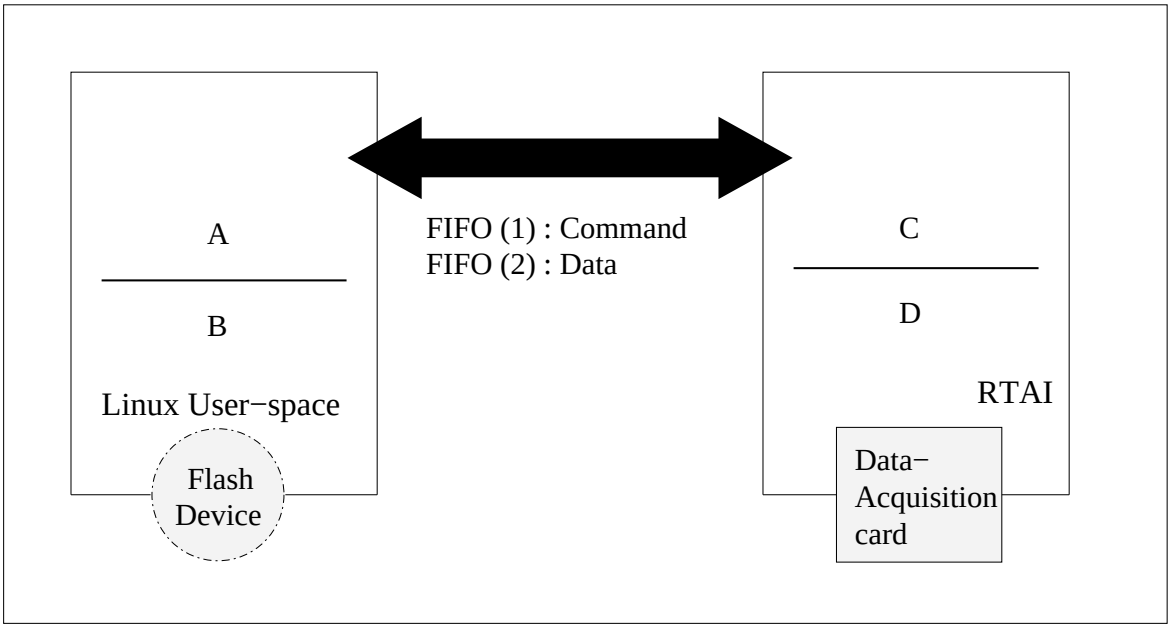

Figure 4.7: High level view of kernel space (RTAI) and User-space split-architecture design of system [15, fig. 2]

3. Provide a command handler to control the real-time task from the support element running in user-space. The command handler should span both the real-time and user-spaces as shown in figure 4.7 by sections B and D.

4. All time-critical elements are to be developed as hard real-time tasks, shown by section D in figure 4.7. The acquisition function, task scheduling and initialization, and task cleanup are all defined in this step.

5. Define all non time-critical elements which give support to time-critical elements as standard Linux tasks running in user-space. This is represented by B in figure 4.7.

4.5.1 Separating the system

Elements of the system requiring immediate service will be treated as time critical elements. The Geosonde data acquisition application accepts a data stream coming in from an analog-to-digital converter. For successful acquisition, the system must be able to read the A/D output every Xμsec[3].

[3]For testing the system various timer value were used that the 486 microprocessor could handle. In a typical practical application however, the real timer value of the A/D rate would be used.

- This instant where the digitized sample is taken is the only hard-real time element of the whole system. The data acquisition task will be implemented as a real-time task.

Elements of the system that will be implemented as non time-critical task are the following:

- The time-stamping procedure of the acquired waveform.
- The stacking procedure of the acquired waveforms.
- Temporary storage (if any) in the on-board flash device.
- Necessary tasks for control of read/write to system structure for storing sample data.
- Operational mode routines depending on the chosen mode of operation
  - Serial (RS-232) link setup and control routines and data exporting
  - Network (Ethernet) socket-based routines for exporting data

4.5.2 Facilitating communication between real-time and support elements [15]

RTAI Linux provides various IPC (Inter Process Communication) mechanisms. FIFOs are to be used for the implementation phase because of their usefulness in communicating with interrupt handlers. For communication (data and control) between real-time tasks and user-space support tasks the following FIFOs will be defined:

- *fifo1* : Command FIFO for real-time task control
- *fifo2* : Data FIFO for passing acquired data from kernel-space to user-space bound application code.

Both *fifo1* and *fifo2* are shown in figure 4.7 also shown are the communication end-points provided by these FIFOs in the kernel and user-spaces. A command handler makes it possible for the real-time task to be controlled by the user-space application code via the Command FIFO (*fifo1*). Extra FIFOs will also be defined for the sole purpose inter-kernel-space communication between the acquisition task and command handler task.

4.5.3 Developing hard real-time tasks

For developing real-time tasks under RTAI, the separate RTAI API will be used. All development will be done in kernel-space instead of utilizing RTAI's LinuX-Real-Time (LXRT) module. This decision is based on the fact that only a single task (*acq_func*()) will be coded in kernel-space. Development in kernel-space offers a direct hard real-time response as opposed to using LXRT. However it must be noted that developing in kernel-space does not come with the luxury of memory protection common in all user-space development.

The following functions are to be defined for hard real-time tasks:

- *acq_func()* : The acquisition function
- *init_module()* : A function to allocate required system resources, declare tasks and start tasks. This function is invoked by *insmod()*[4] when this module is loaded. This function is an absolute necessity for any development done in kernel-space.
- *my_handler()* : The command handler for user-space bound code to issue commands to real-time tasks.
- *cleanup_module()* : This function is invoked by *rmmod()*[5] whenever the module is removed. It does all the house-keeping like deleting tasks and releasing all resources allocated during the lifetime of the module. It complements the *init_module()* function and as a result it also required.

4.5.4 System support elements

All system support tasks are implemented as simple user-space Linux tasks. They will have access to Linux system calls, C library calls and all the services provided by a basic Linux system.

4.6 Conclusions

This chapter has explained in detail the software design for the Geosonde subsystem. The design of system components has been detailed and reasons given for certain design choices that were made. All the data structures used by the subsystem for

[4] see: *man 8 insmod* under GNU/Linux man pages
[5] see: *man 8 rmmod* under GNU/Linux man pages

temporary storage of data are mentioned and described. All assumptions made in order to have a proper system software design are also outlined. Design specific issues for interfacing to the subsystem are essential and as a result were also mentioned for each of the two operational modes of the Geosonde subsystem.

The bulk of the design chapter strives to concentrate on major software design issues without delving into implementation specifics such as the real-time kernel used. However a small section of the design focuses on the design model for designing software under the RTAI Linux sub-kernel as there are a number of issues to contend with for this type of system.

Chapter 5

Geosonde subsystem Implementation

This chapter describes the process of building a minimal Linux file system for an embedded system and all the tools employed in the process. The basic tools used to meet the project requirements are also mentioned as well as the necessary configuration options for the various tools and the kernel configuration. In compiling the tools various compiler options and optimization keys are used. This is important in most embedded systems due to restricted resources imposed by the hardware. These options and optimizations are also mentioned were necessary.

5.1 Building the minimal Linux system

Implementing a highly stripped-down Linux system follows a process much similar to that of building Linux boot disks used in recovering from system failures [17]. A minimal set of system directories are essential to support a fully-embeddable Linux system.

The root file system contains the following directories and sub-directories:

/ ——— the root directory

- /bin– Essential command libraries
- /dev–Device files
- /lib–Essential shared libraries and kernel modules for run-time support
- /proc–Stub under which *proc* file system is placed when system is running

- /etc–System specific configuration files
- /usr–Additional utilities and applications
- /boot–Boot loader specific files

The busybox and uClibc root file system was build around *buildroot.tar.gz* [18] with additions and changes to the directories and configuration files made were appropriate for the embedded root file system.

5.1.1 Kernel configuration options

A customized kernel image had to be configured with a minimal set of features required to support the underlying hardware and the required system functionality as outlined in the project requirements .

The following sub subsections describe some of the features to be linked in which are essential to meet the project requirements

Networking options

This feature has to be enabled with support for TCP/IP networking. DHCP support is also enabled to support kernel level auto configuration during boot time[1]. Network device support for cs89x0 chipset based Ethernet controllers is enabled to support the on-board Ethernet controller.

File systems supported

The minix file system was enabled during the testing phases of the file system on boot floppies, this is no longer essential and can be removed as a trade-off between size and system features. The */proc* file system is supported to provide a virtual file system which provides information about status of the system during run time. ext2, the second extended file system (the default Linux file system) is used for testing and final deployment of system using an IDE device[2].

[1] A DHCP server has to be available on the local network for assigning IP address of the device.

[2] For future system upgrades, a compact-flash card may be used. This will require the MTD (Memory Technology Devices) feature to support solid-state file systems.

Processor type

The Elan option for the AMD Elan processor family is chosen for the PC/104 board as the processor type for optimization purposes. The board has an Elan SC410 processor.

RTHAL (The Real-Time Hardware Abstraction Layer)

The RTHAL patch to a standard Linux kernel is essential for RTAI to provide a hard real-time environment as part of Linux.

Other configuration options are available in the kernel configuration file (see *Config_elan* in enclosed CD) used in the build process.

5.1.2 GRUB bootloader

The choice of which bootloader to use for the project is limited to only two of the most popular methods of booting-up the system under Linux. These are LILO and GRUB. GRUB was chosen as the bootloader for the project. The decision was based on the rich set of features GRUB supports.

> " GRUB is a GPLed bootloader intended to unify bootloading across x86 operating systems. In addition to loading the Linux kernel, it implements the Multiboot standard, which allows for flexible loading of multiple boot images (needed for modular kernels such as the GNU Hurd). "[19]

Listed below are some of the features GRUB supports that are important to the project.

- Supports a text configuration file with preset commands
- Provides a simple menu interface to support the preset commands in the configuration file
- Has an easily adaptable command-line interface which is accessible from the menu. This can be used to edit the preset commands
- Supports network booting to load operating systems from the network

- Has remote terminal support. As a result GRUB on the target PC/104 board can be controlled from the host computer.

The text configuration file (see *menu.lst* in enclosed CD) was used to set the menu commands for the bootloader.

5.2 Tools and software used in building the Linux system

This section describes the tools and software used during the implementation phase of the project.

5.2.1 uClibc-0.9.14

uClibc is a C library for developing embedded Linux systems that offers a smaller footprint than the standard GNU libc (glibc) [20]. Almost all applications supported by glibc work well under uClibc.

The uClibc development environment was installed on a Linux development host workstation. A configuration file was used for compiling and building uClibc (see *uClibc-config* in enclosed CD).

All the C-code developed for the project was compiled and tested under the uClibc environment. The runtime environment was installed to the target PC/104 system through special commands as described in the INSTALL file in the uClibc distribution. Only the files necessary to run the binaries compiled against uClibc during development are installed to the target IDE device. These are essentially shared libraries installed under */lib* directory of the system.

Most of the tools mentioned in the following subsections were build using the uClibc C compiler.

5.2.2 busybox-0.60.3

BusyBox is often referred to as the *Swiss Army Knife of Embedded Linux.* It offers small versions of many standard Linux utilities and bundles them into a single stripped-down executable. These utilities are mostly GNU fileutils and shellutils among others.

> "The utilities in BusyBox generally have fewer options than their full-featured GNU cousins; however, the options that are included provide the expected functionality and behave very much like their GNU counterparts. BusyBox provides a fairly complete POSIX environment for any small or embedded system" [21]

BusyBox is extremely modular as a result commands or utilities can be added or excluded at compile time to customize the embedded system for the PC/104 target. A configuration file (see *busybox-config* and *busybox-makefile* in enclosed CD) was used to select features to be compiled into the image.

5.2.3 tinylogin-1.02

> "TinyLogin is a suite of tiny Unix utilities for handling logging into, being authenticated by, changing one's password for, and otherwise maintaining users and groups on an embedded system. It also provides shadow password support to enhance system security. TinyLogin is, as the name implies, very small, and makes an excellent complement to BusyBox on an embedded system (though it can of course be used without)." [22]

Like BusyBox, TinyLogin is also modularized making it possible to build only the required components. It also supports the use of a configuration file to select desired components and features. The configuration file used for the project development is available (see *tinylogin-config* and *tinylogin-makefile* in enclosed CD).

5.2.4 setserial-2.17-24

The setserial program is used to configure and allocate the serial port resources [23]. It is also used to report on the status of the serial port. A number of attributes of a serial port can be configured. Among these, listed below are the most common attributes to set:

- The I/O port of the serial port
- The hardware IRQ
- The UART type

The PC/104 target system serial port is configured and initialized by a startup shell script located in */etc/init.d* directory of the system. Only the first two (COM1 and COM2) serial devices are configured for the system. See *S20setserial* in enclosed CD.

5.2.5 udhcpc 0.9.6-3

udhcpc is a small, fully functional RFC compliant DHCP client [24]. The udhcpc program receives an IP address and other information assigned to it by a DHCP server located somewhere on the local network . Its major task is to automatically configure the network interface at boot-up time using the Ethernet protocol.

The network interface and other network specific information are also configured through a shell script located in the */etc/init.d* directory. See *S40networking* in enclosed CD.

5.2.6 lrzsz 0.12.21-4

> "lrzsz is a Unix communication package providing xmodem, ymodem and zmodem file transfer protocols." [25]

The package provides programs to send (sz, sx and sb) and receive (rz, rx and rb) files over a dial-in serial port using XMODEM, YMODEM and ZMODEM protocols. These programs can be run from a number of different programs under different operating systems.

For the project only the ZMODEM program to send files sz, was build for the target board. On the host computer, all three protocols are supported by minicom, see section 5.3.1.

5.2.7 util-linux 2.11n-4

util-linux is a collection of essential system utilities for Linux [27]. Only *hwclock* was compiled for the target board. hwclock is a program that queries and sets the hardware clock (RTC). The hwclock program is run from a shell (see *S18hwclockfirst* and *S50hwclock*) script during the boot-up process, and is used for setting the system clock using the hardware clock.

5.2.8 ntpclient

> "ntpclient is an NTP (RFC-1305) client for unix-alike computers. Its functionality is a small subset of xntpd, but has the potential to function better within that limited scope. Since it is much smaller than xntpd, it is also more relevant for embedded computers. " [26]

The ntpclient software has a number of user options to be passed to the program at the command line. The aim is to be able to synchronize the PC/104 board to an external source of time using the NTP protocol. This is to be performed before any acquisition is performed so that all time stamping of acquired data is as close to the "real time" as possible.

The program can be run again at a later stage to ensure that the PC/104 board is in synchronization with other sensors on the local network. The system clock often strays from the RTC (Real Time Clock) time and synchronizing to an external source of accurate time is the only way true time can be kept.

5.3 Serial Communications Control for PC/104

The RS-232 port on the PC/104 board is the basic interface to the Geosonde subsystem. On the target side low level routines setup the serial port as well as setting up such things as the terminal device for reading input from the user. The *inittab* boot-up script located in the /etc directory ensures that a getty is running on the chosen serial port and a communication program on the host side connects to the target board.

minicom was used on the host side to communicate with the Geosonde subsystem.

5.3.1 minicom

minicom is a menu driven serial communication program that runs under Linux. It is basically a clone of the MS-DOS TELIX communications program and supports ANSI and VT102 terminals. A number of features are supported by minicom such as automatic zmodem downloading and capturing to file.

minicom supports saving all configuration options to a file. The configuration script (see *minirc.dfl* in enclosed CD) was used for all serial interface testing of the project.

5.4 XML software for Control and Monitoring

The Internet interface for the Geosonde subsystem follows a typical client-server programming model as illustrated in figure 5.1. The Control and Monitoring software that runs on the PC (Windows NT 4.0 workstation) opens up server ports and waits for the acquisition system to connect to it.

Two ports per acquisition subsystem are opened by the server.

PORT1 This port is used for issuing commands to the client over a network link. In this case the Geosonde subsystem is the client that receives the commands.

PORT2 The second port of the subsystem-controller interface is used by the client for telemetry purposes.

Both ports are simply sockets with matching port numbers on each side (server and client side), i.e. PORT1 has same identifier on the client side as PORT1 on the server side.

5.4.1 Host side software

The control software that runs on the PC as shown in figure 5.1 is part of an ongoing research project at NASA's Goddard Space Flight Center. The basic idea is to use XML to facilitate data exchange between remote sensors like the Geosonde and a host computer connected to the network running the control software.

The group has designed an XML vocabulary called IML (Instrument Markup Language) [28] to control distant sensors via Internet.

IML [29]

IML was written using DTD. It basically provides a framework for the use of XML in the exchange of data and commands between a host computer and remote sensors or instruments.

Below are some of the benefits of using IML:

- ASCII messages are sent from control software to sensor instead of the actual XML

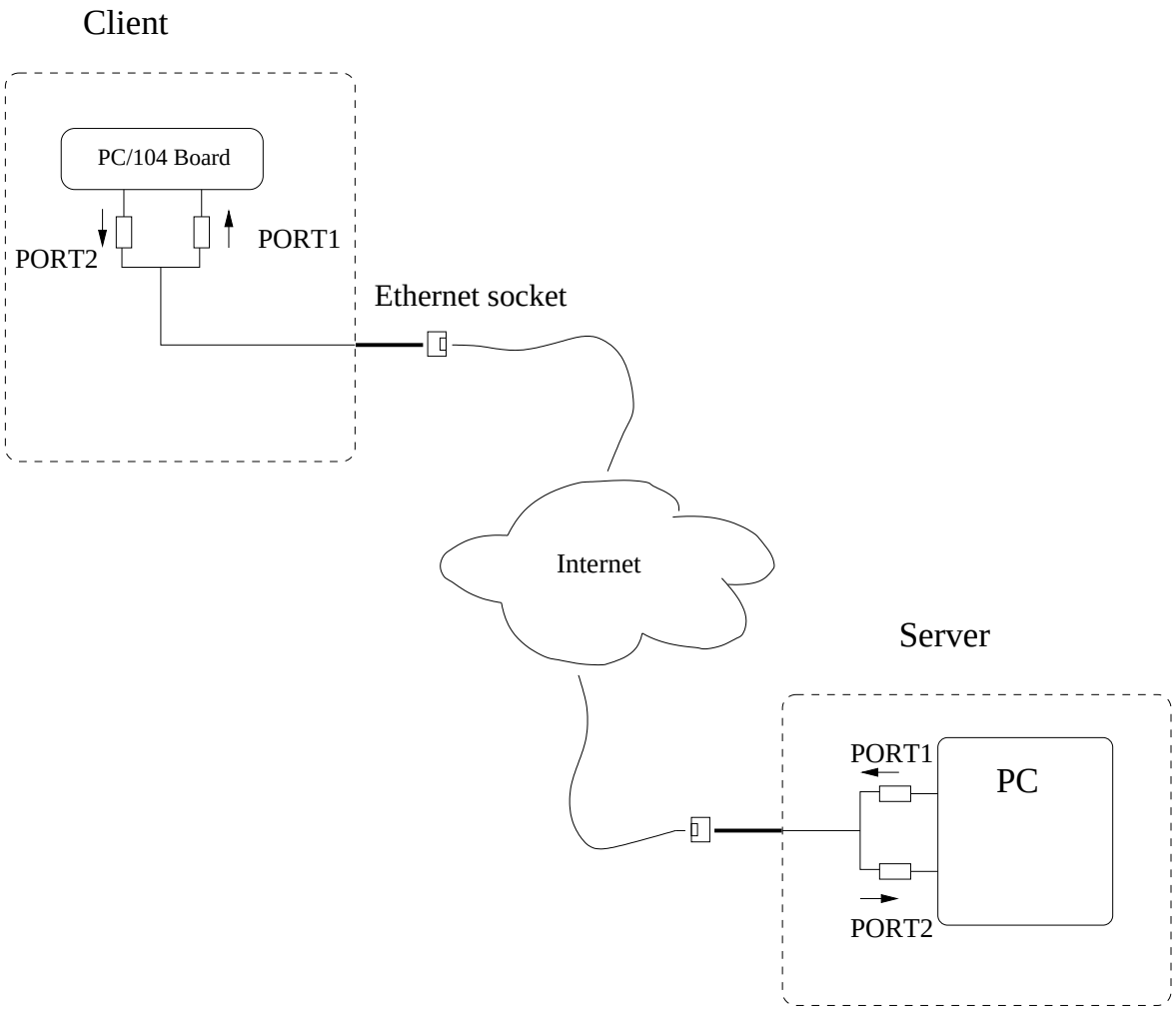

Figure 5.1: Client-server model of PC to Geosonde

- A Java based tool reads the XML document using IML and creates a user interface for issuing commands to the sensor.
- Received data from the sensor can be interpreted by the Java based tool and presented properly on the screen using the XML specification.

The XML specification used for the Geosonde subsystem is available in a plain text file (see *hawc.xml* in enclosed CD). Only a sub-set of the original Geosonde commands are implemented. It is however easy to extend the supported commands to the whole set by editing the *XML script (hawc.xml)*. This is one of the advantages of using XML as means of control for one or more set of networked sensors. Each new sensor in the system only has to be added to the XML file describing how it is to be interfaced to and the Java software automatically creates the necessary user interfaces.

5.4.2 Target side software

On the client side, the networking code was written using basic socket level programming. The networking code is embedded in the actual application code and interfaces directly with the kernel space module using FIFOs as described in section 4.5.

5.5 rtai-24.1.9

In order for the real time code to work under Linux, the application needs an RTHAL-enabled kernel and the necessary RTAI modules as well the device nodes for the RTAI FIFOs. This is all accomplished during the build process of the RTAI distribution, see *README.INSTALL* file of the RTAI distribution in enclosed CD.

To compile the RTAI modules, the rtai-24.1.9 distribution source includes a *menuconfig* interface tool to configure RTAI. The menuconfig tool automatically generates a configuration script which is in-turn used by the *make* command. See *RTAI-config* in enclosed CD for the configuration script used in the project.

Among others, the following configuration options are essential to the project.

Schedulers RTAI offers a total of three schedulers. The UP (uniprocessor) scheduler was used for the project.

Features A number of standard features are available. These are listed below

POSIX The POSIX API threading

RT memory_ manager A dynamic memory manager for RTAI tasks

During configuration, an option of choosing between built-in and standalone modules is available.

5.6 Conclusions

This chapter has described the process of building a minimal Linux file system for an embedded system. All the necessary tools and packages necessary for a fully functional embedded Linux system are mentioned, as well as the Geosonde specific utilities that were used to fulfill the project goals. The various open-source programs used for implementing and finally testing both interfaces of the Geosonde subsystem are mentioned as well as the necessary configuration scripts for certain tools. All the software responsible for communication between the client and target side for both interfaces is also described.

The implementation of a minimal file system was followed by testing of both interfaces.

Chapter 6

Geosonde Testing and Results

This chapter describes the testing procedures of the Geosonde subsystem to ensure that all the project requirements as stated in earlier chapters are met.

The system will be tested for correct operation of both the serial and network interfaces. Correct operation of the RTAI Linux real time sub-kernel under Linux will also be tested to ensure that data exchange does indeed take place between both the kernel and user space.

6.1 Data Used for Testing

Due to time constraints the final implementation of the project does not include an A/D card as used by the current DOS system. Instead the data acquisition routines in the kernel module of the code emulate the operation of the data acquisition card by writing short integers to an array. The size of the array is the same size as the array that would be produced using an actual acquisition card.

This is accomplished by the *get_wave* function which should contain A/D device specific code.

6.2 Geosonde Setup and Synchronization

To setup the board for testing requires a 12V power supply for the board and the IDE device. The testing configuration of the whole subsystem for both the serial and the Ethernet mode of operation is shown in figure 6.1.

The NTP client software is run during boot-up (see *S51ntpclient* in enclosed CD) to synchronize to a source of accurate time. After the boot-up process of the PC/104

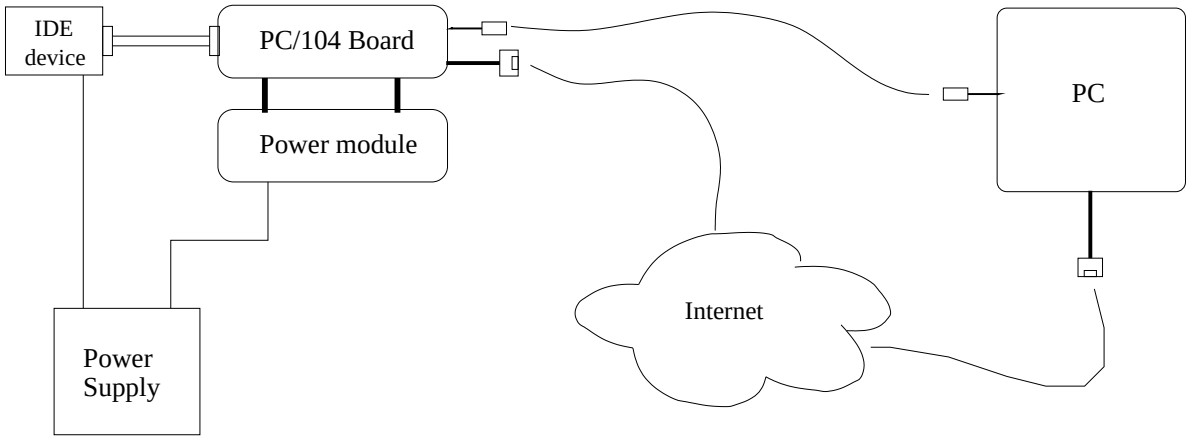

Figure 6.1: Test configuration of subsystem

board, the ntpclient program can be optionally run from an xterm to synchronize the RTC to external source of accurate time. Some of the public NTP servers used for time synchronizing are listed in the file *ntp-servers* in the enclosed CD.

6.3 RS-232 Interface

To test the serial interface between the Geosonde board and a host PC running Linux, the two are directly connected by a null modem cable which provides for hardware flow control (RTS/CTS).

The PC/104 boots-up and runs a getty program on the connected serial port with the same configuration as on the host PC side. In this manner it is possible to control the target board in an interface similar to that provided by standard Linux xterm.

All acquired data is written to a file. The file is transferred to host PC over serial port using the *sz* (a program to send data over serial port using ZMODEM error correction protocol) program .

A couple of the shell scripts that are run at boot time which make it possible to operate the PC/104 board as a text-terminal are the files *inittab* and *S20setserial* in the enclosed CD.

The minicom script for configuring the serial port for the host computer is the file *minirc.dfl* in the enclosed CD.

6.4 Ethernet Interface

In testing the Internet interface mode of operation, the Java based control software which runs on a Windows NT workstation must be run first before the Geosonde subsystem network application code can be run. This is because the control software runs as a server and all sensors connecting to it are run as clients.

Unlike the serial interface, all data acquired is immediately send over the Internet to the control software. The XML file *hawc.xml* (see enclosed CD) describes the interface to be created for the received data from the PC/104 system. The data can thus be viewed in any manner desired by the user. All that has to be done is to define in the XML file how the data received should be presented (these are essentially the data formats the control program expects from the Geosonde).

6.5 RTAI Linux

The RTAI Linux sub-kernel and its modules were fully integrated into the whole Linux system. Task communication and data exchange between kernel and user space was observed to be fully operational. Different timer values were passed to the RTAI Linux scheduler to test whether adequate timing of tasks was taking place.

6.6 Conclusions

This chapter has described the hardware setup for the Geosonde subsystem and the testing procedure for both the serial and the network mode of operation. The data used for testing purposes to emulate ab A/D acquisition card is also described. Both modes of operation and communication were tested and found to be working. Other functional requirements of the system were also tested and found to be working properly. These include RTAI Linux integration into subsystem and implementation of the NTP specification for client synchronizing

Chapter 7

Conclusions and Future Improvements

7.1 Conclusions

The literature has shown that MS-DOS is not a suitable operating system of choice for the Geosonde subsystem. It also showed that a Linux system endowed with real-time capabilities is a more viable option. The benefits of using an XML based interface to the subsystem were also shown. The manner in which new sensor descriptions are added to the control software also shows the simplicity with which new sensors can be added in the Data Logging and Control system.

The objective of this thesis was to firstly look into upgrading the MS-DOS system by introducing network capabilities to the system. As a more viable alternative to the DOS system part of the project requirements was to investigate and implement a real-time solution for the Geosonde radar system and provide for a network interface to the subsystem while still maintaining the serial mode of operation as a basic form of interaction with the subsystem.

The thesis objectives were achieved by first investigating the viability of MS-DOS as a suitable operating environment bearing in mind the required functionality to be introduced. A number of real-time kernels and/or operating systems were investigated and a choice was made taking into consideration the requirements imposed by the application. A software design was carried out and a minimal Linux system was built.

7.1.1 Software design

The software design of the Geosonde subsystem follows closely on the workings of the current system. The system requirements were thus drawn from the existing software. All the data requirements of the application were considered taking into account the limited resources of the PC/104 platform. The sub-kernel approach for implementing real-time under Linux influenced some of the design decisions made.

7.1.2 System implementation

A highly stripped-down Linux system was developed with only the minimal set of features required for basic system operation being implemented. A number of tools and packages were used in the build process. The RTAI Linux abstraction layer was included into the Linux kernel and several RTAI modules were included in the file system to provide runtime support for the application.

System testing of both modes of operation were carried out to ascertain whether data was indeed being exchanged between the PC/104 and a console workstation.

7.2 Future Work

Some of the points that are within the scope for future work are:

- The deployment phase of the Geosonde subsystem has to be preceded by further testing of the whole subsystem with the A/D card integrated into the system. Comedi [30] is a library of low-level driver modules and tools for common data-acquisition plug-in cards. The library provides Linux kernel modules that can provide an interface to the device for real-time tasks in kernel space. The Comedi library provides a common framework for writing drivers as a result acquisition devices that are not supported can be easily added to the library without much work.

- As part of future upgrades to the subsystem, the IDE device used in the implementation and testing of the project can be replaced by a Compact Flash card as this provides a similar interface as far as the TB486 BIOS and the Linux operating system is concerned. The footprint of the whole subsystem is about 1.8MB which means that a standard size of about 4MB to 16MB ought to be appropriate. The Linux 2.4 kernels come standard with Flash media support through the Journaling Flash File System[31]. All that is necessary to achieve

this is to configure the kernel for the specific device. By using JFFS, the subsystem can effectively be run as a disk-less embedded device.

- All the real-time development for the project was written in kernel-space which offers no memory protection. The RTAI Linux distribution offers an alternate approach to real-time development, by using LXRT. The LXRT module offers near real-time response and eases the development process of coding in kernel space. RTAI Linux makes it possible to convert LXRT tasks to hard real-time tasks after they have been thoroughly tested by calling a single function.

Appendix A

Geosonde Linux System scripts

Due to the amount and size of the various configuration scripts used by the embedded system as well as the build scripts for the various tools, all scripts and Makefiles have been omitted from the report.

See the enclosed CD for all scripts and Makefiles.

Each directory in the CD contains a README file with short descriptions of the contents.

Bibliography

[1] M.R. Inggs. Top Level Design of a Sensor Based Recording and Processing System.

[2] TB486 PC/104 Compatible computer, Technical Reference Manual.
`http://www.dspdesign.com/tb486prod.htm`

[3] `http://www.delorie.com/djgpp/`

[4] `http://www.bgnett.no/~giva/`

[5] D.B. Stewart. Introduction to Real Time. *Embedded Systems Programming,* 11/01/01, 12:40:42 PM EDT.
`http://www.embedded.com/story/OEG20011016S0120`

[6] K. Dankwardt. Real Time and Linux, Part 1. *ELJonline*, January 2002.
`http://www.linuxdevices.com/articles/AT5997007602.html`

[7] Getting started with eCos^{TM} (i386 PC edition July 2001) manual (ecos-tutorial-i386PC.pdf)
`http://sources.redhat.com/ecos/docs-latest/`

[8] `http://www.redhat.com/embedded/technologies/ecos/brochure.pdf`

[9] E. Nisley. Rating Real Time: Count the Ways. *Dr. Dobb's Journal,* September 2001.
`http://www.ddjembedded.com/resources/articles/2001/0109m/0109m.htm`

[10] T. Bird. *Comparing two approaches to real-time Linux,* Dec. 21, 2000.
`http://www.ddjembedded.com/resources/articles/2001/0109m/0109m.htm`

[11] `http://www.mvista.com/`

[12] `http://people.redhat.com/mingo/lowlatency-patches/`

[13] C.Williams. *Linux Scheduler Latency (Which is better – the preempt patch, or the low-latency patch? Both!*), March 20, 2002.
`http://www.linuxdevices.com/articles/AT8906594941.htm`

[14] K. Dankwardt. Real Time and Linux, Part 3: Sub-Kernels and Benchmarks. *ELJonline,* 2002.
http://www.linuxdevices.com/articles/AT6320079446.html

[15] D. Beal, S. Papacharalambous. Create Hard Real-time Tasks With Precision Under Linux. *Electronic Design*, 23 July 2001.

[16] I. Sommerville. *Software Engineering*, Fifth Edition. Addison and Wesley

[17] T. Fawcett. *The Linux Bootdisk HOWTO*, January 2002
`http://www.ibiblio.org/pub/Linux/docs/HOWTO/other-formats/html_single/Bootdisk-HOWTO.html`

[18] `ftp://uclibc.org/buildroot.tar.gz`

[19] `http://packages.debian.org/stable/doc/grub-doc.html`

[20] `http://www.uclibc.org/`

[21] `http://busybox.net/`

[22] `http://tinylogin.busybox.net/`

[23] `http://packages.debian.org/stable/base/setserial.html`

[24] `http://packages.debian.org/stable/net/udhcpc.html`

[25] `http://www.uclibc.org/uClibc-apps.html`

[26] `http://doolittle.faludi.com/ntpclient/`

[27] `http://freshmeat.net/projects/util-linux/?topic_id=861`

[28] `http://pioneer.gsfc.nasa.gov/public/iml/`

[29] D. Cox. XML for Instrument Control and Monitoring.Dr. *Dobb's Journal,* November 2001.
`http://www.ddjembedded.com/resources/articles/2001/0111i/0111i.htm`

[30] COMEDI driver library:
`http://stm.lbl.gov/comedi/`

[31] Journaling Flash File System:
http://www.developer.axis.com/software/jffs/