

AN EMBEDDED LINUX PLATFORM TO COLLECT, ANALYZE AND STORE
CRITICAL DATA FOR THE NAVIGATION OF AN AUTONOMOUS VEHICLE.

by

Sonia Thakur

A thesis submitted to the faculty of
The University of North Carolina at Charlotte
in partial fulfillment of the requirements
for the degree of Master of Science in the
Department of Electrical and Computer Engineering

Charlotte

2006

Approved by:

Dr. James M. Conrad

Dr. Andrew Willis

Dr. Bharat S. Joshi

ABSTRACT

SONIA THAKUR. An Embedded Linux Platform to Collect, Analyze and Store Critical Data for the Navigation of an Autonomous Vehicle (Under the direction of DR. JAMES M. CONRAD).

The purpose of a data acquisition system is to provide reliable and timely information. The received information can be then analyzed in real-time or remotely to assess the state of the measured environment. Similarly for an autonomous underwater navigation system it is critical to analyze the data received from the inertial measurement unit and other acoustic sensors to calculate its position and direct the vehicle. This thesis work is an effort toward the development of a data logging system based on an embedded Linux platform for an unmanned underwater vehicle.

Linux has been gaining popularity as an embedded operating system because of enormous advantages, like reusable device drivers and application programs, more convenient development environment, and easy resolution of problems within the open source community. An embedded Linux based board has been chosen as the core data processing unit of the Autonomous Underwater Vehicle (AUV). This thesis work implements device drivers required for interfacing the inertial sensors and GPS unit to the core processing unit. This system is intended to provide a development base for implementing various navigation algorithms, like Kalman Filter, to calculate the direction of the vehicle.

ACKNOWLEDGEMENTS

I would like to thank my graduate adviser Dr. James M. Conrad for his extraordinary support and understanding in guiding me through this thesis successfully. I would also like to express my gratitude to Dr. Bharat Joshi and Dr. Andrew Willis for serving on my thesis committee.

I would like to thank Sandeep Sirpatil for providing me valuable help to accomplish my thesis. I would also like to express my sincere appreciation and thank my friends Shweta Kautia, Christina Warren and other fellow students for their support throughout. I also want to express my gratitude towards the developers who post valuable information and their experience on the embeddedarm yahoo group.

TABLE OF CONTENTS

LIST OF FIGURES.....	vii
LIST OF TABLE.....	iii
LIST OF ABBREVIATIONS.....	x
CHAPTER 1: INTRODUCTION.....	1
1.1 Motivation	2
1.2 Current Work.....	4
1.3 Completed Thesis Work and its Contributions	5
1.4 Organization of Thesis	6
CHAPTER 2: INTRODUCTION TO LINUX.....	8
2.1 Introduction to Linux Operating System.....	8
2.2 Introduction to ARM9 Architecture	11
2.3 Overview of CirrusLogic EP903	12
2.3.1 Overview of TS-7200	12
CHAPTER 3: HARDWARE DESCRIPTION.....	15
3.1 Choice of Components	15
3.2 MAX197 External ADC Overview	15
3.3 Overview of GPS Receiver	17
3.4 Overview of RS-232.....	18
3.5 Overview of Accelerometer	20
3.6 Overview of Analog Compass R1655.....	22
3.7 Overview of Thermistor	24
3.8 The Complete Hardware Setup	25

CHAPTER 4: SOFTWARE DEVELOPMENT	26
4.1 Linux Environment	26
4.2 Introduction to GCC Compiler	26
4.2.1 Basic Operation of GCC	27
4.2.2 Input options	27
4.2.3 Output options	27
4.2.4 Makefile	29
4.3 Description of Main Programs	30
4.3.1 MAX197 External ADC	30
4.3.1.1 Address types used by Linux	30
4.3.1.2 Memory Mapping:	32
4.3.2 Timer 3	33
4.3.3 Register Configuration	35
4.4 GPS Receiver	35
4.4.1.1 RS232 Interface Development	36
4.4.1.1.1 Opening Serial Port	36
4.4.1.1.2 Configuring Serial Port	36
4.4.1.1.3 Ioctl Function	38
4.4.1.1.4 Reading and Writing to the Serial Port	38
4.4.1.1.5 Closing Serial Port	39
4.5 Compact Flash Card	39
4.6 Operation of the System	40
CHAPTER 5: CONCLUSION	43

5.1	Future work	44
REFERENCES.....		43
APPENDIX: CODE.....		46

LIST OF FIGURES

FIGURE 1.1: A Hydroid Remus AUV.....	1
FIGURE 1.2: An Autonomous Underwater Vehicle.....	2
FIGURE 1.3: Nekton Ranger AUV.....	2
FIGURE 2.1: A Split View of the Kernel	10
FIGURE 2.2: TS-7200 Hardware Components.....	14
FIGURE 3.1: Schematic of 8 channel A/D	16
FIGURE 3.2: EM-406 GPS Receiver Board.....	17
FIGURE 3.3: TS-7200 Interfaced with GPS Receiver.....	18
FIGURE 3.4: RS-232 Voltage Levels	19
FIGURE 3.5: ADXL311EB Schematic Diagram.....	21
FIGURE 3.6: TS-7200 Interfaced with Accelerometer.....	22
FIGURE 3.7: The Expected Sensor Output Voltages	23
FIGURE 3.8: TS-7200 Interfaced with Compass.....	23
FIGURE 3.9: Thermistor Resistance Vs Temperature (linear scale).	24
FIGURE 3.10: LM35 Typical Connection.....	25
FIGURE 3.11: The Complete System Setup.....	25
FIGURE 4.1: Steps to Produce an Executable Program or Shared Object	27
FIGURE 4.2: Address Types used in Linux.....	31
FIGURE 4.3: The Output from the ADC and the GPS Receiver.....	42

LIST OF TABLE

TABLE 3.1: TS-7200 10 pin Header output	20
TABLE 4.1: Output File Selection	28
TABLE 4.2: Timer 3 Register Map.....	34
TABLE 4.4: A/D Registers MAX197	35
TABLE 4.3: Termios Structure Members	37

LIST OF ABBREVIATIONS

ADC	Analog to Digital Converter
API	Application Program Interface
ARM	Advanced RISC Machine
AUN	Autonomous Underwater Navigation
AUV	Autonomous Underwater Vehicle
CPU	Central Processing Unit
DHCP	Dynamic Host Configuration Protocol
DTE	Data Terminal Equipment
ext2	Extended File System
FAT	File Allocation Table
GCC	GNU Compiler Collection
GPL	General Public License
GPS	Global Positioning System
IDE	Integrated Development Environment
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
IMU	Inertial Measurement Unit
I/O	Input/ Output
ISM	Industrial, Scientific, and Medical
ITU-T	Telecommunication Standardization Sector
MIPS	Million Instructions Per Second
MMU	Memory Management Unit

NFS	Network File System
NGIMU	Non Gyro Inertial Measurement Unit
NEMA	National Marine Electronics Association
OSS	Open Source Software
PC	Personal Computer
PDA	Personal Digital Assistant
POSIX	Portable Operating System Interface
PWM	Pulse Width Modulation
RAM	Random Access Memory
RISC	Reduced Instruction Set Computer
ROM	Read Only Memory
RTOS	Real Time Operating System
SBC	Single Board Computer
SDRAM	Synchronous Dynamic RAM
SPI	Serial Peripheral Interface
SRAM	Static Random Access Memory
SSH	Secure Shell
tty	Teletypewriter
UART	Universal Asynchronous Receiver/Transmitter
UI	User Interface
UUV	Unmanned Undersea Vehicle

CHAPTER 1: INTRODUCTION

Autonomous underwater vehicles (AUVs) are unmanned and undeterred submarines. They provide marine researchers with long range and low cost solution with which they gather not only oceanographic data but strategic military data as well. Underwater navigation for an autonomous vehicle is a major subject of concern for both the AUV community and their end users [1]. Rapid development on sensors and electronics technology in the past decade has made it possible that smaller, better performing and lower power AUVs can be built. The practicality of AUVs for commercial and military operations is dictated by not only the vehicle operational cost but also the design and development cost. Therefore, a low cost inertial measurement unit consisting of sensors like a magnetic compass, accelerometer and GPS are interfaced with the Embedded Linux based board. Figure 1.1, 1.2, and 1.3 shows an example of Autonomous Underwater Vehicle (AUV)



Figure 1.1: A Hydroid Remus AUV [2]

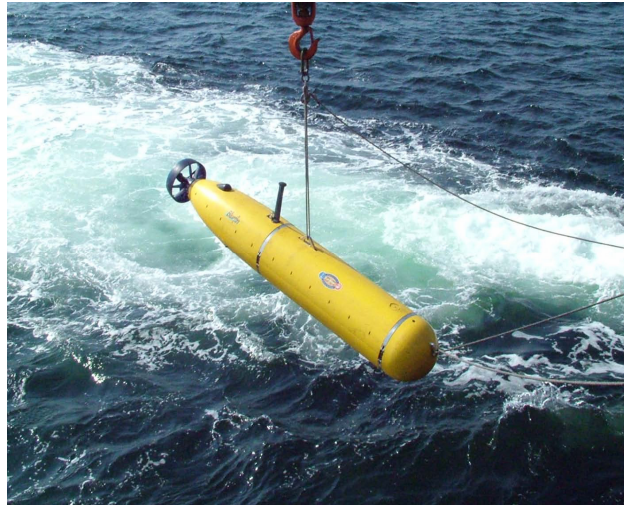


Figure 1.2: An Autonomous Underwater Vehicle

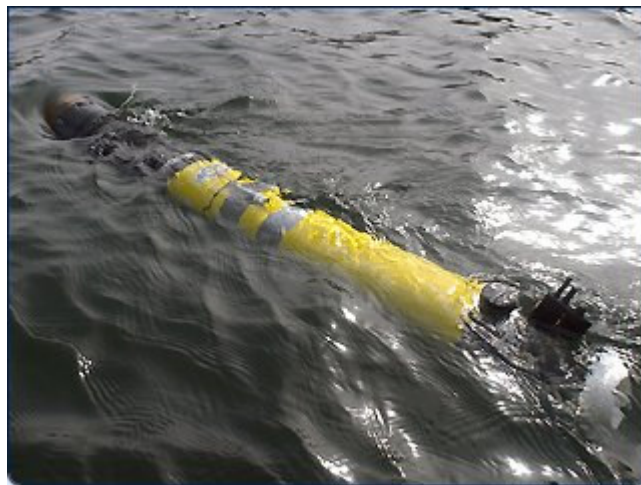


Figure 1.3: Nekton Ranger AUV

Some of the applications of AUVs are seabed mapping, environmental monitoring, research and inspection work for offshore industry and mine counter measures.

1.1 Motivation

The motivation of this thesis is to design an underwater navigation system composed of a low cost Embedded Linux platform and small sized sensors. For air or

ground vehicles, the Global Positioning System (GPS) with differential corrections (DGPS) can provide very precise and inexpensive measurements of geodetic coordinates. Unfortunately these GPS radio signals cannot penetrate beneath the ocean surface, and this poses a considerable constraint on the overhead of a vehicle mission, which must surface to obtain GPS fixes. Therefore, data from a low cost inertial measurement unit are integrated with GPS to produce continuously accurate navigation information [3].

The primary impetus for this thesis was to use a Linux-supported processor. Linux is available under the GNU General Public License (GPL) and is a part of the open source software (OSS) community [4]. According to a survey conducted by Venture Development Corporation as part of its Embedded Software Strategic Market Intelligence Program, commercial Embedded Linux owns approximately 50 percent more of the new project market than either Microsoft or Wind River Systems [5]. The considerations that make Linux attractive in the embedded domain:

- No licensing fee is needed, as it is available under GNU GPL.
- It is open source, which implies direct access to the source code. Therefore, individual parts of the operating systems can be customized and optimized which is impossible with rival commercial operating system.
- Linux is highly robust. The up-time is measured in weeks or years and a system rarely requires rebooting. The Linux kernel is also highly modular and scalable, giving it an advantage in resource-constrained environments.
- It is relatively straightforward to develop and debug an application on a host system running Linux.

Linux is a multi-tasking, multi-user, multi-processor operating system and supports a wide range of hardware processor platforms, such as x86, Alpha, SuperH, PowerPC, SPARC and ARM. After reviewing commercially available Autonomous Underwater Navigation (AUN) systems it was found that none of them utilized the Embedded Linux operating system for data processing.

1.2 Current Work

Most of the work in embedded Linux has been done in kernel development. Work by Lee [6] proposes to customize Linux as an application specific OS. To achieve this, processes based on reengineering called Call-Graph are implemented. Call-Graph has the ability to depict a program's calling structure. The basic concept used is to construct a kernel's calling structure and to remove the unnecessary code according to each specific application.

One of the important requirements for implementing real time operations in an operating system is to support scheduling algorithms. Work by Lin and Wang [7] addresses this by designing Red-Linux, which supports real time systems and non real-time jobs with different performance requirements. RT Linux addresses the issue of latency by inserting pre-emption points in the kernel.

Work by Kato [8] discusses the transition from the conventional RTOS to Linux for mobile phones. The requirements for mobile phone are memory size, stability, boot time/UI response time and power consumption. To reduce the memory size, executable binaries are read directly from the ROM, thereby reducing the use of the RAM. To reduce the user response time/boot-up time, Prelink was used, which locates the virtual address of each shared library uniquely and resolves symbol references before run time.

Work by Hill and Culler [9] describes the difference between the requirements of the wireless devices like mobile phone or PDA and wireless sensors. The MICA wireless platform has been introduced, which is suited for self-configuring multihop wireless networks with sensing, communication and I/O capabilities. The MICA hardware platform uses the TinyOS multithreading execution model. TinyOS performs high-level, long-running application processing in special execution contexts, called tasks.

Work by Li and Chiang [10] proposes the implementation of a TCP/IP stack as a self-contained component, which is independent of operating system and hardware. For adapting TCP/IP stack as a self-contained component for embedded systems, zero-copy mechanism has been incorporated for reducing protocol-processing overhead, memory usage and power consumption. In this mechanism data from a Network card is directly received in the user buffer and the data from the user buffer is directly sent to the network card.

Inertial Measurement unit (IMU) consists of an accelerometer and an angular rate sensors (gyro meters) to track the motion of the vehicle. The paper written by Wang, Ding and Zhao [11] proposes a configuration of nine accelerometer based non-gyro inertial measurement unit (NGIMU). Here the authors have expressed the angular acceleration in terms of the linear combination of the accelerometer outputs.

1.3 Completed Thesis Work and its Contributions

The main contribution of this thesis work, towards the research in AUV, is to use an embedded Linux supporting SBC as the core processor. The proposed work is to interface the Inertial Measurement Unit consisting of an accelerometer, compass, and

temperature sensor, along with a GPS receiver with an embedded Linux board. One of the main requirements for this thesis work was to write the code as generic as possible so that it could be ported to other Linux based SBC's like gumstick, which is utilized by Durham-based Nekton Research Inc. for their small autonomous vehicle.

It was required to identify a well-supported and inexpensive Linux based single board computer, selecting and installing a Linux distribution with the associated development tools and implementing device drivers for integrating the sensors. Many single board computers were considered and evaluated for their performance, features, price and software support. A SBC with inadequate support from the manufacturer in the form of device drivers, kernel patches and operating system will require enormous individual effort. TS-7200 SBC was selected for its features, good support by the manufacturer and a huge user community. The TS-7200 manufacturer supports the Debian distribution, it is well recognized for its stability and component package management, and it also has a large advanced user community.

The hardware platform setup and the device driver is expected to serve as a development platform for implementing a well designed Kalman filter or any other algorithm for processing the raw data accumulated from the sensor and calculating the exact location of the vehicle.

1.4 Organization of Thesis

The thesis report is divided into five chapters. Chapter 2 introduces the Linux operating system. It discusses the organization of the Linux kernel. Chapter 3 describes the hardware setup and the features of the individual components. Chapter 4 introduces the software component. It describes the software development environment, the Linux

environment, and the test setup. Chapter 5 details results and conclusions and suggests future work.

CHAPTER 2: INTRODUCTION TO LINUX

2.1 Introduction to Linux Operating System

Linux is an open-resource OS and supports diverse application, packages and device drivers. It was developed by Linus Torvalds at the University of Helsinki. It started as a hobby inspired by Andy Tanenbaum's Minix, a small UNIX like system, but has now grown to become a complete system of its own. It had its first official release in October 1991.

Linux can generally be divided into three major components: the kernel, the environment, and the file structure. These three together form the basic operating system structure. The kernel is the core program that manages the hardware devices. The environment receives the command from the user and sends them to the kernel for execution. The file structure organizes the files into directories. Each directory can be subdivided into subdirectories and store files [12].

Linux has the whole operating system – process management, memory management, file systems and drivers - contained in one binary image. The Linux kernel always resides in the memory. Each application program is loaded from disk to memory for its execution. When the program stops executing then the memory it occupies is discarded; that is, the program is unloaded. Because of this dynamic load unload capability Linux can support a range of features.

The kernel's role can be split, as shown in Figure 2.1 into the following parts:

Process management

The kernel creates and destroys processes and handles their connection (input and output). The scheduler, which controls how processes share the CPU, is part of process management.

Memory management

The different parts of the kernel interact with the memory-management subsystem through a set of function calls. On top of the limited available resources the kernel builds up a virtual addressing space.

File systems

The kernel builds a structured file system on top of unstructured hardware, and the resulting file structure is used throughout the whole system. Linux supports multiple file system types for example ext2, FAT etc. The Linux file system contains files and executables that the kernel requires as well as executables for the system. The kernel mounts a hard disk partition on the / (root) directory. The directories that exist beneath the / directory are:

/proc - Directory required by the proc filesystem

/etc - System configuration files and scripts

/sbin - Critical System binaries

/bin - Basic binaries considered part of the system

/lib - Shared Libraries to provide run-time support

/mnt - Mount point for maintenance

/usr - Additional utilities and applications

/dev - The dev directory is required to perform input/output for devices. Each file in this directory may be created using the mknod function.

Device control

All device control operations are performed by the driver that is specific to the device being accessed. The kernel has a device driver for every peripheral present on a system

Networking

The routing of packets and their address resolution is implemented within the kernel. The incoming packets are asynchronous events and also these network operations are not specific to a process. Therefore, Networking is managed by the operating system. The packets are collected, identified, and dispatched before a process takes care of them.

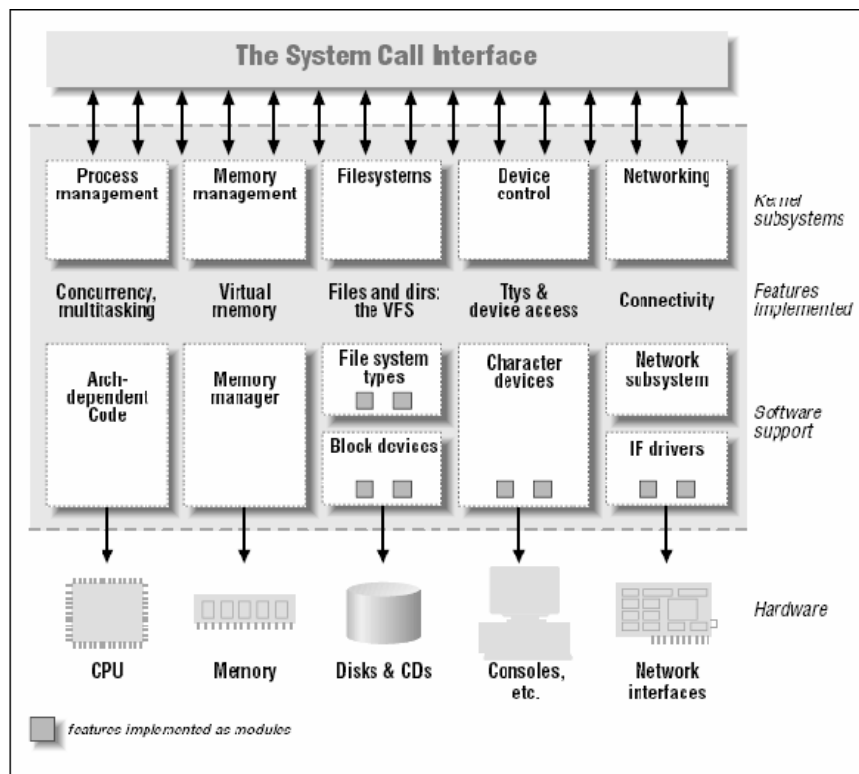


Figure 2.1: A Split View of the Kernel [13]

Several user-interface tools and programs enhance the versatility of the Linux basic kernel. Linux, ranges from being a stripped-down micro-kernel with memory management, process management and timer application to a full-fledged server supporting a complete range of file system and network services.

Software upgrades are modular in Linux, that is, applications can be upgraded and drivers can be loaded on the flash while the system is running. Configuration information and runtime parameters can be stored as data files on the flash.

A minimal Embedded Linux system needs just three essential elements [12]:

- A boot utility
- The Linux micro-kernel, composed of memory management, process management and timing services
- An initialization process

To do anything useful while remaining minimal, one also needs to add:

- Drivers for hardware
- One or more application processes to provide the needed functionality

As additional requirements become necessary, one might also want:

- A file system (perhaps in ROM or RAM)
- TCP/IP network stack
- A disk for storing semi-transient data and swap capability

2.2 Introduction to ARM9 Architecture

The ARM processor is based on 32bit RISC architecture. It is a simple load store architecture with a large register set, a reduced number of instruction classes, and a simple pipeline. The ARM9TDMI is an integer core whereas ARM940T is a cached

processor. Higher performance has been achieved by the ARM9TDMI by increasing the depth of the pipeline to 5 stages as compared to 3 stages in the ARM7TDMI. Forwarding paths have also been introduced to the pipeline in order to reduce the number of interlocks and hence have reduced the average number of clocks per instruction, CPI. The ARM9 is a Harvard architecture processor core with separate 16Kbyte instruction and data cache. The five-stage pipeline realized in the processor core consists of Instruction fetch, Instruction decode, execute, data memory access, and register write [14].

2.3 Overview of CirrusLogic EP903

The EP9302 features an advanced ARM920T processor design with a memory management unit (MMU) that supports Linux and many other operating systems. The included 16Kbyte instruction cache and 16Kbyte data cache provide zero cycle latency to the current program and data, or can be locked to provide guaranteed, no latency access to critical instruction and data. The core supports both the 32bit ARM and 16bit Thumb instruction set. The core can operate in big and little endian mode. Endianess affects both the address and data interfaces [15].

2.3.1 Overview of TS-7200

The TS-7200 single board computer (SBC) runs on a 200MHz EP9302 ARM9 processor with power as low as ½ Watt. As a general-purpose controller, it provides a standard set of peripherals on board [14].

The features of TS-7200 are:

- TSLinux Embedded Operating system Installed

- 200MHZ ARM9 CPU with MMU
- 8MB on-board Strata Flash
- 32MB RAM
- 2 USB 2.0 compatible ports
- 2 serial ports (up to 230 Kbaud)
- 10/100 Megabit Ethernet port
- 20 total digital I/O lines.
- Single +5VDC power supply
- PC/104 expansion bus

The TS-7200 features a true IDE compact Flash socket. A Compact Flash card in the socket appears as a hard drive to the operating system. Compact Flash cards are available in a wide range of capacities. A 512 MB card is sufficient to install a customized version of the Debian Linux distribution. The board also features USB and Ethernet ports which can be used to connect to other devices.

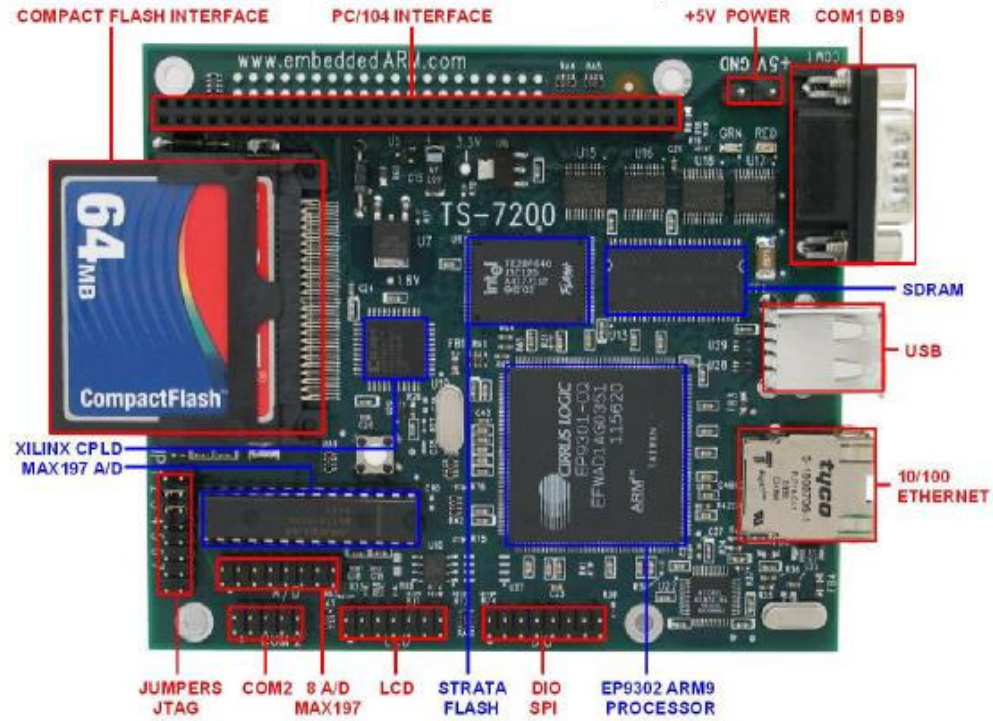


Figure 2.2: TS-7200 Hardware Components

This board was selected because of the above features. Other main features that are included are its low power consumption, availability of precompiled Linux kernel and the cross toolchain and the technical support that the company provides for Linux. This board is small and is priced economically.

CHAPTER 3: HARDWARE DESCRIPTION

3.1 Choice of Components

During the start of this work all the requirements to develop an autonomous navigation system were carefully considered and the choices of hardware components were made. Following section gives a brief description of the various hardware components used for the development of an AUV.

3.2 MAX197 External ADC Overview

The TS-7200 supports an optional 8 channel ADC, MAX197, with a conversion time of 12 μ s. This allows up to 60,000 samples per second. The MAX197 is a multi range fault tolerant data acquisition system. It uses successive approximation and internal input track/hold (T/H) circuitry to convert analog signal to 12bit digital output. Each channel is independently software programmable for a variety of analog input ranges.

Figure 3.1 shows the schematic of the MAX197 ADC. The dedicated MAX197 A/D header brings out all the 8 A/D channels.

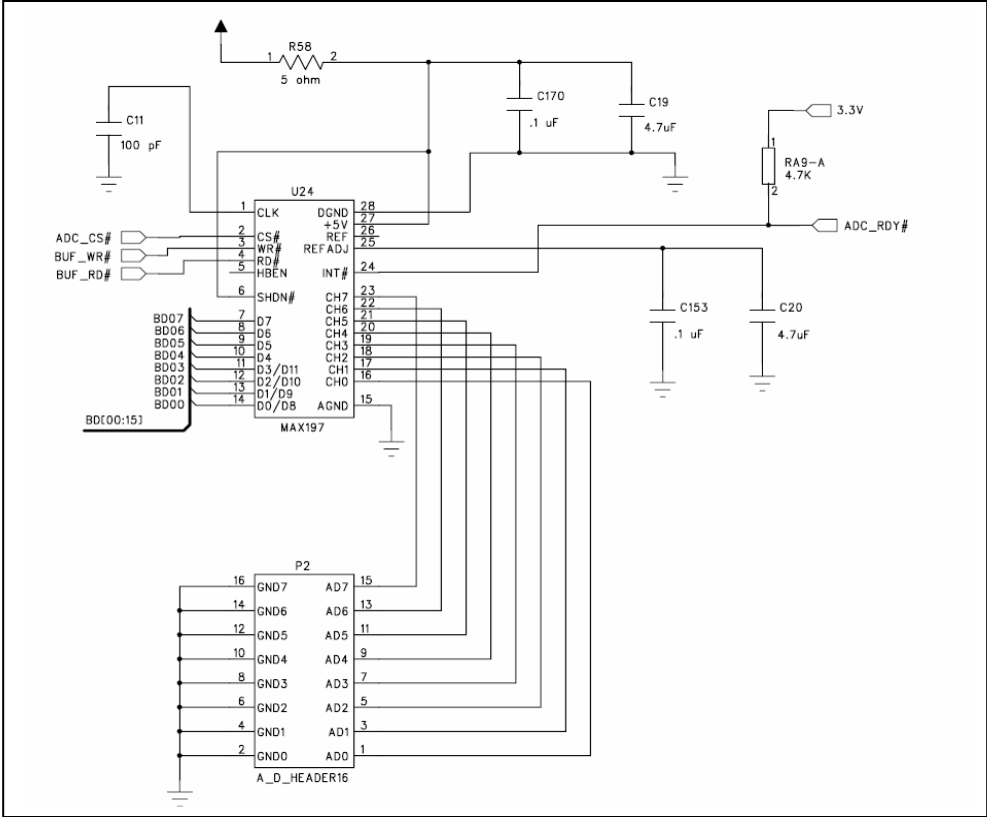


Figure 3.1: Schematic of 8-channel A/D [17]

Some of the key features of MAX197 ADC are as follows [17]:

- 12-Bit resolution, 1/2LSB linearity
- single +5V operation
- software-selectable input ranges: $\pm 10\text{V}$, $\pm 5\text{V}$, 0V to 10V, 0V to 5V
- fault-protected input multiplexer ($\pm 16.5\text{V}$)
- 8 analog input channels
- $6\mu\text{s}$ conversion time, 100ksps sampling rate
- internal 4.096V or external reference
- two power-down modes
- internal or external clock

3.3 Overview of GPS Receiver

To provide the absolute positioning of any object, GPS uses a constellation of 24 satellites. These satellites are monitored continuously from the five widely separate ground stations. Four satellites are visible at all times from any point on the earth's surface. Provided the four visible satellites have a very accurate clock, a four equation, four unknown systems can be used to extract the vehicle's position accurately. To further improve the accuracy and reduce the error from 20m to less than 5 m, differential correction for the GPS is used. In this, a precisely known ground station is used to estimate the range error in the GPS signal [1].

The GPS receiver selected for this application is SiRF star III based EM 402. This is a low cost, self-contained GPS unit with a passive antenna [19].



Figure 3.2: EM-406 GPS Receiver Board

Some of the main features of the GPS receiver are:

- SiRF star III high performance GPS chip set
- Supports NMEA 0183 data protocol
- Built in Super Cap to reserve system data for rapid satellite acquisition
- Built in patch antenna

The National Marine Electronics Association (NEMA) 0183 standard defines electrical signal requirements, data transmission protocol timing and specific sentence formats for a 4800 baud serial data bus.

Figure 3.3 shows the photograph of GPS module interfaced with the TS-7200. The GPS receiver is connected to the MAX232 chip, which is a RS232 serial driver. The serial output is connected to the COM2 port of the TS-7200.

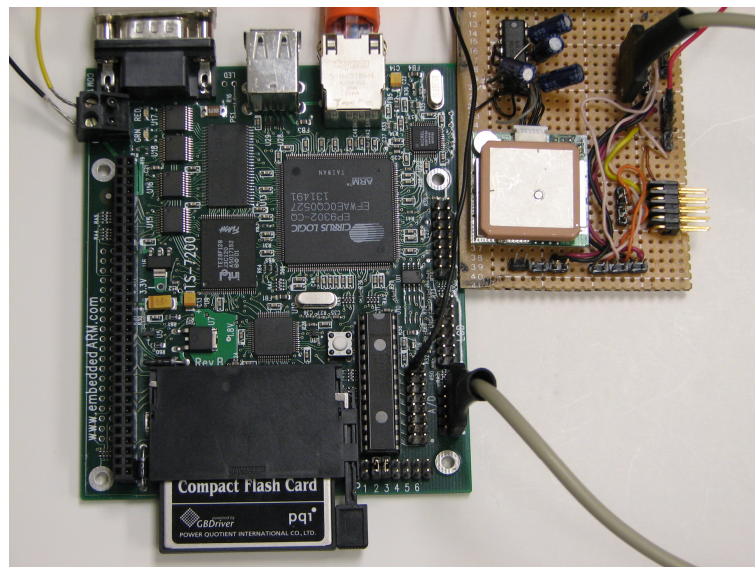


Figure 3.3: TS-7200 Interfaced with GPS Receiver

3.4 Overview of RS-232

RS-232 is a well known single-ended serial communication protocol. It is intended to support efficient data communication at low baud rates (<20kbps). Voltage levels with respect to common ground represent the RS-232 signals. Signal definitions in RS232 standard are:

- Ground
- Primary communication channel for data exchange and flow control.

- Secondary communication channel for controlling remote modem and for handshaking.

A signal between -3v to -25v signifies logic '0' while a signal between +3v to +25v signifies logic '1'. Thus, a serial communication in RS232 has a 50v voltage swing. The region between -3v to +3v is not defined for RS232 standard. Figure 3.4 illustrates the RS232 signal. [20]

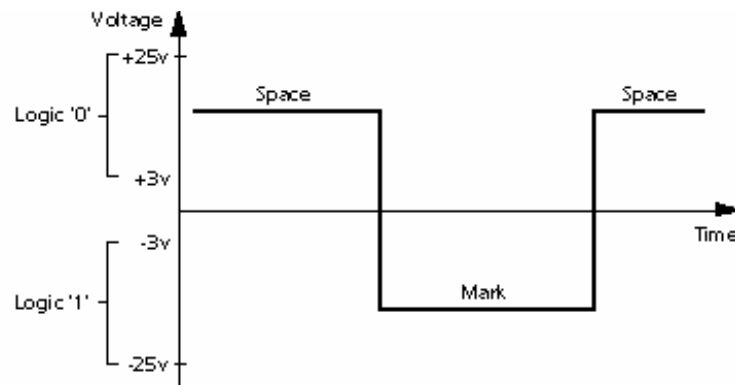


Figure 3.4: RS-232 Voltage Levels

Ports used for RS232 interface are of two types. D-type 25-pin connectors and D-type 9-pin connectors. D-type 25 pin connectors are obsolete but D9 are commonly used. Connecting just the primary channel of these connectors make them a Null Modem. A Null Modem is used to connect two DTE's (Data Terminal Equipment) together. This is a commonly used method in communicating an embedded device with a PC. It involves just a three wire (transmit, receive, and ground) connection. The grounds of both terminals are supposed to be common. Programming a serial port is fairly easy. Exchanging data by way of RS-232 port is similar to reading from or writing to a file.

The base address of COM2 on the SBC appears in the physical address space at 0x808D_0000, defined by the device driver provided by Technologic. COM2 has RS-

232 support only for the TXD and RXD signals. COM2 is accessible on a 10-pin header labeled COM2.

The pin out of the RS232 lines from the 10-pin COM2 header, and the pins to connect to a male DB9 standard DTE are shown in the Table 3.1

Table 3.1: TS-7200 10 pin Header output

COM2	Header	DB9
Pin 3	RS232 RXD	Pin 2
Pin 5	RS232 TXD	Pin 3
Pin 9	Ground	Pin 5

3.5 Overview of Accelerometer

An accelerometer is a device for measuring the linear acceleration of a rigid body along a single axis. Accelerometers are used along with gyroscopes, which measure the angular acceleration of a rigid body, to estimate the position of the inertial navigation system. It has been proposed [21] to calculate the angular motions of the rigid body relative to a fixed inertial frame from the accelerometer output. By integrating the output of the accelerometer, the velocity of the system is obtained. By further integrating the velocity the change of position of the body is evaluated along the accelerometer axis.

The ADXL311EB is an evaluation board, which is used to evaluate the performance of ADXL311 dual axis accelerometer. The output signals obtained from the board are analog voltage proportional to acceleration. The ADXL311 is capable of measuring both positive and negative acceleration to at least ± 2 g. The sensor is a surface micromachined polysilicon structure built on top of the silicon wafer.

Polysilicon springs suspend the structure over the surface of the wafer and provide a resistance against acceleration forces. Deflection of the structure is measured using a differential capacitor that consists of independent fixed plates and central plates attached to the moving mass. Acceleration deflects the beam and unbalance the differential capacitor, resulting in an output square wave whose amplitude is proportional to acceleration [22].

The schematic and parts of the ADXL311EB are shown in the Figure 3.5 Two 100 nF capacitors are installed at X out and Y out to reduce the bandwidth to 50Hz.

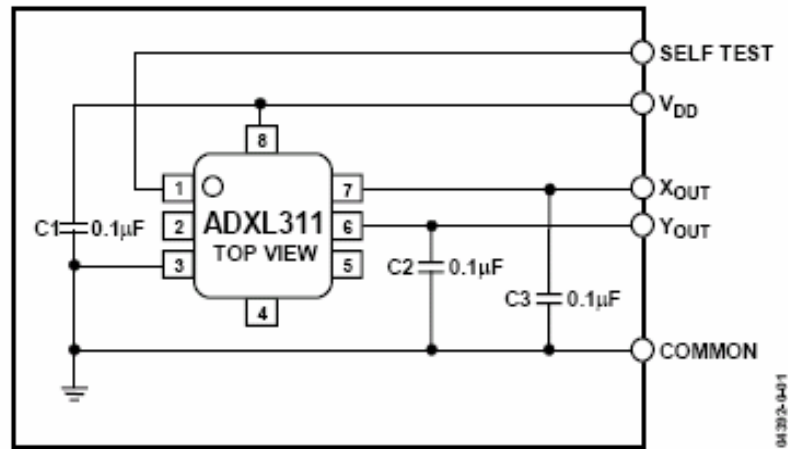


Figure 3.5: ADXL311EB Schematic Diagram

The Figure 3.6 shows the photograph of accelerometer interfaced with TS-7200. The accelerometer is connected to the SBC through the ADC. After every 5 seconds the data is fetched from the accelerometer. This collected data is logged in the Compact Flash, as well as displayed on the telnet screen.

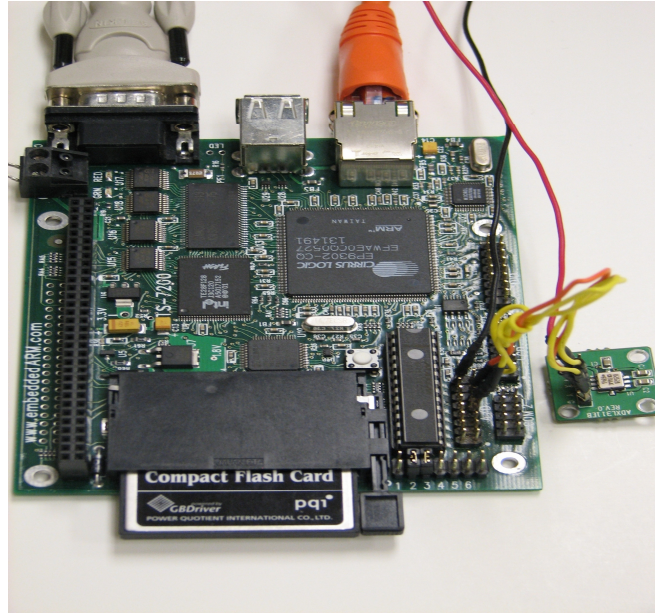


Figure 3.6: TS-7200 Interfaced with Accelerometer

3.6 Overview of Analog Compass R1655

Dinsmore Analog Compass R1655 is based on Hall-effect technology. The Hall effect is observed when a magnetic field is applied at right angles to a rectangular sample carrying an electric current. Due to an electric field applied at right angles to both the current and the magnetic field a voltage appears across the sample.

The sensor is constructed to operate in a vertical position with the leads down. The sensor is designed to measure the direction of the horizontal component of the earth's flux field [22]. The analog output from the compass closely resembles a sine-cosine set of curves, which cross at approximately 2.5 volts and peak at approximately 3.1 volts and floor at about 1.9 volts. The Figure 3.7 shows the expected output from the sensor. Each output drives up to 4 mA.

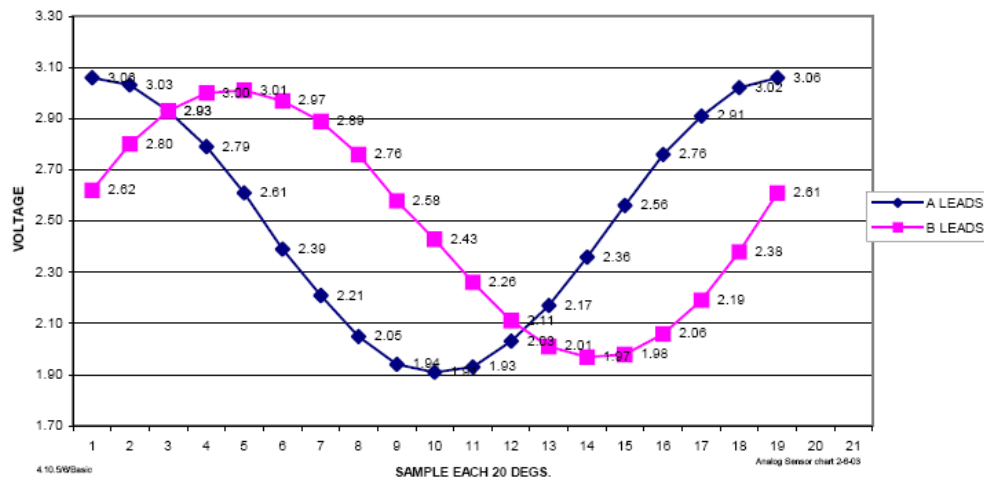


Figure 3.7: The Expected Sensor Output Voltages

Figure 3.8 shows the photograph of the analog compass interfaced with the TS-7200. The compass is connected to the TS-7200 through the ADC.

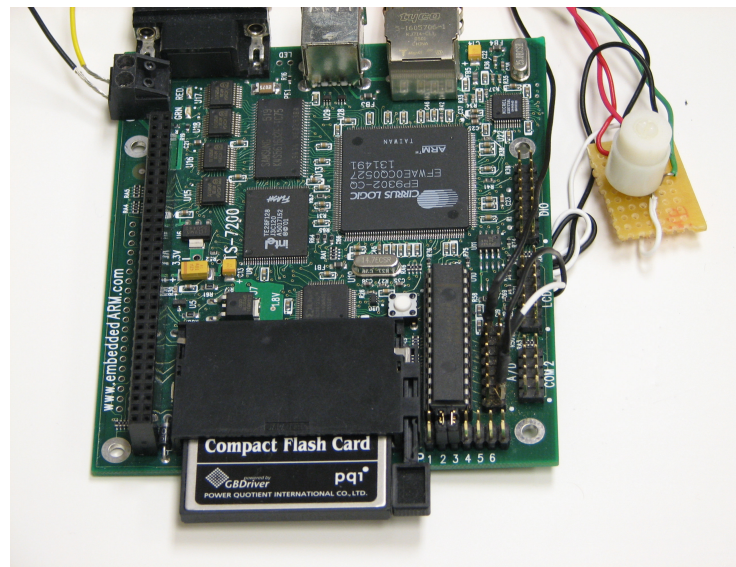


Figure 3.8: TS-7200 Interfaced with Compass

3.7 Overview of Thermistor

Thermistors have a negative temperature coefficient, that is, the resistance of the device decreases as temperature increases. The resistance of Thermistors is highest at lower temperatures. Figure 3.9 shows the decrease in Thermistor resistance with increase in temperature.

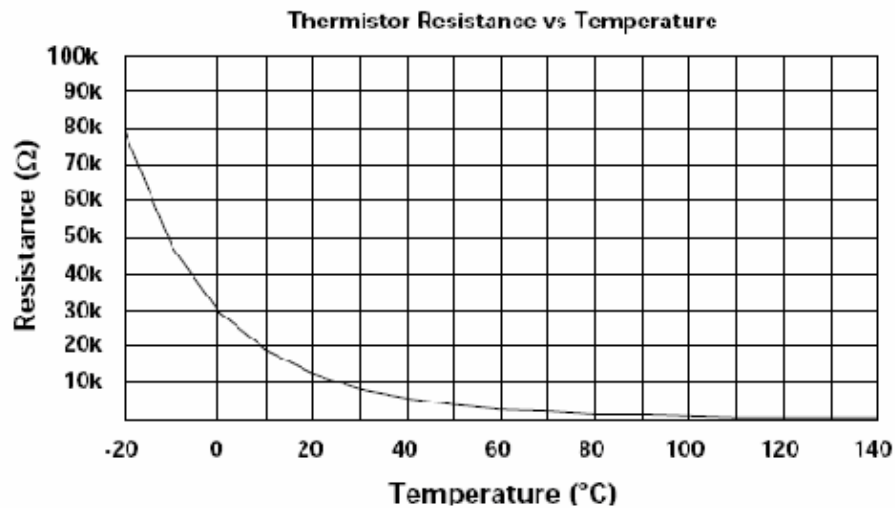


Figure 3.9: Thermistor Resistance Vs Temperature (linear scale).

The LM35 is a three terminal device that produces output voltages proportional to $^{\circ}\text{C}$ ($10\text{mV}/^{\circ}\text{C}$), so the nominal output voltage is 250mV at 25°C and 1.000V at 100°C . These sensors can measure temperatures below 0°C by using a pull down resistor from the output pin to the voltage below the ground pin. Figure 3.10 illustrates the typical connection of LM35 for any application [24].

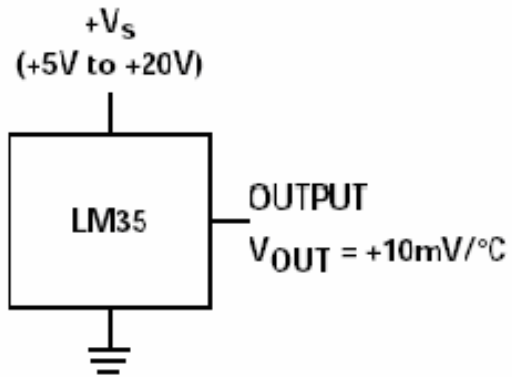


Figure 3.10: LM35 Typical Connection

3.8 The Complete Hardware Setup

A photograph of the complete hardware module is taken and shown in Figure 3.11. The interface used to communicate between single computer board and other sub modules is RS-232 and the external ADC.

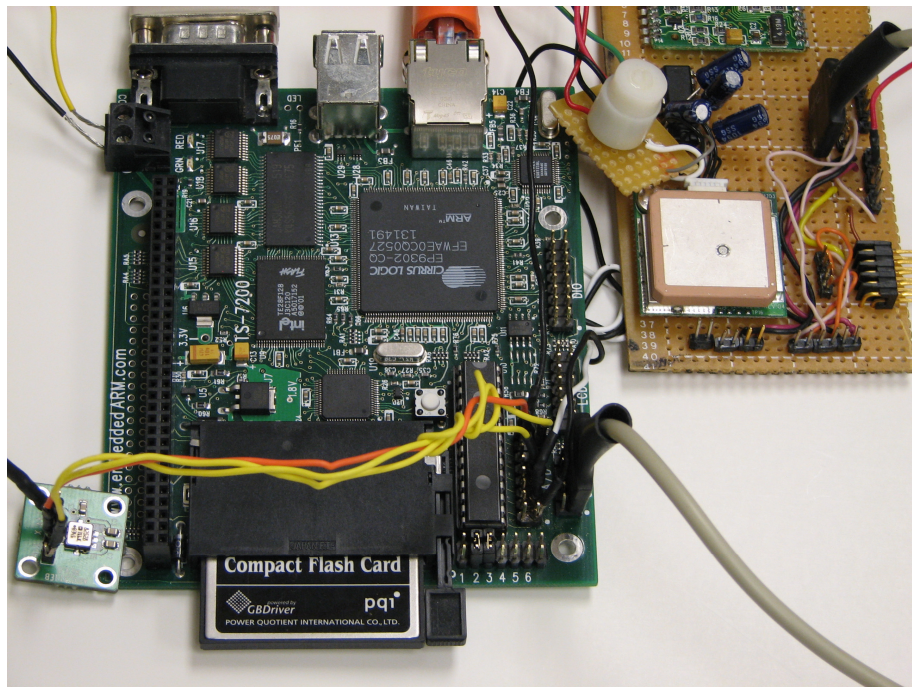


Figure 3.11: The Complete System Setup

CHAPTER 4: SOFTWARE DEVELOPMENT

4.1 Linux Environment

TS-Linux embedded distribution is installed on the on-board Flash memory of TS-7200. TS-Linux is a compact distribution, which is based on BusyBox and is therefore, ideal for a small executable footprint. BusyBox combines tiny versions of many common Linux/UNIX utilities into a single small executable. A full-featured Linux distribution can also be run on either the Network File System (NFS) root file system or installed on a large Compact Flash card.

The on-board Flash contains the TS-Linux kernel, which is a standard kernel with patches to customize for this hardware. The version of the on-board kernel was ts-8.

4.2 Introduction to GCC Compiler

The GNU C compiler is part of the GNU tool chain, which works with the gdb source level debugger. Together they provide all the software tools needed for the development of an embedded Linux system [25].

Earlier GCC used to refer to the GNU C Compiler, but now it is called as GNU Compiler Collection. GCC is a collection of integrated compilers for C, C++, Objective-C, Java, FORTRAN, and Ada programming languages.

4.2.1 Basic Operation of GCC

Preprocessing, compiling, assembly, and linking constitute the basic GCC operations. The options passed to GCC are either directed to the compiler or are directed to of these other components of the toolchain. The exception to this is platform selection, optimization flags, and debugging, which can pass arguments to both the compiler and other components.

Figure 4.1 shows the steps taken by the GNU Toolchain to produce an executable program. The GCC driver accepts options to control the toolchain indirectly.

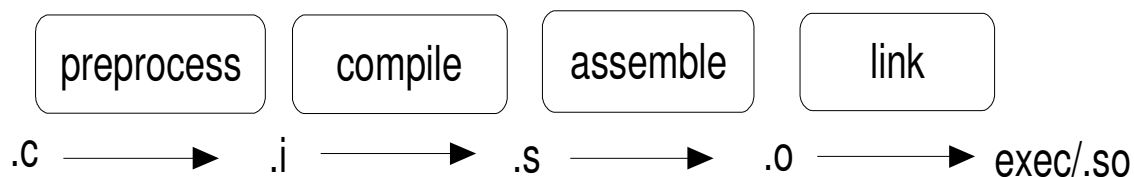


Figure 4.1: Steps to Produce an Executable Program or Shared Object

4.2.2 Input options

A C source file is compiled differently than a C++ source file. Therefore the compiler must know the type of the input file to be processed. The file extension of the source file is an implicit option to the compiler and determines which GCC compiler is invoked. For example, `file.c` invokes the C compiler, while `foo.C` invokes the C++ compiler.

4.2.3 Output options

With the command line options or with the source file name extension the output of the compiler can be controlled. The GCC driver can pass instructions to the rest of the toolchain to either manufacture a complete executable, or it may stop at a

number of intermediates. Therefore it is essential for the compiler to know the type of output that is expected.

Table 4.1 summarizes output file selection.

Table 4.1: Output File Selection

Flag	Action	Output/Format
-C	Assemble, but do not link	.o file
-S	Compile, but do not assemble	.s file
-E	Preprocess, but do not compile	Preprocessed .c file

The Technologic Systems not only provides a full-featured Linux OS, but also GNU installation for ARM. To compile the code with gcc cross compiler it is essential to ensure that the CrossTool binaries are in the systems path. The command to export the path of the gnu gcc compiler is:

```
export PATH=$PATH:/opt/croostool/arm-unknown-linux-gnu/gcc-3.3.2-glibc-2.3.2/bin
```

The example below creates the executable file, hello.exe, of the helloworld.c file:

```
arm-unknown-linux-gcc -Wall -o hello helloworld.c
```

The -Wall option uses many of the warnings, but not all. The warnings that are issued by -Wall is a rule-of-thumb. [26]

The -O option is the grouped optimization flag, which allows the compiler to select a group of optimization options with one flag. These options are used to optimize for either the performance or for reducing the code size. The -O flag is a group optimization flag that includes optimizations and does not take a lot of compile time to

implement. Even at the expense of performance –o optimizes for program size. There is no group option for selecting all performance optimizations available to the compiler.

To automate the compilation process make tool was implemented.

4.2.4 Makefile

Make speeds up the edit-compile debug process. It minimizes rebuild time because it can determine which files have changed, and thus only rebuilds files whose components have changed. A Makefile is a text file database containing rules that tell make what to build and how to build it. A rule consists of the following.

- A target, which is usually a binary or object file that one wants to create.
- A list of one or more dependencies required to build the target.
- A list of commands to execute in order to create the target from the

specified dependencies

Makefile Rules for compiling the code for the system:

```
#path to the include directory
includedir    = $(HOME)/conrad3/kernel24/include

# path to the crosstool
INSDIR        = /opt/crosstool/gcc-4.0.1-glibc-2.3.5/arm-unknown-linux-
gnu/bin

#to make an executable
extern_adc:   $(extern_adc)

$(CC) $(nov1) $(LDFLAGS) -o $@
```


4.3 Description of Main Programs

Source Code of this project primarily can be divided into three main programs:

- MAX197 external ADC
- Header file for the external ADC
- RS232 configuration code

These codes are described in detail in the section below.

4.3.1 MAX197 External ADC

A user space program was written for the MAX197 external ADC. The physical memory cannot be accessed directly through the user development side. The following sections describe the various address types used by Linux and how to memory map them. We also discuss how timer 3 is initialized to periodically access the data after every 5 seconds from the external ADC.

4.3.1.1 Address types used by Linux.

Linux is a virtual memory system. This means that the addresses seen by user program do not directly correspond to the physical addresses used by the hardware. With Virtual memory, programs running on the system can allocate far more memory than the system's available physical memory. [29]

User Virtual address

These are the addresses seen by user-space programs. User addresses are either 32 or 64 bits long depending on the underlying hardware architecture, and each process has its own virtual address space.

Physical Addresses

The addresses used between the processor and the system's memory. Physical addresses are 32 or 64 bit quantities; even 32 bit systems can use 64 bit physical addresses in some situations.

Bus addresses

Bus addresses depend on the architecture of the system. These are the addresses used between the peripheral buses and the memory.

Kernel logical addresses:

The kernel logical addresses map most or all of the main memory, and are often treated as physical addresses. They constitute the normal address space of the kernel.

Kernel virtual addresses

They do not always directly map to physical addresses. That is why they are different from the kernel logical addresses.

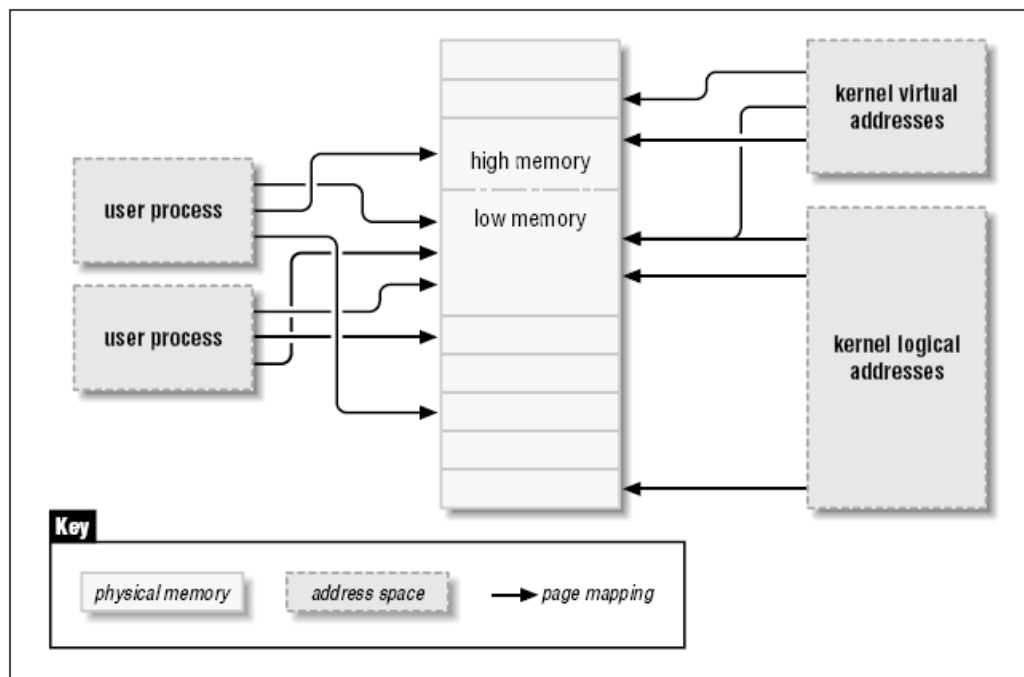


Figure 4.2: Address Types used in Linux

4.3.1.2 Memory Mapping:

The direct access to the device memory is provided by Memory Mapping (mmap). The mmap is part of the file operation structure. When mmap system call is issued, it gets invoked. The “/dev/mem” device implements a way to access the physical memory from the protected user space. It allows the readings and writings to any specific memory register. The following is an example on using the “/dev/mem” device with C:

```
unsigned char *dat, *start;

off_t addr = 0x11e00100;

int fd = open("/dev/mem", O_RDWR);

off_t page = addr & 0xffff000;

start = mmap(0, getpagesize( ), PROT_READ|PROT_WRITE, MAP_SHARED, fd,
page);

dat = start + (addr & 0xfff);

close(fd);
```

The prot argument describes the desired memory protection and should not conflict with the open mode of the file.

PROT_EXEC

Pages must be executed

PROT_READ

Pages may be read

PROT_WRITE

Pages may be written

PROT_NONE

Pages may be accessed

The flag parameters specify the type of the mapped object and the mapping options. It also specifies whether the modifications made to the mapped copy of the page are to be shared with other references or are private to the process. The flag parameters are:

MAP_FIXED

This option stipulates to the kernel not to select a different address than the one specified. If the memory region specified by start and length overlaps pages of any existing mapping, then the overlapped part of the existing mapping is discarded.

MAP_SHARED

This option specifies the kernel to share this mapping with all the other processes that map this object.

MAP_PRIVATE

This option specifies the kernel to create a private copy. A store to the region does not affect the original file. It is not specified whether the changes made to the file after the `mmap()` call are visible in the mapped region or not.

`fd` should be a valid file descriptor.

Offset should be multiple of the page size as returned by `getpagesize()`

4.3.2 Timer 3

Timer 3 is a 32-bit counter and is referred to as TC3. The counter is loaded with a value written to the data register. This value is then decremented on the next active clock edge after the write. When the timer's counter decrements to "0", it generates an

interrupt. There are two clock sources available one is 5.8 KHz and the other is 2 KHz.

Both of these clock sources are synchronized to the main system bus clock.

Table 4.2: Timer 3 Register Map

Address	Read Location	Write Location	Size
0x8081_0080	Timer3Load	Timer3Load	32 bits
0x8081_0084	Timer3Value	-	32 bits
0x8081_0088	Timer3Control	Timer3Control	32 bits
0x8081_008C	Reserved	Timer3Clear	1 bit

Timer 3 Load

The load register contains the initial value of the timer. In the periodic timer mode it is used as the reload value. The timer is loaded by writing to the load register when the timer is disabled.

Timer 3 Value

The value register gives the current value of the timer. When the timer load register is initialized, the value register is also updated with this value.

Timer 3 clear

The interrupt generated by the timer can be cleared by writing any value to the clear location.

Timer3 control

The control register provides enable/disable and mode configurations for the timer.

Another alternative to transmit data from the external ADC after every 5 seconds is to use the sleep () function.

```
unsigned int sleep(unsigned int seconds);
```

sleep () makes the current process sleep until the defined seconds have elapsed

4.3.3 Register Configuration

Table 4.4 shows the memory addresses to which the external ADC is memory mapped. The MAX197 ADC is an optional external ADC. Therefore, to check whether A/D option is installed on the SBC or not, a read to I/O location 0x2240_0000 bit 0 is done. If the return value is a “1” then A/D option is installed on the SBC. One of the disadvantages of this external ADC is that 0x1080_0000 has to be polled continuously to check if the conversion is completed. [29]

Table 4.3: A/D Registers MAX197

I/O Address	Action
0x10F0_0000 Write	Initiate A/D Conversion
0x10F0_0000 Read	LSB of Conversion
0x10F0_0001 Read	MSB of Conversion
0x2240_0000 Read	Bit 0 = 1 if A/D option installed
0x1080_0000 Read	Bit 7 = 0 when Conversion completed

4.4 GPS Receiver

As shown in Section 3.3, the GPS receiver is interfaced to the TS-7200 through the serial port. Linux distinguishes devices into three fundamental device types: a char module, a block module, or a network module.

A character (char) device is one that can be accessed as a stream of bytes; a char driver implements this behavior. Char devices are accessed by means of filesystem nodes, such as /dev/tty1 and /dev/lp0.

4.4.1.1 RS232 Interface Development

Developing a serial class is classified as: opening serial port, configuring serial port, checking for input data, reading from port, writing to port, and closing the port.

4.4.1.1.1 Opening Serial Port

Serial port is considered as a "char device" in Linux. Each serial terminal line is represented as a file in the /dev (device) directory. The ttyS [0-3] is the file under which each serial port is available in the system, where 0-3 represents the comp port. The "COM2" of the TS-7200 is connected to the GPS, which can be accessed by the device file "/dev/ttyAM1". To edit the serial port file open () is used [30].

```
int fd; /* File descriptor for the port */

fd = open("/dev/ttyAM1", O_RDWR | O_NOCTTY | O_NDELAY);

if (fd == -1)
```

The first argument is the file to open. The second argument is to set the read write permission to the file. The O_NOCTTY flag indicates to the system that this program is not the controlling terminal for that port. The O_NDELAY flag indicates to the system that this program doesn't care - whether the other end of the port is up and running or not. If this flag is not specified then the process is put to sleep until the data carrier detect signal is the space voltage.

4.4.1.1.2 Configuring Serial Port

To set the attributes of the serial port, a termios structure is configured. The termios.h header file is included in the code development. The two functions used are *tcgetattr()* and *tcsetattr()*. These get and set terminal attributes, respectively and

provide a pointer to the termios structure. The options available in the termios structure are:

Table 4.4: Termios Structure Members

Member	Description
c_cflag	Control options
c_lflag	Line options
c_iflag	Input options
c_oflag	Output options
c_cc	Control characters
c_ispeed	Input baud(new interface)
c_ospeed	Output baud(new interface)

The `cfsetospeed` and `cfsetispeed` functions are provided to set the output and input baud rate respectively in the termios structure.

```
cfsetispeed(&options, Bbaudrate)
```

```
cfsetospeed(&options, Bbaudrate)
```

The following example shows how to set the parameters for the serial port for 8N1:

```
options.c_cflag &= ~PARENB
```

```
options.c_cflag &= ~CSTOPB
```

```
options.c_cflag &= ~CSIZE;
```

```
options.c_cflag |= CS8;
```


4.4.1.1.3 ioctl Function

The **ioctl** function manipulates the underlying device parameters of special files like the serial terminal file. The `sys/ioctl.h` header file is included in the code development.

```
int ioctl(int fd, int request, ...)
```

The first argument is an open file descriptor, the second is the request code number and the third is either an integer value, possibly unsigned or a pointer to data.

```
The fionread functions returns when data is available to read.
ioctl(fd,FIONREAD,&bytes);
if (bytes <= 0 ) return 0;
```

The ioctl functions returns a positive value equal to the number of characters in a read buffer.

4.4.1.1.4 Reading and Writing to the Serial Port

To write data to the port - `write()` system call is used.

```
n = write(fd, "ATZ\r", 4);
if (n < 0)
    fputs("write() of 4 bytes failed!\n", stderr);
```

The `write` function returns the number of bytes sent or -1 if an error occurred.

The port was configured for raw data bytes therefore, the `read()` system call returns the number of characters that are actually available in the serial input buffers.

```
read(fd, buf, count);
```

In the above example the `read ()` attempts to read up to count bytes from file descriptor fd into the buffer defined by buf.

4.4.1.1.5 Closing Serial Port

To close the serial port, use the *close()* system call:

```
close(fd);
```

4.5 Compact Flash Card

The data received from the ADC and the GPS module is continuously displayed on the telnet screen. This data is simultaneously logged in the CompactFlash, which is mounted on the board, to analyze the data received. Following are the steps, which show how to format the CompactFlash card and mount it on the SBC.

- -using a standalone Linux pc, bring up a terminal window
- -connect the compact flash card to the PC using USB adapter
- -type fdisk /dev/sda1
- format the compact flash. The instruction are given in the [27]
- m for displaying the help menu
- d (to delete the existing partitions. Repeat for all the partitions)
- n (for new partition)
- p (for primary partition)
- 1 (make the new partition primary number 1)
- hit the enter key for the default starting cylinder
- hit the enter key again for the default last cylinder
- to make the first partition bootable , enter 'a' at the command prompt and then enter 1 to make the first partition bootable.

- to commit these changes to the disk, enter 'w' to write out to the new partition table

At this point the compact card has been partitioned. It still needs to be formatted with the Linux compatible format. Type the following command:

```
$mkfs.ext2 /dev/sda1
```

The compact flash card is now formatted and can be removed from the USB adapter. Plug the compact flash card into the TS-7200 when it is OFF (the compact flash is not hot swappable)

- turn on the TS-7200 and log on as root
- fdisk -l command is used to check for the partitions recognized by the Linux
- to mount the drive, use the following command where mnt/cf is the mount point for the CF card.

```
mount -t ext2 /dev/ide/host0/bus0/target0/lun0/part1 /mnt/cf
```

This however will mount the drive for the current boot, when the system is rebooted the compact flash will not be mounted.

- to make the compact flash drive to mount on boot up, go to /etc/fstab and add the following line to this file using vi as a text editor

```
/dev/ide/host0/bus0/target0/lun0/part1 /root/cf ext2 defaults 0 0
```

The drive will now be mounted automatically on boot up. The files can now be retrieved from this drive.

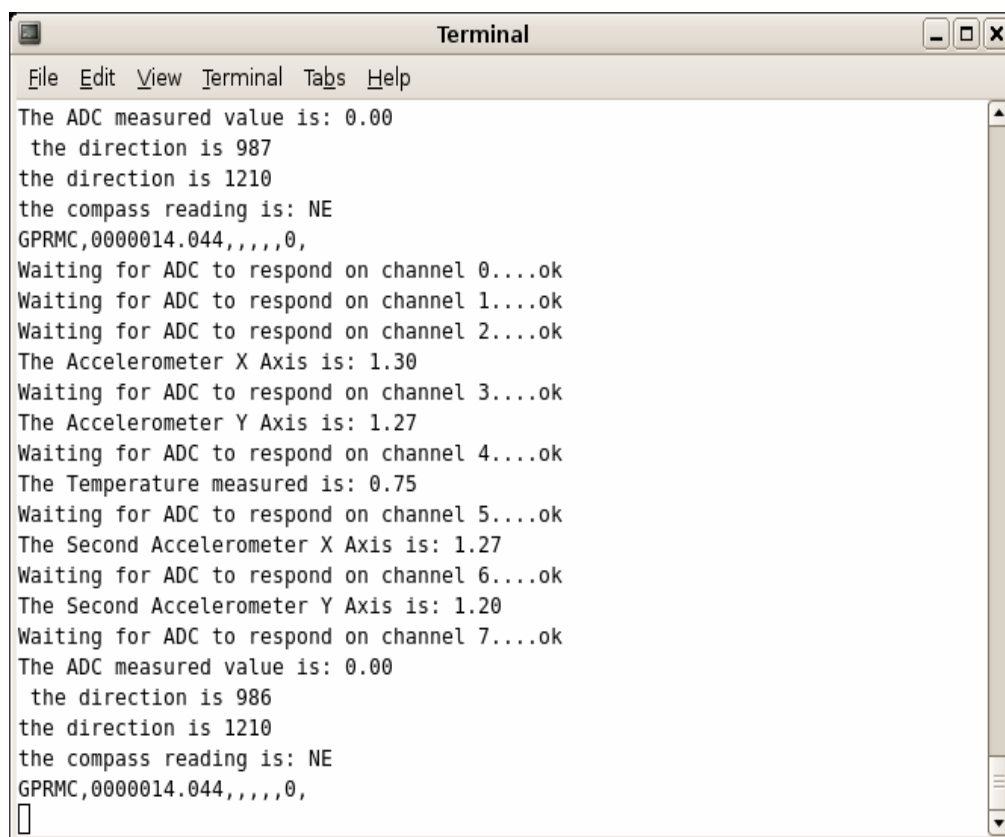
4.6 Operation of the System

The system was setup as described in the section 3.7 and illustrated in Figure 3.11. The ADC mode selected is unipolar and the range selected is 10V. All the 8

channels are sampled and there data is displayed on the minicom and logged in the Compact Flash card. The vehicle is manned by a ship on the surface and therefore data is transmitted to it through the Ethernet port. The SBC is accessed through telnet and the data is sent in the interval of 5 seconds to the PC.

The device drivers were written for the external ADC on the SBC. The timer 3 was initialized, which can be accessed from the user space, to transmit the ADC data after every 5 seconds. The Comp 2 of the SBC was configured to collect the data from the GPS module. The baud rate selected for the serial communication is 4800bps.

Figure 4.3 shows the data collected from the sensors. Channel 2 and 3 of the ADC are connected to the Accelerometer one and channel 5 and 6 are connected to the second accelerometer. The channel one and two are connected to the Compass and the value obtained from it is utilized to calculate the direction. Channel 4 is connected to the Thermistor, which is used to find the temperature of the vehicle. The GPS module continuously sends data to the SBC. The NEMA code displaying the longitude and latitude is selected and displayed.

A screenshot of a terminal window titled "Terminal". The window has a menu bar with "File", "Edit", "View", "Terminal", "Tabs", and "Help". The terminal displays a series of text outputs from an ADC and a GPS receiver. The outputs include ADC measured values, direction readings, compass readings, and GPRMC data. There are also several "Waiting for ADC to respond on channel X...ok" messages. The terminal ends with a cursor on a new line.

```
File Edit View Terminal Tabs Help
The ADC measured value is: 0.00
  the direction is 987
the direction is 1210
the compass reading is: NE
GPRMC,0000014.044,,,,,0,
Waiting for ADC to respond on channel 0....ok
Waiting for ADC to respond on channel 1....ok
Waiting for ADC to respond on channel 2....ok
The Accelerometer X Axis is: 1.30
Waiting for ADC to respond on channel 3....ok
The Accelerometer Y Axis is: 1.27
Waiting for ADC to respond on channel 4....ok
The Temperature measured is: 0.75
Waiting for ADC to respond on channel 5....ok
The Second Accelerometer X Axis is: 1.27
Waiting for ADC to respond on channel 6....ok
The Second Accelerometer Y Axis is: 1.20
Waiting for ADC to respond on channel 7....ok
The ADC measured value is: 0.00
  the direction is 986
the direction is 1210
the compass reading is: NE
GPRMC,0000014.044,,,,,0,

```

Figure 4.3: The Output from the ADC and the GPS Receiver.

CHAPTER 5: CONCLUSION

The objectives of the thesis work have been achieved with the implementation of driver for the external ADC and the GPS receiver on a Linux SBC and demonstrate the use of such a setup in AUN system. The SBC with Linux setup and the development tools form the embedded Linux development system. The complete system with the hardware setup and the software code can be used in the following ways.

- As an embedded Linux development system with the development tools, interfaces such as Ethernet and USB, and many other applications.
- Investigation and development of Unmanned Land Surface Navigation system.
- Evaluation of various aspects of Autonomous Underwater Navigation system.
- As a tool to implement a non gyro inertial navigation unit.
- As a data logger/analyzer of various sensor measurements for Marine research.
- As an educational tool in teaching Autonomous Navigation system and embedded Linux.

5.1 Future work

This thesis work can be extended in many ways. The exact position of the vehicle can be determined by processing the logged sensor data using the Kalman filter. The current version does not have a driver for the Digital I/O port. This can be implemented to increase the number of sensors that can be interfaced with the SBC.

The TS-7200 has an on board Apache server. Therefore the data logged from the SBC can be displayed on a webpage. For implementing this, a cgi script in shell can be written which will display the file opened by the SBC to log data on the Compact Flash. Thus, data can be transmitted through Ethernet to a webpage at <http://192.168.0.50>.

REFERENCES

- [1] Grenon, G., Edgar, P., Smith, S., and Healey, A., "Enhancement of the inertial navigation System for the Morpheus Autonomous Underwater Vehicles", *IEEE Journal of Oceanic Engineering*, Vol. 26, No. 4, October, 2001.
- [2] Autonomous Underwater Vehicle
<http://www.blueviewtech.com/?page=applications§ion=2>
- [3] Yun, X.; McGhee, R.; Whalen, R.; Roberts, R; Healey, A, and Zyda, M., "Testing and Evaluation of an Integrated GPS/INS System for Small AUV Navigation" *IEEE Journal of Oceanic Engineering*, Vol. 24, Issue 3, pp.396–404, 19-20 July, 1999.
- [4] Lennon, A., "Embedding Linux," *IEEE Review*, Vol. 47, Issue 3, pp. 33 - 37, May 2001.
- [5] Henkel, J., "Software development in embedded Linux – informal collaboration of competing firms," *Proceedings of the 6th International Tagung Wirtschaftsinformatik*, Vol. 2, pp. 81-99, September, 2003.
- [6] Lee, Che-Tai; Hong, Zeng-Wei, and Lin, Jim-Min, "Linux kernel Customization for Embedded Systems by Using Call Graph Approach," *Design Automation Conference, 2003. Proceedings of the ASP-DAC 2003. Asia and South Pacific*, pp. 689 – 692, January, 2003.
- [7] Lin, K. J., and Wang, Y.C., "The Design and Implementation of Real-Time schedulers in RED-Linux," *Proceedings of the IEEE*, Vol. 91, No. 7, July 2003.
- [8] Kato, K.; Yamamoto, T.; Hirota, T., and Mizuyama, M., "Embedded Linux Technologies to Develop Mobile Phones for the Mainstream Market," *Consumer Communications and Networking Conference, 2006. CCNC 2006. 2006* *IEEE*
3rd
Vol. 2, pp. 1073 – 1077, 8-10 Jan. 2006.
- [9] Hill, J.L., and Culler, D.E., "Mica: a wireless platform for deeply embedded networks," *IEEE Micro*, Vol. 22, Issue 6, pp. 12 – 24, November, 2002
- [10] Li, Yun-Chen, and Chiang, Mei-Ling, "LyraNet: A Zero-Copy TCP/IP Protocol Stack for Embedded Operating Systems," *Proceedings of the 11th IEEE International Conference on embedded and Real-Time Computing Systems and Applications*, pp. 123-128, August, 2005.

- [11] Wang, Q., Ding, M., and Zhao, P., "A New Scheme of Non-gyro Inertial Measurement Unit for Estimating Angular Velocity," *29th Annual Conference of Industrial Electronics Society, IECON*, Vol. 2, pp. 1564-1567, November, 2003.
- [12] Wall, K., *Linux Programming by Example*, Indianapolis, Ind. Que, ISBN: 0789722151, 2000
- [13] Rubini, A., and Corbet, J., *Linux Device Drivers*, 2nd Edition, O'REILLY Online, 2002.
- [14] Segars, S., "The ARM9 Family – High Performance Microprocessors for Embedded Applications," *Proceedings International Conference on Computer Design: VLSI in Computers and Processors*, pp. 230 - 235, 5th October, 1998.
- [15] Cirrus Logic EP9301 User's Guide
http://www.embeddedarm.com/downloads/Components/EP9301_User_Guide.pdf
- [16] Linux for ARM on TS-7200 User's Guide
<http://www.embeddedarm.com/Manuals/linuxarm-guide-rev2.0.pdf>
- [17] MAX197 Multi-Range ($\pm 10V$, $\pm 5V$, $\pm 10V$, $+5V$), Single $+5V$, 12-Bit DAS.
<http://www.embeddedarm.com/downloads/Components/MAX197.pdf>
- [18] Jalving, B., Gade, K., Hagen, K., and Vestgard, K., "A Toolbox of Aiding Techniques for the HUGIN AUV Integrated Inertial navigation System," *OCEANS 2003 Proceedings*, Vol. 2, pp. 1146-1153, September, 2003.
- [19] GPS Receiver Engine Board Manual
http://www.sparkfun.com/datasheets/GPS/EM-406%20Product_Guide1.pdf
- [20] RS-232 Specification and Standard
http://www.lammertbies.nl/comm/info/RS-232_specs.html
- [21] Tan, W., and Park, S., "Design of Accelerometer-Based Inertial navigation Systems," *IEEE Transaction on Instrumentation and Measurement*, Vol. 54, Issue 6, pp. 2520-2530, December, 2005.
- [22] Analog Devices ADXL311 Datasheet
http://www.analog.com/UploadedFiles/Data_Sheets/39398238692761ADXL311_a.pdf
- [23] Dinsmore Analog Compass R1655 Datasheet
<http://www.jelu.se/shop/pdf/sensor%20information.pdf>

- [24] LM35 Thermistor Datasheet
<http://www.national.com/ds/LM/LM35.pdf>
- [25] Hollabaugh, C., *Embedded Linux Hardware, Software, and Interfacing*, Addison Wesley, Indianapolis, ISBN 0672322269, 2005.
- [26] GCC Manual
<http://gcc.gnu.org/onlinedocs/gcc-3.4.3/gcc>
- [27] Nakajima, T.; Sugaya, M., and Oikawa, S., “Operating Systems for Building Robust Embedded Systems,” *Proceedings of the 10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems*, pp. 211-218, 2-4 February, 2005.
- [28] Perez, S., and Vila, J., “Building Distributed Embedded Systems With RTLinux-GPL” *Proceedings of the IEEE Conference Emerging Technologies and Factory Automation*, Vol. 1, pp.161 – 168, 16-19 Sept. 2003.
- [29] TS-7200 Hardware Manual, 200 MHZ ARM9 Embedded PC with 32 MB RAM.
<http://www.embeddedarm.com/Manuals/ts-7200-manual-rev2.2.pdf>
- [30] Serial Communication How to
<http://www.tldp.org/HOWTO/Serial-HOWTO.html>
- [31] NMEA Reference Manual
<http://www.sparkfun.com/datasheets/GPS/NMEA%20Reference%20Manual1.pdf>
- [32] <http://www-128.ibm.com/developerworks/linux/>
- [33] <http://www.linuxdevices.com>
- [34] Linux How To
<http://www.tldp.org>

APPENDIX

External ADC code

```

/*****
*   File Name: rs232.c
*   Version:   2.0
*   Author:    Sonia Thakur
*   License:   GNU General Public License
*   Purpose:   to configure the RS232 port(comp 2)
*              For serial Communication
*
*****/

#include <unistd.h>
#include <sys/types.h>
#include <stdio.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <termios.h>
#include <stdlib.h>
#include <string.h>
#include <sys/stat.h>
#include <sys/uio.h>
#include <errno.h>

int open_port(char* device_name) {
    int fd;
    fd = open(device_name, O_RDWR | O_NOCTTY| O_NDELAY);
    if(fd == -1)
    {
        printf("Failed to open device \n");
    }
    return fd;
}

/*****
*   File Name: rs232.c
*   Version:   2.0
*   Author:    Sonia Thakur
*   License:   GNU General Public License
*   Purpose:   to read from the RS232 port(comp 2)
*              For serial Communication
*
*****/
int read_port(int fd) {
    char buffer[100];
    char bufptr[6] = "GPRMC";
    char bufptr1[6] = "GPGGA";
    char *bufstring;

```

```

int count = 0;
int lines= 0;

    int i=0;

buffer[i] = 0;
while (lines < 110)
{
    ioctl(fd,FIONREAD, &count);
    //printf("the value of count is: %d\n",count);
    if(count <= 0) return 0;
    count =read(fd, buffer,60);
    //printf("the count is: %d\n",count);
    lines++;
    /* Add terminator and print if non-zero */

    if(count > 0)
    {
        bufstring = buffer;
        if(strncmp (bufstring,bufptr, 5) == 0)
        {
            usleep(4);
            printf("%s\n",buffer);
        }
        else if(strncmp (bufstring, bufptr1, 5)==0)
            printf("%s\n", buffer);

    }
    else printf("not a valid data\n");
}

return count;
}
/*****
*   File Name: rs232.c
*   Version:   2.0
*   Author:    Sonia Thakur
*   License:   GNU General Public License
*   Purpose:   to configure the RS232 port(comp 2)
*               For serial Communication
*
*****/

void port_configuration(int fd) {
    struct termios options;
    fcntl(fd, F_SETFL, FNDELAY);
    tcgetattr(fd, &options);
    printf("speed in %d out %d\n", options.c_ispeed,
options.c_ospeed);
    printf("mode in %d out %d\n", options.c_iflag,
options.c_oflag);
    printf("control flag %d\n", options.c_cflag);

```

```

printf("local flag %d\n", options.c_lflag);

options.c_cflag |= (CLOCAL|CREAD);
options.c_cflag &= ~PARENB;
options.c_cflag &= ~CSTOPB;
options.c_cflag |= CS8;
options.c_cflag &= ~CRTSCTS;
cfsetispeed(&options,B4800);
cfsetospeed(&options,B4800);

/* Enable data to be processed */
options.c_lflag &= ~(ICANON | ECHO | ECHOE | ISIG);
options.c_iflag &= ~(IXON | IXOFF | IXANY);
options.c_oflag &= ~OPOST;
tcsetattr(fd, TCSANOW, &options);
}

int main(int argc, char*argv[]) {
    extern int errno;
    int fd;
    /* check the command */
    if(argc < 2)
    {
        printf("serial_tester<device>\n");
        return 0;
    }

    /* Open the port */
    printf("Opening device %s\n", argv[1]);
    fd = open_port(argv[1]);
    printf("the fd id is : %d\n",fd);
    /* Get port configuration */
    port_configuration(fd);

    /* Set the port to non blocking and read the port */
    while (1)
        if(read_port(fd)!=0)printf("the error is :
%d\n",errno);
        //sleep(1);
        //read_port(fd);
        //printf("the number of lines is: %d\n",numb);
        close(fd);
        return 0;
}

```

```

*****
#*      File Name: Makefile                                *
#*      Version:   1.0                                    *
#*      Author:    Sonia Thakur                          *
#*      License:   GNU General Public License            *
#*      Purpose:   A general makefile to compile the     *
#*                  Programs                              *
#*                                                         *
*****

SHELL          = /bin/sh

DEMOS          =
PROGS          = adcfinal

srcdir         = .

# *****

includedir     = $(HOME)/conrad3/kernel24/include
INSDIR        = /opt/crostoool/gcc-4.0.1-glibc-2.3.5/arm-
unknown-linux-gnu/bin

#MANDIR        = /usr/local/man/man1
#LOCALINS     = $(HOME)/bin
#LOCALMAN     = $(HOME)/man/man1

# *****

#CC            = gcc
CC            = arm-unknown-linux-gnu-gcc
CFLAGS        = -O
CPPFLAGS      = -I. -I$(includedir)

CCFLAGS       = $(CFLAGS) $(CPPFLAGS)

LINK          = $(CC)

# libraries

#LDFLAGS       = -static

adcfinal      = adcfinal.o

# if there is an include file uncomment this. Add anyother
dependancies

.c.o:
    $(CC) -c $(CCFLAGS) $<

all: $(PROGS)

```

```

clean:
    rm -rf *~ *.o $(PROGS)

backup: dist

dist:
    rm -rf *~ *.o
    (d=`basename $$PWD` ; cd .. ; tar cfz $$d.tgz $$d)

adcfinal:      $(adcfinal)
               $(CC) $(adcfinal) $(LDFLAGS) -o $@

install:       $(PROGS)
               @for f in $(PROGS) $(DEMOS) ; do \
                   install -D --backup $$f $(INSDIR)/$$f; done

/*****
*   File Name: adcfinal.c                               *
*   Version:   3.0                                       *
*   Author:    Sonia Thakur                             *
*   License:   GNU General Public License               *
*   Purpose:   A general include file for the whole     *
*               application                               *
*****/

#include<unistd.h>
#include<sys/types.h>
#include<sys/mman.h>
#include<stdio.h>
#include<fcntl.h>
#include<assert.h>
#include<time.h>
#include<stdlib.h>

#define      PLD_ATOD_SET          0x01
#define OPTIONS                    0x22400000
#define CTRL_BYTE                  0x10f00000
#define BUSY                       0x10800000
#define CHANNEL_0                  0x50
#define CHANNEL_1                  0x51
#define CHANNEL_2                  0x52
#define CHANNEL_3                  0x53
#define CHANNEL_4                  0x54
#define CHANNEL_5                  0x55
#define CHANNEL_6                  0x56
#define CHANNEL_7                  0x57
#define BUSYBIT                    0x80

```

```

int main(int argc, char **argv)
{
    //FILE *fp;
    volatile unsigned char *options, *controlByte, *busy;
    unsigned char *registerValue;
    unsigned short *results;
    double percentage;
    int ctrlByte, n;

    int RawCompass, Compass, Crosszero, Zerocross;
    int Uplimit, Lowlimit;
    int Scale, Spread;
    int A_Curve, B_Curve;
    //int a ,b;
    int Value;
    Uplimit = 1200;
    Lowlimit = 900;
    Scale = 35;

    int fd = open("/dev/mem", O_RDWR|O_SYNC);
    assert(fd != -1);
    setvbuf(stdout, NULL, _IONBF, 0);

    /* lets initialize our pointer */
    options=mmap(0,getpagesize(), PROT_READ|PROT_WRITE,
MAP_SHARED, fd, OPTIONS);
    assert(options !=MAP_FAILED);

    controlByte=mmap(0,getpagesize(), PROT_READ|PROT_WRITE,
MAP_SHARED, fd, CTRL_BYTE);
    assert(controlByte != MAP_FAILED);

    busy=mmap(0,getpagesize(), PROT_READ|PROT_WRITE,
MAP_SHARED, fd, BUSY);
    assert(busy != MAP_FAILED);

    printf("checking if MAX197-ADC option is set...");
    registerValue=(unsigned char *)options;
    if(*registerValue & PLD_ATOD_SET) {
        printf("ok\n");
    }
    else {
        printf("FAIL, not set\n");
        return 1;
    }
    while(1)
    {

        /*lets do data conversion */
        for(ctrlByte = CHANNEL_0; ctrlByte <= CHANNEL_7;
ctrlByte++)
        {

```



```

        *controlByte = ctrlByte;

        printf("Waiting for ADC to respond on channel
%d....",ctrlByte-CHANNEL_0);
        registerValue = (unsigned char *)busy;
        n=0;
        while(n<14 && (*registerValue & BUSYBIT) !=
0x0) {
            usleep(1<n);
            n++;
        }
        if (n ==14){
            printf("FAIL, time out\n");
            return 4;
        }
        else {
            printf("ok\n");
        }
        if(ctrlByte ==(CHANNEL_0))
        {

            results = (unsigned short *) controlByte;
            /*percentage =(((double) *results) * 10)/4096; */
            /*printf("the adc result got is %3.2f\n",
percentage);
            fp = fopen("received_data", "ab");
            fprintf(fp, "%3.2f\n",percentage);
            fclose(fp); */
            A_Curve = *results;
        } else if(ctrlByte == (CHANNEL_1))
            {
                results = (unsigned short *) controlByte;
                B_Curve = *results;

            }
        else if(ctrlByte == (CHANNEL_2))
            {
                results = (unsigned short *)controlByte;
                percentage = (((double) *results) * 10)/4096;
                printf("The Accelerometer X Axis is: %3.2f\n",
percentage);

            }
        else if(ctrlByte == (CHANNEL_3))
            {
                results = (unsigned short *)controlByte;
                percentage = (((double) *results) * 10)/4096;
                printf("The Accelerometer Y Axis is: %3.2f\n",
percentage);

            }
        else if(ctrlByte == (CHANNEL_4))

```

```

        {
            results = (unsigned short *)controlByte;
            percentage = (((double) *results) * 10)/4096;
            printf("The Temperature measured is: %3.2f\n",
percentage);
        }
        else if (ctrlByte == (CHANNEL_5))
        {
            results = (unsigned short *)controlByte;
            percentage = (((double) *results) * 10)/4096;
            printf("The measured ADC value is:
%3.2f\n",percentage);
        }
        else if (ctrlByte == (CHANNEL_6))
        {
            results = (unsigned short *)controlByte;
            percentage = (((double) *results) * 10)/4096;
            printf("The measured ADC value is:
%3.2f\n",percentage);
        }
        else (ctrlByte == (CHANNEL_7))
        {
            results = (unsigned short *)controlByte;
            percentage = (((double) *results) * 10)/4096;
            printf("The measured ADC value is:
%3.2f\n",percentage);
        }
    }

    printf(" the direction is %3d\n",A_Curve);
    printf("the direction is %3d\n", B_Curve);
    //fp = fopen("received_data", "ab");
    //fprintf(fp, "%3.2f\n", percentage);
    //fclose(fp);

    if(A_Curve == B_Curve)
    {
        if(A_Curve > 2048){
            Uplimit = A_Curve;
        }
        else if(A_Curve < 2048)
        {
            Lowlimit = A_Curve;
        }
        Spread = 9000 / (Uplimit - Lowlimit);
        Scale = (Spread * 10)/4;
    }
    if(A_Curve < Lowlimit)
    {

```

```

        RawCompass = (Uplimit - B_Curve) * Scale;
        Compass = (( RawCompass/10) * 4)/10;
        Value = Compass + 900;
        printf("the compass reading is: %3s\n", "NW");
    }
    else if(A_Curve > Uplimit)
    {
        RawCompass = (B_Curve - Lowlimit) * Scale;
        Compass = ((RawCompass/10) * 4) /10;
        printf("the compass reading is:
%3s\n", "SE");

        Crosszero = Compass + 2700;

        if( Crosszero >= 3600)
        {
            Zerocross = Crosszero - 3600;
            printf("the compass reading is Zerocross:
%3d\n", Zerocross);
        }

    }

    else if(B_Curve > Uplimit)
    {
        RawCompass = (Uplimit - A_Curve) * Scale;
        Compass = ((RawCompass/10) * 4)/10;
        printf("the compass reading is:%3s\n", "NE");
    }
    else if(B_Curve < Lowlimit)
    {
        RawCompass = (A_Curve - Lowlimit) * Scale;
        Compass = ((RawCompass/10) * 4)/10;
        printf("the compass reading is: %3s\n", "SW");
    }
    else if((A_Curve == 1200) | (A_Curve <= 1220))
        printf("the compass reading is: %2s\n", "S");
    else if((B_Curve == 1200) | (B_Curve <= 1220))
        printf("the compass reading is: %2s\n", "E");
    else if((A_Curve == 900) | (A_Curve <= 930))
        printf("the compass reading is:%2s\n", "N");
    else if((B_Curve == 900) | (B_Curve <= 930))
        printf("the compass reading is: %2s\n", "W");
    sleep(5);
}

return 0;
}

```

```

/*****
*   File Name: adcfinal.h
*   Version:   3.0
*   Author:    Sonia Thakur
*   License:   GNU General Public License
*   Purpose:   A header file for the external ADC
*
*****/

#define ADC_Busy          0x10800000
#define Conversion_Mask   0x00000000
#define Reg_val           0x22400000
#define Reg_Val_Mask      0x00000001
#define Extern_ADC_Result 0x10f00000
#define Data_Mask         0xffff

/* declaration */
static char is_extern_ADC(unsigned char IO);
int read_channel(unsigned char lsb, unsigned char control);
static char extern_ADC_busy(unsigned char complete);

/*functions */

static char is_extern_ADC(unsigned char IO)
{
    unsigned short val;
    val = PEEK32(IO);
    if((val & Reg_Val_Mask) == Reg_Val_Mask){
        return 1;
    }
    else {
        return 0;
    }
}

int read_channel(unsigned char lsb, unsigned char control)
{
    unsigned short val;
    POKE32(lsb, control);
    while((extern_ADC_busy(complete)) != 0x0);
    val = PEEK32(lsb);
    val = val & Data_Mask;
    return val;
}

static char extern_ADC_busy(unsigned char complete)
{
    int n;
    unsigned short val;
    val = PEEK32(complete);
    n=0;

```

```
while(n<14 && (val & Conversion_Mask) != 0x0){
    usleep(1<<n);
    n ++;
}
if(n==14) {
    printf("FAIL, timed out\n");
    return 0;
}else {
    printf("ok\n");
    return 1;
}
}
```