An Embedded Linux Platform to Collect, Analyze and Store Critical Data for the Navigation of an Autonomous Vehicle

Sonia Thakur
University of North Carolina at Charlotte
sthakur@uncc.edu

James M. Conrad
University of North Carolina at Charlotte
jmconrad@uncc.edu

Abstract

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 work is an effort toward the development of a data logging system based on an embedded Linux platform for an unmanned underwater vehicle. An embedded Linux based board has been chosen as the core data processing unit of the Autonomous Underwater Vehicle (AUV). This 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.

1. Introduction

Autonomous Underwater Vehicles (AUVs) are unmanned and undeterred submarines. They provide marine researchers with a long range and low cost solution with which they gather not only oceanographic data but strategic military data. Underwater navigation for an autonomous vehicle is a major subject of concern for both the AUV community and their end users [1]. Some of the applications of AUVs are seabed mapping, environmental monitoring, research and inspection work for offshore industry and mine counter measures. Figure 1 shows an example of an AUV.

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.

To meet these goals of high functionality and low cost, we have designed and implemented an inexpensive inertial measurement unit consisting of sensors and a Global Positioning System (GPS) unit and interfaced them with an Embedded Linux based board.

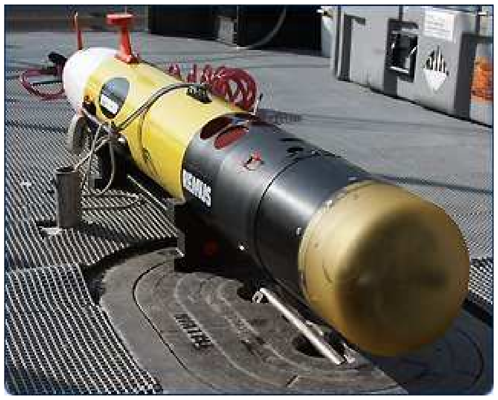


Fig. 1. A Hydroid Remus AUV [2]

1.1. Motivation

The motivation of this work is to design an underwater navigation system composed of a low cost Embedded Linux platform and small sized sensors. For air or ground vehicles, a GPS receiver with differential corrections (DGPS) can provide very precise and inexpensive measurements of geodetic coordinates. Unfortunately these GPS radio signals cannot penetrate beneath the ocean's surface, and this poses a considerable constraint as the vehicle 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 work 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].

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. 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. 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 Li and Chiang [9] 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.

The Inertial Measurement Unit (IMU) consists of an accelerometer and an angular rate sensor (gyro meter) to track the motion of the vehicle. The paper written by Wang, Ding and Zhao [10] 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 Work and its Contributions

The main contribution of this work, towards the research in AUV, is to use an Embedded Linux supporting a single board computer (SBC). The proposed work was 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 work was to write the code as generic as possible so that it could to be ported to other Linux-based SBC's like the Gumstick and Kontron.

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.

The paper is divided into five sections. Section 2 introduces the Linux operating system. It discusses the organization of the Linux kernel. Section 3 describes the hardware setup and the features of the individual components. Section 4 introduces the software component. It describes the software development environment, the Linux environment, and the test setup. Section 5 details results and conclusions and suggests future work.

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 [11].

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 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. 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.

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.

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.

2.2. Introduction to the ARM9 Architecture

The ARM processor is based on 32-bit 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 the 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 [12].

2.3. The CirrusLogic EP9302 and TS-7200

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 32-bit ARM and 16-bit Thumb instruction set. The core can operate in big and little endian mode. Endianess affects both the address and data interfaces [13].

The TS-7200 SBC runs on a 200MHz EP9302 ARM9 processor with power as low as 0.5Watts. As a general-purpose controller, it provides a standard set of peripherals on board [14].

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. The board also features USB and Ethernet ports which can be used to connect to other devices. TS-7200 SBC was selected for its features, good support by the manufacturer and a huge user community.

3. Hardware Description

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.

Figure 2 shows a photograph of the complete hardware module system. The RS232 and the external ADC are the interfaces that are used to communicate between Single Board Computer (SBC) and other sub modules.

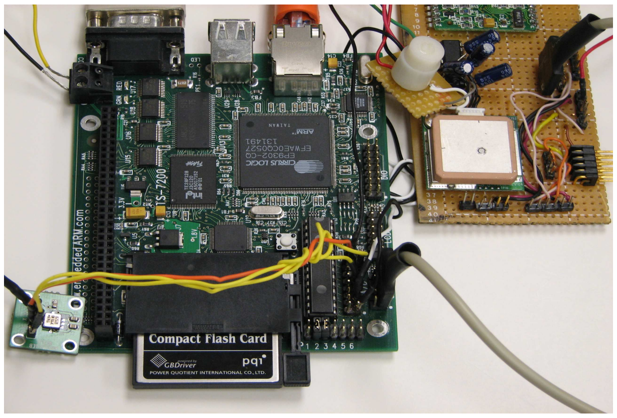


Fig. 2. The Complete System Setup

3.1. MAX197 External ADC Overview

The TS-7200 supports an optional 8 channel ADC, MAX197, with a conversion time of 6 µs. 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 an analog signal to a 12-bit digital output. Each channel is independently software programmable for a variety of analog input ranges.

3.2. 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 system 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. This GPS receiver supports NMEA 0183 data protocol.

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.

3.3. RS-232 Interfacing

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.

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. Exchanging data by way of the RS-232 port is similar to reading from or writing to a file.

3.4. 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 [15] 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 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 unbalances the differential capacitor, resulting in an output square wave whose amplitude is proportional to acceleration.

3.5. R1655 Analog Compass

The 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 designed to measure the direction of the horizontal component of the earth's flux field. The analog output from the compass closely resembles a sine-cosine set of curves.

3.6. 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. The LM35 is a three terminal device that produces output voltages proportional to ^{0}C (10mV/^{0}C), so the nominal output voltage is 250mV at 25^{0}C and 1.000V at 100^{0}C.

4. Software Development

4.2. Linux Environment

TS-Linux embedded distribution is installed on the on-board Flash memory of the 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. 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. The 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 [16].

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 any 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.

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

To automate the compilation process make tool was implemented.

4.3. Description of Main Programs

The 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 (described below)

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.

Linux is a virtual memory system. This means that the addresses seen by a 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 [17].

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: The kernel virtual addresses does not always directly map to physical addresses. That is why it is different from the kernel logical addresses.

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 & 0xfffff000;
    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.
MAP_SHARED: This option directs the kernel to share this mapping with all the other processes that map this object.
MAP_PRIVATE: This option stipulates to the kernel to create a private copy. A store to this region does not affect the original file.

fd should be a valid file descriptor.

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

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.

4.4. GPS Receiver

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.

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

Opening a 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 comm 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 [18].

Configuring the 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 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)
```

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.

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

Reading and Writing to the Serial Port: To write data to the port - write( ) system call is used.

The *write* function returns the number of bytes sent or -l 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.

Closing the Serial Port: To close the serial port, use the *close( )* system call: close(fd);

4.3. Compact Flash

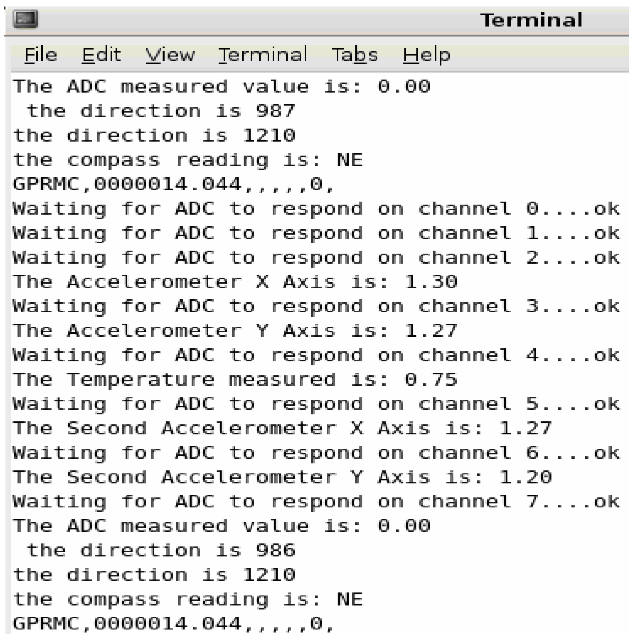
The data received from the ADC and the GPS module is continuously displayed on the telnet screen. This data is simultaneously logged in the CompactFlash, which is mounted on the board, to analyze the data received.

4.4. Operation of the System

The system was setup as described in the Section 3 and illustrated in Figure 2. The ADC mode selected is unipolar and the range selected is 10V. All the 8 channels are sampled and the 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 Comm 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 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.



```
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,
```

Fig. 3. Output from the ADC and the GPS Receiver.

5. Conclusion and Future Work

The objectives of this work have been achieved with the implementation of a 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.
- 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.

This 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.

6. 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&section=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.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.

[5] D. Geer, "Survey: Embedded Linux Ahead of the Pack" *Distributed Systems Online*, IEEE, Vol. 5, Issue 10, pp. 1-6, Oct. 2004

[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 3rd IEEE* Vol. 2, pp. 1073 – 1077, 8-10 Jan. 2006.

[9] 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.

[10] Wang, Q., Ding, M., and Zhao, P., "A New Scheme of Non-gyro Inertial Measurement Unit for Estimating Angular Velocity,"*29th Annual Conference of the Industrial Electronics Society, IECON,* Vol. 2, pp. 1564-1567, November, 2003.

[11] Wall, K., *Linux Programming by Example*, Indianapolis, Ind. Que, ISBN: 0789722151, 2000

[12] Segars, S., "The ARM9 Family – High Performance Microprocessors for Embedded Applications," *Proceedings of the International Conference on Computer Design: VLSI in Computers and Processors*, pp. 230 - 235, 5th October, 1998.

[13] Cirrus Logic EP9301 User's Guide http://www.embeddedarm.com/downloads/Components/EP9301_User_Guide.pdf

[14] Linux for ARM on TS-7200 User's Guide http://www.embeddedarm.com/Manuals/linuxarm-guide-rev2.0.pdf

[15] 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.

[16] 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.

[17] Rubini, A., and Corbet, J., *Linux Device Drivers*, 2nd Edition, O'REILLY Online, 2002

[18] Serial Communication How to http://www.tldp.org/HOWTO/Serial-HOWTO.html