Wireless Sensor Data Transfer using Bluetooth Technology

Prepared for:
The Department of Electrical Engineering for partial fulfillment of the requirements for the Degree of Master of Science in Electrical Engineering

Prepared by:
Patrick Dolz

University of Cape Town, July 2003

TERMS OF REFERENCE

This thesis project report describes the research and development done by the author under the supervision of Prof. J. Tapson. The area of research is the investigation of the Bluetooth technology in combination with developing and programming an application under Linux and using this technology to send sensor data within a wireless network. This thesis was commissioned by Prof. J. Tapson, on the 1st April 2002. The goals were as follows:

- Investigate the research done on this topic by previous researchers.
- Investigate current applications and standards of development concerning this topic.
- Develop and program an application under Linux for the Etrax100LX developer board which ensures the following:
  - Collect Sensor Data from the serial port.
  - Save the Sensor Data in a file on the board.
  - Send the Packets over the Bluetooth Radio to an Internet Access Point.
- Make use of the programming language C.
- Draw conclusions regarding the performance, effectiveness and limitations of the system and software.
- Make recommendations for future project developments.
- Submit the thesis by the 15th of July, 2003.

SYNOPSIS

The objective of this thesis is to design and program an application for the Linux based Axis Etrax100LX developer board using the C programming language. This application will collect sensor data (4-20mA signals as the standard analog inputs and 0-5V as the digital inputs) from the serial port of the Etrax100LX developer board, save them to a file on the developer board and make this data available for remote access. The idea behind using the Etrax100LX development board and writing software for it is to build a wireless, web-based monitoring system for electrical and mechanical rotating machinery which makes the data available for remote access.

Bluetooth is a wireless, data transmission technology [13] designed specifically for use in Personal Area Networks (PANs) where up to seven Bluetooth enabled computing devices such as Personal Computers and Personal Digital Assistants (PDAs) can form either a Group ad-hoc Network (GN or Bluetooth Piconet) or a network where one of them acts as a proxy, router or bridge (Network Access Point, NAP) between an existing network infrastructure (typically LAN) and the other Bluetooth devices [14].

The development resource used for the project was the Axis Etrax100LX developer board and the included software development kit (SDK). The SDK uses Linux 2.4 as the base platform and the latest tested kernel version 2.4.19 includes Etrax drivers for Ethernet, serial ports, parallel ports, USB, General Purpose I/O (GPIO) and IDE [15].

Throughout the research and investigation of this project, a few adjustments had to be made and the original system design had to be changed. This thesis will cover this process plus the development of the required software.

A main part of this thesis consisted of research. It was very difficult to find the right documentation and software to meet the requirements of this project. Bluetooth technology under Linux is a fairly new topic and the development an ongoing process. A lot of time was spent on downloading new available software and testing it out. It took a long time to find the right software to set up the desired bluetooth network using the available hardware to fullfill the requirements. The unsuccessful attempts are left out in this thesis due to the fact that they where numerous and did not help to reach the goal of this project.

Important conclusions drawn were that the software written can only serve as a starting point for further investigation or development concerning this project. Due to the fact that only a temperature sensors was available, sufficient multiple input data was missing and therefore the written software can only be seen as a basic tool to read input data from a serial port. The Bluetooth connection is stable and works well within a range of 10m. The PAN profile is a very powerful tool, due to its versatility; any type of network protocol is supported and can be send over a Bluetooth connection using the PAN profile.

As a result of this project the following recommendations were made: To be able to design a monitoring system as required within this project, the availability of multiple sensors has to be guaranteed. This is essential for future development. Since the developer board only has one serial port and not sufficient memory to run big applications on it, it might be an idea to create a board on its own, since the Axis company offers to sell the Etrax100LX chip on its own. A suitable board with a sufficient amount of memory and interfaces could make this project even more versatile for maybe monitoring devices where a lot more sensors are required. Further more, the data captured from the serial port could be packed into XML packages and made available to a lot more applications and programs. A database could be implemented to handle the amount of sensor data. The required 16 sensors for this monitoring system can produce a lot of data throughout the day, and bigger machinery might even require a lot more sensors. Further work, maybe the scope of a thesis, has to be done to convert this project into a monitoring system suitable for industry.

Contents

List of Figures

Chapter 1

Introduction

Web-based monitoring systems for electrical and mechanical rotating machinery have gained more and more popularity over the last couple of years. This thesis is premised on an attempt to take this technology a step further and create a monitoring system which is wireless and self configuring towards an existing network structure. Wireless communication networks such as GSM (Groupe Special de Mobile communication, the mobile phone standard) are state of the art, but wireless computer networks are still uncommon. Although, Wireless LAN (Wireless Locale Area Network) is gaining more and more popularity and UMTS (Universal Mobile Telecommunication System) is lifting the wireless communication networks to the next level, wireless networks are hardly ever found in industrial environments due to their sensitivity towards electromagnetic fields. The Bluetooth technology makes use of frequency hopping [27]and therefore is said to be extremely robust against electromagnetic influences and performs very well within a range of over 20m [13]. The Axis developer Board for Bluetooth Etrax100LX was designed for embedded Bluetooth development and offers a platform with Bluetooth and Ethernet connections. The platform is using Standard Linux 2.4 with the latest kernel patch version 2.4.19 and offers an ideal base for the scope of this thesis.

1.1 Background Information

In the Personal Area Network (PAN) arena, the Bluetooth technology has emerged as the technology of choice for the transport of data and voice between short range wireless computing devices. Its ad-hoc connectivity, low power consumption and robustness against electromagnetic influences due to frequency hopping [27] make it the ideal technology for use in a short range wireless network within an industrial environment. In the present application, the Bluetooth technology using PAN is self-configuring and therefore, installation of such a monitoring device will be a lot easier since there is no network administrator required. On top of that, up to seven similar monitoring devices can function in one PAN structure since the Bluetooth specifications [8] for PAN allow seven active PANU (Personal Area Network User) connections to a NAP (Network Access Point) [14]. Hence it would be very easy to add more monitoring devices to an already existing monitoring network.

1.2 Purpose of the Thesis

This thesis is part of designing and implementing of a wireless, web-based monitoring system for electrical and mechanical rotating machinery which makes the data available for remote access. Its main focus lies on developing and implementing software for the Axis Etrax100LX developer board which is capable of reading in sensor data from the serial port of the board, saving this data into a file on the board, and making it accessible via a Bluetooth connection for remote access. As a starting point this thesis required a lot of research based on the available technologies and methods in order to find suitable components and software for this project. Throughout the research and development of this project a number of changes were required due to the availability of new updates for the software used on the developer board and to the availability of new Bluetooth software.

1.3 Objectives

The objectives of this research in connection with the development of software for the Axis Etrax100LX developer board were to:

- Investigate the research done on this topic by previous researchers.
- Investigate current applications and standard of development concerning this topic.
- Develop and program an application under Linux for the Etrax100LX developer board which ensures the following:
  - Collect Sensor Data from the serial port.
  - Save the Sensor Data in a file on the board.
  - Send the Packets over the Bluetooth Radio to an Internet Access Point.
- Make use of the programming language C.
- Draw conclusions regarding the performance, effectiveness and limitations of the system and software.
- Make recommendations for future project developments.

This project will only consider the development of a test-based piece of equipment. Further work would therefore have to be done to convert this project into a monitoring system suitable for industry.

1.4 Thesis Layout

The content of this thesis is divided into eight different chapters and four different appendices. A summary of each one of them is given below:

CHAPTER 1 introduces the reader to the topic of this thesis. It gives a short introduction, some background information, the purpose and the objectives of the thesis, as well as the layout of the thesis.

CHAPTER 2 introduces the reader to the Bluetooth wireless technology and briefly describes each layer within the Bluetooth stack. The concept of Bluetooth profiles is covered as well as a closer look at the three most common open source Bluetooth stacks (BlueDrekar, openBT and BlueZ). Finally, the PAN profile is briefly described and a general overview on other available Bluetooth profiles given.

CHAPTER 3 gives the reader a brief overview of the Linux operating system. It talks about the Kernel structure, the Shell, the file system and the system/user programs. It also talks about the most common Linux Distributions, in particular the SuSE 8.1 Distribution which was used for this project.

CHAPTER 4 describes the features of the Bluetooth hardware used in this project: The Axis Etrax100LX developer board on both the hardware and software side as well as the Network Access Point side, which used a Bluetooth USB adapter.

CHAPTER 5: This chapter explains the system design in more detail. The first section talks about the original layout as required by the supervisor Prof. J. Tapson. The second section explains the current layout (including the used hardware and its implemented software features) after evaluation of the available hard- and software features. The last section talks about sensor requirements for monitoring electrical and mechanical rotating machinery, although the development of this thesis did not emphasise that since only single sensor data was available during the process of development.

CHAPTER 6 explains in detail how the software for the used hardware was installed and configured. Throughout the research and developing process of this thesis, the Axis Etrax100LX for Bluetooth was updated with new software, which made it possible to have some of the software requirements already implemented. This updating process is also explained in detail.

CHAPTER 7 covers the configuration of the Bluetooth network between the two used Bluetooth components as well as the development and implementation of the written software.

CHAPTER 8 covers the conclusions reached within the scope of this thesis as well as the state of development concerning the final project. The text ends by making recommendations for future work which can utilize the current develop-

ment environment with regard to Bluetooth wireless technology.

Chapter 2

BLUETOOTH TECHNOLOGY

Bluetooth wireless technology is a specification designed to enable wireless communication between small, mobile devices. The inspiration behind this technology was the concept to eliminate the need for proprietary cables, which are currently required to enable device connectivity. For instance, in order to transfer images from a digital camera to a laptop PC, a cable is needed in order to connect the camera to the laptop. Each camera manufacturer and model has a different cable requirement. In fact every hand held device manufactured which allows connectivity with a PC has a different cable configuration. A perfect scenario would be where both the laptop PC and the digital camera use Bluetooth wireless technology. In this case there would be no need for cables to transfer data between devices. Expanding that idea to include all hand held mobile electronic devices is, in a nutshell, the *vision* of the Bluetooth wireless technology [4]. The Bluetooth standard also allows for ad-hoc network connectivity, a characteristic of the very mobile Personal Area Network (PAN). The link supports both voice and data traffic and is perfectly suitable for this project. The standard is aimed at achieving true inter-operability between network devices.

2.1 History

Harald I Bluetooth (Danish Harald Bltand) was the King of Denmark between 940 and 985 AD. The name "Bltand" was probably taken from two old Danish words, 'bl' meaning dark skinned and 'tan' meaning great man. He was born in

910 as the son of King Grom the Old (King of Jutland, the main peninsula of Denmark) and his wife Thyre Danebold (daughter of King Ethelred of England). By 960 he was at the height of his powers, ruling over both Denmark and Norway. He then created a monument that read: "King Harald raised this monument to the memory of Grom his father and Thyre his mother. Harald conquered all of Denmark and Norway and made the Danes Christian". These words were also carved in stone called rune stones. Harald was killed in a battle in 985. Harald completed the country's unification begun by his father, converted the Danes to Christianity, and conquered Norway. Old Harald Bluetooth united Denmark and Norway, Bluetooth of today is trying to unite the worlds of computers and telecom (hopefully with longer and greater success than the few years Harald's Viking kingdom survived). In 1994 Ericsson Mobile Communications initiated a study to investigate the feasibility of a low-power low-cost radio interface between mobile phones and their accessories. In Feb 1998, five companies Ericsson, Nokia, IBM, Toshiba and Intel formed a Special Interest Group (SIG). The group contained the necessary business sector members - two market leaders in mobile telephony, two market leaders in laptop computing and a market leader in digital signal processing technology. It is estimated that Bluetooth is a built-in feature for more than 100 million mobile phones and several million communication devices ranging from handsets and portable PCs to desktop computers and notebooks [6].

Some historical facts [2]:

- Early 1998 - Special Interest Group formed
  - Code name "Bluetooth"
  - Promoter Companies: Ericsson, IBM, Intel, Nokia, and Toshiba
- May 20, 1998 - Bluetooth publicly announced
- July 26, 1999 - Bluetooth 1.0 Specification Release
- Today - Bluetooth 2.0 work is ongoing
  - Promoter Companies: 3Com, Ericsson, IBM, Intel, Lucent Technologies, Microsoft, Motorola, Nokia and Toshiba
  - Currently 1883 SIG Members

- Harald Bltand - 10th Century Viking King
  - Literally translated to "Bluetooth"
  - Bltand or "Bluetooth", had nothing to do with blue teeth
  - Referred to Harold's dark complexion and his very dark hair

2.2 Overview

In addition to eliminating the need for cables and dongles to connect devices, Bluetooth enables devices to form small, ad hoc wireless networks called piconets or Personal Area Networks (PANs). These wireless connections are established using a radio transceiver embedded within each Bluetooth device. The radio operates in the 2.4 GHz Industrial, Scientific, and Medical (ISM) band which is globally available. The Bluetooth Radio is designed to operate in a noisy radio environment and to provide a fast, robust, and secure connection between devices. A full duplex data exchange rate of up to 1 Mb/s may be achieved in which a Time-Division Duplex (TDD) scheme is used. Stability is ensured by implementing a frequency hopping scheme which enables Bluetooth Radio modules to avoid interference from other signals after transmitting or receiving a data packet. Security within Bluetooth connections is implemented at the hardware layer, with the option of using one of three security levels. While Bluetooth wireless technology has many features unique to its own specification, it has borrowed heavily from several existing wireless standards, including Motorola's Piano, IrDA, IEEE 802.11, and Digital Enhanced Cordless Telecommunications (DECT). Motorola's Piano was developed with the concept of forming ad hoc Personal Area Networks (PANs). This concept was adopted by the Bluetooth SIG to expand the capabilities of the original Bluetooth concept beyond simple cable replacement. The voice data transmission capabilities of Bluetooth are derived from the DECT specification. The object exchange capabilities (the ability to share business card, contact information, messages, etc.) are derived from the IrDA specifications. Bluetooth also inherits the use of the 2.4GHz ISM band, Frequency Hopping Spread Spectrum (FHSS), authentication, privacy, power management, and LAN capabilities provided by the IEEE 802.11 specification [4]. An overview of the Bluetooth stack (or protocol architecture) is shown in

figure 2.1, next page. The abbreviations used in the figure are explained in the following section.

Bluetooth Protocol Architecture Overview: The Bluetooth protocol stack is shown in Figure 2.1. These protocols can be divided into four categories [22]:

- core protocols
- cable replacement protocol
- telephony control protocol
- adopted protocols

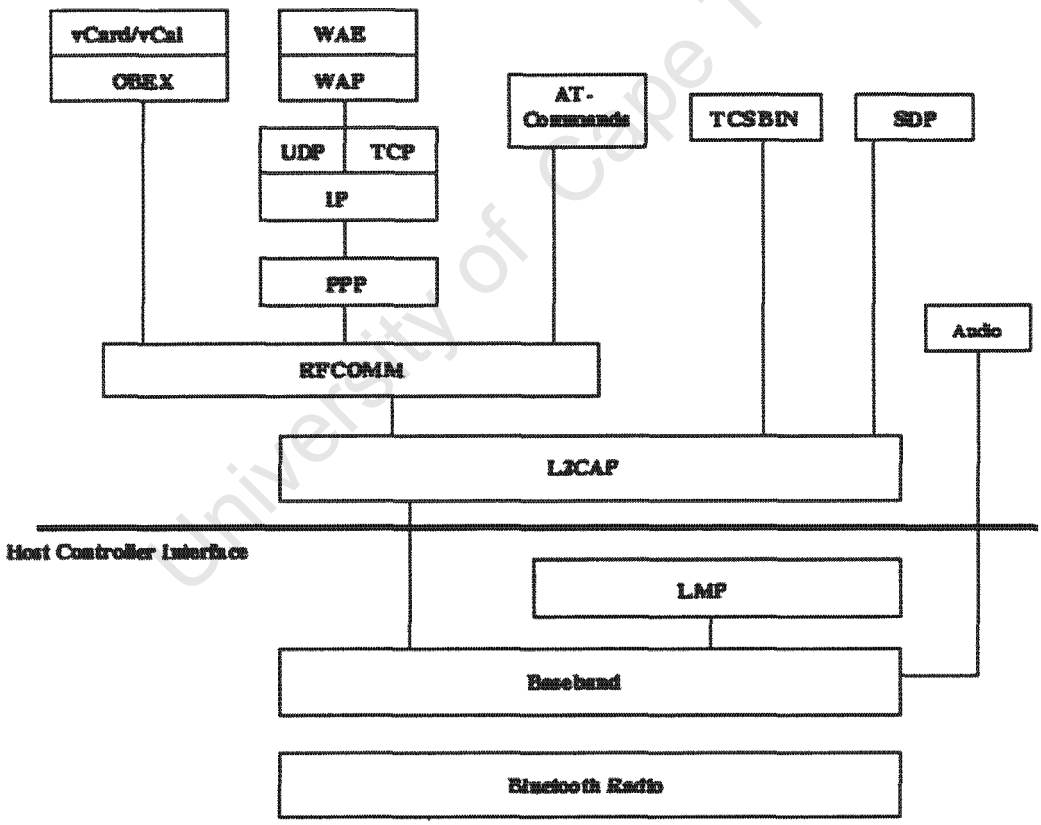

Figure 2.1: **The Bluetooth Protocol Stack**

2.2.1 Bluetooth Radio

The Bluetooth Radio (layer) is the lowest defined layer of the Bluetooth specification. It defines the requirements of the Bluetooth transceiver device operating in the 2.4GHz Industrial Scientific Medical (ISM) band. The Bluetooth radio accomplishes spectrum spreading by Frequency Hopping Spread Spectrum (FHSS) usage in 79 hops displaced by 1 MHz, starting at 2.402GHz and finishing at 2.480GHz. In a few countries (e.g. France) this frequency band range is (temporarily) reduced, and a 23-hop system is used. In order to comply with out of band regulations in each country, in both systems a guard band is used at the lower and upper band edge [3].

2.2.2 Baseband *core protocol*

The Baseband and Link Control layer enables the physical RF link between Bluetooth units forming a piconet. This layer controls the Bluetooth unit's synchronization and transmission frequency hopping sequence. The two different link types defined in Bluetooth, Synchronous Connection Oriented (SCO) and Asynchronous Connectionless (ACL), are also managed by this layer. The ACL links, for data, and the SCO links, mainly for audio, can be multiplexed to use the same RF link [24].

2.2.3 Link Manager Protocol (LMP) *core protocol*

LMP is responsible for link set-up between Bluetooth units. It handles the control and negotiation of packet sizes used when transmitting data. The Link Manager Protocol also handles management of power modes, power consumption, and the status of a Bluetooth unit in a piconet. Finally, this layer handles generation, exchange and control of link and encryption keys for authentication and encryption [24].

2.2.4 Host Controller Interface(HCI) *core protocol*

HCI provides a uniform interface method for accessing the Bluetooth hardware capabilities. It contains a command interface to the Baseband controller and link manager and access to hardware status. It also contains control and event registers [24].

2.2.5 Audio *core protocol*

Audio transmissions can be performed between one or more Bluetooth units, using many different usage models. Audio data do not go through the L2CAP layer but go directly, after opening a Bluetooth link and a straightforward set-up, between two Bluetooth units [24].

2.2.6 Logical Link Control and Adaptation Protocol (L2CAP) *core protocol*

The Bluetooth logical link control and adaptation protocol, L2CAP, is situated over the Baseband layer and beside the Link Manager Protocol in the Bluetooth protocol stack. The L2CAP layer provides connection-oriented and connectionless data services to upper layers. The four main tasks for L2CAP are: [24]

- *Multiplexing:* L2CAP must support protocol multiplexing, since a number of protocols (e.g. SDP, RFCOMM and TCS Binary) can operate over L2CAP.
- *Segmentation and Reassembly:* Data packets exceeding the Maximum Transmission Unit, MTU, must be segmented before being transmitted. This and the reverse functionality, reassemble, is performed by L2CAP.
- *Quality of Service:* The establishment of an L2CAP connection allows the exchange of information regarding current Quality of Service for the connection between the two Bluetooth units.
- *Group abstraction:* The L2CAP specification supports a group abstraction that permits implementations for mapping groups on to a piconet.

2.2.7 RFCOMM *cable replacement protocol*

This is the cable replacement protocol, whose purpose is to emulate a serial port. The protocol covers applications that use serial ports of the kind used in PCs. Thus, RFCOMM emulates RS-232 control and data signals over the Bluetooth baseband. It provides transport capabilities for upper level services, such as OBEX (Object Exchange protocol) [23].

2.2.8 Service Discovery Protocol (SDP) *core protocol*

SDP defines how a Bluetooth client's application shall act to discover available Bluetooth servers' services and their Bluetooth characteristics. The protocol defines how a client can search for a service based on specific attributes without the client knowing anything of the available services. The SDP provides means for the discovery of new services becoming available when the client enters an area where a Bluetooth server is operating. The SDP also provides functionality for detecting when a service is no longer available [24].

2.2.9 Telephony Control - Binary *telephony protocol*

The Telephony Control Specification - Binary, TCS Binary or TCS BIN, is a bit-oriented protocol, which defines the call control signaling for the establishment of speech and data calls between Bluetooth units. The protocol defines the signaling for establishment and release of calls between Bluetooth units as well as signaling to ease the handling of groups of Bluetooth units [25].

2.2.10 Telephony Control - AT-Commands *telephony protocol*

A number of AT-commands are supported for transmitting control signals for telephony control. They use the serial port emulation, RFCOMM, for transmission [25].

2.2.11 PPP *adopted protocol*

The IETF Point-to-Point Protocol (PPP) in the Bluetooth technology is designed to run over RFCOMM to accomplish point-to-point connections. PPP is a packet-oriented protocol and must therefore use its serial mechanisms to convert the packet data stream into a serial data stream [21].

2.2.12 TCP/UDP/IP *adopted protocols*

The TCP/UDP/IP standards are defined to operate in Bluetooth units allowing them to communicate with other units connected, for instance, to the Internet. Hence, the Bluetooth unit can act as a bridge to the Internet. The TCP/IP/PPP protocol configuration is used for all Internet Bridge usage scenarios in Bluetooth 1.0 and for OBEX in future versions [21].

2.2.13 OBEX Protocol *adopted protocol*

IrOBEX, as is the correct term, is an optional application layer protocol designed to enable units supporting infrared communication to exchange a wide variety of data and commands in a resource-sensitive standardized fashion. OBEX uses a client-server model and is independent of the transport mechanism and transport API. The OBEX protocol also defines a folder-listing object, which is used to browse the contents of folders on remote device. RFCOMM is used as the main transport layer for OBEX [21].

2.2.14 Wireless Application Protocol (WAP) *adopted protocol*

WAP is a wireless protocol specification that works across a variety of wide-area wireless network technologies bringing the Internet to mobile devices. Bluetooth can be used like other wireless networks with regard to WAP, it can be used to provide a bearer for transporting data between the WAP Client and its adjacent WAP Server. Furthermore, Bluetooth's ad hoc networking capability gives a

WAP client unique possibilities regarding mobility compared with other WAP bearers [21].

2.3 Bluetooth Stacks

When using an Internet connection on a computer, one needs a network adapter, appropriate driver software and the protocol software (called the TCP/IP stack) in order to communicate with other devices. With Bluetooth the case is very similar. One needs a Bluetooth adapter, driver software and a Bluetooth Protocol stack. As many Bluetooth devices present themselves as a serial device, there is usually no special driver needed. For Linux there are several Bluetooth stacks to choose from [18] but only three open source stacks.

2.3.1 BlueDrekar

BlueDrekar is a middleware project from IBM that includes a complete Bluetooth stack. Only the transport driver is open source. The BlueDrekar project team in IBM Research is a part of the Pervasive Security and Networking Department led by Mahmoud Naghshineh. The team has been an active participant of the Bluetooth Special Interest Group (SIG) since its inception in early 1998 and it has led IBM's involvement in the technical activities of the SIG. It has contributed heavily to the development of the Bluetooth 1.0 specification. Team members have served as editors for the HCI RS-232 transport and Service Discovery Application Profile (SDAP) parts of the specification. The team is a strong advocate of the Bluetooth wireless technology both inside and outside IBM, where they have been invited to speak about the technology on several occasions [10]. The bluedrekar stack is not used in this project.

Project Homepage: http://www.alphaWorks.ibm.com/tech/bluedrekar

Latest official version: 1.3.0

Download: http://www.alphaWorks.ibm.com/tech/bluedrekar

Supported Chipsets: Ericsson

2.3.2 The Axis openBT Stack

OpenBT was the first Linux Bluetooth stack available. It was originally developed by Axis Communications Inc. and is now an open source project hosted at Sourceforge. It is part of the software running on the Axis Etrax100LX developer board for Bluetooth [18].

Project Homepage: http://developer.axis.com/software/bluetooth/

Latest official version: 0.0.8

Download: http://sourceforge.net/projects/openbt/

Supported Chipsets: CSR, Ericsson, ...

2.3.3 The BlueZ Stack

BlueZ was originally developed by Qualcom and later made OpenSource. Since version 2.4.6 it is included in the Linux kernel districution [18]. It is the official Linux Bluetooth protocol stack.

Project Homepage: http://bluez.sourceforge.net/

Latest official version: 1.1

Download: http://bluez.sourceforge.net/

Supported Chipsets: CSR, Ericsson, Broadcom, Silicon Wave, ...

2.4 The Bluetooth Personal Area Network (PAN)

The developers of Bluetooth have refer to this type of network as a PAN (Personal Area Network), although this could be misleading and is not quite correct:

Personal area networks (PAN) actually refer to using a near field electric field to send data across various devices using the body as a medium. Personal area networks should not be confused with networks formed using Bluetooth technology. Bluetooth is a far field radio technology enabling communication between devices in the ISM band. Unfortunately, you now see Bluetooth and PAN being used

interchangeably in many technical articles, which is not correct. Bluetooth can in fact be used as an enabler to extend the reach of body networks to other devices and networks. (as taken from [26])

To focus on Bluetooth: the Bluetooth technology supports point-to-point and point-to-multipoint connections (it can communicate with multiple devices). A connection of two or more Bluetooth devices is called a piconet. A piconet can consist of up to eight devices. A scatternet is a collection of piconets. Each device in a scatternet can only see up to eight devices. The piconets in a scatternet are linked via a different frequency-hopping sequence and the devices within each piconet synchronize with the unique sequence of the piconet they want to communicate with [16]. The piconet is controlled by one of the devices in the network called the master. This device coordinates the connection and synchronizes with other piconet devices by going through various clocking and hopping sequences. The other devices in the piconet are referred to as slaves. Slaves have a peer relationship. Each device has a 3 bit MAC (Media Access Control) address to identify it in the piconet. Some units are parked, which means they are synchronized in the piconet but have given up their MAC addresses, while other units may go into the "sniff and hold" mode, which means they have gone into power-saving mode but are ready to wake up if necessary [28]. The device waits for a connection in standby mode. Every 1.28 seconds it checks for messages from other bluetooth devices. According to [28] these devices listen on up to 32 hops frequencies and a device that wants to make a connection sends a message to the device indicating that it wants to establish a connection. The two devices then form a master slave relationship.

Participants in a BlueZ (which was used for this project) PAN can take on the following roles (see [7]):

PAN user (PANU): Client of a NAP or client-type member of a GN (see below).

Group ad-hoc Network (GN) controller: Forwarding node in a peer-to-peer style network (Bluetooth Piconet). Interconnects up to 7 (active) PANUs to a real peer-to-peer network.

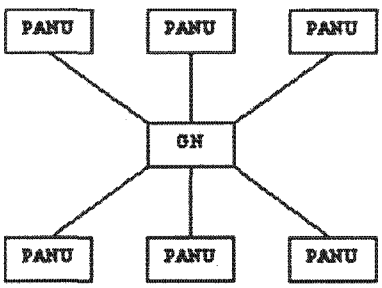

Figure 2.2: **Group ad-hoc Network Controller**

Network Access Point (NAP): Acts as proxy, router or bridge between an existing network infrastructure (typically LAN) and (up to 7 active) wireless clients (PANUs).

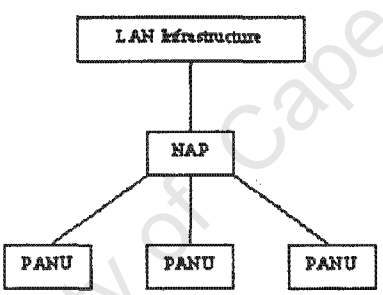

Figure 2.3: **Network Access Point**

There are two types of links (as listed in [28]):

SCO (Synchronous Connection Oriented): A symmetric link that supports real-time voice traffic. The packets in this link are transmitted at reserved intervals to ensure high voice quality. However, lost or corrupted packets are not transmitted.

ACL (Asynchronous Connectionless): Supports asymmetric packet-oriented traffic in which the master unit controls the bandwidth and how much of that is given to each slave. Slave devices are polled before they can transmit data. A broadcast message mode is also provided.

2.4.1 Other Bluetooth Profiles

Here are some other Bluetooth profiles and a short description of each one of them (as taken from [19]):

GAP Profile:

The Generic Access Profile defines the generic procedures related to discovery of Bluetooth devices and link management aspects of connecting to Bluetooth devices. It is the core on which all other Profiles are based.

DNP or DUN Profile:

The Dial-up Networking Profile defines the requirements that shall be used by devices (modems, cellular phones) implementing the usage model called "Internet Bridge". The Gateway role is typically used in a mobile phone with Bluetooth dial-up networking capability and the Terminal role in PDAs or laptops. Thus DUN has two roles: terminal and gateway.

SPP Profile:

The Serial Port Profile defines the requirements for Bluetooth devices necessary for setting up emulated serial cable connections using RFCOMM between two peer devices.

LAP Profile:

The LAN Access Profile defines how Bluetooth enabled devices can access the services of a LAN using PPP. Also, this profile shows how the same PPP mechanisms are used to form a network consisting of two Bluetooth-enabled devices.

SDAP Profile:

The Service Discovery Application Profile defines the features and procedures for an application in a Bluetooth device to discover services registered in other Bluetooth devices and retrieve any desired available information pertinent to these services.

CTP Profile:

The Cordless Telephony Profile defines the features and procedures that are required for interoperability between different units active in the 3-in-1 phone

use case. This profile also shows how the use case can be applied generally for wireless telephony in a residential or small office environment.

IP Profile:

The Intercom Profile defines the requirements for Bluetooth devices necessary for the support of the intercom functionality within the 3-in-1 phone use case. This is also refereed to as the "walkie-talkie" usage of Bluetooth.

HS Profile:

The Headset Profile defines the requirements that shall be used by devices implementing the usage model called "Ultimate Headset".

FP Profile:

The Fax Profile defines the requirements for Bluetooth devices necessary to support the Fax use case. This allows a Bluetooth cellular phone (or modem) to be used by a computer as a wireless fax modem to send/receive a fax message.

GOEP Profile (obex):

The Generic Object Exchange Profile lays the basis (defines the protocols and procedures) for Bluetooth devices necessary for the support of the object exchange usage models. The usage model can be the "Synchronization", "File Transfer", or "Object Push" model.

OPP Profile:

The Object Push Profile defines the requirements for applications providing the object push usage model. Typical scenarios covered by this profile involve the pushing/pulling of data objects between Bluetooth devices.

FTP Profile:

The File Transfer Profile defines the requirements for applications providing the file transfer usage model. Typical scenarios involve a Bluetooth device browsing, transferring and manipulating objects on/with another Bluetooth device.

SP Profile:

The Synchronization Profile defines the requirements for applications providing the synchronization usage model. Typical scenarios covered by this profile in-

volving manual or automatic synchronization of PIM data when two Bluetooth devices come within range.

Chapter 3

LINUX OPERATING SYSTEM

Linux is a 32 bit multi-tasking, multi-user operating system that feels exactly like UNIX. It was written by Linus Torvalds, a Finnish student. Linux runs on many platforms including intel, alpha, sparc, etc and is distributed under the conditions of the GNU Public License. Linux uses a virtual file system, that makes it possible to read various filesystems (such as fat16, fat32, vfat)[1]. Most software for Linux are free.

The Linux operating system (OS) has become a viable alternative for anyone with a PC. It brings all the power and flexibility of a Unix workstation, as well as a complete set of Internet applications and fully functional Desktop Interfaces.

Linux is a fully functional Unix operating system that has all the standard features of powerful Unix systems. There are shells for managing commands. Linux uses two of the more advanced Unix shells. The Bourne Again Shell (Bash) and the TCSH shell(an enhanced, but completely compatible version of the Berkeley UNIX C shell [csh]). Each supports a complete shell programming language that can be used to create specific shell scripts.

Linux has the same level of system administration features that can be found on standard Unix systems. It has the same multi-user and multi-tasking capabilities. Accounts can be set up for different users and each one can access the system at the same time. Each user can have several programs running concurrently. With Linux access can be controlled, new connections can be set up and new devices

[1] 16 bit, 32 bit or virtual file allocation table

can be installed.

Linux also has very powerful development tools for creating specific Linux applications. These include the GNU C compiler with large numbers of programming tools such as debuggers and revision managers. With the shell programming capabilities of different shells, specific Linux commands can be created.

Linux can be generally divided into four major components: *the Kernel, the Shell, File structure, and the System and User Programs.* The kernel is the core program that runs programs and manages hardware devices such as disks and printers. The shell provides an interface for the user. It receives commands from the user and sends those commands to the kernel for execution. The file structure organizes the way files are stored on a storage device such as a hard disk. Files are organized into directories. Each directory may contain any number of subdirectories each holding files. The System and User Programs are generally programs running on the operating system. These four components are described in more detail in the following section. Linux comes in numerous free distributions, such as Redhat, Debian, Mandrake, and SuSE, just to name a few.

3.1 The Kernel

At its most simple, the kernel acts as a mediator for programs and hardware. It would be a waste of time and hard drive space to write all the code to access a hard drive into every single program that needs to be able to do this. So the kernel is the first component in the system, controlling "low-level" access to all hardware, and other functions critical to an operating system. They include [1]:

- *Booting* - The kernel is the first piece of code executed when the system is booting up. The booting process before that will include the BIOS initializing the hardware, then looking for the bootsector (Master Boot Record, MBR) on the hard drive for instructions of what to boot (which might simply be *"boot this OS"* or get *LILO (LInux LOader)* or *GRUB (GRand Unified Bootloader)* to ask what to boot), and the bootsector instructions will then be to uncompress and initialize the kernel. The kernel then sets up all the low- and high- level routines required by the system. All this can be seen as the system boots (all that text flying by). Once the kernel

has finished booting, it hands things over to *init*, which starts up programs and enables users to log in.

- *Process management* - Once booted, the system will be running lots of different programs, each of which is called a process. To see what processes are running on the machine at a time, the command "ps -ax" will list them . As in any other Operating System a lot of background processes are running concurrently. The Linux kernel controls all of these, ensuring that each of them gets a fair share of the memory (RAM and swap) and of processor cycles. The processor's time is divided into cycles, and processes take it in turn to use some of the processor's cycles. The CPU speed is defined in terms of how many cycles per second it can perform.

- *Inter Process Communication (IPC)* - The kernel also controls communication between processes - a central feature of any POSIX (Portable Operating System Interface for UNIX) operating system like GNU/Linux. To put it simply, different processes need a way of communicating with each other, and the kernel provides the medium through which they go, using shared memory (where each program names the portion of memory given to it, and gives it the usual read/write/execute permissions, allowing other programs to share the actual memory, thereby sharing data, and communicating), pipes and FIFOs (First In First Out).

- *Hardware interaction* - An operating system with no access to hardware would be completely useless, so the Linux kernel provides access to hardware devices for other programs. To do this, it has a set of general instructions for types of device (for example it has "read" and "write" instructions for storage devices), and then device-specific drivers which tell the kernel how to interpret those general instructions for a specific device (instructions on exactly how to read and write to, for example, a zip drive). Programs can then send generalized instructions to the kernel, like "read this data from the zip drive", and the kernel will use the device driver to read the data, and then send it back to the program.

- *Virtual Filesystem (VFS)* - In order to support multiple filesystems (e.g. Ext2, Vfat and ReiserFS), Linux contains a special kernel interface called VFS (Virtual Filesystem Switch). This provides a unified way of treating filesystems, even if the filesystems themselves are very different. So

a program sends the instruction "write to section x of the device", and the VFS takes this command and translates it into instructions that make sense for the particular filesystem. So the VFS is like a device driver, but for filesystems. It is a core component of the kernel.

That is a fairly short and simplified list of the main functions of a kernel. The Kernel development is an ongoing process due to constantly new available computer hard- and software. The newest versions are available under *http://www.kernel.org.*

For this project, kernel version 2.4.19 was used and later updated to version 2.4.20. The following subsection explains the use of modules, which are a very important part of the kernel structure and its functionality.

3.1.1 Kernel Modules

Another important feature of the kernel structure in Linux operating systems is the use of modules. When compiling a kernel with all the necessary support features for the used hard- and software on a PC (e.g. supported file systems, sound cards, graphic cards, etc.), the user has a choice of building the software support features either within the kernel or as a module.

A kernel is described as being monolithic, which means that it is a large structure of code that is indivisible (so it cannot work being divided into separate components) and uniform (each function is performed in the same way every time by the one big kernel). That does not mean that it can only exist as one long piece of code that functions as one huge executable program. In fact a kernel is very modular, and so all of its functions are divided into smaller chunks of separate, manageable code. Technically speaking, a monolithic kernel keeps all of its functions in the same memory address space, and the functions communicate within this shared memory. To ensure stability, the kernel puts each driver into a "virtual bubble", ensuring that if one driver or piece of code malfunctions, this does not end up causing the whole kernel to crash.

The Linux kernel being modular means, that all of its functions are allocated to modular chunks of code, that can be integrated into the compiled kernel, or compiled separately as modules which the kernel can then load and use at

any time. So, for example, the functions to use a sound card are all saved in one module, and they can either be compiled into the kernel, or compiled as a module, which can be loaded whenever the sound card needs to be used. This is one of the key benefits of a modular kernel - a kernel does not need to have every single function it might make use of saved in the computer's memory at all time, but can instead load functions on demand, whenever they are needed.

Distributors use modules extensively in the pre-compiled kernel versions they distribute with their software. They often compile as much as they can into the kernel itself and then add almost all other modules that users might want to use as separate, loadable modules.

Having a modular system also makes it a lot easier to maintain the kernel and the overall system. If a user wants to write a driver for a device not yet supported by the Linux kernel, it can be written and tested without affecting the stability and functionality of the working kernel. As long as it can be loaded by the kernel, and fulfills the required specifications of a module, it will function as a valid kernel module and can be included in kernel source code at any time.

Most distributions will therefore provide a compiled kernel version (zImage, or bzImage if compressed) and various compiled modules. The two combined, kernel and modules, make the so called "kernel" [1].

3.2 The Shell

The shell, also known as the command prompt, the terminal, or the console, is one of the most useful and powerful tools in GNU/Linux. It is useful because you can do tasks extremely quickly in it, far more quickly than through graphical tools like a filemanager. It is a program which interprets commands, either typed in directly by the user, or contained in a file called a "shell script", which is a simple interpreted program. The equivalents under Windows would be "command processor" for shell, "COMMAND.COM" or "CMD.EXE" instead of bash, and ".BAT files" instead of shell scripts.

A shell is accessible in several ways. Under KDE (K Desktop Environment) or GNOME (GNU Network Object Model Environment, another desktop environ-

ment under Linux), a terminal/console can be brought up and the user will be logged in and ready to use it. If that does not work, another way of opening a shell terminal is to press *Alt+F2* to bring up the Run Command prompt, type *konsole* and *Run*. The third option is to use the *Ctrl+Alt+F1* (up to F6) combination, where there are six different shells always available for the user.

Linux has a variety of different shells, but certainly the most popular ones are bash (Bourne-Again Shell), csh (C Shell), ksh (Korn Shell), and tcsh (enhanced but completely compatible version of csh).

3.3 File Structure

One of the most important features of Linux is its support for many different file systems. This makes it very flexible and able to coexist with many other operating systems. Just to name a few supported file systems: ext, ext2, ext3, minix, reiserfs, ntfs, msdos, vfat, proc, smb, ncp, iso9660, sysv, hpfs, affs and ufs.

In Linux, the separate file systems the system may use are not accessed by device identifiers (such as a drive number or a drive name) but instead they are combined into a single hierarchical tree structure that represents the file system as one whole single entity. Linux adds each new file system into this single file system tree as it is mounted. All file systems, of whatever type, are mounted onto a directory and the files of the mounted file system cover up the existing contents of that directory. This directory is known as the mount directory or mount point. When the file system is unmounted, the mount directory's own files are once again revealed.

In the Linux file structure files are grouped according to purpose. For example: commands, data files, or documentation. Parts of a Unix directory tree are listed below. All directories are grouped under the root entry "/". That part of the directory tree is left out of the hierarchy of a Linux OS shown below [20].

- *root* - The home directory for the root user.
- *home* - Contains the user's home directories along with directories for services:

  - *ftp*
  - *HTTP*
  - *samba*
  - *george*

- *bin* - Commands needed during bootup that might be needed by normal users.
- *sbin* - Like bin but commands are not intended for normal users. Commands run by LINUX.
- *proc* - This filesystem is not on a disk. It is a virtual filesystem that exists in the kernels imagination which is memory.
  - *1* - A directory with info about process number 1. Each process has a directory below proc.
- *usr* - Contains all commands, libraries, man (short for manual) pages, games and static files for normal operation.
  - *bin* - Almost all user commands. Some commands are in /bin or /usr/local/bin.
  - *sbin* - System admin commands not needed on the root filesystem. e.g., most server programs.
  - *include* - Header files for the C programming language. Should be below /user/lib for consistency.
  - *lib* - Unchanging data files for programs and subsystems.
  - *local* - The place for locally installed software and other files.
  - *man* - Manual pages.
  - *info* - Info documents.
  - *doc* - Documentation.
  - *tmp* - temporary file system.
  - *X11R6* - The X windows system files. There is a directory similar to usr below this directory.
  - *X386* - Like X11R6 but for X11 release 5.

- *boot* - Files used by the bootstrap loader, LILO or GRUB. Kernel images are often kept here.
- *lib* - Shared libraries needed by the programs on the root filesystem.
  - *modules* - Loadable kernel modules, especially those needed to boot the system after disasters.
- *dev* - Device files.
- *etc* - Configuration files specific to the machine.
  - *skel* - When a home directory is created it is initialized with files from this directory.
  - *sysconfig* - Files that configure the linux system for devices.
- *var* - Contains files that change for mail, news, printers log files, man pages, temp files:
  - *file*
  - *lib* - Files that change while the system is running normally.
  - *local* - Variable data for programs installed in /usr/local.
  - *lock* - Lock files. Used by a program to indicate it is using a particular device or file.
  - *log* - Log files from programs such as login and syslog which logs all logins and logouts.
  - *run* - Files that contain information about the system that is valid until the system is next booted.
  - *spool* - Directories for mail, printer spools, news and other spooled work.
  - *tmp* - Temporary files that are large or need to exist for longer than they should in /tmp.
  - *catman* - A cache for man pages that are formatted on demand.
- *mnt* - Mount points for temporary mounts by the system administrator.
- *tmp* - Temporary files. Programs running after bootup should use /var/tmp.

3.4 System and User Programs

System programs, mostly written by the GNU (GNU is a recursive acronym for "GNU is Not Unix"; it is pronounced "guh-NEW") Project, give a collection of useful commands like "move" (mv), "change directory" (cd), "copy" (cp), etc. These commands are run in a shell only and enable the user to complete tasks without using desktop environments or user programs.

User programs, on the other hand, are running on top of a shell, such as an ftp server, or a text editor. These have been written by thousands of different programmers around the world, and are mostly released under the GPL (General Public Licence).

3.5 Distributions

A Linux Distribution is a copy of the Linux operating system, its related utilities and software programs from a specific vendor. For example, Redhat offers a Linux Distribution, as does Debian, SuSE, and several others.

There are several differences between each distribution from different vendors. First of all, there is the price. Although most vendors offer a free download of their distribution from their web sites, a full version can be purchased to avoid long download periods. Their prices differ, and most have prices for a Personal version and a Server version, each with a different set of programs that go with it. Secondly, different distributions include different utilities, software programs, and features. Also, different distributions have different support offers from the vendors.

The most popular Linux distribution is Red Hat Software. Red Hat has a simple installation utility and offers many different versions for purchase. It also has 30 days of support with additional support available for purchase.

Another popular Linux distribution due to its easy installation process is Caldera Linux. It is aimed at the corporate market, with a copy of Staroffice and ease of use for workstations.

SuSE is also a popular distribution of Linux, especially its German version. The

SuSE 8.1 distribution will be introduced in the next section.

3.5.1 The SuSE 8.1 Distribution

SuSE 8.1 runs on the 2.4.19 Linux kernel and includes support for USB 2.0 and FireWire devices. Plus, SuSE ships with a seemingly endless supply of programs, ranging from sophisticated graphics and video tools to utilities and servers galore. It also includes the OpenOffice 1.0.1 office suite.

SuSE, like Mandrake, has always placed emphasis on desktop software and usability features. Though SuSE also offers strong server features, the maturity of the desktop packaging, documentation, and overall appearance of the product is evident. SuSE is a participant in the United Linux initiative, but the features of United Linux are found only in the Enterprise Server products.

SuSE Linux AG prefers carefully controlled development and testing cycles. For that reason, SuSE Linux AG does not engage in open Beta tests, nor do they make such tests available to the general public. In contrast, some of the other vendors, especially Mandrake and Red Hat, involve the community more in both development and testing. SuSE Linux AG is very involved in bringing its innovations to the GNU/Linux community, but the company is more concerned about protecting its software assets than getting new code out to the overall community right away.

The choice fell on SuSE 8.1, only because this distribution was freely available to the author.

Since the programming of the Axis Etrax100Lx developer board takes place on a low level, shell based, C programming editor, which is included in the Etrax100Lx Software Development Kid (SDK), only the running Kernel Version 2.4.19 was of importance. Any other Linux distribution, running on the same kernel version, would have been as good as the SuSE 8.1 distribution.

Chapter 4

BLUETOOTH HARDWARE USED IN THIS WORK

This chaper serves as a short overview on the used Bluetooth hardware. Both hardware and software features of the Bluetooth hardware used will be listed in this chapter. For further details please visit the manufacturer's homepages:

- *http://www.mitsumi.com*
- *http://www.developer.axis.com*

4.1 Axis Etrax100LX developer board for Bluetooth

The Axis Etrax100LX developer board for Bluetooth was originally purchased for a different project under the supervision of Prof. J. Tapson. It was used for this project because of its versatility and its implemented Bluetooth technology. The following section gives a short overview of its features [5], both hardware and software.

4.1.1 Hardware

The Developer Board for Bluetooth is shipped with the following hardware features:

- ETRAX 100LX, highly integrated 100MIPS 32bit RISC CPU
- Ethernet 10/100 Mbps Twisted Pair, RJ-45 connector with integrated magnetics
- 1 RS-232 serial port, 9 pin male D-SUB with RX, TX, RTS, CTS signals
- 1 Asynchronous serial port with 3.3 V levels, pin header
- 1 Bluetooth module. Alps UGTZ4-121A
- 1 Bluetooth Antenna
- Power, Radio and Network LEDs
- TEST button - to make a factory default
- BOOT button - to enable network boot
- Power: 9-24 V AC (or DC) on a standard connector and on screw terminal block
- FLASH: 4 MByte
- RAM: 16 MByte DRAM

The picture in figure 4.1 presents the Developer Board for Bluetooth and does not include a description of its components. For a detailed component description please visit the Axis developer homepage at:

- *http://developer.axis.com/doc/hardware/etrax100lx/des-ref.html*

Figure 4.1: **The Axis Developer Board for Bluetooth**

4.1.2 Software

The Axis Etrax100LX developer board for Bluetooth is shipped with embedded Software and a Software Development Kit (SDK) which feature the following:

- Linux 2.4 kernel.
- Support for 7 Bluetooth clients.
- Bluetooth profiles included are: LAN, Dialup and Serial.
- More than 1.5 Mbyte free flash space for own applications etc.
- Configured as a Bluetooth Access Point at delivery.

The SDK includes a comprehensive list of applications. Below is a summary of these applications and what they do. Many of these applications are just standard applications that have been compiled for the ETRAX platform, but some are developed by Axis specifically for the ETRAX platform.

- **init** Implements a mini init daemon.

- **telnetd** Simple Telnet server - allows you to log on to the developer board.
- **inetd** This program invokes all internet services as needed.
- **in.telnetd** Telnet server. This is the default telnet server.
- **sftpd** Lightweight FTP server.
- **sftpclient** Lightweight FTP client.
- **shells/ash** Ash is a fairly small Bourne compatible shell which offers the power and flexibility of shell scripting for an embedded system.
- **boa** High performance web server with support for CGI.
- **smtpclient** This program is a minimal SMTP client that takes an email message body and passes it on to a SMTP server (default is the MTA on the local host).
- **bootblocktool** Sets/gets bootblock parameters using a specified device.
- **hwtest/IO** Input / Output test utility.
- **hwtest/serial** Program to test serial port functionality.
- **hwtest/hwtest** Program for setting or getting status from parallel port serial port, button, and some other useful devices.
- **sysklogd** Sysklogd provides two system utilities which provide support for system logging and kernel message trapping. Support of both internet and Unix domain sockets enables this utility package to support both local and remote logging.
- **editors/easyedit** An easy to use text editor.
- **editors/editcgi** This is a simple CGI based editor and file browser.
- **login** Login is used when signing onto a system. It can also be used to switch from one user to another at any time (most modern shells have support for this feature built into them, however).
- **dhcp** Complete DHCP client.

- **busybox** Busybox combines tiny versions of many common UNIX utilities into a single small executable.
- **ipsetd** Daemon for setting IP address with ARP+Ping in user-space.
- **iptables** Implements a NAT (Network Address Translation) firewall.
- **tools/gdbserver** The gdbserver will help you debug user applications running on the Developer Board for Bluetooth.
- **utils/eraseflash** Flash erasing utility.
- **utils/readbits** Utility for reading bits on ETRAX general purpose I/O (GPIO) ports.
- **ppp-2.4** PPP daemon, which negotiates with the peer to establish the link and sets up the PPP network interface.
- **bluetooth/experimental** Contains a number of Bluetooth related applications.
- **bluetooth/sdp-server** Sdp-server is used for searching the SDP database stored in the XML file sdp.xml.

4.2 The Mitsumi Bluetooth USB Adapter

The Mitsumi Bluetooth USB Adapter was purchased in order to set up a functional Bluetooth PAN Network with the Etrax100lx developer board for Bluetooth. It also functions as the Network Access Point (NAP) in this project. The following section gives a short overview on its features, both hardware and software.

4.2.1 Hardware

The MITSUMI Bluetooth USB Adapter allows all Bluetooth compatible PCs and notebooks to be enhanced by the Bluetooth functionality. All hardware information is taken from [9].

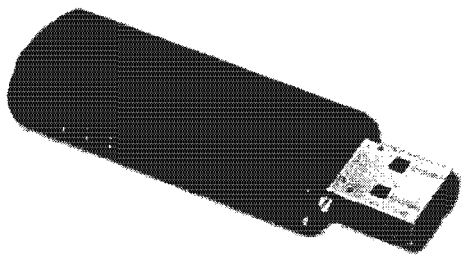

Figure 4.2: **The Mitsumi Bluetooth USB Adapter**

Hardware Specifications:

- Transmitter range: 0 - 10m
- Supported profiles: Bluetooth Specification 1.1
- Maximum data transfer rate: 723,2kbps (Asymmetric), 433,9kbps (Symmetric)
- RF Band: 2400MHz -2483,5MHz
- Interface: USB
- Weight: 9g

Supported Operating Systems:

Windows 98SE, Windows Me, Windows 2000, Windows XP, Linux 2.4. and later

System requirements:

IBM PC/AT compatible system with pentium processor or newer.

4.2.2 Software

In this project, the Mitsumi Bluetooth USB Adapter is run with the official Linux Bluetooth protocol stack, the BlueZ Stack. The BlueZ Linux Bluetooth stack includes a generic driver for Bluetooth USB devices which are according

to the Bluetooth USB Transport Protocol defined in the Bluetooh Specification (Section H2).

Here is a small introduction to the stack:

General:

BlueZ provides support for core Bluetooth layers and protocols. It is a flexible, efficient and modular implementation.

BlueZ features:

- Support for multiple Bluetooth devices
- Multithreaded data processing
- Hardware abstraction
- Standard socket interface to all layers
- PSM level security support

Currently BlueZ consists of:

- BlueZ Core
- HCI UART, USB, PCMCIA and Virtual HCI drivers
- L2CAP protocol module
- RFCOMM protocol module
- PAN/BNEP protocol module
- SCO module
- Configuration and testing utilities

Supported hardware platforms:

- x86 (UP and SMP)

- SUN SPARC (32/64 UP and SMP)
- ARM (Intel StrongARM, XScale)
- PowerPC
- Motorola DragonBall

Supported Linux Distributions:

RedHat, Debian, SuSe, etc.

Chapter 5

SYSTEM DESIGN

This chapter explains the system design in more detail. The first section describes the original system design as required by the supervisor Prof. J. Tapson. The second section describes the current system design and its limitations. Because the development of the Bluetooth technology under Linux is an ongoing process a lot of extensive research had to be done. New software versions, sometimes including new features, are being released constantly. Due to this fact a great part of this thesis dealt with research for the right software components to meet the requirements of the project. The last section of this chapter gives the reader an idea about the sensor requirements for such a monitoring device.

5.1 Original and Overall System Design

The overall idea behind this project is to create a web-based monitoring system for electrical and mechanical rotating machinery. These have different monitoring requirements for the motive systems, but the rotating mechanisms can be treated as identical. The intention was to provide a single solution for both types of rotating systems. The original system requirements (at the beginning of the thesis) are explained below:

The monitoring system should consist of a central data logger (see figure 5.1), which should contain sufficient RAM memory to record and hold spectral infor-

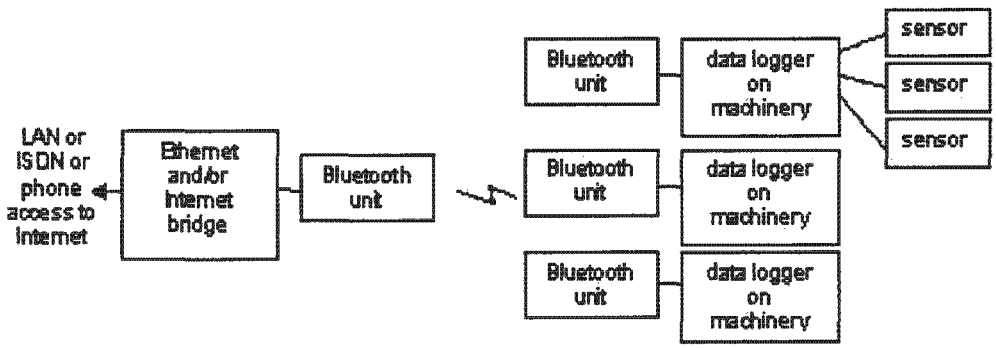

Figure 5.1: **Basic System Layout**

mation until this is downloaded. The sensors which measure system conditions should be wired into the data logger, and should be sampled, with analog to digital conversion in the data logger. The data logger should accept 4-20mA signals as the standard analog inputs, and 0-5V as the digital inputs.

The Bluetooth wireless data network should be used to connect the data loggers, situated at the machinery, to a unit which is situated at the nearest connection point to the Internet. The connection point may be the customer's LAN, an analog or digital phone line, a digital cellular telephone line, or other appropriate connections. In some cases, it should be possible to connect the data loggers directly to the LAN without the use of the Bluetooth unit. Bluetooth radios occupy the license-free 2.4 GHz ISM band and use a spread-spectrum modulation which is extremely reliable in an electromagnetic environment.

Either the data logger or the bridge unit should be a protocol bridge which converts the data into an internet-compatible format (Html, XML for example) and encrypts it for security during transmission over non-secure networks. The encrypted data should then be posted to a central database, where it can be available for remote access using, for example, an internet web browser.

Simple alarm conditions (such as threshold values) can be detected and raised on-site by the data logger. Sophisticated fault detection and diagnosis software should be maintained on the central database server; each incoming data set is screened by a neural network system which compares it with standard waveforms for that particular machinery set, and should detect quite marginal signs of future failure.

Alarms should be tracked by the central database software and should be escalated if not acknowledged within a preset time. All alarms should be logged, and a full audit trail should be available which tracks both alarm responses and any alteration to alarm thresholds and other setpoints.

Physical Details:

The core hardware should be the data logger. The logger should be supplied either with or without an enclosure for OEM purposes; the circuit boards should be approximately 150mm x 110mm x 50mm and should be mounted using four 4mm screws or bolts. The sensors should be connected using Phoenix springloaded connectors which allow quick wire connection in the field.

Electrical Details:

Power Supply: The data loggers should require 15-24V DC supply and should draw no more than 500mA each, depending on sensor configuration.

Local Networking: Data loggers should be connected using the Bluetooth wireless network, or directly to a LAN using 10 Mb Ethernet. The units should be supplied with a standard RJ-45 ethernet connector.

These are the original requirements on the system. After considerations on the available hardware, a few changes had to be made. Those are explained in the next section.

5.2 Current System Design

After extensive research, installation of the used components, updating those components with the latest available software, and due to the availability of only one sensor and its data output, the original layout of the system design had to be restructured. This section will explain the changes made and their impact on the project. The main focus of this thesis changed to developing and implementing a software application which reads in general data information from the serial port of the developer board and saves this data in a file on the board. Furthermore,

this file had to be made accessible for remote access using the Bluetooth connection. Figure 5.2 gives an overview of the current system including the used hardware components.

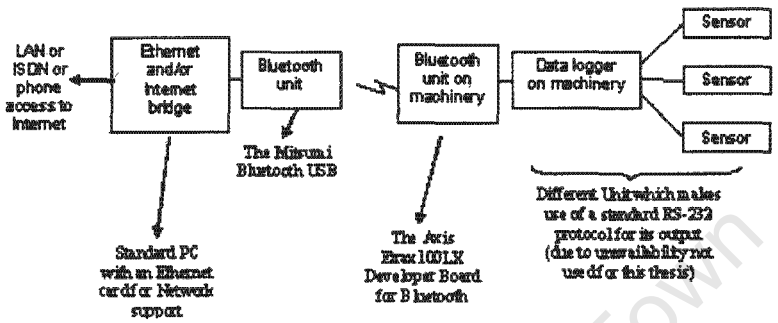

Figure 5.2: **Current System Design**

5.3 Sensors

Due to the availability of only one temperature sensor and its data output, this project did not focus on dealing with the actual sensor data. Hence, its main focus was to generally read in data from the serial port of the developer board. Nevertheless, this section will give a short overview on the required sensors for building a monitoring system for electrical and mechanical rotating machinery. Furthermore, it will give a short list of recommended sensors.

Required Sensor Inputs: The standard unit should offer 16 high-speed analog channels (suitable for high-frequency variables such as vibration) and 4 slow analog channels (suitable for variables such as temperature). The high-speed channels could also be used for low-speed variables. In addition, up to 8 digital inputs should be added for the application. A tachometer[1] input should be offered in the standard unit.

Analog Inputs:

- Range: 4-20mA

[1]This is a sensor which measures revolutions per minute (RPM)

- Bandwidth: DC - 16kHz (fast), DC - 1kHz (slow)

Digital Inputs:

- Range: 0-5V
- Bandwidth: DC - 1KHz

Tachometer Input:

- Range: 12V digital (can be changed)
- Pulse rate: 0.1Hz - 100Hz

Sampling:

- Basic Sample Rate: 40kHz
- Resolution: 8/12/16 bit (User specified), 16 bits available at lower sampling rates
- Memory Depth: 128 kilosamples

Suggested Sensor Pack:

A general purpose sensor pack is recommended, with additional specialized sensors according to whether the it is a mechanical or electrical rotating machinery.

Temperature:

- 8 analog sensors
- 4-20mA output
- DC-0.2Hz

Can be used for:

- Ambient Temperature (which was used during development)

- System Water Temperature
- Exhaust Gas Temperature
- Outboard Bearing Temperature
- Inboard Bearing Temperature
- Discharge Water Temperature
- Engine Water Temperature
- Pumping Fluid Temperature

All temperature sensors could be silicon sensors scaled to the range -25°C to 150°C, except for exhaust gas temperature which could be a PT-100 scaled to 100°C-800°C. The sensor used in this thesis was a Dallas DS 1820 temperature sensor in combination with a Microchip Pic 16F84 flash based 8-bit CMOS which provided the serial input of the developer board with digital data.

Vibration:

- 4 in total (2 x 2-axis sensors)
- Analog, 4-20mA or 0-5V output
- DC-5kHz
- ±10G full-scale output

Pressure:

- 2 sensors
- Analog, 4-20mA output
- DC-1kHz
- 20bar full-scale output

Speed:

- 1 sensor
- Digital, 12V pulses
- 0.1 - 100Hz PRF

In addition, for mechanical rotating machinery, the following extra sensors will be required:

Oil pressure:

- 1 sensor
- Analog, 4-20mA output
- DC-1kHz
- 10 or 20bar full-scale output

Fuel level:

- 1 sensor
- Analog, 4-20mA output
- DC-0.1 Hz
- Ultrasonic or rheostat - sized for fuel tank

In addition, for electrical rotating machinery, the following extra sensors will be required:

Current transducers:

- 3 sensors
- Analog, 4-20mA output
- DC - 5kHz
- Rogowski coil or CL Hall effect
- Scale according to motor size -100, 400 or 1000A f.s.

Chapter 6

SOFTWARE INSTALLATION

This chapter describes how the software used for this project was installed.

6.1 Axis Etrax100LX developer board

First of all, the software packet for the developer board had to be installed, since it is the only possible way to create executable software for the developer board. In order to develop an executable, the source code has to be written, cross compiled and linked on a PC. Once the program is created, it can be transferred to the hardware of the board in basically two different kind of ways - it can be either flashed onto the board, or transferred onto it using ftp. This section will explain how the software for the board has to be installed on the PC side, as well as what the general configurations have to look like in order to be able to access the board through either one of its interfaces.

6.1.1 Connecting the Developer Board to the PC

The Axis Developer Board for Bluetooth comes with a RS-232 serial port and a RJ-45 Ethernet 10/100 Mbps connector. It is recommended to configure both connectors in order to have full access to the board.

First of all, it is advisable to set the IP address for the board. As a default,

the Ethernet address of the developer board is the same as its serial number. The serial number is found on the label on the board. There are several ways of setting the IP address either temporarily or permanently:

Setting the IP Address Temporarily:

The developer board has a default IP address (e.g. 192.36.253.80) that needs to be changed in order to make the board work within a local network. The board does not support automatic IP setting, like DHCP(Dynamic Host Configuration Protocol) when delivered. DHCP must be activated manually later. In order to work within a network, the board requires a unique IP address. For the initial setting of the IP address the board needs to be connected to the same network segment as the PC client.

The following methods describe how the IP address can be set temporarily. This has to be done within thirty minutes after booting the developer board:

ARP[1] and ping from UNIX or GNU/Linux:

Either one of the following commands has to be typed in a shell either as super user ("su") or as "root" on the PC client side:

- arp -s <IP address> <Ethernet address>
- ping -s 408 <IP address>

Example:

- arp -s 123.45.67.89 00:40:8c:12:34:56
- ping -s 408 123.45.67.89

The "-s" stands for set.

This will set the IP address of the board temporarily. In order to make this new IP address permanent, the IP address, subnet mask and network address have to be changed in the *network.conf* file on the developer board (found in the

[1]Address Resolution Protocol, a TCP/IP protocol used to convert an IP address into a physical address, such as an Ethernet address.

/etc/network/ directory), otherwise the default network settings will be restored again when the board is rebooted and all changes are lost.

If setting the IP address temporarily should not work for some reason, it can still be set permanently using the serial port (see next).

Setting the IP Address Permanently:

Once the IP address has been set temporarily it can be set permanently using HTTP, telnet or FTP. If the temporary setting of the IP address did not work, it can still be set permanently, using a serial connection to login to the board. These methods are explained as follows:

HTTP:

Any type of a web browser can be used to http into the board. The command *http://<IP address>/* (example: http://123.45.67.89/) will log the user into the board. After editing the variables according to the local network under *"Edit network settings"*, the changes will be saved permanently with the *"Save file button"*.

telnet:

To Telnet to the IP address of the developer board, the following command has to be typed in a shell:

- telnet <IP address> (e.g. telnet 123.45.67.89)

The default user and password settings for the board are:

- login: root
- Password: pass

Now, the *network.conf* file in the */etc/network/* directory has to be edited using the editor /bin/easyedit.

Example:

- easyedit /etc/network/network.conf

- Change the settings
- Save the file and exit easyedit by pressing <ESC>
- Close the connection by typing exit

FTP:

The following steps have to be done in order to use ftp to change the settings:

- Open an FTP client on the PC client side.
- Open a connection to the IP address of the board.
- Log into the board using the default user and password settings for the board:
  - login: root
  - Password: pass
- Download the file */etc/network/network.conf*
- Change the settings in the file.
- Upload the file to the developer board.
- Close the connection to the board.

Login from serial port:

If connecting to the developer board over the network fails the board can be accessed from a serial port instead and the *network.conf* file can be edited through there.

To Connect the RS232 port on the developer board to a serial port on a PC, an ordinary null modem cable with RX/TX and RTS/CTS has to be used to connect the two. After that, a serial connection to the board has to be established by using a terminal program (e.g. cu or minicom). The settings for this connection should be:

- 115200 baud connection speed

- no parity
- eight data bits
- one stop bit

If everything went right, the text *"AxisProduct login:"* comes up. After the connection is established, the same steps as when using the telnet configuration are required.

The network entry will be used for the route to the local network in the routing table of the kernel. If this does not match the actual address of the local network it will be impossible to reach the developer board over the network after reboot. (In that case only a logging through the serial port is possible to make changes to the *netwok.conf* file.)

It is recommended to test the configuration by rebooting the developer board and pinging it.

Example:

- Reboot the board:
  - Press and release the reset button on the developer board
  - Wait for approximately 20 seconds (until the developer board is up and running again)
- Ping the board by typing *ping <IP Address> (e.g. ping 123.45.67.89)* on the PC client side.

6.1.2 Installing the Developer Board for Bluetooth Software

This and the following section describe the installation, setup and configuration of the Developer Board for Bluetooth Software [11] as well as the recommended update for the board, enabling it to run on a Linux 2.4.19 kernel and having the PAN profile support implemented.

Any Linux distribution with a kernel 2.4.14 or later should be working, the developers from Axis have tested it on Redhat and Debian and it was working on SuSE 8.1 for this project. First of all, the SDK requires some other software, which has to be installed on the PC side as well:

- gcc C compiler
- cris compiler tools (version 1.16 or later)
- GNU make
- awk
- bc
- byacc (or yacc if byacc is a link to it)
- lex or flex
- patch
- perl
- sed

The PC this software is installed on should also have an ethernet network interface.

The Developer Board for Bluetooth Software does *not* include the Linux kernel. The original SDK (called devboard_bt-Rx_y_z.tgz) uses Linux 2.4.14 so if the PC the SDK should work on uses a later kernel version, the source code of the Kernel 2.4.14 needs to be installed in the */axis/devboard_bt/os* directory (this will be explained later). The source code of this kernel can be downloaded from http://www.kernel.org or any of the mirror sites. The file is called linux-2.4.14.tar.gz and can be downloaded from (as done for this project):

- http://www.kernel.org/pub/linux/kernel/v2.4/ *or*
- ftp://ftp.kernel.org/pub/linux/kernel/v2.4/

Installing devboard_bt-Rx_y_z.tgz

Then latest version of the Developer Board for Bluetooth Software (*devboard_bt-Rx_y_z.tgz*) can be downloaded from the directory:

- http://developer.axis.com/download/devboard_bt/latest/

To install this software packet, the downloaded tgz-file needs to be "Ungziped" and "untared" in the desired working-directory:

- cd (to desired working-directory)
- tar -zxvf /path/to/the/downloaded/devboard_bt-Rx_y_z.tgz

It does not matter where the source code tree is "untared", in the case of this thesis it was /usr/local/share/2.4.20devboard.

Installing the Kernel 2.4.14 source code

The installation of the kernel source code is explained in the file: *axis/devboard_bt/README*. This software uses the Linux 2.4.14 kernel and the installation script needs to know where the source code for it is located. It is recommended to "ungzip" and "untar" it in the directory axis/devboard_bt/os:

- cd axis/devboard_bt/os
- tar -zxvf /path/to/linux-2.4.14.tar.gz

If the source code for kernel 2.4.14 is already installed on the PC in a different directory (e.g. /usr/src/linux) it can be left there and the install script will create a symbolic link (axis/devboard_bt/os/linux)to that directory.

Now the software can be installed by running the install script:

- cd axis/devboard_bt ./install

Whenever this distribution of the software is used, it is necessary to "source" the init_env file in the axis/devboard_bt directory. This is how it is done:

- cd axis/devboard_bt
- . ./init_env *or*
- source ./init_env

To be able to see the boot log from the developer board, the serial port on the developer board has to be connected to the serial port on the PC using an ordinary null modem cable (as explained in 6.1.1). A terminal program is needed to listen to the serial port on your computer. The easiest way is to run cu or minicom. However, any terminal program can be used when using the following settings:

- 115200 baud connection speed
- no parity
- eight data bits
- one stop bit

Any terminal program usually listens to the COM1 port (/dev/ttyS0) as a default. When using another COM port, this has to be kept in mind and according changes have to be made (e.g. /dev/ttyS1).

6.1.3 Linux 2.4.19 and PAN profile support

This section explains the required steps for updating the developer board and the SDK for using Linux kernel 2.4.19 and adding PAN profile support for devboard_bt-R1_0_0 [12]. It is advisable to save all old files before they are replaced with the new ones. In case the update fails for some reason, the old setup can be reinstalled very easily. The best way to do that is to move the files that are going to be replaced. This is done by just adding the extension *.old* to all the files or directories which are going to be replaced during the installation process. Those so called "safety precautions" will be emphasized throughout this section.

The whole upgrading process requires an installed and working devboard_bt-R1_0_0, as explained in 6.1.2 and the required files for this update should be downloaded from:

- http://developer.axis.com/download/devboard_bt/latest/linux-2.4.19+pan/

When the Axis Developer Board for Bluetooth is successfully upgraded, the default PIN-code for connecting via Bluetooth will be "12345" (for both LAN Access and PAN profile). This default setting can be changed in the file /etc/btsec.conf after successfully installing this upgrade.

Upgrading the Kernel to version 2.4.19:

First of all, to upgrade the kernel version of the developer board, the source code file *linux-2.4.19.tar.gz* has to be extracted into the directory axis/devboard_bt/os and a symbolic link (called "linux") to the directory linux-2.4.19 has to be created:

- cd axis/devboard_bt/os
- tar -zxvf /path/to/linux-2.4.19.tar.gz
- ln -sf linux-2.4.19 linux

If a directory called "linux" should already exist (as created in 6.1.2), it has to be renamed before creating the symbolic link:

- mv linux linux.old (to save the old directory!)

After that, the kernel patch called *linux-2_4_19-openbt.diff.gz* has to be extracted and applied as following:

- gunzip /path/to/linux-2_4_19-openbt.diff.gz
- cd linux
- cat /path/to/linux-2_4_19-openbt.diff — patch -p1

After the kernel patch was applied successfully, the old kernel configuration files *kernelconfig and kernelconfig.latest* should be replaced with the new ones (by renaming the old ones and extracting the new ones into the same directory):

- cd axis/devboard_bt

- mv kernelconfig kernelconfig.old (to save the old file!)
- mv kernelconfig.latest kernelconfig.latest.old
- cp /path/to/kernelconfig.gz
- gunzip kernelconfig.gz

When this is done successfully, the kernel can be rebuild with:

- make kernel

Installing the PAN Profile Applications

First of all, the directories *apps/bluetooth/experimental* and *apps/bluetooth/sdp_server* should be renamed and replaced with the new versions as following:

- cd axis/devboard_bt
- mv apps/bluetooth/experimental apps/bluetooth/experimental.old (to save the old directory!)
- tar -zxvf /path/to/apps-bluetooth-experimental-R1_0_23.tgz
- mv apps/bluetooth/sdp_server apps/bluetooth/sdp_server.old (to save the old directory!)
- tar -zxvf /path/to/apps-bluetooth-sdp_server-R1_0_19.tgz
- tar -zxvf /path/to/apps-bluetooth-btsec-R1_0_22.tgz
- tar -zxvf /path/to/apps-bluetooth-bttool-R1_0_11.tgz
- tar -zxvf /path/to/apps-bluetooth-pan-R1_0_8.tgz
- tar -zxvf /path/to/apps-bridge-utils-R1_0_0.tgz
- tar -zxvf /path/to/packages-bridge-bnep-eth-development-R1_0_0.tgz
- tar -zxvf /path/to/packages-param-bluetooth-sdp-lan+pan-R1_0_0.tgz
- tar -zxvf /path/to/packages-param-bluetooth-security_development-R1_0_0.tgz

After having done that, the old makespec file must be replaced with the new one containing the new applications and packages:

- mv makespec makespec.old (to save the old file!)
- cp /path/to/makespec.gz
- gunzip makespec.gz

After that, the old inittab file has to be replaced with the new one:

- mv files/etc/inittab files/etc/inittab.old (to save the old file!)
- cp /path/to/inittab.gz files/etc/inittab.gz
- gunzip files/etc/inittab.gz

Now, the new applications and packages have to be prepared to be built for cris and glibc with:

- make cris-axis-linux-gnu

Finally, the new applications and packages can be built and installed (including the new inittab) with:

- make install

To be able to upgrade the running software on the developer board, a new flash image has to be created as the SDK uses a flash script to update or install software on the board:

- make images
- set the board in network boot mode (by holding the boot button while powering up the board)
- ./flashit

6.2 Mitsumi Bluetooth USB Adapter

In order to get the Mitsumi Bluetooth USB adapter running on a Linux operating system, a lot of research had to be done, since the adapter only gets delivered with supporting software and drivers for Windows operating systems (like any other adapter as well). The first approach was to also use the openBT stack, since it is running on the developer board. After having compiled numerous kernels and modules and having tried out almost any possible configuration for the needed software to set up the adapter, it was decided not to use it on the PC side. Instead, BlueZ was given a try.

A good overview and *"howto"* can be found at [14], the official BlueZ stack site. Nevertheless, to get the Mitsumi Bluetooth USB Adapter running under Linux, including the PAN features (which are very important for this project), it was necessary to update the PC kernel to version 2.4.20 and to use the latest BlueZ stack. This section will explain how that was done.

First of all, the Kernel running on the Linux operating system should be of version 2.4.20. It was used for this project and it was not tested with any other version, although any newer kernel version will most probably have the BlueZ features implemented. When compiling the kernel version 2.4.20 for BlueZ support, it is recommended to deactivate any but the following Bluetooth support features in the kernel "make menuconfig" file:

Bluetooth support–>

- <M> Bluetooth subsystem support (M for module)
- < > L2CAP protocol support (not activated)
- < > SCO link support (not activated)
- Bluetooth device drivers–>
  - <M> HCI USB driver (M for module)
  - [ ] USB zero packet support (not activated)
  - < > HCI UART driver (not activated)
  - < > HCI VHCI (Virtual HCI device driver) (not activated)

and for USB support, since it is a USB adaptor, only the following should be activated:

USB support–>

- <M> Support for USB (M for module)
- [*] Preliminary USB device filesystem (* for building it inside the kernel, not as a module!)
- <M> UHCI (Intel PIIX4, VIA, ...) support (M for module)
- <M> UHCI Alternate Driver (JE) support

The rest of the options *must* be deactivated! After that, when the kernel was compiled and installed successfully (with make dep clean bzImage && make modules && make modules_install) the following files are required to install the BlueZ stack:

- bluez-kernel-2.3.tar.gz
- bluez-libs-2.2.tar.gz
- bluez-utils-2.2.tar.gz
- bluez-sdp-1.0.tar.gz
- bluez-pan-1.1rc2.tar.gz
- bluez-hcidump-1.5.tar.gz
- bluez-hciemu-1.0.tar.gz
- bluez-bluefw-0.9.tar.gz

The latest versions of those files can be downloaded at:

- http://bluez.sourceforge.net/download/download.html

After having downloaded the latest versions of those files, they need to be unpacked and installed in a directory of choice. Each one of them will create subdirectories:

- tar -xvfz "name of file" (e.g. tar -xvfz bluez-libs-2.3.tar.gz)
- cd "name of directory created" (e.g. cd bluez-libs-2.3)
- ./configure
- make
- make install

When installing the bluez-kernel-2.3.tar.gz and having successfully finished the above steps, a file called *./create_dev* (create devices) has to be run as well to create the software Bluetooth devices in the /dev directory of the operating system. It is most likely that this file is not included in the .tar.gz file but it can be downloaded from:

- http://cvs.sourceforge.net/cgi-bin/viewcvs.cgi/bluez/kernel/Attic/create_dev

When downloading the *./create_dev* file, it should be saved as "create_dev" and its status should be changed to being executable with:

- chmod +x create_dev

After having made those changes it can be run and the Bluetooth devices will be created in /dev.

Chapter 7

SOFTWARE DESIGN

After extensive and long software research, the right components were found and installed, as explained in the last chapter, to meet the requirements of this project. This chapter will explain in detail how the Bluetooth connection was established. Furthermore it will explain how the *sensorprogram* application, which was written for this project, was developed and how it functions.

7.1 Setup PAN

In order to meet the requirements for this project, it had to be made sure that the Axis Etrax100LX developer board would connect itself automatically to an existing network access point (NAP). This was achieved by using the PAN profile.

Setup PAN connection on NAP side (PC side)

After having installed the BlueZ software packet (see chapter 6), it is necessary to start the stack and the PAN profile on the PC side manually, although this only has to be done once. The following steps are required:

- *hcid -f /etc/bluetooth/hcid.conf* (Starts the HCI daemon)
- *modprobe bnep* (Loads the bnep.o module into the kernel)
- *sdpd* (Starts the SDP daemon)

- pand *–listen –role NAP* (starts the PAN profile and sets the PC in the master mode for the NAP)

Now the Network Access Point (NAP) is up and running and is "listening" for other PAN or bnep devices. Nevertheless, the connection has to be established from the slave side in this case the Etrax100LX developer board. In order to achieve that, since the developer board is running on a different Bluetooth stack (openBT), some changes had to be made to the bootup process on the board.

Setup PAN connection on the developer board

First of all, after the board is updated with the latest software, it is automatically configured as a Network Access Point (NAP) and all seven bnep devices are bridged to the ethernet port on the board. These configurations had to be changed by editing the bridge configuration file *"bridge"*, in the */devboard_bt/packages/bridge/bnep-ethernet-development* directory. This was done, by commenting out the eth0 and bnep0 entry in that file. Furthermore, the bnep0 device was given an own IP-address (in this case 10.0.0.2), so it could be accessed as a normal ethernet device through the Bluetooth network. The changes made can be found in APPENDIX A.

To be able to have an automatic connection between the Bluetooth devices established (bnep0 on both sides), the inittab (configuration file at bootup) of the developer board had to be configured differently. This was necessary due to the fact that the two devices are running on different stacks. The automatic Bluetooth connection was achieved by adding the following line to the Bluetooth configuration within the inittab file (also in APPENDIX A):

- bnep:3:once:/bin/bnep –connect 00:a0:96:20:42:a0 (Bluetooth Address of USB device)

To be able to make those changes work on the board, a new *flashimage* had to be built to update the board:

- cd axis/devboard_bt/ (on the PC side)
- make cris-axis-linux-gnu (to prepare the software to be build for CRIS and glibc)

- make install (to install the new bridge and inittab files)
- make images (to build a new flash image for the board)
- ./flashit (to write the new flash image to the board, the board has to be in network boot mode to do so!)

After having made those changes and rebooted the board, a PAN connection is established between the two devices. Both sides are running a bnep0 device connection, which can be seen as a virtual ethernet port. On the developer board, this bnep0 device was configured during bootup, giving it its own unique Ip-address (10.0.0.2). On the PC side, a configuration script was written to configure the bnep0 device automatically when ever it is created. This small configuration file is called *"ifcfg-bnep0"* and can be found in the */etc/sysconfig/network/* directory on the PC. Its purpose is basically to set the boot-protocol, the IP-address, the netmask and the startup-mode of the bnep0 device whenever it is created. This is what it looks like:

ifcfg-bnep0

BOOTPROTO=static
IPADDR=10.0.0.2
NETMASK=255.0.0.0
STARTMODE=hotplug

These settings are chosen to meet the settings of the bnep0 device on the developer board.

To be able to access the board through a web browser on the PC side, the bnep0 device and the eth0 device (on the PC) have to be "bridged" in order to make the web browser function via the Bluetooth connection. The PAN profile fully supports *"bridging"* and comes with a configuration tool which is easy to use. The tool is called *brctl* and is able basically to create a virtual bridge between any network devices. The example shows how the bridge was created on the PC side:

- brctl addbr pan0 (creates a bridge called pan0)
- ifconfig pan0 10.0.0.3 (sets the IP-address of the bridge)

- brctl addif pan0 eth0 (adds the eth0 interface to the bridge)
- brctl addif pan0 bnep0 (adds the bnep0 interface to the bridge)

Having created this bridge enables both interfaces to interact with each other and fully access the functionalities of one another. In the case of this project it is now possible to access the developer board through a web browser by only using the Bluetooth connection.

7.2 The "microcontroller" setup

The microchip pic microcontroller had to be programed to read in the data of the Dallas DS 1820 temperature sensor and convert it into a digital signal to feed the serial port of the developer board. The program code is attached to this thesis in APPENDIX C. This short program code was written to feed the serial port of the developer board with a 9600 baud, 8bit,no parity and one stop bit signal(9600 8N1). The hardware flow is also switched on.

7.3 The "sensorprogram" application

One of the main focuses of this project, besides creating an automatic Bluetooth connection between the two devices, was to create an application which enables the developer board to read in data information through its serial port. Furthermore, this information had to be saved automatically on the board's filesystem in order to be accessible through the established Bluetooth connection for remote access.

The biggest obstacle was the fact of having only one serial port available. The serial port of the developer board is the only interface which guarantees full access on the board itself. To be able to move, copy, and erase files or applications, as well as starting or terminating programs situated on the board, it is the only fully available and fully supporting interface. Nevertheless, all requirements were met and the software was developed as follows:

The idea behind the development of this software was to make it as flexible as

possible, in acknowledgement of the fact that this project is an ongoing process and further research and development should be able to build on to this thesis. Therefore a program was developed and implemented which is easy to use for further development. The programming language C was used to develop this application.

The application created is the executable *"sensorprogram"* which is located in the */bin/* directory of the developer board. It consists of three parts of software code:

- the header file *sensorport.h*
- the C file *sensorport.c*
- and its executable C file *sensorprogram.c*

The header file is in charge of declaring the developed main functions, which are:

- **OpenAdrPort(char* sPortNumber)**, which opens the serial port,
- **WriteAdrPort(char* psOutput)**, which can send data or commands through the serial port,
- **ReadAdrPort(char*psResponse, int iMAx)**, which reads information from the serial port, and
- **CloseAdrPort()**, which closes the serial port again.

The C file *sensorport.c* is in charge of establishing and ensuring the full functionality of those functions, where the main application *sensorprogram.c* makes use of those functions to create the desired application. All the source code can be found in APPENDIX B.

The applications executable can be found in the bin directory of the developer board. *"sensorprogram 0"* will execute the application by itself when starting up the developerboard using the standard serial port 1 (ttyS0), where, for example *"sensorprogram 1"* would execute the application using serial port 2. This available choice was put in for future development where multiple serial ports could be available. The application was programmed to read in sensordata for

20 seconds because at the moment it would not make sense to let the application capture the data infinitely, due to the lack of memory space. The data will be saved in a file called *sensordata*, which is situated in the */etc/* directory of the developer board, where it is accessible for remote access through the established Bluetooth connection.

7.4 Flow chart of the Application

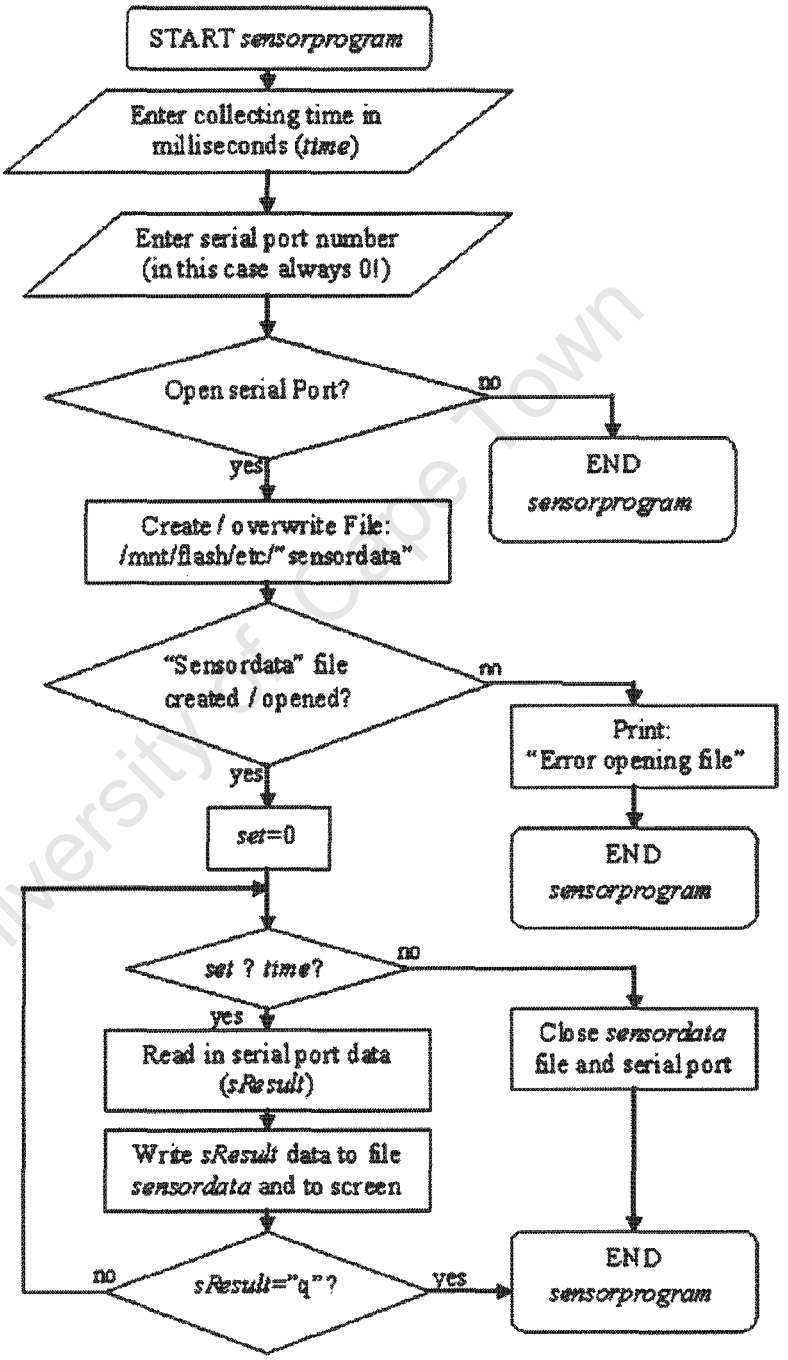

Figure 7.1: **Flowchart of the Application**

Chapter 8

CONCLUSIONS AND RECOMMENDATIONS

8.1 Conclusions

Based on the research and development the following conclusions can be drawn.

8.2 The Software

The software written can only serve as a starting point for further investigation or development concerning this project. Due to the fact that only one sensor was available, sufficient input data was missing and therefore the written software can only be seen as a basic tool to read input data from a serial port. The Bluetooth connection is stable and works well within a range of 10m. The PAN profile is a very powerful tool, due to its versatility; any type of network protocol is supported and can be send over a Bluetooth connection using the PAN profile. Concerning this project it was a little tricky to set it up but only due to the fact that the two Bluetooth devices were run on different stacks.

8.3 The Hardware

- **The Axis Etrax100LX** developer board for Bluetooth is a powerful and very versatile product. It suited the scope of this project perfectly. Due to the ongoing development concerning the Bluetooth technology, especially under Linux operating systems, a lot of changes and adjustments had to be made in order to fulfill the requirements. The only disadvantage of the developer board is the fact that it only comes with one serial port.

- **The Mitsumi Bluetooth USB adaptor** is a nice USB device. Within the scope of this project its performance was sufficient enough, although its performance in an industrial environment was never tested.

  The overall scope of this project has a lot of potential because monitoring systems in industry have gained more and more popularity over the last few years. The fact that wireless technologies like the Bluetooth PAN profile have the potential to be self-configuring will open up a lot of opportunities for similar systems in industrial and other environments.

8.4 Recommendations

The following recommendations can be made:

1. To be able to design a monitoring system as required within this project, the availability of multiple sensors has to be guaranteed. This is essential for future development.

2. Since the developer board only has one serial port and not sufficient memory to run big applications on it, it might be an idea to create a board on its own, since the Axis company offers to sell the Etrax100LX chip on its own. A suitable board with a sufficient amount of memory and interfaces could make this project even more versatile for maybe monitoring devices where a lot more sensors are required.

3. The data captured from the serial port could be packed into XML packages and made available to a lot more applications and programs.

4. A database could be implemented to handle the amount of sensor data.

The required 16 sensors for this monitoring system can produce a lot of data throughout the day, and bigger machinery might even require a lot more sensors.

5. The software written for this project should be running continuously on the board. With the implementation of actual multiple sensor data, this could be the scope of a new thesis.

6. A thesis could be proposed, testing out new available Bluetooth devices, their performance and range within an electro-magnetic environment.

7. Further work, maybe the scope of a thesis, has to be done to convert this project into a monitoring system suitable for industry.

Bibliography

[1] An Introduction to the Linux Kernel. [Online]. Available: http://www.newtolinux.org.uk/tutorials/linuxkernel.shtml [2003, April].

[2] Bluetooth History. [Online]. Available: http://www.ietf.org/pro-ceedings/00jul/SLIDES/ipobt-agenda/tsld004.htm [2003, April].

[3] Bluetooth Radio. [Online]. Available: http://www.palowireless.com/infotooth/tutorial/radio.asp [2003, April].

[4] Bluetooth Wireless Technology Overview. [Online]. Available: http://archive.devx.com/wireless/articles/Bluetooth/BtoothIntro.asp [2003, April].

[5] Developer Board for Bluetooth Software. [Online]. Available: http://developer.axis.com/software/devboard_bt/index.html [2003, April].

[6] History. [Online]. Available: http://www.cs.utk.edu/ dasgupta/bluetooth/history.htm [2003, April].

[7] "Specification of the Bluetooth System", Volume 1. [Online]. Available: http://www.bluetooth.com/pdf/Bluetooth_11_Specifications_Book.pdf [2003, April].

[8] The Bluetooth Specifications. [Online]. Available: http://www.bluetooth.org/specifications.htm [2003,April].

[9] The Bluetooth USB Adapter WIF-0402C. [Online]. Available: http://www.mitsumi.com [2003, April].

[10] BlueDrekar. [Online], April 2003. Available:
http://www.alphaWorks.ibm.com/tech/bluedrekar [2003, April].

[11] How to Install the Developer Board for Bluetooth Software. [Online], 2003. Available:
http://developer.axis.com/doc/software/devboard_bt/install-howto.html [2003, April].

[12] Linux 2.4.19 and PAN profile support for devboard_bt-R1_0_0. [Online], 2003. Available:
http://developer.axis.com/download/devboard_bt/latest/linux-2.4.19+pan/INSTALL [2003, April].

[13] The Official Bluetooth SIG web site. [Online], April 2003. Available:
http://www.bluetooth.org [2003, April].

[14] The Official Linux Bluetooth protocol stack Web Site. [Online], April 2003. Available:
http://bluez.sourceforge.net [2003, April].

[15] The Web Site for Axis Developers. [Online], April 2003. Available:
http://developer.axis.com [2003,April].

[16] A. Dornan. *The Essential Guide to Wireless Communication Application.* Prentice Hall, New Jersey, April 2001.

[17] Benny Bing. *High Speed Wireless ATM LAN's.* Artech House Publishers, Boston, April 2000.

[18] Frank Karle. Bluetooth Installation HOWTO. [Online], September 2001. Available:
http://medien.informatik.uni-ulm.de/frank/bluetooth/bluetooth-howto.html [2003, April].

[19] Hans-Cees Speel. BlueZ User HowTo. [Online], April 2003. Available:
http://www.hanscees.com/bluezhowto.html [2003, April].

[20] Mark Allen. CTDP Linux Files and Command Reference Version 0.8.0. [Online]. Available:
http://www.comptechdoc.org/os/linux/commands/linux_crfilest.html [2003, April].

[21] Praveen Yalagandula. Adopted protocols. [Online], September 2000. Available: http://www.cs.utexas.edu/users/ypraveen/surveys/wlan_security/node10.html [2003, April].

[22] Praveen Yalagandula. Bluetooth Protocol Architecture Overview. [Online], September 2000. Available: http://www.cs.utexas.edu/users/ypraveen/surveys/wlan_security/node6.html [2003, April].

[23] Praveen Yalagandula. Cable replacement protocol. [Online], September 2000. Available: http://www.cs.utexas.edu/users/ypraveen/surveys/wlan_security/node8.html [2003, April].

[24] Praveen Yalagandula. Core Protocols. [Online], September 2000. Available: http://www.cs.utexas.edu/users/ypraveen/surveys/wlan_security/node7.html [2003, April].

[25] Praveen Yalagandula. Telephony control protocol. [Online], September 2000. Available: http://www.cs.utexas.edu/users/ypraveen/surveys/wlan_security/node9.html [2003, April].

[26] Puneet Gupta. "Personal area networks. How did they come in existence?", April 2002. Available: http://www.wirelessdevnet.com/channels/bluetooth/features/pans2.html [2003, April].

[27] Ramjee Prasad. *Universal Wireless Personal Communications.* Artech House Publishers, Boston, 1998.

[28] T. Sheldon. *Encyclopedia Networking & Telecommunication.* McGraw-Hill, New York, April 2001.

Appendix A

Inittab and Bridge Configuration

A.1 Inittab

```
#
# inittab       This file describes how the INIT process should set up
#               the system in a certain run-level.
#
# Default runlevel is indicated by initdefault.
#
# The runlevels used by Elinux are:
#   0 - Halt.
#   1 - User mode 1
#   2 - User mode 2
#   3 - User mode 3
#   4 - User mode 4
#   5 - User mode 5
#   6 - Reboot
#
#  <ID>:<RUNLEVEL>:<ACTION>:<PROCESS>
#
# Possible actions are:
#
#     respawn      - restarted whenever it terminates
#     initdefault  - indicates default runlevel for initd
#       once         - started once
#       wait         - started once and init will wait for its termination

# Default runlevel
id:3:initdefault:

# -n means no fork. -o is max log size in bytes before rotation.
syslogd:3:respawn:/bin/syslogd -n -o 20000

# Simple FTP server
sftpd:13:respawn:/bin/sftpd

# -n means no daemon
inetd:3:respawn:/sbin/inetd -n

# dnrd:3:once:/bin/dnrd -i eth0

# Web server
boa:3:respawn:/bin/boa -c /etc/httpd

# Enable login from serial port
#agetty:3:respawn:/sbin/agetty -L console 115200
# We are using in.telnetd from inetd so we do not want telnetd running.
# telnetd:3:once:/bin/telnetd
# We use ipsetd for ARP+Ping IP-setting
# timeout 1800 secs, change IP immediately, -l for packet size
ipsetd:3:once:/bin/ipsetd -i eth0 -t 1800 -l 408 -c
# Bluetooth
btsc:3:respawn:/bin/btsec
btdm:3:respawn:/bin/btdm -R -u /dev/ttyS3 -m 1 -s 921600
bttl:3:wait:/bin/bttool -serno
btcf:3:wait:/bin/btconfig
brgd:3:once:/sbin/bridge
bnep:3:once:/bin/bnep --connect 00:a0:96:20:42:a0
sensorprogram:3:once:/bin/sensorprogram 0
```

A.2 Bridge Configuration

```
#!/bin/sh

# Disable IP of bnep interfaces
echo "Disabling IP of eth0 and bnep0-6"
ifconfig eth0 0.0.0.0
ifconfig bnep0 10.0.0.2 up
ifconfig bnep1 0.0.0.0 up
ifconfig bnep2 0.0.0.0 up
ifconfig bnep3 0.0.0.0 up
ifconfig bnep4 0.0.0.0 up
ifconfig bnep5 0.0.0.0 up
ifconfig bnep6 0.0.0.0 up

# Add the bnep bridge device
echo "Adding brbnep bridge"
brctl addbr brbnep

# Set minimum learning period
brctl setfd brbnep 0
brctl stp brbnep disable

echo "Adding interfaces eth0 and bnep0-6 to brbnep"
# brctl addif brbnep eth0
# brctl addif brbnep bnep0
brctl addif brbnep bnep1
brctl addif brbnep bnep2
brctl addif brbnep bnep3
brctl addif brbnep bnep4
brctl addif brbnep bnep5
brctl addif brbnep bnep6

# Get network parameters
. /etc/network/network.conf

# Configure brbnep interface
echo "Setting IP $IP on brbnep"
ifconfig brbnep $IP netmask $NETMASK broadcast $BROADCAST up

# Add default gateway for brbnep
if [ -n "$GATEWAY" ]; then
      echo "Adding default gateway $GATEWAY on brbnep"
      route add default gw $GATEWAY dev brbnep
fi

# NAT must use brbnep interface now
echo "Masquerading outgoing packets on brbnep"
iptables --table nat --flush
iptables --table nat --append POSTROUTING --out-interface brbnep \
      --jump MASQUERADE
```

Appendix B

Sensorprogram Application

B.1 sensorport.h

```
// sensorport.h
// Copyright Patrick Dolz
int OpenAdrPort (char* sPortNumber);
int WriteAdrPort(char* psOutput);
int ReadAdrPort(char* psResponse, int iMax);
void CloseAdrPort();
```

B.2 sensorport.c

```
// sensorport.c - Serial Port Handler
// Copyright Patrick Dolz

// Permission is hereby granted to freely copy,
// modify, utilize and distribute this example in
// whatever manner you desire without restriction.

#include <sys/time.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <asm/termios.h>
#include <string.h>
#include <errno.h>
#include "sensorport.h"

static int fd = 0;
struct termios my_termios2;
int OpenAdrPort(char* sPortNumber)
{
    int portSettings = 0;
    int rtsSettings=0;
    char sPortName[64];
    sprintf(sPortName, "/dev/ttyS%s", sPortNumber);

    //first make sure port is closed
    CloseAdrPort(fd);
    portSettings= (  O_RDWR | O_NOCTTY );
    portSettings |= CRTSCTS;

    fd = open(sPortName, portSettings);
    if (fd < 0)
    {
        printf("open error %d %s\n", errno, strerror(errno));
    }
    else
    {
        struct termios my_termios;
        tcgetattr(fd, &my_termios);
        tcflush(fd, TCIFLUSH);
        my_termios.c_cflag = (B9600 | CS8 | CLOCAL | CREAD | CBAUD | CSIZE);
      my_termios.c_cflag &= ~CRTSCTS;
      my_termios.c_cflag &= ~PARENB;
      my_termios.c_cflag &= ~CSTOPB;
        my_termios.c_cflag &= ~CSIZE;
        my_termios.c_cc[VMIN] = 1;
        my_termios.c_cc[VTIME] = 10;
      my_termios.c_lflag = 0;
      my_termios.c_iflag |= (INLCR);
      my_termios.c_iflag &= ~(IXON | IXOFF | IXANY);
      my_termios.c_iflag = IGNPAR;
      my_termios.c_oflag = OPOST;


      cfsetospeed(&my_termios, B9600);
      cfsetispeed(&my_termios, B9600);
```

```
    tcsetattr(fd, TCSANOW, &my_termios);
      tcflush(fd, TCIFLUSH);


      ioctl(fd,TIOCMGET, &rtsSettings);
      rtsSettings &= ~TIOCM_RTS;
      rtsSettings |= TIOCM_CTS;
      ioctl(fd,TIOCMSET, &rtsSettings);

      //Enable the line below to not read raw data
      //fcntl(fd, F_SETFL,0);
    } // end if
    return fd;
} // end OpenAdrPort

int WriteAdrPort(char* psOutput)
{
    int iOut;
    if (fd < 1)
    {
        printf(" port is not open\n");
        return -1;
    } // end if
    iOut = write(fd, psOutput, strlen(psOutput));
    if (iOut < 0)
    {
        printf("write error %d %s\n", errno, strerror(errno));
    }
    else
    {
      printf("wrote %d chars: %s\n", iOut, psOutput);
    } // end if
    return iOut;
} // end WriteAdrPort

// read string from the serial port
int ReadAdrPort(char* psResponse, int iMax)
{
    int iIn;
    int rtsSettings=0;

      ioctl(fd,TIOCMGET, &rtsSettings);
      rtsSettings &= ~TIOCM_RTS;
      rtsSettings |= TIOCM_CTS;
      ioctl(fd,TIOCMSET, &rtsSettings);

    if (fd < 1)
    {
        printf(" port is not open\n");
        return -1;
    } // end if
    strncpy (psResponse, "   % ",  iMax<4?iMax:4);
    iIn = read(fd, psResponse, iMax-1);
    if (iIn < 0)
    {
      if (errno == EAGAIN)
      {
            return 0; // assume that command generated no response
```

```
      }
      else
      {
            printf("read error %d %s\n", errno, strerror(errno));
      } // end if
    }
    else
    {
      psResponse[iIn<iMax?iIn:iMax] = '\0';
          //printf("read %d chars: %s\n", iIn, psResponse);
    } // end if

    return iIn;
} // end ReadAdrPort


// closes the serial port
void CloseAdrPort()
{
    if (fd > 0)
    { close(fd);} // end if

} // end CloseAdrPort
```

B.3 sensorprogram.c

```
// sensorprogram.c - Serial Port Test Example
// Copyright Patrick Dolz


//#include <termios.h>
//#include <sys/types.h>
//#include <sys/stat.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include "sensorport.h"
#include <fcntl.h>
#include <assert.h>
// this is the main program with a time setting of 40 seconds
int main(int argc, char *argv[])
{
      char sResult[254];
      FILE *Fout;
      int fileWrite;
      int time = 40;
      int set;
      int fdResult;
      int counter;
      counter = 0;
//for different data collecting time enable below!!!
      //printf("Enter the period of time you want to collect data!\n");
      //printf("(in milliseconds, e.g. enter 300 -> collects for 3 seconds)\n");
      //printf("?:");
      //scanf("%d", &time);
//    printf("First Argument:\s\n",argv[0]);


// for choosing different serial ports enable below
      //    if (argc < 2 || argc > 2)
      //    {
      //          printf("Please enter the number of the serial port\n");
      //          printf("it should be 0 to connect to /dev/tty0\n");
      //          return 0;
      //} // end if
      //fileWrite = open("./dataFile", O_CREAT| O_WRONLY| O_APPEND, 0666);
      fileWrite = open("/mnt/flash/etc/serialdata", O_CREAT| O_WRONLY| O_APPEND,
0666);
      assert(fileWrite>=0);
      write(fileWrite,"Opened File\n",12);
      if (fdResult = OpenAdrPort(argv[1]) < 0)
      {
          write(fileWrite,"Could not open Serial Port\n",27);
          return 0;
      } else{
          write(fileWrite,"Opened Serial Port!\n",20);
      }
      sleep(1);               //some time for the file to be written

      while(counter<time)
            {
            ReadAdrPort(sResult,253);
            write(fileWrite,sResult,14);
```

```
            counter++;
                        }
                                    //      end while {1}
write(fileWrite,"End of File\n",12);
close(fileWrite);
CloseAdrPort();
return 0;
} // end main
```

Appendix C

Pic program

C.1 pic program

```
'*****************************************************************
'* Name    : QC5.BAS                                             *
'* Author  : Patrick Dolz                                        *
'* Notice  : Copyright (c) 2005                                  *
'*         : All Rights Reserved                                 *
'* Date    : 22/04/2005                                          *
'* Version : 5.0                                                 *
'* Notes   :                                                     *
'*****************************************************************
define osc 4

' Set LCD Data port
'DEFINE LCD_DREG PORTB
' Set starting Data bit (0 or 4) if 4-bit bus
'DEFINE LCD_DBIT 0
' Set LCD Register Select port
'DEFINE LCD_RSREG PORTC
' Set LCD Register Select bit
'DEFINE LCD_RSBIT 4
' Set LCD Enable port
'DEFINE LCD_EREG PORTC
' Set LCD Enable bit
'DEFINE LCD_EBIT 5
' Set LCD bus size (4 or 8 bits)
'DEFINE LCD_BITS 4
' Set number of lines on LCD
'DEFINE LCD_LINES 2

'Include "modedefs.bas"

' Set receive register to receiver enabled
DEFINE HSER_RCSTA 90h
' Set transmit register to transmitter enabled
DEFINE HSER_TXSTA 20h
' Set baud rate
DEFINE HSER_BAUD 9600
'Set SPBRG directly (normally set by HSER_BAUD)
DEFINE HSER_SPBRG 25


temperature Var Word          ' Temperature storage
count_remain Var Byte         ' Count remaining
count_per_c Var Byte          ' Count per degree C

i var byte
DQ Var PORTA.2                ' One-wire data pin

DEFINE CHAR_PACING 1000
DEFINE LOADER_USED 1          'uses a bootloader
DEFINE HSER_BAUD 9600         'defining serial baud rate

CR    con 13                  'carriage return
LF    con 10                  'line feed

cdelay var word
cdelay = 5                    'clock delay
sdelay var word
sdelay = 5                    'short delay
```

C.2 pic program

```
sm
   bsf 03,5
   bcf 01,7
endasm

low porta
low portb

temperature = 0

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
main:       high portb.3
         pause 500
         low portb.3
         serout portb.6,2,[#Temperature,CR,LF]     'sending values to PC via RS232
         pause 500



sensetemp:     OWOut DQ, 1, [$CC, $44] ' Start temperature conversion

         high porta.0

waitloop:     OWIn DQ, 4, [count_remain] ' Check for still busy converting

         If count_remain = 0 Then waitloop

         OWOut DQ, 1, [$CC, $BE] ' Read the temperature
         OWIn DQ, 0, [temperature.LOWBYTE, temperature.HIGHBYTE, Skip 4, count_remain,
count_per_c]
         Temperature = temperature / 2
         low porta.0

         goto main
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
```