Δ17
1027

DATA ACQUISITION, TRANSPORT OVER A GSM NETWORK AND PRESENTATION OVER THE WEB

by

Dimitris Giakoumis

Technical University of Crete

2002

Supervisory Committee

Prof. Dollas Apostolos (Supervisor)
Prof. Mihalis Paterakis
Prof Kostas Kalaitzakis

Acknowledgements

The author wishes to express sincere appreciation to Professor Apostolos Dollas for his guidance and assistance during the design and implementation of this system. In addition, special thanks to Professors Michalis Paterakis, and Kostas Kalaitzakis, for participating in the Supervisory Committee and evaluating this work.

I would also like to thank Christos Vosnidis, Dimosthenis Stellakis, and Tassos Prigouris for allowing me the use of their hardware in the development stage of the system.

Last but not least, very special thanks go to everyone posting at comp.arch.embedded and the forums at www.avrfreaks.net who tried to help me.

1. Introduction

Information in this day and age is so important we call it "the information age". Over the years much effort has gone into various technologies with the sole purpose of distributing information, making it accessible to a widely distributed public. The improvement of available technology has led to a variety of progressively more advanced systems to process and distribute information. Beginning with the simple concept of a newspaper that was first and simplest form of information delivery to a large group of people, every advancement of technology caused the development of a faster and more direct form of information delivery. The introduction of electronics brought about the use of radio and later television, aerospace technology gave us satellite communication, and finally the invention of computer networks led to the single most widely spread information exchange network on the planet, the Internet.

Information is often contained in data, or derived from it, and this makes data acquisition systems an important part of any information delivery infrastructure. The applications of such a system are virtually limitless, as are the circumstances requiring fast delivery of data from a remote location. Stock and commodity values can be made available all aver the world. Data about traffic can be obtained by web enabled in-car systems. Meteorological, geological and medical data can reach scientists in real time, wherever they are, even on the move and from practically anyware. These are only some of the applications of such a technology. Even things such as parking availability reporting systems and vehicle tracking systems can be implemented through a data acquisition system that samples data from an embedded system. Connectivity of the embedded system through the GSM network enables us to locate the data source system practically anyware within GSM covered area (which is almost everyware today), without any need for cable connectivity or any wireless access point being in the vicinity.

The purpose of this work is to develop a data acquisition system. The system must transport data from an embedded system to a central server via a GSM network. From there the data is presented upon request in a web interface over a TCP/IP network, in a text or graph form. Multiple clients must be able to connect to the server at once.

Chapter 2 gives an overview of relative products in the market, thus reveling the penetration of this type of technology to the consumer. In chapter 3 the design of the system is discussed, and this includes the whole design process, various choices that had to be made and the architecture of the system. Chapter 4 describes how the system components were implemented and how they were integrated. Chapter 5 describes how the system was tested and validated. Chapter 6 presents possible future work that can be done to improve the system.

2. Research and related products

In this chapter we present products that have similar functionality with the system developed. The study of existing similar products provides a variety of ideas on how to approach the design of the system. Although a great variety of data acquisition systems exist for a variety of different needs, the following chapter focuses on embedded system based solutions that make use of either TCP/IP or the GSM network to transfer data.

2.1. Embedded Ethernet

One type of related products are embedded Ethernet applications. These applications interface directly into an Ethernet network and use the TCP/IP protocol to send data over it. In many cases HTTP, FTP or E-mail protocols are used over TCP/IP for easier access to the acquired data.

2.1.1.1.1. WebDAQ by SuperLogics

WebDAQ is a Multi-Function Networkable A/D, Digital I/O and D/A Device. Every webDAQ contains a complete web server built into the box. It can be connected it to an Ethernet network, or direct to an Ethernet card in a computer, and accessed by a web browser. The user has complete control over channels, rates and other acquisition parameters, a dynamic view of the acquired data, and direct download in a variety of formats.

32 separate channels can be sampled with 12 bit accuracy in up to 500 KHz total sampling rate (#ch * rate). This unit can also e-mail and offer access via FTP to the collected data, handle data conversion according to the type of sensor that was used and has 8 programmable analog outputs for circuit stimulus. It also supports multiple users and a command line interface for programming. The WebDAQ data acquisition system can use up to 128 MB of RAM in SIMM package and utilizes specialized DSP hardware. [Ref. 1]

2.1.1.1.2. PicoWeb Server

The PicoWeb ServerTM allows World Wide Web access to digital I/O and serial I/O signals without the need for assistance from external PCs or Unix computers. It is a stand-alone device with a real-time networking kernel, TCP/IP stack, and an HTTP Web server. Once plugged into a network with Internet access, using its twisted pair Ethernet connector (i.e., 10baseT RJ-45 plug), the PicoWeb can be accessed via a standard Web browser from anywhere in the TCP network. User designed web pages can be implemented to process "CGI like" devices. Web pages (i.e., HTML, text, images, etc.) can be retrieved in the normal manner from the PicoWeb's on-board flash memory.

The PicoWeb Server uses Atmel's AT90S8515 microcontroller, which offers an 8 MIPS low power RISC processor, 8 Kbytes of flash program memory, 512 bytes of EEPROM, 512 bytes of RAM, 32 I/O lines, and a built-in UART. The PicoWeb development board also includes a 32K-byte serial EEPROM chip, useful for holding Web pages and images. A Realtek RTL8019AS a single chip NE2000-compatible Ethernet controller chip provides the 10baseT network interface [Ref 2]

2.1.1.1.3. Axis network camera

The Axis 2100 is a network camera running TCP/IP with built in web server to be used with a web-browser. It is a camera connected directly to the network, a real web cam (not some pc connected camera) which has an integrated web server and uses normal Ethernet. The camera can be connected in three ways:

Directly to an Ethernet network running TCP/IP where the camera is a web server on its own.

In a dial-up mode, the user can attach an external modem to the serial port to have the camera dial-up to an ISP. The Axis 2100 can be set to deliver snapshots at certain intervals or triggered by external events, e.g. to send you an email in case of an alert.

In a dial-in mode the user also attaches a modem to the serial port but can then dial in to the camera using windows standard dial-up-networking to connect to the camera. Images are then monitored through your web-browser.

The Axis 2100 Network Camera is a network camera that attaches directly to a network providing live video with high quality images. In contrast to existing video servers it is using digital technology with Ethernet networks. The Internet style software used, based upon the TCP/IP protocol, makes it easy to manage. The camera can be monitored locally (intranet) or wherever desired in the world by taking advantage of the Internet. As it is based on open standards, the users can take and view pictures remotely over the network with a standard web browser, such as Netscape Navigator or Microsoft Internet Explorer.

2.2. GSM interfaces

Another product type used the GSM network to pass the acquired data through. The GSM modem was used to dial to a remote server, or to an ISP to facilitate acquisition.

2.2.1.1.1. M@xks Application Microserver

M@xks Application Microserver is a modular data acquisition system based on the CAN bus. The main module is connected with other M@xks sensors or with custom embedded systems via the CAN[1] bus. This line of products is also using an embedded TCP/IP protocol, but it runs over a dialup connection that can be run over a modem, be it an analog, ISDN or GSM modem. Over the modem connection a PPP protocol is run.

Once connected to the TCP/IP network the device can send data to a web server using the HTTP protocol (utilizing a perl script), the FTP or the SMTP mail protocol. [Ref 3]

2.2.1.1.2. Accord's Fleet Management Software

Accord's Fleet Management Software (AFMS) is a powerful yet cost effective web based solution for tracking vehicles in a fleet. It acquires location data from automobiles in real time to maintain the history of their movement and generates valuable information for the fleet management. AFMS helps the Fleet operators to track their vehicles' location coordinates accurately no matter where they are. AFMS facilitates help to reach vehicles immediately in the event of any emergency such as accident or breakdown. AFMS accomplishes these functions by integrating the GPS receivers (locating devices) and GSM cellular modems (communicating devices) with a user-friendly graphical user interface on a personal computer.

---

[1] Controller Area Network (CAN) is a serial bus system especially suited to interconnect smart devices to build smart systems or sub-systems.

The attributes of a Controller Area Network (CAN) are

* the multi-master capabilities that allow building smart and redundant systems without the need of a valuenerable master,
* the broadcast messaging that is the first piece of the gurantee for 100% data integrity as any device within the network uses the very same information,
* the sophisticated error detecting mechanism and the retransmission of faulty messages which is the second piece of the guarantee for 100% data integrity,
* the availability of more than 50 controllers from low-cost devices to high-end chips from more than 15 manufacturers,
* and the availability of CAN for the next 15 years as its use within the European automotive industry and the decision for CAN from the US and Japan automotive industry.

In Vehicle Equipment consists of NAV2300R GPS Receiver with embedded GSM Interface and any standard GSM modem compliant with the ETSI 07.07 and 07.05 standards. The GPS receiver determines the vehicle's position, speed and direction by tracking the NAVSTAR GPS satellites. The GSM modem is used to report the vehicle's location coordinates to the Nerve Centre.

Control Station has two software components; GSM Driver and Fleet Server. GSM Driver interacts with Mobile Unit and provides vehicle data to the Fleet Server. Fleet Server maintains vehicle data in MS-Access database. This data is served to web-server components when triggered by AFMS users.

AFMS provides web-based interface to vehicle data. This interface displays vehicle data and helps users interact with Control Station. The web-components are written in Java Server Pages (JSP) and Java Beans. These web-components generate HTML-based information to the browsers. AFMS can be browsed on Internet Explorer 4.0+. [Ref 4]

2.2.1.1.3. Dragonix VZ

The Dragonix VZ board is an embedded linux solution maintained as "open hardware". It uses a Motorola Dragonball microprocessor (68VZ328) to run the linux operating system and has an Ethernet interface. [Ref 5] It can interface with a universal wireless communication board that includes an onboard Siemens MC35 GSM module. Combined with the Ethernet controller on the Dragonix mainboard, it is possible to use the two board set as a Gateway.

2.3. Analysis

All of the analyzed systems had some of the characteristics that were required of the developed system, but none met all of them.All systems that have a TCP/IP acquisition option run the server in the embedded system. This results in either bad performance (especially when a lot of clients are connected) or a high-end embedded system, more expensive and certainly not with better performance than a standard PC. Some of these systems are implemented for specific applications, such as the Axis Camera or Accord's Fleet Management Software. Of all products reviewed, the M@xks microserver came closer to being a match for the requirements, by featuring both a platform independent web interface and out-of-the-box GSM connectivity.

	Connectivity via GSM	Internet/Web interface	Generic (not for	Platform Independent	Low cost

			single application)		
WebDAQ		?	?	?	
PicoWeb		?	?	?	?
Axis Camera	?	?		?	
M@xks server	?	?	?	?	
Accord's Fleet Management	?	?			
Dragonix VZ		?	?	?	

3. System Architecture

3.1. Introduction

This chapter describes the design of the data acquisition system. It describes not only how the system was structured and designed, but also what technologies were used and the reasons behind the choices that led to the use of these technologies. Also the reasons for adopting the architecture that was used and the benefits this architecture bestows on the system are discussed.

3.2. Requirements

Given the fact that not much is known for the exact type of data transferred through the system (meteorological data, biometric data, etc). The system should be able to accommodate a single stream of data or many scarcely sampled variables. It should also be able to maintain the data acquired by multiple data acquisition units, while serving multiple requests over the web.

3.3. Design choices

The design choices that are discussed here were influenced by many factors. They are presented in the context of technical issues, as well as availability, cost, and suitability to this project.

3.3.1. GSM equipment

In the choice of a mobile phone to use with the system, various compatibility, protocol openness and availability issues came into play. Firstly, not all mobile phones are designed for data transfer, so the one selected had to be one equipped with a GSM modem. The second requirement was a RS-232 interface. The popularity of data enabled mobile phones with laptops and other portable devices has led many companies to adopt an infrared port as a way to allow communication with a PC. There exist, however, models that use external cables to communicate via a serial port.

In theory any mobile phone with a built-in GSM modem and a serial connection option could be set up to communicate, at least on a serial link level, with an embedded system featuring a RS-232 interface, by configuring the microcontroller to behave like a PC would. Most mobile phones however utilize propertary protocols to run the standard modem AT command protocol over. These models were unsuitable because technical information about the internal protocol used to communicate between the phone and the driver was unavailable (in fact denied) by the manufacturing company.[2]

The answer came in the usage of a microcontroller-driven serial cable that comes with some mobile phones to provide data connectivity. This type of peripheral enables the mobile phone to appear as a standard modem to the PC. Thus we can program a microcontroller to mimic the PC's behavior without knowledge of any specific proprietary protocol.

3.3.2. Simple Dialup Vs embedded TCP/IP

A system for over-the-internet data delivery from an embedded system could be designed by attempting to give access to both the embedded system and the client side software on the same TCP/IP network. This technique is used on all WAP enabled mobile phones, but with the embedded system (the mobile phone) as a client. The same method can be used, but with an http server set up to run on the microcontroller.

Although possible, this technique has a number of drawbacks, all of which concern the microcontroller's limited resources. Actually this technique would be ideal for a similar system with a normal desktop computer in place of the embedded system.

The protocols that make connection to the internet possible were created with standard computers in mind. Indeed, to access a web client through the internet, a computer system should run no less than four distinct protocols. These are the modem's AT command protocol (that results in a transparent link after initial connection, and therefore requires no memory, nor does it increase the workload of the system), a PPP or SLIP protocol, the TCP/IP protocol and the HTTP protocol. The protocols mentioned above run one on top of the other, with HTTP being top level and running over TCP/IP, which in turn runs over PPP (or SLIP), that run over the serial link provided by the connected modem.

This set of protocols would have both workload and memory requirements far exceeding those of the microcontroller used (and any in the low cost range, requiring

[2] The gnokii project is a project to achieve connectivity between Nokia phones and the popular open source Linux operating system. Nokia has been contacted many times by members of the open source community for information on the inner workings of their protocols and has announced its unwillingness to make any information available. The project continues by reverse engineering.

high power embedded processors to be implemented). It was therefore necessary to use a simple protocol over the serial link of the mobile phone to transfer the data to a gateway computer. This gateway will in turn access the internet and send the data through.

3.3.3. Data storage location

Another choice that had to be made was where in the system to store the data that was acquired. Storage in the embedded system and dynamic acquisition upon request, while theoretically possible, would obviously have been a poor choice because of limited resources, both in terms of memory and bandwidth (especially having to serve each requesting user separately). It would also require the embedded system to be on-line for as long as the system would operate, rather than connecting only to update the content, leading to higher application costs. Streaming the data to the clients and storing it there is also not the best choice. Apart from each user having to be on-line when data is transmitted or miss a part of the stream, we would have to solve several consistency and data redundancy problems, ensuring that the same data resides an all clients.

The wisest choice would be to store the data in the gateway PC where resources are plenty and data storage can be centralized. Data should be stored in a way that would allow access by many clients and update by the data source simultaneously. A database was chosen to store the data primarily because of its built-in resolution of these consistency problems. A database is also easily accessible from most modern programming languages and from scripting languages used in server side scripting. This ease in connectivity even permits the parallel use of another data delivery system, for example a stand alone client written in any programming language supporting a database access protocol (ODBC is supported by almost every modern language) retrieving data without affecting the rest of the system. Finally a database eases the implementation of security and access control should it become necessary.

3.3.4. Over-Network delivery

Having decided to use a gateway computer gave a wider range of choices on the technologies that could be used to transfer the data to the client software. Both the gateway PC and the client software will be on the same TCP/IP network and so the

TCP/IP protocol could be used as a basis for the data transfer process. Either TCP/IP could be employed to transfer the data or another protocol could be employed over TCP/IP to be used in data transfer.

The main aspect of the system that influenced this choice was system and platform independence. The client part of the system was to be deployable on the broadest range of operating systems and platforms possible. To fail that requirement would be to exclude diverse TCP/IP network resources (namely the internet). Therefore the main concern in this choice was to insure that the client portion of the system is independent of the platform and version of the operating system used.

3.3.4.1. Architectural Alternatives and Tradeoffs

The first solution would involve client software specifically written to receive data through an open TCP/IP socket and display the results. This client could be implemented in three ways: using a stand alone application, or using Java or ActiveX technology, none of which offers true platform independence.

3.3.4.1.1. a) Standalone application

A standalone application would be by definition platform specific. Standard executable programs cannot be run on a different operating system than the one they were designed to be used with. There are times in witch even the same operating system will require different compilations of a program depending on the hardware. Noted cases are non Intel (alpha) processors running windows environments, windows CE devices and executables using specific instruction sets like MMX, SSE and 3DNOW.

Variations on the standalone application scenario include use of either ActiveX control technology or of Java. Both these technologies can be used to create applets, client programs that load automatically from a web server and appear to run within the web browser, giving the illusion of a web interface, but are otherwise client programs running in the client's computer and system architecture would be no different from the stand-alone client scenario.

3.3.4.1.2. b) ActiveX technology

ActiveX technology uses normally running compiled binaries that run from within the web browser, thus having all the drawbacks of a standalone executable client. ActiveX controls require use of the Microsoft Internet Explorer web browser to operate, being still more restrictive platform-wise.

3.3.4.1.3. c) Java Technology

Java technology appeared as a promising choice for a cross-platform implementation of a client. Either a stand alone client or an applet seemingly running on a web browser could be written using Java and it would run on any platform with the same Java Virtual Machine (JVM) it was written for and even across some different versions of JVM's. However, even being designed for platform independence Java technology still cannot guarantee it. Microsoft's implementation of the Java Virtual Machine is quite different from Sun's and there are compatibility issues between them. So even if there is no platform compatibility problem, on Windows OS there is the issue of JVM compatibility (Windows may run either version of the JVM). Recently, after Sun Microsystems' legal action against Microsoft the latter's implementation of the JVM, and the deployment of Microsoft's competitive technology (.NET), Microsoft announced its intended discontinuation of Java support in the Windows operating systems. This means that continued compliance of Windows (especially newer versions of Windows like XP) cannot be guaranteed.

3.3.4.2.

Any of these designs would also have problems using an available TCP/IP connection because of the widespread use of firewalls. Inability to use some internet application because of the TCP/IP port it is using is closed by a firewall is a commonplace scenario, made worst by the frequent inability of the end user to alter firewall settings. Using a standard TCP/IP port that is commonly used and therefore routinely left open by firewalls would minimize problems of this type.

3.3.4.3. Choice of network technology

In the end, a true web interface was chosen. Dynamically created HTML content is delivered over a standard HTTP connection to a web browser. The web server uses a server side script to access the stored data according to a request sent to him and render the response content. Server side scripting generated web content is the most widely used design for dynamic web content delivery today, and server side scripting technology is constantly researched to allow web developers more power still. Choosing to use server side scripting to deliver the data was probably the most important design choice, and despite its unique advantages, it also has certain limitations.

3.3.4.3.1. Advantages of Server side scripting

The main advantage of server side scripting is platform independence. It is possible to generate HTML content that does not contain browser-specific tags or client-side scripts, allowing almost all browsers in most operating systems to view the content, albeit with minor visual differences. The breadth of platforms on which clients can be deployed is maximized by the use of this technology. Also, most firewalls allow web access by default, so no network configuration is required to allow the operation of the system. This extends to the lack of system component installation on the client computer in most modern operating systems. Virtually any post-1998 operating system in widespread use comes with a pre-installed web client, and in those that do not, the possibility of the user having installed a web browser is high. Thus this system is easily usable by a novice user without technical knowledge, or from a public computer not specifically set up for use with the system.

An added advantage is compliance to standard web design practices. Integrating this system to a commercial web host's existing web page would be quite a trivial task. Furthermore most commercial web hosts dislike binary programs running on their web servers, as they pose a serious security hazard. This system is based on technology already in widespread use on web hosts, designed with security in mind, and therefore supported by virtually all commercial web hosts.

3.3.4.3.2. Disadvantages of Server side scripting

Use of server-side scripting also has downsides. HTML content lacks the ability to update text or graphics without a new request for the entire page. Therefore on each data request a snapshot of the data that is in the system's database is shown. Consequent refreshes of the web page will update the shown content.

Also, because the visual representation of the data is shown, bandwidth use can be higher than it would have been had the numeric values of the data been sent.

3.4. Architecture and organization of the system

There are seven distinct components up the mobile data acquisition system: The microcontroller, the GSM equipment (mobile phone), the gateway modem, the database gateway software, the database itself, the web server, and one or more web clients. The microcontroller is responsible for acquiring the data from sensors, or other data sources. The mobile device is used to connect to the gateway modem. Once connected to the latter it permits the flow of information between the microcontroller and the gateway software. The getaway software is responsible for interpreting the data send to it by the microcontroller and storing it in the database. Data stored in the database can be accessed by the web server and is used for generating the web pages to be sent at the web clients according to their requests.

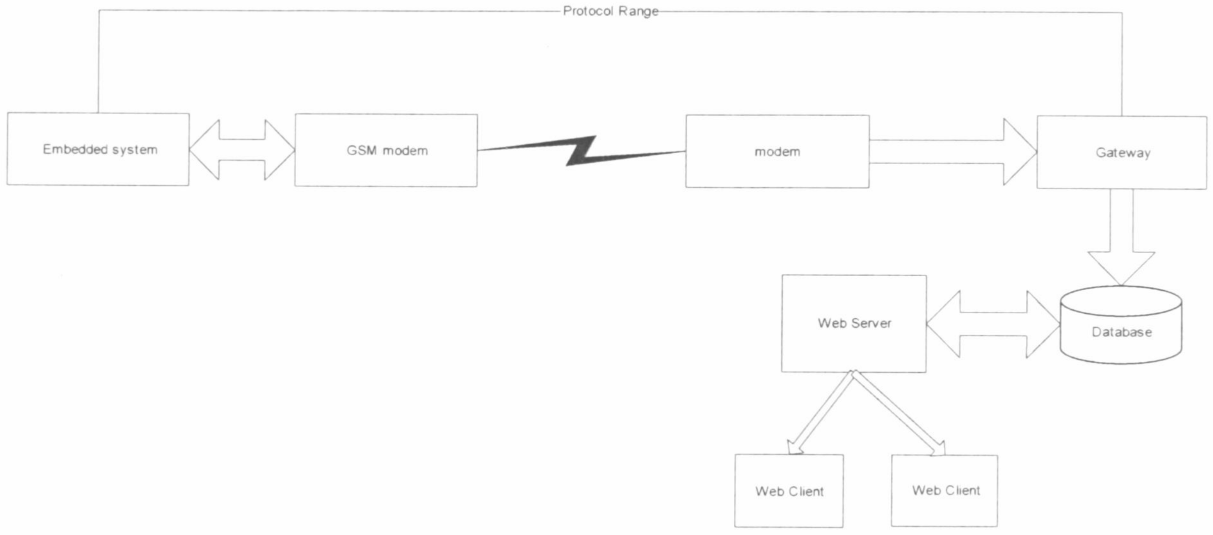


Figure 1 : System architecture

3.4.1. Microcontroller

This component acquires information by a routine that is specific to the type of data source used and stores it in its internal memory. The maximum size of the data block that can be stored into the microcontroller memory to be sent in one data acquiring transaction with the gateway software (one data packet) is 255 bytes. (This size is hereafter referred to as "packet payload size")

The existing RS232 compatible port and baud generator of the microcontroller was used as basis for the serial interface. Hardware flow control was necessary for interfacing with the mobile equipment's modem and was implemented in software.

The microcontroller performs any data acquisition from the data source. The use of internal memory as an intermediate buffer is necessary in case of a preemptive data source. In case of a client-behaved data source that provides data on request, no intermediate buffering is needed.The microcontroller runs one end of the data transfer protocol once a transparent link is established through the GSM device.

3.4.2. GSM equipment

The GSM equipment is essentially a serially driven data enabled mobile phone with an openly available serial (RS232) protocol. It connects to a normal modem on the server computer and thus provides a transparent serial link between the microcontroller and the gateway software.

3.4.3. Database Gateway software

This piece of software is responsible for the dial-up connection on the modem's side. Once a connection is achieved it uses the modem to run the other end of the data transfer protocol. The gateway parses the packets and stores the data in the database.

3.4.4. Database Server

This is an ODBC supporting database. The database gateway uses it to store the data. The web server scripts also use it to access the stored data and generate the web content.

3.4.5. Web Server

The web server accepts requests through an http connection from the web clients and generates the appropriate web content dynamically, using the data stored in the database. It then returns the response content through the http connection.

The web server scripts also utilize an additional component of Microsoft Office, the Office Web Components, to render the graphs.

3.4.6. Web Clients

These can be any frame supporting web client on the same IP network as the web server. It is used to access the web server content as well as the dynamically created content of any data request.

3.5. Subsystem Connectivity

The microcontroller and the GSM equipment are typically bundled with the data acquisition subsystem (typically a sensor or some type of memory) on site where the data is to be sampled from. The gateway software should run on a PC with an analog modem connected to a phone line. This PC should be on the same IP network with the PC's running the SQL server, the web server and the web clients. The web clients are typically on a different PC on the same IP network (most commonly the internet, but the system can be set up on an intranet as well). In a typical configuration however the web server, database server and the gateway program can run on the same computer to avoid network-related performance hits. Another popular configuration would be for the PC with the modem to run the gateway program and the web and database servers on another. This would be the proposed way of setting up the system while using a commercial web server, as security reasons normally disallow the execution of custom programs on the web server. The use of ODBC access to the database server however allows enough flexibility for the system to be distributed along a network and accommodate these types of limitations.

3.6. Subsystem Interfaces

3.6.1. Communication between Microcontroller and GSM equipment

The GSM equipment chosen supported connection with a serial port. A PC's serial port is based on the RS-232 serial communication protocol supported by the AVR microcontroller which we used, but it also uses a number of control signals. In order for the serial connection to work we had to implement the output signals for the mobile phone to detect. Also, we had to use some sort of flow control to prevent data loss at the send buffer of the mobile phone should it overflow by the microcontroller continuously sending data faster than the phone can send through the GSM network

PCs have 9 pin male SUB-D connectors. The pin layout is as follows (seen from outside your PC):

Not all RS232 signals are typically used, therefore we need to examine what signals are needed in the context of this system.

RxD, TxD

These lines carry the data and were connected through the level shifter to the respective lines of the mobile phone.

RTS, CTS

These are used by the PC and the modem to start/stop a communication. The PC sets RTS to 'high', and the data set responds with CTS 'high'. If the data set wants to stop/interrupt the communication (eg. imminent buffer overflow), it drops CTS to 'low'; the PC uses RTS to control the data flow. CTS was routed through the level shifter and to an input pin on the microcontroller. Before each byte is sent, the microcontroller checks the pin's status, ensuring that no overflow will occur.

DTR, DSR

These lines are used to establish a connection at the very beginning, ie. the PC and the data set perform handshake in order to assure that they are both present. The PC sets DTR to 'high', and the data set answers with DSR 'high'. Modems often indicate hang-up by resetting DSR to 'low' (and sometimes are hung up by dropping DTR).

(These six lines plus GND are often referred to as '7 wire'-connection or 'hand shake'-connection.)

DCD

The modem uses this line to indicate that it has detected the carrier of the modem on the other side of the phone line. The signal was not used and was left unconnected; the carrier is instead detected by a message through the serial port upon connection.

RI

The modem uses this line to signal that 'the phone rings' . The signal was not used and was left unconnected; instead rings could be detected by a message through the serial port upon ring, though no ring detection was used in this system, as the microcontroller was the one to dial the PC.

The 'signal ground', ie. the reference level for all signals. It was connected with the microcontroller's ground.

Protective ground

This line can be connected to the power ground of the serial adapter. It was left unconnected, as the microcontroller power supply had a signal ground but lacked earth ground. .

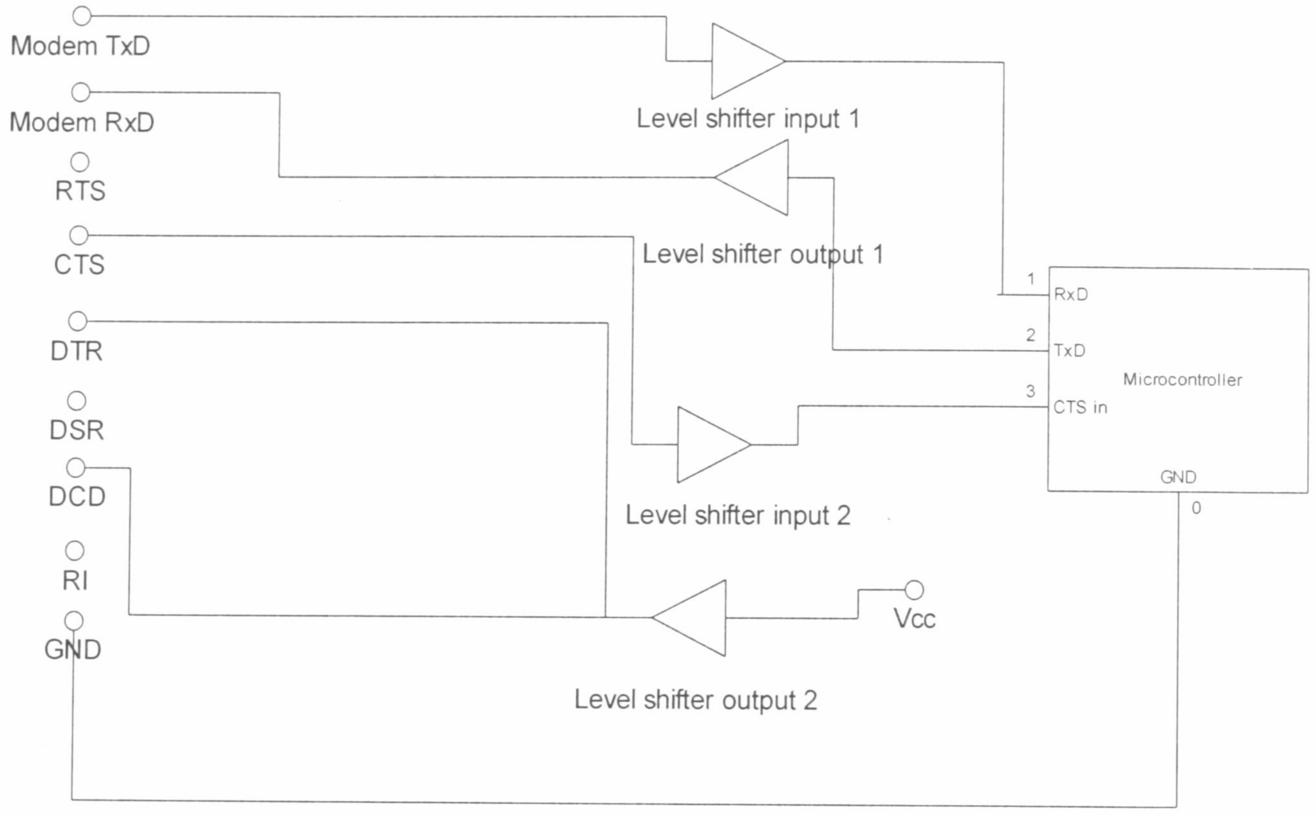


Figure 2 : Connection of the Mobile phone on the Microcontroller

The most important lines (the ones used in a typical Microcontroller to PC connection) are RxD, TxD, and GND. Others are used with modems, printers and plotters to indicate internal states.

Once connected on a hardware level, AT commands may be used to control the modem until a connection is achieved. The ATV0 command is be used to set the modem so numerical responses are used to the AT commands. (0 instead of ok or an error code instead of a descriptive error message) more easily parsed and recognised by the microcontroller. The ATE0 command disables command echo. ATX0 command limits the return code range for easier parsing (that is, causes the modem to return a generic "connected" response upon success or an "error" on failure, rather than a more descriptive, but unnecessary and harder to parse message). The AT&K3 sets hardware flow control. All these settings may be initialised by the microcontroller at start-up, but can be omitted if the mobile phone is pre-set with those, as they are persistent. The ATDT command is used to dial the gateway PC's modem. After the gateway answers and the modem connects (the 0 character signifies successful connection) the modem creates a transparent serial link that appears as a cable based serial connection to both the microcontroller and the gateway software. Since all modern modems and the mobile phone support the MNP5 error correcting protocol, data sent through this connection are protected against transmission errors, so long as they occur between the two modems. The error protection scheme used in the serial communication protocol was implemented mainly to protect against errors occurring between the microcontroller and the mobile phone's serial interface.

3.6.2. Communication between Gateway software and Modem

A modem whose drivers provide a COM port interface must be used with the gateway software. All serial modems can be accessed this way and the vast majority of internal and USB modems offer a virtual com port interface to the system to facilitate use with software. The ATE0 command should be used in the init string to disable command echo. ATX0 command limits the return code range for easier parsing. The ATS0=1 at command is used to configure the modem to answer automatically after one ring (in the present implementation a dedicated phone line was assumed, although the implementation of an interactive pick-up mode is trivial).

3.6.3. Communication between Gateway software and Microcontroller

3.6.3.1. Link Transparency

Once the mobile phone's modem dials the gateway computer and the modem on the latter picks up, and upon successful handshaking both modems signal a successful connection. After that, the link between the microcontroller and the PC is a transparent serial link, until an escape sequence is sent through. An escape sequence consists of a pause of at least one second, and sending the modem's escape character (character 43 in the ascii table or '+' by default but configurable) within a one second time period. After that the modem disconnects and can be hung up by the ATH AT command. The other modem disconnects after some time due to carrier loss automatically (unless expressly set not to). Although the occurrence of an escape sequence at random in the transferred data is improbable, steps have been taken in the protocol design to prevent it.

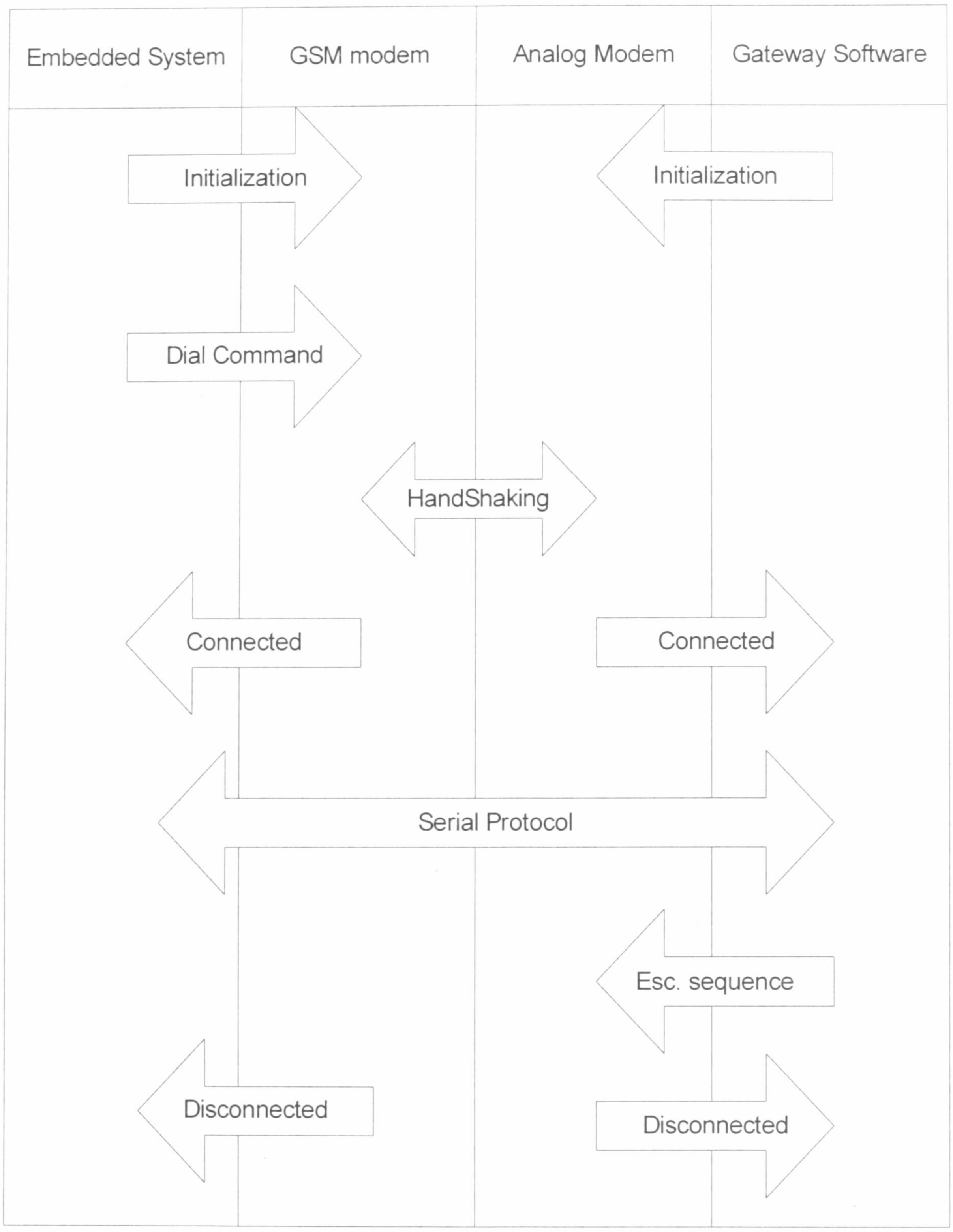


Figure 3 : Connection Use case

3.6.3.2. Data acquisition protocol

3.6.3.2.1. Protocol Requirements

A simple protocol was designed to control the connection and encapsulate the transferred data. Although many protocols exist to transfer data over a serial line (zmodem, xmodem, Kermit) connecting two computers, the presence of an embedded system and its limited resources, mainly in memory, combined with the potential for "on the fly" data transmission from it, as opposed to the file transfer use of the other protocols disallowed their use. This new protocol needed to be simple in its implementation and light on necessary resources, especially on the side implemented in the microcontroller. Lastly, since more than one system may dial the gateway, a one byte ID (referred to as device ID) was used to identify a connecting server.

3.6.3.2.2. Protocol design

The protocol was very loosely based on the way xmodem works but heavy modification was needed to facilitate the requirements of the present system. The protocol is a client server protocol with the gateway software at the client side. This enables timeouts to be handled on the client (i.e. PC software) side, so that the timeouts won't use one of the microcontroller's few timers/counters. A command to the server consists of a single byte, and it responds accordingly. Commands include ID, ALIVE, REQUEST, NEXTREQUEST, DISCONNECT and EXTENDED (the actual op-codes of these commands are discussed below). Other one-byte responses from the server are ACK (acknowledgement), NAK (no acknowledgement) and DATA (data packet header). The server responds to these commands accordingly, but reception of any other character results in a NAK response. The use of each command is as follows:

ID

The ID command identifies the particular embedded system to the gateway software. Following the reception of the ID command the server responds by issuing either the ACK or NAK response followed by the device ID. An ACK response indicates that the device is ready to send data while a NAK response signifies that data is unavailable. This is useful in case the data acquisition subsystem has to be initialized or has not received data. In case of a memory request the NAK request may signify as an empty memory.

ALIVE

The ALIVE command is used to verify proper function of the remote embedded system. It always returns a single byte as a result: ACK.

REQUEST

This command is used to perform the actual data transfer and causes either a data packet or a NAK response to be sent from the server. A data packet begins with the DATA byte header and continues to transmit a number of payload bytes equal to the packet size (packet size is discussed later). A single checksum byte follows at the end of the packet, calculated as the 8 least significant bits of the sum of all payload bytes. Care must be taken if the packet is not stored in memory but generated dynamically on request. It may be possible for the data source subsystem to include delays. However the transmission of a single packet should never span over more than a second for fear of the data forming an escape sequence and disconnecting the modem. No assumption was made as to the stability of this value, and it can change between packet transfers. This however would have to be a gateway initiated action through a "custom command" as the gateway has to receive a predefined number of payload bytes to successfully complete a data packet, and therefore must "know" the exact packet payload size each time a packet is sent.

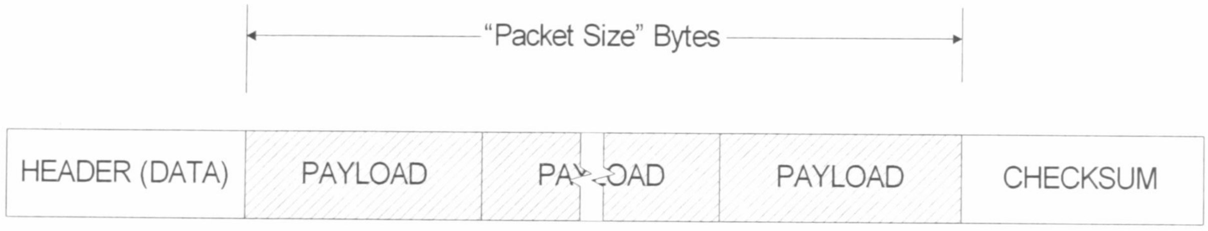


Figure 4: Data packet structure

NEXTREQUEST

This command is used to facilitate transfer of large data blocks stored in memory. Its use is identical to the REQUEST command, except for the fact that it implies successful delivery of the previous packet. So if the contents of a memory attached to the microcontroller were to be transferred, consecutive NEXTREQUEST commands would transfer sequentially all the contents of the memory, while if a checksum error occurred the next command issued by the gateway would be a REQUEST command to retransmit the last packet transmitted. In a per-request polling of the data source where the resulting data isn't stored between requests but is instead sent immediately to the gateway, this command should be implemented to act exactly as the REQUEST command.

DISCONNECT

This command is used to signify the end of the data acquisition session and is issued prior to an escape sequence from the gateway software. The server can either not respond at all, waiting for the disconnection from the client side or issue an escape sequence to disconnect his side of the phone line (requires timer/counter use due to long waiting periods associated with escape sequence) . In any case it should ignore all following data from the serial port (until its next dial) as the escape sequence would pass through to it.

EXTENDED

Since this protocol is designed for a multitude of applications, the need for expanding the protocol for an application specific command is obvious. Changing the sampling rate or target channel of an A/D converter, addressing a memory for the next data request to fetch data from, reading the status of a secondary sensor, controlling subsystems connected to the microcontroller and triggering some operation are only some of the uses that might demand expandability of the protocol. The EXTENDED command offers a way for these application-specific operations to be implemented. After an EXTENDED command, other commands can be implemented using the same op-codes or different ones to cover application-specific needs. To check the status of a slave microcontroller attached to the main microcontroller for example, the EXTENDED command could be followed by an ALIVE command. This sequence would cause the main microcontroller to access its slave to ascertain if it is functioning properly and then return an ACK or NAK code in response.

If no such custom commands are implemented, invoking the EXTENDED command should result in a NAK response. The assumptions made about the protocol used inside the EXTENDED "sub-protocol" are:

1. At some point the EXTENDED session ends and the server exits back into parsing the protocol's original commands.
2. The sub-protocol used inside the EXTENDED session cannot send randomly occurring escape sequences. This can be assured either by limiting the timeframe of each session or by packing the transferred data into limited timeframe packets with a header not beginning with the modem's escape character.

3.6.3.2.3. Packet Size

The packet size may need to be different for each application, especially if one wishes to use the packet to transport bundled sets of data, for example data from different channels of an A/D converter taken (almost) simultaneously. The packet size can be set anywhere from a single byte to 256 bytes. While a static packet size, unchanging throughout the session would be enough for most applications, a variable packet size could be implemented through the use of an EXTENDED command. The negotiation of the new packet size value remains a matter for the application specific command.

3.6.3.2.4. Command op-codes

The actual byte values representing each command can be almost arbitrarily selected. The only real restriction is that the DATA packet header and the values for the ACK and NAK responses should not be the same as the modem's escape character. This is necessary in order to exclude an escape sequence from occurring, as there might well be a long pause before a packet transaction takes place and an escape character as an opcode in conjunction with data that may well contain the escape character might constitute en escape sequence.

In order to maximize the robustness of the serial link, the selection of the opcodes can follow certain rules. First if all the 0x00 and 0xFF values are most commonly the result of errors, being composed fully from 1s or 0s (especially 0xFF is very common). Second, opcodes that differ in a single bit from each other are also a bad practice, as a single mistransmitted bit could result in a different operation taking place. The small number of commands allows us to construct opcodes by arbitrarily selecting the first halfbyte and inverting it to form the second. This gives us 16 distinct command opcodes, more than enough to accommodate the protocol's needs.

Protocol element	*Byte opcode*
ACK	0x1E
NAK	0x2D
ALIVE	0x3C
DATA	0x4B
REQUEST	0x5A
NEXTREQUEST	0x69
DISCONNECT	0x78
EXTENDED	0x87

The above table indicates the generated op codes that were used. (Note: during most phases of development, a different set of opcodes corresponding to ASCII codes were used to facilitate debugging)

3.6.3.2.5. Error rates

While no errors were observed in actual system operation, a checksum was used in case noise on the serial port caused data loss. A simulation of how the protocol reacts over a noisy line was used to calculate how error rates will behave in various packet sizes. The simulation program was written in C. Every randomly generated byte is XORed with a bit mask, generated based on the selected bit error rate (1 denoting a bit hit by noise). Two checksums are generated, one on the original bits and one on the bits that have been subjected to the noise bit mask. The "noisy" checksum is then subjected to the noise bit mask itself (as the transmitted checksum is also subject to noise on the actual protocol) and the two checksums are compared. Track is also kept on all packets that pass the checksum comparison but have been subjected to errors.

The bit error rate used was $5 \cdot 10^{-5}$ and the simulation was run for 10000000 packets. The following graphs illustrate the checksum mismatch errors that occurred, the errors that occurred without generating a checksum mismatch error, and the bandwidth used for actual payload (given by the formula $\frac{packetsize}{packetsize+2}$ as the overhead for the protocol is two bytes, the request and the checksum byte)

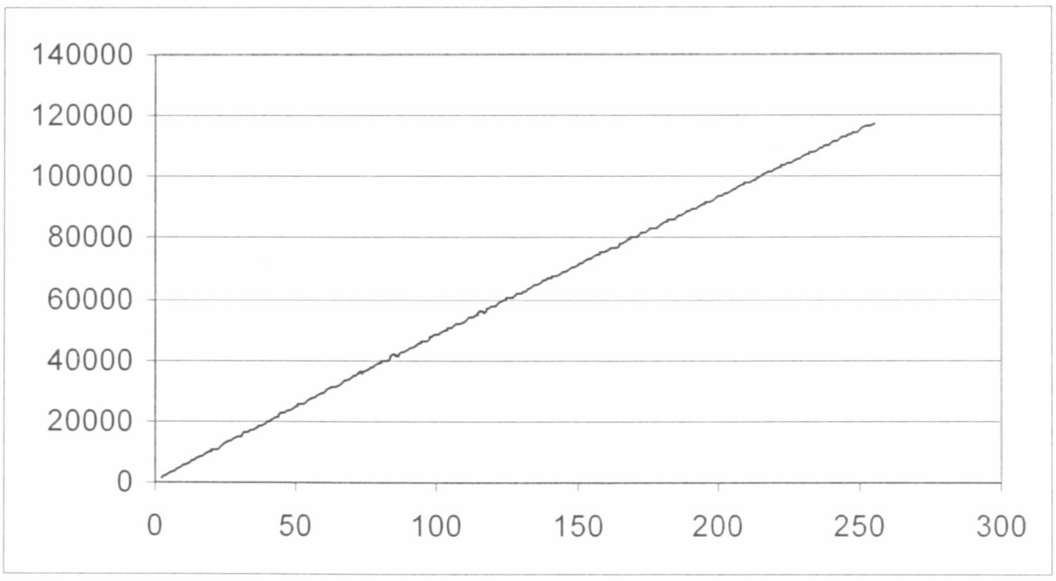


Figure 5: Checksum mismatch errors per packet size

The error rate increases linearly with the packet size, as the probability for an error in the packet increases as it gets bigger.

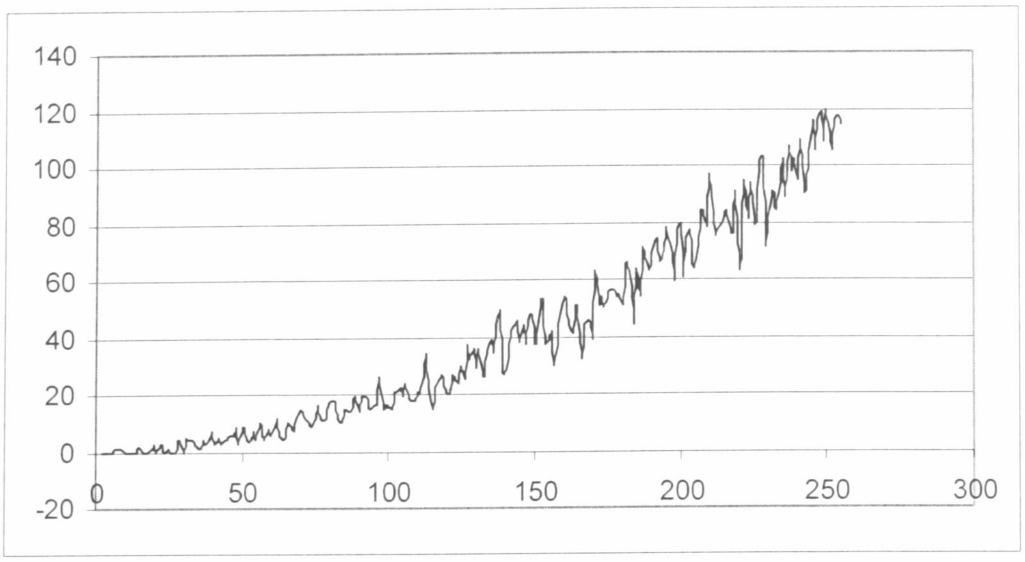


Figure 6: Data errors per packet size

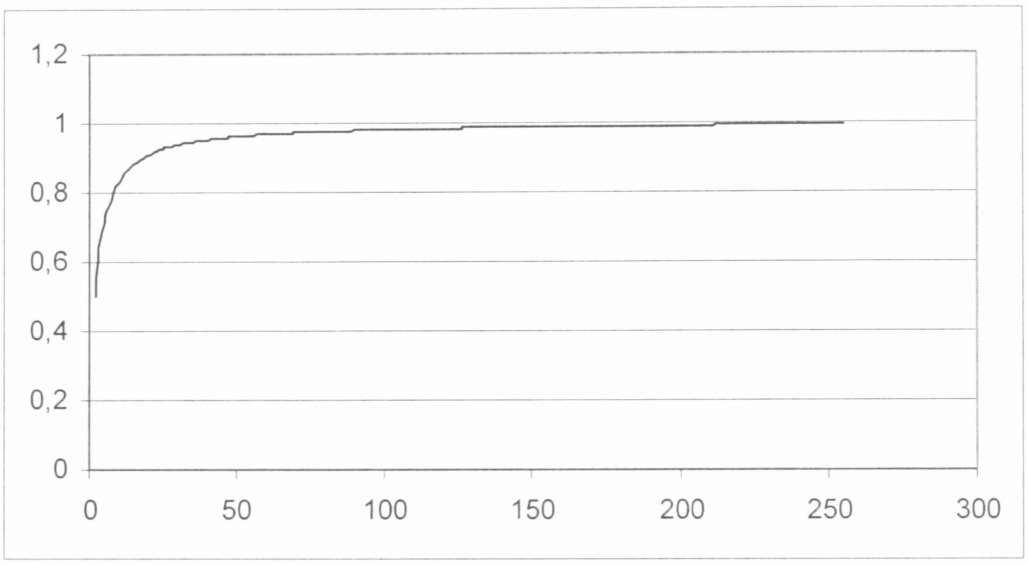


Figure 7: Bandwidth percentage used for payload.

Complete data on the graphs can be found in Appendix 1

3.6.3.2.6. Analysis of error rates

The choice of packet size relies heavily on the type of data that is being transferred and the type of application that is used. In case of transmitting values from multiple sensors that are sampled simultaneously the packet size may have a minimum size. For example if sampling a three axis accelerometer, grouping the tree values might lead to a minimum packet size of six bytes. Also the acceptable error rate depends on the type of data that is being transferred. And of course, depending on the type of sensor used large packet sizes might require more of the microcontroller's resources, such as memory.

With that in mind, a packet size value can be estimated for a bandwidth-intensive application that uses little of the microcontroller's resources in operations other than running the protocol. The estimation is done assuming that the microcontroller has the necessary resources. It is also assumed that the sensors used can produce data with the desired rate. Data stored in an external memory for example can be used with high packet sizes, while for a real time sampling sensor with low enough sample rate it makes no sense to wait for enough data to be sampled before sending them. Considering all that, one can estimate the optimal value at around 50 bytes. Showing near-maximum bandwidth usage and low error rates, it is plain that the optimal packet size is around that value.

3.6.4. Communication between Gateway software and Database

3.6.4.1. Reasons behind database use

Since it was necessary for the gateway software to store the received data, a method for storing it was necessary. In choosing between writing a persistent data structure to store the data in and using a database, the latter option showed a multitude of advantages.

- Consistency: A database could be accessed simultaneously, by many web clients retrieving data while at the same time being updated by one or more instances of the gateway software.
- Connectivity: Almost all modern programming languages offer database connectivity, and the use of database connectivity interfaces (ODBC, DAO) provide an abstraction layer that can be used across platforms and networks. This extends to connectivity with the web server, as all web scripting languages today support native database connectivity and are typically used to access databases.
- Atomicity: The transaction architecture of a database can be used to achieve atomicity of the updates alternatively to a progressive, more real-time update.

For all these reasons, a database was adopted to facilitate data storage.

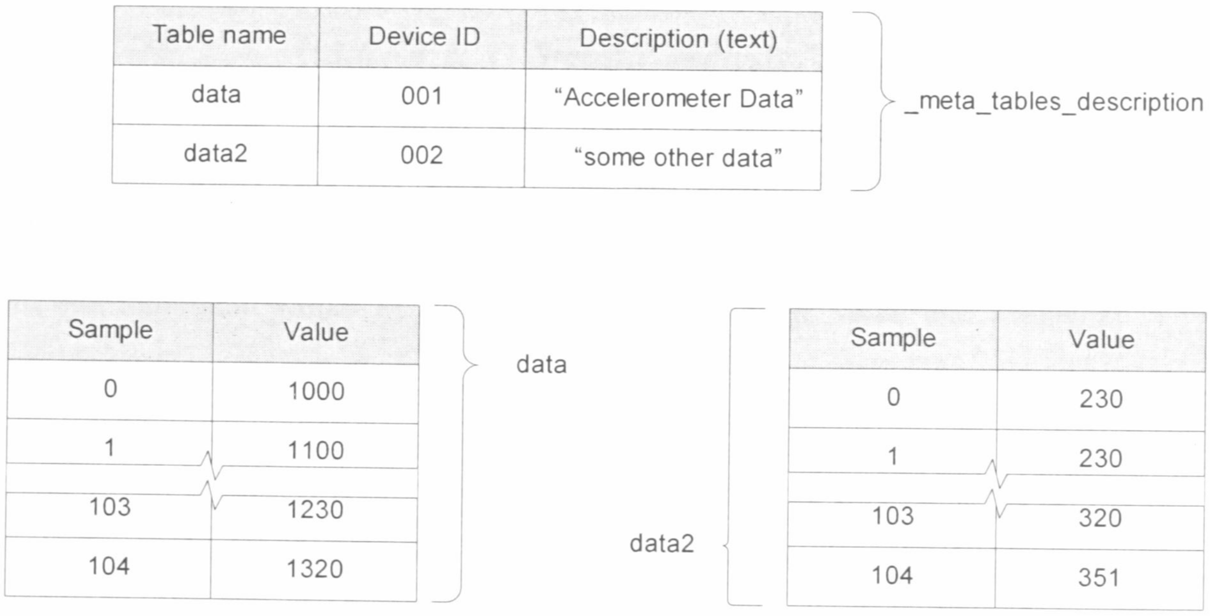
3.6.4.2. Parsing and storing Data

Once a packet is received by the gateway software, its contents must be parsed and a number of data values should be extracted. These values are then stored on the database, commonly appended on a table. The gateway software contains an application specific parse routine that converts the received data to actual values and manages their storage.

3.6.4.3. Database Schema

The data to be stored was simple enough and required no special or complex schema to be designed. Any table with two fields named "sample" and "value", with the "sample" field being the primary key can be used.

As each microcontroller device contains an identification number, the database can have several tables, and each device can be assigned to a different table according to its ID number. The special _META_tables_description table in the database contains information about what tables in the database are active (i.e. accessible from the gateway) along with a short description and a Device ID that can be used to match a data source to the appropriate data table on the database.



_meta_tables_description

Table name	Device ID	Description (text)
data	001	"Accelerometer Data"
data2	002	"some other data"

data

Sample	Value
0	1000
1	1100
103	1230
104	1320

data2

Sample	Value
0	230
1	230
103	320
104	351

Figure 8: Sample database Use case

3.6.5. Communication between Web Server and Database

3.6.5.1. Server side script programming interface

One of the reasons for choosing to use a database was its ease of connectivity with virtually all server side scripting languages. After each request for data, the web page's script queries the database and fetches the appropriate data from the table it is stored in. It then generates the dynamic part of the web content to be sent to the web client.

3.6.5.1.1. Graphical library use

If so requested by the user the script can, call a graphic library to render an image of a graph representing the data, rather than generate a text page with the data. Not all scripting languages support such graphical tools, however. The benefits of such a representation however are apparent, especially when the data samples are too numerous to be red one by one.

3.6.5.1.2. Statistic analysis

The use of a database can also prove convenient when the user requires some statistical analysis of the data and not all its values. Although different databases support a different range of statistical analysis functions, most support finding the minimum and maximum values of a data table, or its average value and standard deviation.

In this chapter the whole process of designing has been described. Not only has it been shown how the subsystems were set to interact with each other and compose the whole system, but also why this particular architecture was chosen and how the design was derived from the requirements of the system.

4. System Implementation

4.1. Choice of Application

The application which was chosen to utilize the remote data delivery system was a solid state accelerometer sensor for biometric data. The accelerometer is sampled on data demand and a two byte value is transferred each time and appended to the rest of the samples. A list of the samples stored or a line graph of the value for each sample is shown upon request on the web interface. The data from the accelerometer is not processed in any way.

4.2. Data Source system

To provide a data source for the data acquisition system a pre-made board with an ADXL210 two-axis solid state accelerometer was used. Only one of its axes was captured. The axes' output pin relays the acceleration measurement using pulse width modulation. A square pulse with a period of 469 Hz that remains constant is output and the duty cycle of the pulse relays the acceleration value. Since the period of the pulse is stable, the acceleration value can be represented by the time the output signal stays high on each period. The output signal is connected to both external interrupt pins of the microcontroller; each transition of the output signal causes an external interrupt on the microcontroller, with one interrupt triggering on the rising and one on the falling edge of the signal.

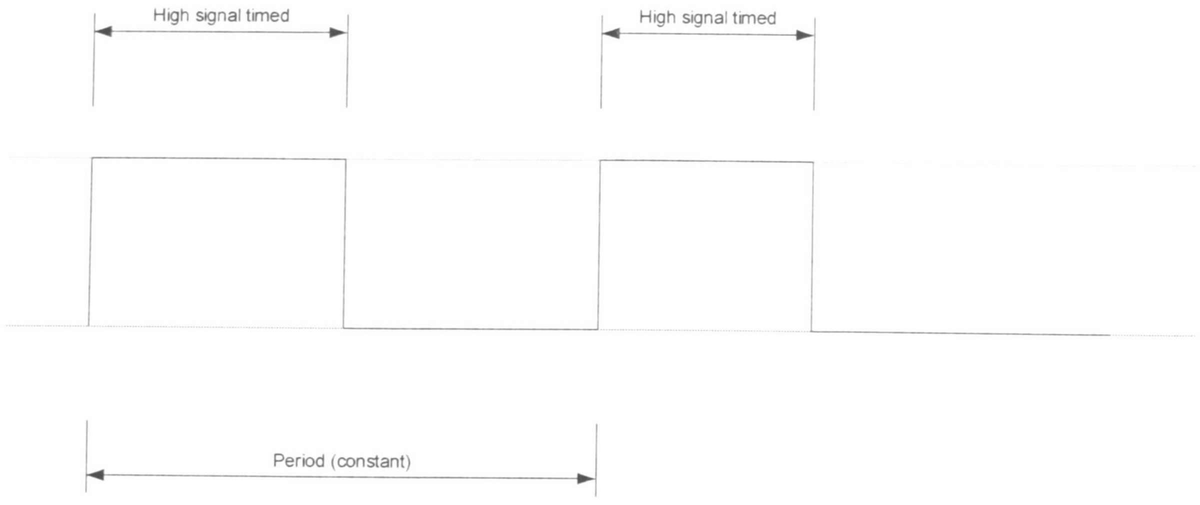


Figure 9 Measured duration of "High" on accelerometer output signal

Since there was no memory in the embedded system, every value that is sampled was temporarily stored in memory, overwriting the old value. On a request for data the last value stored is sent over the link. Thus, the data stream is actually sampled at a frequency set by the frequency or received requests.

The pre-made board that was used was equipped with a 5 volt voltage regulator, and had to be powered with a power supply of 9 volts. To accommodate a second regulator was used to power the rest of the system, so that the power supply could be used.

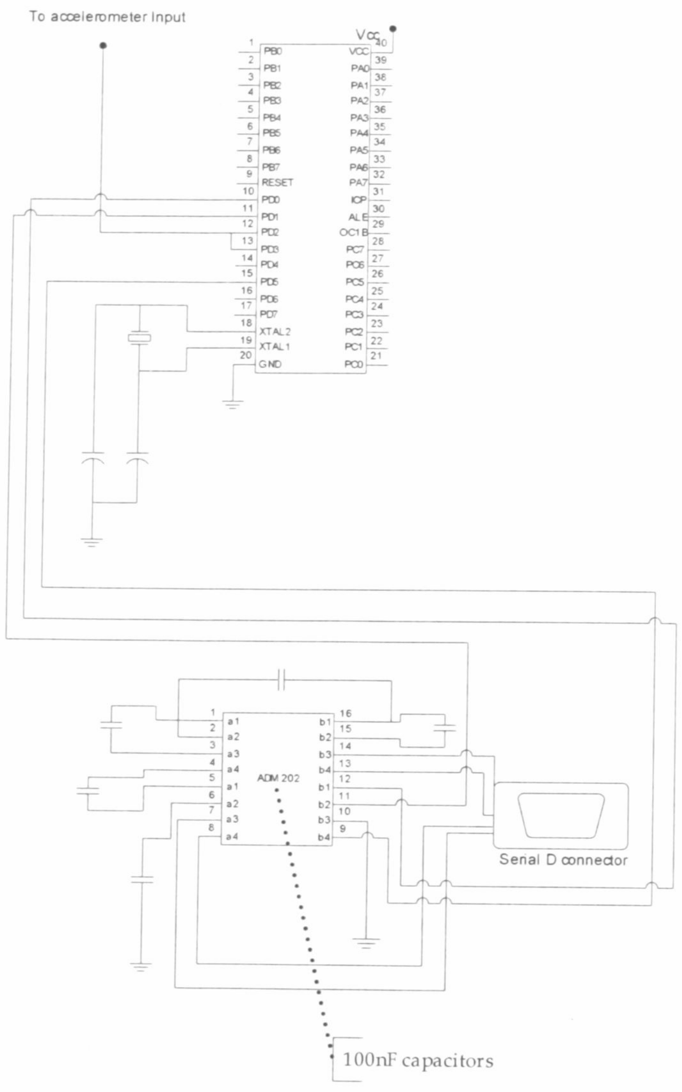


Figure 10 : Embedded system connection

4.3. Mobile Phone

The choice of a mobile phone to use with the system was perhaps the most crucial design choice and influenced the whole system. The mobile phone that was chosen was Nokia's 7110 and it's serial interface cable DLR-3. The foremost reason for this choice was that the standard AT command set could be run directly over the serial connection of the phone's cable. Like all mobile phones that were considered, the 7110 requires a specific proprietary protocol over which to send AT commands and data; the DLR-3 serial cable however contains a microcontroller receives standard AT commands or data and sends it over this protocol, eliminating any need for knowledge of the inner workings of the protocol. In all older Nokia models and in all other mobile phones considered, a driver written specifically for the windows operating system would run the other end of this protocol, making any communication of a non-windows running system with the phone impossible, at least without knowledge of the inner workings of this intermediate protocol. Information about the intermediate protocol of both older Nokia models and Ericsson's data enabled handsets was denied by representatives of both companies. Other phones that were considered and rejected were Panasonic and Alcatel models, rejected due to lack of data cable availability.

The DLR-3 serial adapter is set to run on 19200 bps. When interfacing with a modem in an embedded system it is common practice to use a "3 wire" serial interface, a stripped-down serial connection achieved by driving the modem's status input pins from their respective output pins. DTR is looped back to DSR and RTS is tied to CTS. This effectively sets the modem to read it's own signals as the data terminal's and assume that the microcontroller is always ready to receive and always about to send data. This way, only one of the inputs and one of the outputs of the serial level shifter are used, leaving the other pair for any debugging and/or on-site control the application code may require. This also effectively overwrites any hardware flow control of incoming (towards the microcontroller) data. However, in our case, all input status signals had to be implemented by the microcontroller's serial interface. Apparently the DLR-3 serial cable uses the status signals to power its microcontroller (no technical data was available about this from Nokia, despite queries, yet unofficial sources seem to support this).

Moreover because the serial link between the microcontroller and the modem is faster that the expected bandwidth, the possibility of the microcontroller overflowing the modem buffer was present[3]. To avoid this, the CTS signal is sampled each time a byte is to be sent. The remaining level shifter input is used to convert the modem's CTS signal to TTL level.

4.4. Microcontroller

The microcontroller that was used was a member of Atmel's AVR family of flash microcontrollers, namely the AT90S8515 5-volt microprocessor, running at 8 MHz. The firmware was written in assembly language using Atmel's AVR studio IDE and its integrated assembler.

[3] The protocol permits a maximum of 258 bytes consecutively without a response form the gateway, and thus a buffer above that size would render flow control unnecessary. However, no information was available on the size of the mobile phone's output buffer, and flow control was implemented.

4.4.1. Firmware

4.4.1.1. Firmware overview

In the early design of the microcontroller's firmware it was clear that there would be two separate tasks: Sampling the accelerometers and running the dialer and the data transfer protocol (the data transfer protocol and the dialer need not run simultaneously as the dialer exits when a connection is achieved so the data transfer protocol can be run. The sampling frequency of the accelerometers is high enough to consider it the primary task of the microcontroller, as the limited baud rate of the mobile phone insures that the rate at which the protocol parser will have to serve requests is low. At 9600 bps and using the ID command (being the smallest to execute it would yield the most commands per second) we would have 354 commands each second[4]. The accelerometers on the other hand produce a pulse every 2 milliseconds, thus requiring priority over the dialer. The accelerometer code is entirely in the interrupt service routine thus having priority over the protocol code or the dialer code.

The two "processes" communicate via a shared memory region. As it often happens in shared memory architectures, a consistency problem arises if both routines are allowed to access the memory arbitrarily since memory access is not an atomic action. In this case the interrupt service routine may attempt to write in the shared memory as data is being sent through the serial port. This would happen often, as in the time it takes to send the first byte, several interrupts would occur. A bit in a register is used as a semaphore to prevent that. Before starting transmission of the protocol routine sets this flag and clears it after the reading is finished. The interrupt service routine stores data in the shared memory region only after it checks that the flag is clear.

4.4.1.2. Resources used.

A primary concern while implementing the protocol was leaving as many resources available for the application specific code to use. The resources used by the protocol itself are two registers from the “high” register area (those capable of

[4] This number is calculated theoretically. 9600 bits per second at 9 bits a byte (start bit is included) produce approx 1067 bytes per second. The command requires one byte and its response is one byte, so at two bytes per command we have 534 commands each second. This calculation however does not include latency from the microcontroller (which is negligible) and transmission latency that varies greatly.

operations with immediates), one more from the "low" area, and of course, the microcontroller's RS-232 interface. The shared memory is not considered part of the protocol, as a different kind of sensor (one that generates data on demand) might not require it.

4.4.1.3. Accelerometer PWM sampling

The accelerometer's output (only one axis was used) was connected to the external interrupt pins. External interrupt 0 was set to trigger on a falling edge and external interrupt 1 was set to trigger on the rising edge. On the rising edge of the accelerometer's signal the microcontroller's 16 bit timer/counter is starting, and it stops on the falling edge of the same signal essentially timing the length of time the signal remains high. After stopping, the timer flag0 (the 0 bit on the flags register) is checked and if it is 0 the routine stores the contents of the timer/counter in memory, least significant byte first (in lower memory address). This overwrites the previous value. If the semaphore flag is set, the Interrupt Service Routine exits without saving any data.

4.4.1.4. Phone Dialer

The phone dialer is the part of the software that is used to send the AT commands to the mobile phone prior to the connection. It initializes the mobile phone and sends the ATDT command that causes the modem to dial the gateway PC. The phone number is stored in the program memory, and is thus impossible to change at run time (i.e. without reprogramming the microcontroller). EEPROM memory could alternatively be used or the microcontroller could be upgraded to allow write access to the program memory at run time (some microcontrollers of the AVR mega family are pin compatible with the 8515 and all but the oldest models allow write access to the program memory).

Currently the dialer was set to dial the gateway a few milliseconds after power-up (delay is necessary to allow the serial line level shifter time to charge its capacitors and function properly). After the end of the acquisition session an infinite loop is used to halt the processor. Since no data is being stored across sessions in the microcontroller, resetting the microcontroller can be used to initiate a new session.

4.4.1.5. Protocol parser

This routine runs after the dialer has successfully connected the modems and the transparent link to the gateway has been established. It polls the serial port for protocol commands and responds accordingly.

On an **ID** command the server always answers with an ACK command if a rising edge interrupt has happened and a NAK if no interrupt has happened since power-up (signifying that the accelerometer is not present). The device ID is hard-coded in the firmware (indicated with the "DeviceID" constant in the defined using the .EQU assembler directive).

On a **REQUEST** command the semaphore flag is set to lock further writing in the shared memory, the DATA header is sent and then the data bytes in the shared memory. Before sending a data byte, it is added to an originally cleared register to calculate the checksum. **NEXTREQUEST** does exactly the same.

On an **ALIVE** command, ACK is sent.

No extended commands have been implemented, so **EXTENDED** command causes a NAK response.

On a **DISCONNECT** command the protocol simply exits the protocol parsing subroutine ignoring all serial input until next dial. The AVR does not send an escape sequence as this would require extensive use of the timer/counter. Although it was possible to do this, halting the accelerometer sampling that would be using the 16 bit timer/counter at the time would be necessary and making it part of the protocol would dramatically increase its resource requirements (timers/counters are amongst the most valuable resources on a microcontroller). The mobile phone's modem detects lack of carrier in a few seconds and disconnects on its own. Ignoring serial port input is important as the escape sequence that the gateway will send through the modem to disconnect its end of the connection will be received on this end of the still active connection.

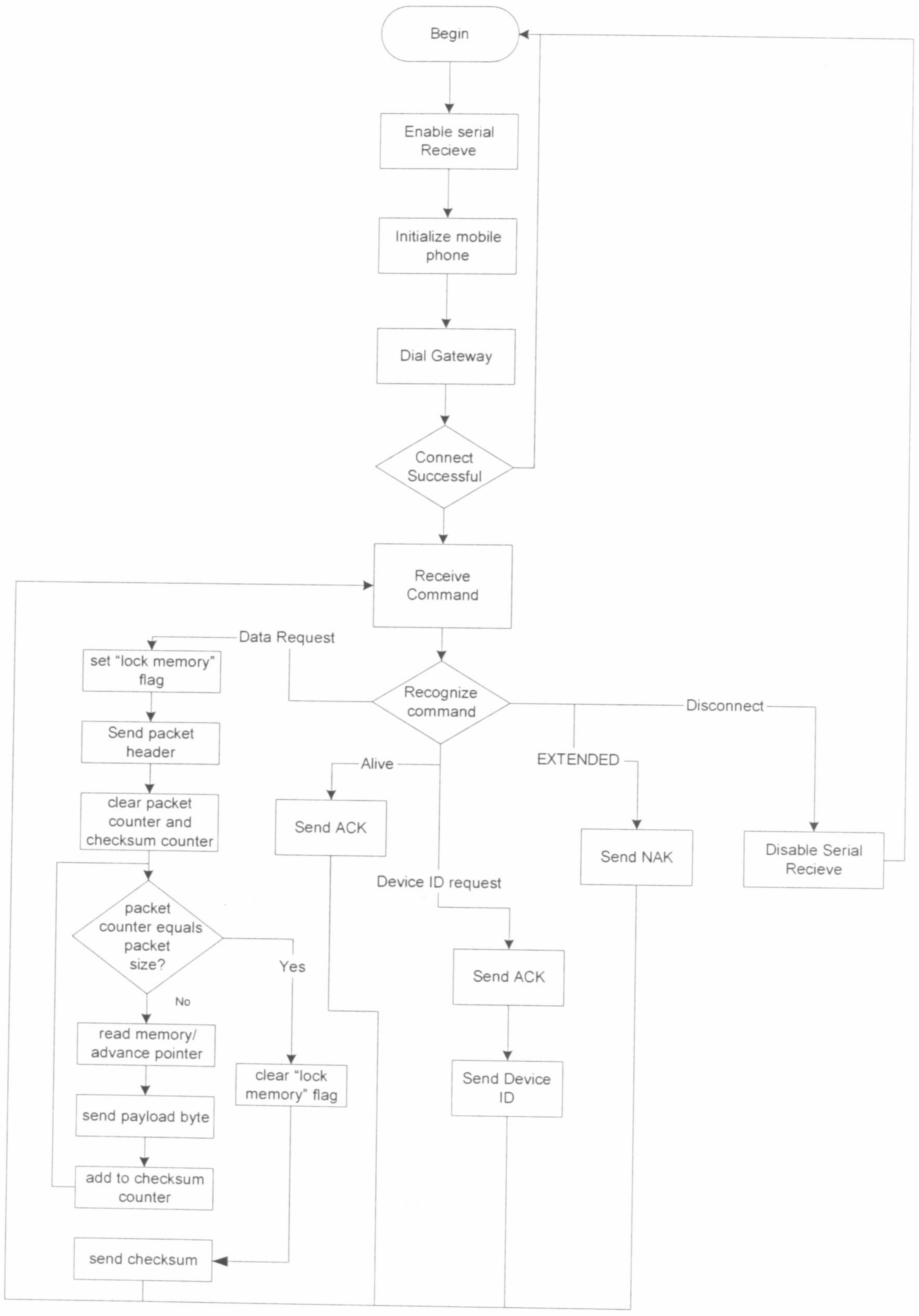
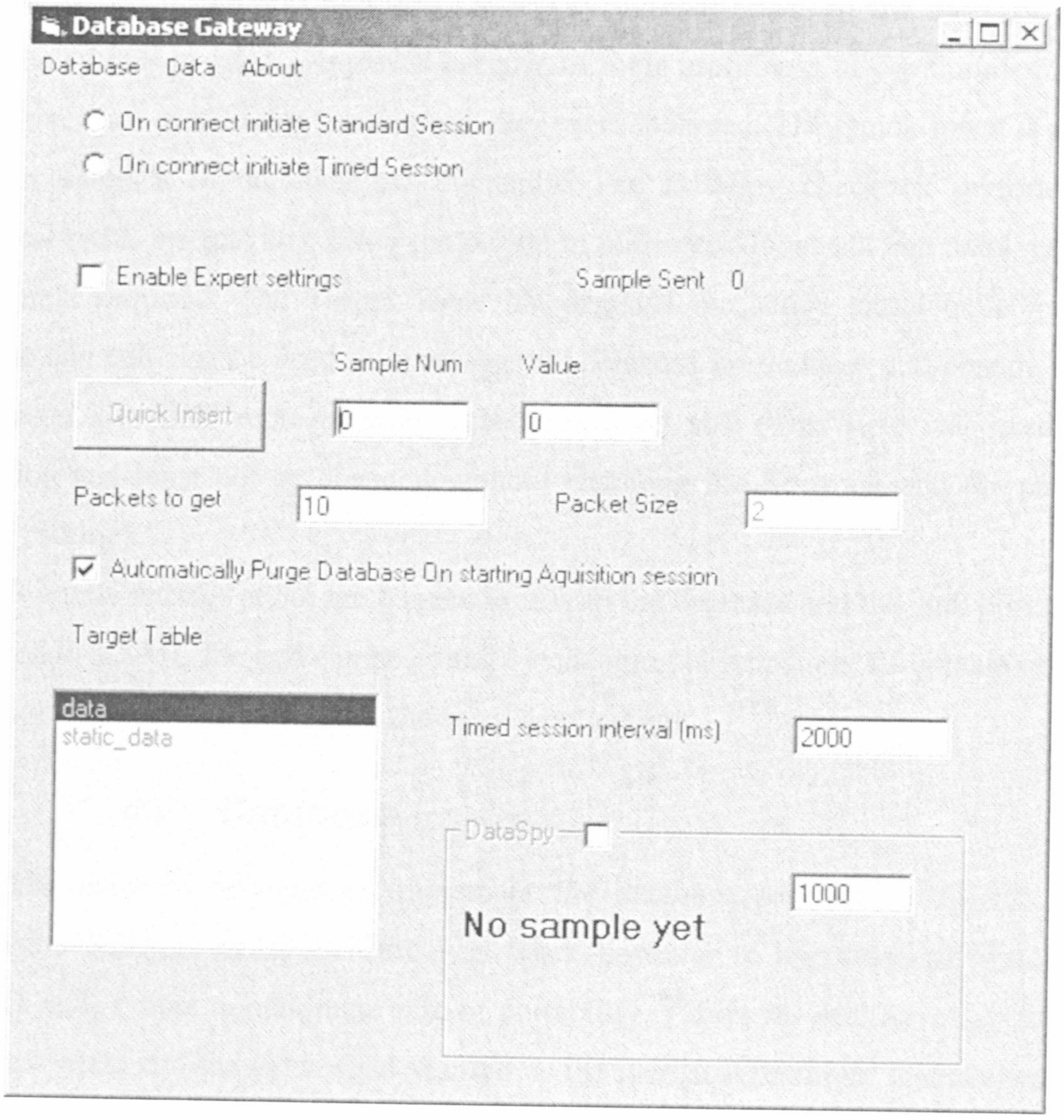


Figure 11 Flowchart representing the function of the microcontroller's main firmware. (ISRs not shown)

4.5. Gateway Software

The gateway software was written in Microsoft Visual Basic 6, because if the languages ease of use, built in connectivity with a database and easy driver-based (as opposed to IO-address based support of older languages) support of serial port communication via an ActiveX control. A graphical interface facilitates the management of the data acquisition session. The various commands are available through the application's menus.



( ) On connect initiate Standard Session
( ) On connect initiate Timed Session

☐ Enable Expert settings

Sample Sent 0

Sample Num 0

Value 0

Packets to get 10

Packet Size 2

☑ Automatically Purge Database On starting Aquisition session

Target Table

data
static_data

Timed session interval (ms) 2000

DataSpy ☐ 1000

No sample yet

Image 1 : Screenshot of the Data Acquisition gateway application

The Database menu contains two options. The "Purge" option empties the contents of the selected active table. The "Exit" option exits the program. The Data menu has three options. "Start" initiates a standard data acquisition session to fetch a number of packets designated by the value in the "Packets to Get" field. The initial sample number is the value of the "Sample Num" field and the sample number increases by one on every subsequent sample stored. Each sample request is made as soon as the previous one is completed. The "Timed session" option has almost identical functionality, except that the requests are made at intervals defined by the value of the "Timed session Interval" field. Timed sessions can be interrupted by selecting the "Stop timed session" option from the Data menu. The two radio buttons can be use to cause the program to initiate either type of session on a successful modem connection.

The other options are enabled by the "enable expert settings" check box. This is because certain combinations can cause key overlap errors in the database and cause the system to crash. However because of their usefulness in development and the relative ease of avoiding such errors, they were included. The Quick Insert is used to insert samples in the database manually. The DataSpy checkbox performs a periodical (with the text box being the period in milliseconds) check and displays the last sample acquired. The Target Table list displays the active target table in the database and can also be used to overwrite the database preferences and change it it. The Packet size field represents the expected packet size (This field was used for debugging and must not be changed without changing the firmware and the packet parsing routine).

Various settings about the connections with the database and the com port used are stored in a XML file and can be edited by changing it. Appendix III contains more information about configuring the Gateway Application.

4.6. Database

The database selected to implement the database part of the system was Microsoft's SQL server 2000. Care was taken however to use standard SQL and Transact-SQL syntax to maximize ease of portability. Tables for data have two fields, the sample field and the value field. Sample is the increasing number that also serves as primary key, and denotes the chronological order of the received samples. Value is the value of the timer for that sample. The type of both fields is long integer, but can be easily changed to better reflect the nature of the data in a different application. The sample number could be changed into datetime type for more loosely sampled values (in out case, sampling frequency is too great for the timedate type to serve as a timestamp, but it would be ideal for data sampled in intervals of more than a few seconds). Value could be changed in real if any processing was done to calculate acceleration, for example in m/sec^2.

Every table is mentioned in a special table named "_meta_tables_description". Every table with two fields named "sample" and "value" (The table must not be named "_meta_tables_description") can be mentioned in this table and be associated with a data source. The _meta_tables_description has three fields: table_name, sensor_ID and description. Sensor_ID is a short integer and the other fields are text fields. Table_name contains the name of the table, and the description field contains a short description of that data the table will represent. Sensor_ID contains the DeviceID of the microcontroller whose data is to be matched to this table.

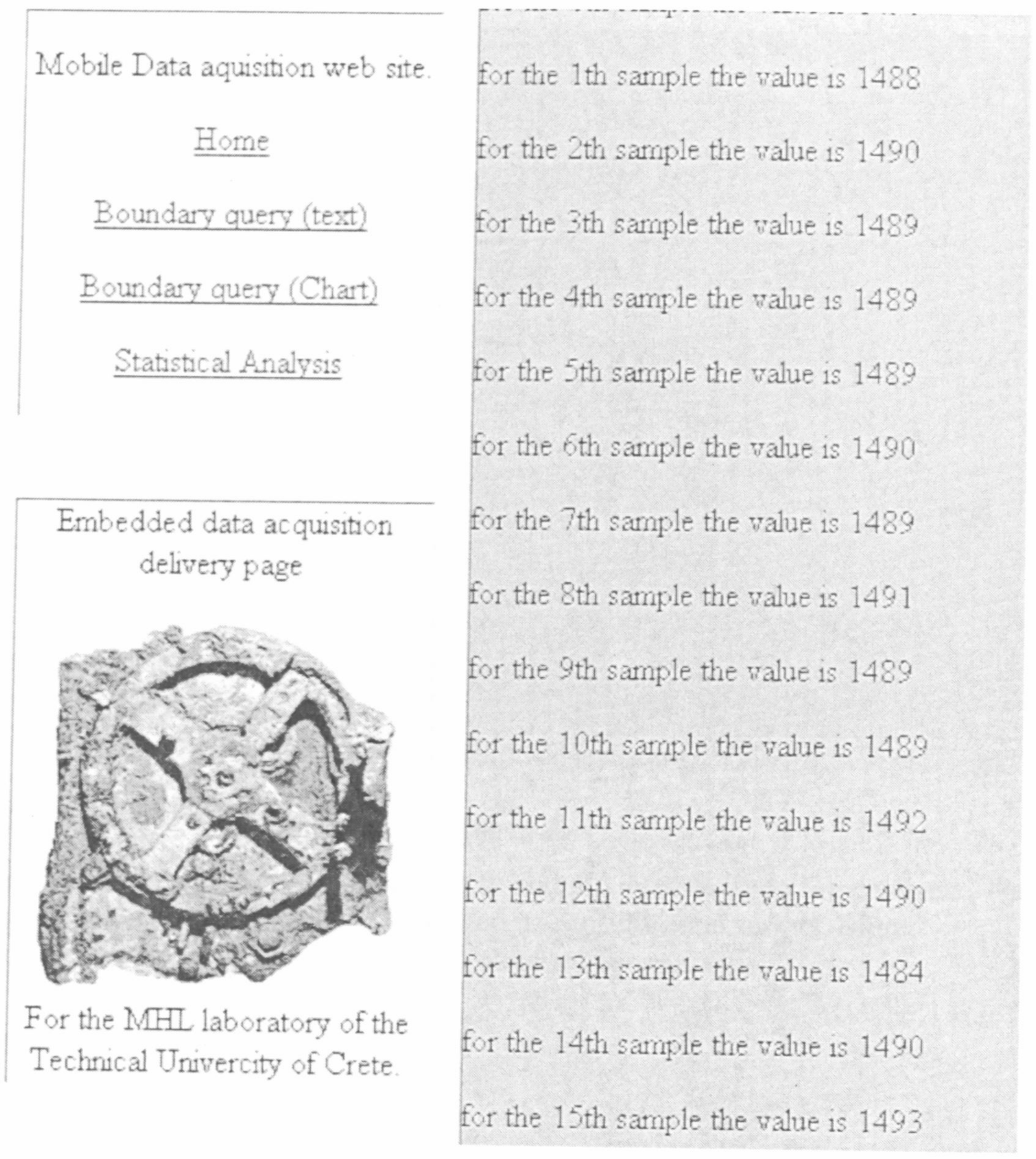
4.7. Web server

The web server is where the scripts responsible for powering the sweb interface. The Internet Information Server provided with Windows 2000 was used, and vbScript was selected as the scripting language of the server side scripts. Client side scripts were not included in any of the web pages to maximize platform and browser independence. Any browser that supports frames, forms and depicting graphics can theoretically use the web interface. (See the testing on browsers). The GET method was used to pass parameters to the server script, both to facilitate bookmarks that contain parameters for pre-made queries and in accordance with the HTML 4.0 guidelines that propose using the GET method for idempotent actions (like database queries).

The web interface at its introductory page presents two links in the sidebar. One is for a standard, text outputting query and the other is for a graphical representation of the data resulting from the query. Pressing either of those links will result in the parameter input page. The basic parameter to select is the table from which to query data (tables present in the _meta_table_discription table appear in the drop down box). If the user wishes he can use a sample range to limit the number of samples that will be displayed as a result of the query. On pressing the submit button the parameters pass back to the server along with the resulting URL.

The first phase of the data retrieval is identical to all queries. First the parameters of the previous page are extracted by parsing the URL string. Then a database connection object is instantiated and the database connection string along with the login information is passed to the object and the connection is open. The parameter connection string is a standard form of passing information necessary for initiating a database connection, that information being the network name of the computer the SQL server runs on and the database to be accessed. The login and password of a database account are also passed into the object at this point along with the protocol that will be used to access the database (SQLOLEDB was used to specify SQL server's OLE protocol). Then the SQL query is constructed using the previously extracted parameters and the query is executed by opening the connection. The result is a recordset, a data structure that represents the returned tuples from the database. Through a recordset a single tuple can be accessed at a time, as if it were a normal record, but the active tuple can be change by moving a notional cursor across the returned table by invoking the recordset's methods.

In the case of a text query a loop is used to access all the tuples serially by moving through the recordset by invoking the movenext method, that moves the recordset's cursor to the next tuple. In each iteration, the sample and value of the current tuple are "printed" on the html response.



Mobile Data aquisition web site.

Home

Boundary query (text)

Boundary query (Chart)

Statistical Analysis

Embedded data acquisition delivery page

For the MHL laboratory of the Technical Univercity of Crete.

```
for the 1th sample the value is 1488
for the 2th sample the value is 1490
for the 3th sample the value is 1489
for the 4th sample the value is 1489
for the 5th sample the value is 1489
for the 6th sample the value is 1490
for the 7th sample the value is 1489
for the 8th sample the value is 1491
for the 9th sample the value is 1489
for the 10th sample the value is 1489
for the 11th sample the value is 1492
for the 12th sample the value is 1490
for the 13th sample the value is 1484
for the 14th sample the value is 1490
for the 15th sample the value is 1493
```

Image 2: Browser client area, results of a text query

In the case of a graphical response however the procedure is different. The recordset is split into two vectors, one containing all sample numbers and one all sample values. Then an Office web component object is instantiated to create a chart. The object's fields are used to set the various parameters of the chart and finally a data series is created based on the sample values vector for its y axis values, with the sample number vector serving as a basis for the x axis. The appropriate method (getPicture) of the chart component is invoked, and the response is piped on the output of the web page. This way, instead of an HTML page, a bitmap (compressed in gif format) of the resulting graph is sent to the web client and seen in the main frame.

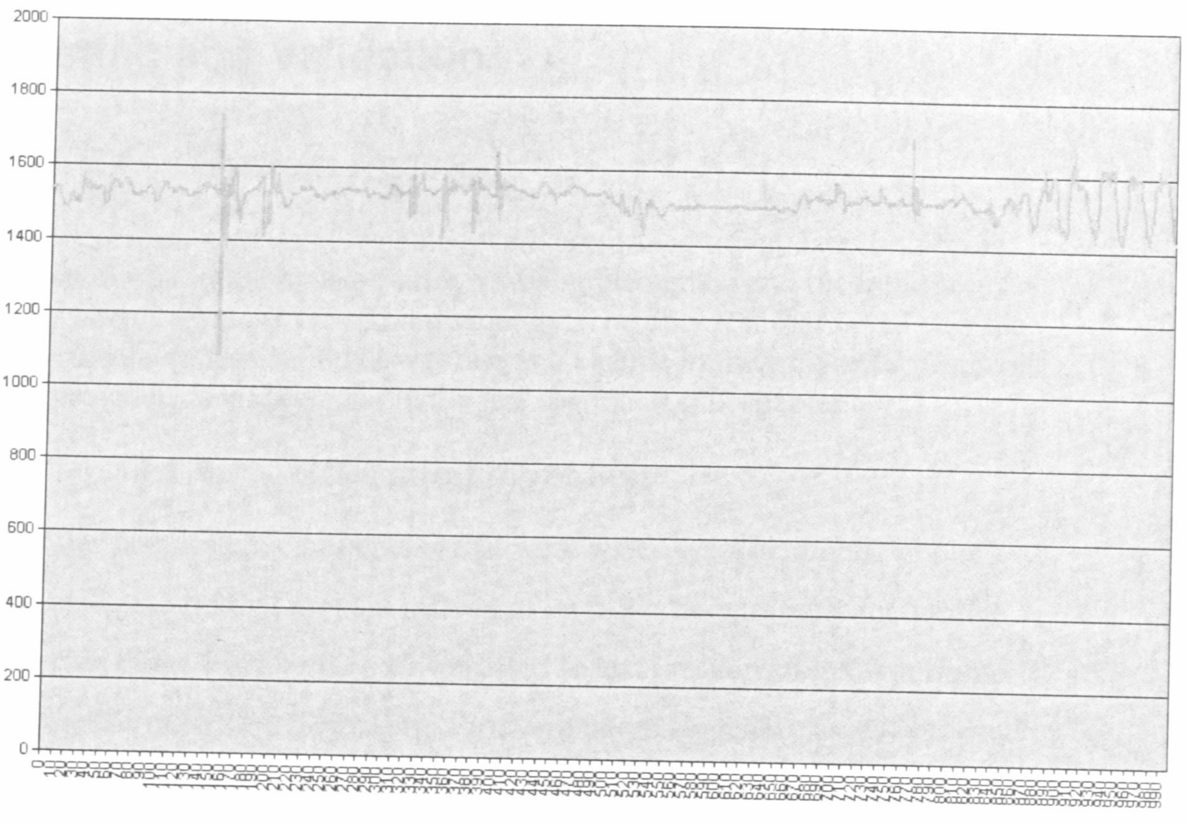


Image 3: Graphic representation of acquired data

Settings for the web server scripts are mainly about connectivity to the database. They are stored in an include file that can be edited to configure the scripts. More information on how to configure it can be found in Appendix III.

5. Testing and Validation

5.1. System testing

This chapter describes a series of tests that were preformed to validate the system. Every subsystem was tested by itself after it was implemented and the interfaces between subsystems were also tested individually. The only test that failed was the testing of the whole system, and that was due to a change in the supported protocols of the GSM provider, as is discussed in the last section of this chapter.

5.1.1. GSM equipment test

One of the first system components that were tested was the mobile phone's ability to connect to a normal modem instead of an ISP's line. Initially the mobile phone connected to the PC's serial port was used to test this capability. A terminal program (HyperTerminal and TerraTerm Pro were interchangeably used) was used to connect to the mobile phone using 19200 port speed, no parity, 1 stop bit and 8 data bits. The ATDT command was then used to dial the phone number of the analog modem (most of the times the modem was connected to an analog port of an ISDN terminal adapter. One test was done on a purely analog phone line but no differences in the modem's behavior were noted). The mobile phone was tested with a variety of modems to ensure compatibility. A Crypto 56K Internal PCI modem, a Diamond SupraExpress Serial modem, and a US robotics Sportster serial modem were used and no differences (except minor variance in handshaking time, considered unimportant) were noted.

The results of this test varied across development time. All tests were successful in early development stages, and connection was consistent. In later tests, after most of the rest of the system was implemented, the same test failed. The ability to perform this type of connection depends on the GSM provider's hardware, as the GSM modem transmits data digitally. A connection with an analog modem is only possible due to the GSM provider utilizing extra hardware that does the analog modulation. This service was apparently discontinued, although repeated attempts to confirm this with the GSM provider's technical support on this subject were unsuccessful.

In the early stages of development, 9600 bps was the maximum data bandwidth the GSM network could attain. Since then, the GSM provider claims that 14400 bps are attainable. Failure in connecting with an analog modem however has prevented confirmation of such claims. However, no assumption was made during the design of the system as to the baud rate of the serial connection, and higher baud rates can be used without change to the system. Indeed, all null modem cable tests were done at 19200 (the baud rate the DLR-3 serial cable connects) without any adjustment.

5.1.2. Embedded system testing

The next component that had to be tested was the embedded system and its serial connection, both directly to the PC but, more importantly, to the mobile phone. First the serial link between the microcontroller and the mobile phone was tested by having the microcontroller issue an ATDT dialing command, just to verify the microcontroller's ability to send AT commands. The next test was dialing the modem and sending a repeating byte pattern (ASCII A through Z was used). The output of the modem was captured (using a terminal program) and checked for transmission errors. No errors were detected in a transmission of no less than 3 minutes (roughly 190 kb of data). This was expected as the modems use an error correction protocol (MNP5) across the serial link (although this does not protect against errors occurring between the microcontroller and the mobile phone's serial interface).

5.1.3. Gateway software testing

Testing the gateway software consisted of testing the arrival of data to it and testing its ability to store the data to the database. In early stages of development the DataSpy feature was used to display the received data in real time form the embedded system connected to the PC's serial port via a null modem serial cable (the null modem serial cable emulates a pair of connected modems). This test was particularly important while implementing, and later testing, the accelerometer interface. This test also verified the proper function of the accelerometers. The quick Insert function was used to introduce dummy data to the database to test database access. Finally the embedded system, connected through a null modem cable, was used to initiate a data acquisition session while storing the data to the database. A non-timed session of 10000 samples to a table corresponding to the embedded system's ID was used so verify system stability. No checksum errors were reported in three separate repetitions of the 10000 sample test.

5.1.4. Web interface testing

The web interface was originally tested with dummy data manually entered in the database, and later with actual data from the accelerometers. The main concern for the web interface was its proper function across different platforms and browsers. In theory any frame-supporting browser will work. A wide range of browsers were tested, and apart from minor visual differences (mainly in the way form controls look and in spacing) the interface works similarly in all of them. The following browsers were tested: Microsoft internet explorer 6.0, Microsoft internet explorer of windows CE for pocket PC, Opera 6.00, Netscape 6.20, mozilla 1.0 and 1.1 and KDE's konqueror browser were used. The display was optimized for 1024x768 pixel resolution, but the dimensions of the generated image can change easily to make viewing in other resolutions easier.

5.1.5. Complete test

Because when the complete system was ready, connection with an analog modem was unattainable, the system could not be tested in its entirety. The remaining and final step that had to be taken to achieve the desired specifications was substitution of the null modem serial cable with the mobile phone / analog modem pair. However such connection was not possible. All other tests however have validated the rest of the subsystems and based on their results, no problems are expected to occur should data connections become available.

6. Future work

The system is far from perfect and there are a number of things that could be done to improve it. These improvements mainly consist of making use of newer technologies that have become available. Not all are applicable or useful to every application, and not every one would result to a low cost solution (especially the ones requiring changes in the embedded system). However there are applications that can benefit greatly from such improvements.

6.1.1. Embedded TCP/IP

One of the drawbacks of the system is its inability to connect with any Internet Service Provider and access the internet directly. This would solve problems with the GSM provider and give more versatility to the system, as it would eliminate the need to set up a server connected to a phone line, and allow the server to reside anyware in the network. Implementation of this architecture will require running a SLIP or PPP protocol over the serial line, and over that the TCP/IP protocol. Running these protocols is costly however, especially in memory, and any such endeavor will lead to an embedded system with external memory, at the very least, and maybe a more powerful microcontroller.

6.1.2. Embedded Web Server

One step further than the embedded TCP/IP improvement is an embedded web server system. This system would dial an ISP and connect to the internet, and then run a web server. With this architecture no server is required. Every connecting client would connect directly to the embedded system and get requests. This type of embedded system would have to be of considerable power, and possibly run some operating system to enable high-level software development.

6.1.3. VoiceXML interface

VoiceXML technology is the equivalent of HTML in voice recognition applications. It is a trivial task to change the server side scripts so that they generate a VoiceXML output to be parsed by a voice browser. Information can be accessible through the telephone, or in an automotive environment where visual interfaces are unusable. Parking availability and traffic information systems can benefit greatly by an interface of this type. A conventional Web interface can function simultaneously with the voice interface.

6.1.4. WML interface

Along the lines of creating a VoiceXML interface is the creation of a WAP interface. Modifying the server side scripts to generate WAP content is easier than generating VoiceXML, given the similarities between HTML and WML. Such a modification would enable data queries from WAP enabled mobile phones. As with the previous modification, WML generating scripts can run simultaneously with normal HTML generating scripts offering access to the same data.

6.1.5. GPRS support

GPRS technology permits high bandwidth access to the internet from mobile phones. Also GPRS access is "always on-line", and is priced by the amount of traffic each user generates rather than on-line time. This makes a GPRS enabled system an economically attractive solution for applications where small amounts of data are often required.

Every day new technologies become available and new ways to transfer data and interact with computers appear. The use of a database and the standardized interfaces make future updates with newer technologies a fairly easy process.

7. Appendix I: Bibliography and other sources

7.1. Bibliography

'Data and Fax Communication', by Robert L. Hummel, pub. Ziff-Davis
ISBN: 1562760777

'Εισαγωγή στα συστήματα βάσεων Δεδομένων', C. J. Date
ISBN: 9603321095 - 9603321109

Digital Design by M. Morris Mano
ISBN: 0130621218

Visual Basic Programmer's Guide to Serial Communications
by Richard Grier, Zane Thomas (Editor), James Shields (Editor), Phounsavan
ASIN: 1890422258

7.2. On-line sources

Comp.arch.embedded. newsgroup
Alt.mobilephones.nokia newsgroup
www.asp101.com

www.w3schools.com
The Microsoft Development network (msdn.microsoft.com) and visual studio MSDN cd-rom.
www.avrfreaks.com
www.atmel.com
forum.nokia.com (Nokia Developer Forum)

8. Appendix II: References

[Ref.1] Superlogics web site, WebDAQ:
http://www.superlogics.com/specpage.asp?Items=8400)
[Ref 2] PicoWeb web page http://www.picoweb.net/
[Ref 3] M@xks web page, M@xks Application Microserver http://www.esw-stolberg.com/webserver/eng/product/roboteng/roboteng.html)
[Ref 4] Accord Web site http://www.accord-products.com
[Ref 5] (http://dragonix.openhardware.net/)
[Ref 6]

9. Appendix III: Implementation Details.

This section provides information on various points in the implementation of the system that are critical for setting up the system or changing its settings. While these details are not important in understanding how the system works, they are important to actually operate the system

9.1. Packet size

In the assembly code of the microcontroller's firmware, the packet size can be changed by changing the value of the "packetsize" constant. On the gateway application a field marked "Packet size" is present that can be edited once the "Expert settings" check box is checked. Note however that packet size changes without change to the packet parsing and packet generating routines in either end of the protocol (microcontroller and gateway application) will cause the system to stop functioning. While the system does not assume constant packet size, changing packet sizes is not implemented.

9.2. Gateway application settings file

An xml file is used to store settings for the gateway application, and can be edited to change them. What follows is a description of these settings.

The information necessary for connection with the database are loaded at startup from the bdsettings.xml file found in the system's drive root directory. This file contains the dbsettings tag that includes the following tags.

- Provider: The database connectivity provider that is used to facilitate connection with the database. The value "sqloledb" was used to communicate with SQL Server

- connection_string: This string consists of three pieces of information. The "server" value is used to point to the IP address of the machine that runs the SQL server, being either the IP address itself or the corresponding DNS name. "VERGINA" was the name used being the network name of the developing machine. The "Database" value points to the database that contains the tables the data is to be stored in. "Mobile" was the active database in our case. Trusted_connection specifies that a login is necessary to access the database (not necessary if the windows login/password is used for authentication). "No" was the value used in this case.

- The "login" tag specifies the database login. "sa" was used.

- The "password" tag specifies the password corresponding to the login used. The accound used had a null password, and this was used as the value of this tag.

- The "COMprot_Number" contains the number of the COM port that is to be used.

- The COMport_settings is the settings string for the serial connection. The value "19200,n,8,1" that was used signifies 19200 bps connection speed, no parity, 8 data bits and 1 stop bit.

9.3. *Web server Settings*

Settings for the web server scripts are mainly about connectivity to the database. They are stored in an include file that can be edited to configure the scripts.

The settings.inc file holds the connection settings for the database. It consists of four vbscript variable assignments, enclosed in an ASP tag. The names and function if these variables are as follow.

Provider: The database connectivity provider that is used to facilitate connection with the database. The value "sqloledb" was used to communicate with SQL Server

connstr: This string is the connection string and in a typical network implementation it should match that of the gateway software (Indeed in most cases the strings should be identical, but not always, especially when a computer employs multiple network adapters). It consists of three pieces of information. The "server" value is used to point to the IP address of the machine that runs the SQL server, being either the IP address itself or the corresponding DNS name. "VERGINA" was the name used being the network name of the developing machine. The "Database" value points to the database that contains the tables the data is to be stored in. "Mobile" was the active database in our case. Trusted_connection specifies that a login is necessary to access the database (not necessary if the windows login/password is used for authentication). "No" was the value used in this case.

The "dblogin" variable specifies the database login. "sa" was used.

The "dbpwd" variable specifies the password corresponding to the login used. The account used had a null password, and this was used as the value of this tag.

10. Appendix III: Protocol Simulation Data

This is the Data used to generate the protocol simulation graphs. The first column is the packet size, the second shows how many mistransmitted packets were intercepted by a checksum mismatch. The third indicates the number of packets that were altered by noise during transmission but were not detected. The forth indicates the bandwidth usage for payload data. The bit error rate that was used in the simulation was $5 \cdot 10^{-5}$ and the simulation length was 10000000 packets.

Simulation Data

packet size	checksum errors	data errors	bandwidth use	packet size	checksum errors	data errors	bandwidth use
2	1517	0	0,5	63	31043	6	0,969230769
3	1956	0	0,6	64	31203	4	0,96969697
4	2385	0	0,666666667	65	31503	5	0,970149254
5	2998	0	0,714285714	66	32300	10	0,970588235
6	3336	0	0,75	67	32716	8	0,971014493
7	3822	1	0,777777778	68	33199	7	0,971428571
8	4324	1	0,8	69	33362	12	0,971830986
9	4807	1	0,818181818	70	34387	15	0,972222222
10	5466	0	0,833333333	71	34501	13	0,97260274
11	5798	0	0,846153846	72	34758	12	0,972972973
12	6373	0	0,857142857	73	35934	10	0,973333333
13	6888	0	0,866666667	74	35778	9	0,973684211
14	7373	0	0,875	75	36071	12	0,974025974
15	7888	2	0,882352941	76	37084	16	0,974358974
16	8453	0	0,888888889	77	37439	12	0,974683544
17	8617	0	0,894736842	78	37734	11	0,975
18	9143	0	0,9	79	38380	12	0,975308642
19	9843	1	0,904761905	80	38860	17	0,975609756
20	10225	3	0,909090909	81	38896	18	0,975903614
21	10661	0	0,913043478	82	39991	18	0,976190476
22	10996	3	0,916666667	83	40053	12	0,976470588
23	11624	2	0,92	84	40423	10	0,976744186
24	11987	0	0,923076923	85	41594	15	0,977011494
25	12679	1	0,925925926	86	41490	15	0,977272727
26	13317	0	0,928571429	87	41982	14	0,97752809
27	13549	0	0,931034483	88	42845	17	0,977777778
28	14004	3	0,933333333	89	42831	19	0,978021978
29	14590	4	0,935483871	90	43349	14	0,97826087
30	15076	0	0,9375	91	44044	19	0,978494624
31	15227	5	0,939393939	92	44072	19	0,978723404
32	15992	4	0,941176471	93	44956	17	0,978947368
33	16559	4	0,942857143	94	45137	15	0,979166667
34	17201	3	0,944444444	95	45526	16	0,979381443
35	17319	2	0,945945946	96	46479	16	0,979591837
36	17845	1	0,947368421	97	46525	26	0,97979798
37	18464	4	0,948717949	98	47355	15	0,98
38	18659	3	0,95	99	47634	16	0,98019802
39	19336	5	0,951219512	100	48151	16	0,980392157
40	20082	7	0,952380952	101	48566	15	0,980582524
41	20120	3	0,953488372	102	48963	20	0,980769231
42	20800	5	0,954545455	103	49424	21	0,980952381
43	21190	3	0,955555556	104	49823	22	0,981132075
44	21574	4	0,956521739	105	50700	19	0,981308411
45	22429	4	0,957446809	106	50607	24	0,981481481
46	22633	6	0,958333333	107	51663	19	0,981651376
47	23233	6	0,959183673	108	51798	18	0,981818182
48	23638	9	0,96	109	52146	18	0,981981982
49	23882	3	0,960784314	110	52682	21	0,982142857
50	24844	9	0,961538462	111	53105	20	0,982300885
51	25194	9	0,962264151	112	53747	28	0,98245614
52	25455	4	0,962962963	113	54002	34	0,982608696
53	25834	4	0,963636364	114	54699	22	0,982758621
54	26528	7	0,964285714	115	54772	15	0,982905983
55	27318	4	0,964912281	116	55801	21	0,983050847
56	27402	10	0,965517241	117	55757	24	0,983193277
57	27713	8	0,966101695	118	56759	27	0,983333333
58	28351	4	0,966666667	119	56836	25	0,983471074
59	29241	8	0,967213115	120	57744	21	0,983606557
60	29204	6	0,967741935	121	57532	20	0,983739837
61	29728	9	0,968253968	122	58475	27	0,983870968
62	30262	12	0,96875	123	58655	26	0,984

packet size	checksum errors	data errors	bandwidth use	packet size	checksum errors	data errors	bandwidth use
124	59473	24	0,984126984	186	87437	54	0,989361702
125	59471	30	0,984251969	187	87329	71	0,989417989
126	60206	25	0,984375	188	88374	69	0,989473684
127	60655	37	0,984496124	189	88559	63	0,989528796
128	61044	32	0,984615385	190	89094	67	0,989583333
129	61646	36	0,984732824	191	89663	71	0,989637306
130	61712	29	0,984848485	192	89598	74	0,989690722
131	62518	36	0,984962406	193	90262	66	0,98974359
132	62866	26	0,985074627	194	90970	70	0,989795918
133	63419	28	0,985185185	195	91215	78	0,989847716
134	63633	35	0,985294118	196	91832	74	0,98989899
135	64386	39	0,98540146	197	91956	69	0,989949749
136	64779	34	0,985507246	198	92403	59	0,99
137	65083	42	0,985611511	199	93134	76	0,990049751
138	65899	49	0,985714286	200	93734	80	0,99009901
139	66033	29	0,985815603	201	93807	61	0,990147783
140	66574	27	0,985915493	202	94360	72	0,990196078
141	66710	32	0,986013986	203	94632	77	0,990243902
142	67677	42	0,986111111	204	95012	68	0,990291262
143	67733	44	0,986206897	205	95617	64	0,990338164
144	68155	46	0,98630137	206	96197	71	0,990384615
145	68910	38	0,986394558	207	96548	83	0,990430622
146	69219	44	0,986486486	208	97079	84	0,99047619
147	69536	37	0,986577181	209	97283	79	0,990521327
148	70453	46	0,986666667	210	98000	97	0,990566038
149	70384	48	0,986754967	211	98328	80	0,990610329
150	71089	38	0,986842105	212	98588	75	0,990654206
151	71598	37	0,986928105	213	98941	77	0,990697674
152	71706	53	0,987012987	214	99599	80	0,990740741
153	72549	52	0,987096774	215	100343	82	0,99078341
154	72934	37	0,987179487	216	100827	84	0,990825688
155	73110	40	0,987261146	217	100879	76	0,99086758
156	73903	42	0,987341772	218	101534	77	0,990909091
157	74573	30	0,987421384	219	101426	91	0,990950226
158	74396	40	0,9875	220	102342	64	0,990990991
159	75172	48	0,98757764	221	102712	75	0,99103139
160	75597	54	0,987654321	222	103154	95	0,991071429
161	75802	51	0,987730061	223	103614	81	0,991111111
162	76694	44	0,987804878	224	103966	94	0,991150442
163	77018	41	0,987878788	225	104372	79	0,991189427
164	76966	51	0,987951807	226	104816	81	0,99122807
165	78019	51	0,988023952	227	105438	101	0,991266376
166	78373	32	0,988095238	228	105790	103	0,991304348
167	78590	42	0,98816568	229	106189	72	0,991341991
168	79343	45	0,988235294	230	106704	78	0,99137931
169	79698	45	0,988304094	231	106975	91	0,991416309
170	79998	40	0,988372093	232	107463	88	0,991452991
171	80173	63	0,988439306	233	107932	84	0,991489362
172	81204	51	0,988505747	234	108318	96	0,991525424
173	81457	54	0,988571429	235	108955	102	0,991561181
174	81549	50	0,988636364	236	109359	89	0,991596639
175	82282	54	0,988700565	237	109685	107	0,991631799
176	82874	56	0,988764045	238	110020	98	0,991666667
177	83291	56	0,988826816	239	110499	102	0,991701245
178	83451	54	0,988888889	240	110998	95	0,991735537
179	84097	55	0,988950276	241	111499	109	0,991769547
180	84915	51	0,989010989	242	111805	90	0,991803279
181	85068	64	0,989071038	243	112220	93	0,991836735
182	85472	66	0,989130435	244	112666	101	0,991869919
183	85869	58	0,989189189	245	113055	116	0,991902834
184	86318	44	0,989247312	246	113455	105	0,991935484
185	86737	64	0,989304813	247	113841	114	0,991967871

248	114578	119	0,992
249	114769	108	0,992031873
packet size	checksum errors	data errors	bandwidth use
250	115434	120	0,992063492
251	115713	108	0,992094862
252	116233	105	0,992125984
253	116743	116	0,992156863
254	116802	117	0,9921875
255	117452	114	0,992217899