MSc Data Communication Systems

Department of Electronic & Computer Engineering

Brunel University

Development of a TCP/IP-to-Serial Protocol Converter Device

MSc Student:
Panagiotis Kenterlis

Supervisor:
Prof. C. Nomikos
(TEI of Athens)

May 2004

A Dissertation submitted in partial fulfilment of the requirements for the degree of Master of Science

MSc Data Communication Systems

Department of Electronic & Computer Engineering

Brunel University

Development of a TCP/IP-to-Serial Protocol Converter Device

Student's name:_____

Signature of student:_____

Declaration: *I have read and I understand the Department's guidelines on plagiarism and cheating, and I certify that this submission fully complies with these guidelines.*

Abstract

The prominence of the Internet and networked computing has driven research efforts into providing support for heterogeneous computing platforms. Thoughts arise of using the current network infrastructure to increase the functionality of legacy devices or reuse obsolete systems.

Most such systems are equipped with a communications port, more often the standard RS-232 serial port. An attempt to use such systems over a computer network could be successful with the use of a communication device, which enables the data normally transferred through the serial port to travel over the network. This requires utilization of the existing transfer protocols on modern computer networks.

Aim of this specific project is the study, design, and development of a complete telemetric system that will transfer the data acquired by a field station to a central station. Objective of this project is to achieve error free transmission of data and real time communication using established TCP/IP network infrastructure.

This report will discuss the steps followed for the design and development of a TCP/IP-to-RS232 protocol interface device.

Acknowledgements

At this point, I would like to thank a number of people without the help of who I would not have been finish this course.

First, I would like to thank my parents for their psychological and financial support through these demanding years of my life. Secondly, the supervisor of this project, Prof. C. Nomikos for his trust and the very interesting project that I was assigned. Third, Mr. Grigoris Koulouras, a PhD student under supervision by Prof. Nomikos, for his most valuable help in defining the specifications of this project and providing me with information on the protocols to implement.

Finally yet importantly, I would like to thank Mr. John Ellinas and Mr. Panagiotis Drosinopoulos, both of them professors during my undergraduate studies at the Technological Education Institute of Piraeus in Greece and now my associates, for their cooperation, trust, and support for the last four years.

Contents

List of Tables

List of Figures

Summary

The project is concerned with the research, design, and implementation of a TCP/IP-to-serial port protocol converter in hardware. What is presented in this report has been a brief analysis into the problem domain and a small window view on the work that has been performed.

In the first chapter, a brief introduction to the project is attempted and a background on the idea behind the project is presented.

In the second chapter, background information on the mother project that this project is part of is presented. Actual details remain hidden since they are not required in getting the reader acquainted with what this dissertation contains and in fact, they could puzzle him/her.

In the third chapter, the aims and objectives of this project are laid out to help the reader understand the motives and practices that will drive this project.

In the fourth chapter, the currently changing status of the mother project is presented as an introduction to the requirements that must be considered in the next chapter.

In the fifth chapter, a specifications list is drawn as a starting point in defining what guidelines must be followed in this project. Then with these specifications in mind, the rationale behind the selection of the components to form the project is explained.

In the sixth chapter, the design aspects of both the hardware and software of this project are presented: the development tools used, how they were configured, the protocols that were used and design methods used.

In the seventh chapter, testing and debugging methods and procedures followed through the duration of the development stage of the project are analysed. This is an essential part of the project, depicted to provide proof of operation complying with the specifications set in chapter 5.

In chapters numbered 8 to 10, this project is concluded and comments on plans and developments for the future are made. Personal thoughts are presented on the effect this project had on the writer of this dissertation.

Chapter 11 includes brief information on how time was partitioned for this project and how each task was scheduled to occur.

Bibliography and references used as study material for this project are given in chapter 12 categorised according to their application area in the project.

The appendix contains all other material that was not possible to place in the main body of this dissertation and should not be excluded.

1. Introduction

When DARPA (Defense Advanced Research Project Agency) of the Department of Defense of the United States of America created the first large network of computer facilities located in various research institutes more than 35 years ago, no one could ever even dream how this project would evolve. After many breakthroughs in the computer and communications sciences and industries, the first large network grew up from a few nodes to include tens of millions connected together to what is now called "The Internet" (Ref. [9]).

The explosive impulse in the growth of the Internet was given by the TCP/IP protocols suite (Ref. [10]). The Internet Protocol (IP, Ref. [11]) is a network layer protocol and deals with point-to-point routing of packets and establishes an unreliable platform for transferring messages that have no interdependence to each other. The Transfer Control Protocol (TCP, Ref. [12]) is a transport layer protocol which establishes reliable communication sessions through data packets called datagrams. These datagrams include information that relates each datagram to another and data integrity control information. Each node on the Internet has a unique 32-bit IP address by which it can be identified in the network and transmit and receive network packets. Each node has implemented in software internally addressed by TCP a number of data exchange boxes that are called ports. Each port is identified by a 16bit number giving a total of 65536 ports. Network applications use ports to listen for data or send data to another application/node on the network. The combination of an IP address and a TCP port number is called a socket.

Windows and UNIX based operating systems use a basic network-programming interface that is called the Socket API (Refs. [15] & [16]). In both operating system worlds, these APIs are very much standardised allowing applications executing on different systems to talk to each other. The TCP/IP protocols and the Socket API have allowed system interoperability to become the greatest driving force of all networks today, either small or large.

In the past decade, a tremendous interest in the electronics industry to enable new products to connect to the Internet has been observed. Nowadays one can even find home appliances such as refrigerators and ovens that connect to the Internet. However, the most important challenge is to refurbish legacy equipment that has been designed for specific purposes to have network connectivity (Ref. [8]). The cost of replacing this equipment, such as various industrial controllers, is prohibitive and in some cases, there is no rationale in doing so.

Controlling legacy devices over a computer network carries many advantages:

- Share a common wiring infrastructure with telephone and computer network building wiring.
- Distance between devices can be extended from a few meters to the other side of the globe.
- Revival of technologies that were being abandoned by limiting its use on one side of the network interface while replacing the intermediate protocols with newer ones.
- In the case of computer-operated machinery, a modern powerful computer by use of concurrent or parallel execution of tasks can centrally control an entire production line or a factory.

In all cases, control and reduced maintenance cost of legacy equipment is the driving force (Case Study, Ref. [20]).

The most popular communications interface between computers and other equipment is the serial port. Many standards exist, however the RS-232C (Refs. [17] & [18]) can be found on all personal computers. Special protocol converter devices are used to provide these functions (Ref. [19]).

Various connection schemes are possible however, the most popular are:

- Connection between legacy equipment over the network

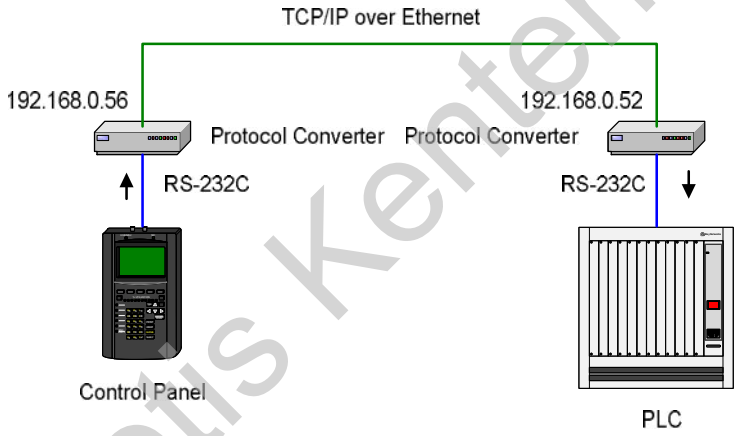

Figure 1 Device to Device connection over the network

- Connection between legacy equipment and PC over the network

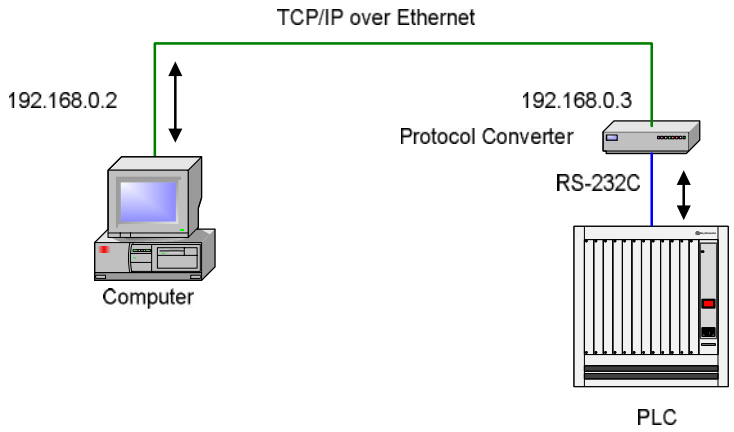

Figure 2 Device to PC connection over the network

This project shall deal with designing a protocol converter for the latter case.

1.1. Project Background

The mission of this project is to redesign from scratch the communication section of an existing system for optimized functionality. The system currently used applies telemetry techniques for scientific research. It was developed to cover the need of collecting seismic data over a leased line network for the Geodynamic Institute of the National Observatory of Athens. The network topology appears in Figure 3 below. This is a seismological research project being run by Professor C. Nomikos, who is also the supervisor for this MSc project.

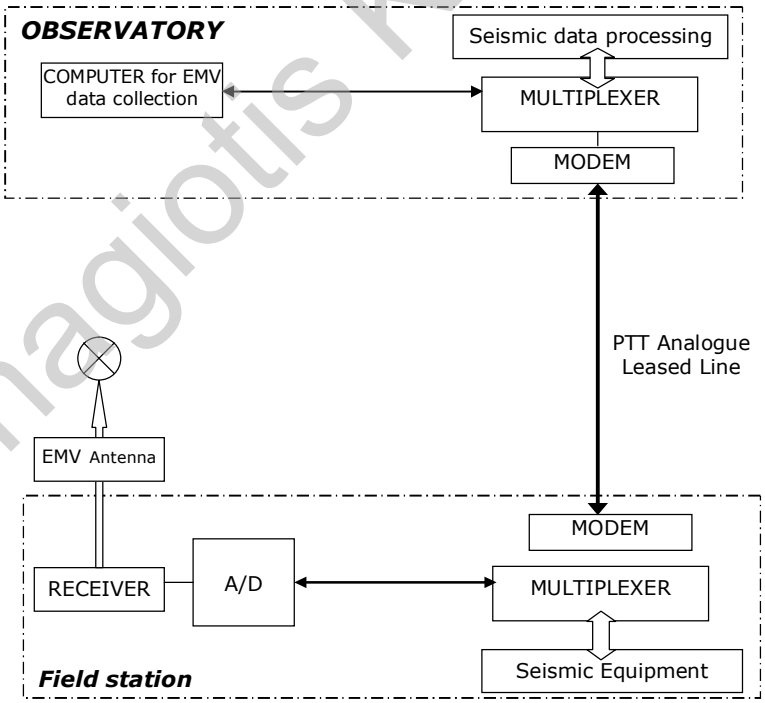

Figure 3 Current Network Topology

The overall system consists of eleven field stations scattered throughout the Greek domain and one controlling station located at the central installations of the National Observatory of Athens. The field stations are equipped with data

acquisition modules measuring variations in the earth's electromagnetic field with the aid of sensitive antennae tuned to specific frequency ranges with a sampling rate of 1 sample/sec. Data acquired is transmitted immediately to the central station through the use of statistical multiplexers and leased line modem links. The central station is responsible for gathering the data forwarded by each field station and stores it locally in a database. From this database, specially coded programs will parse data and apply scientific processing to produce results that could prove valuable in the prognosis of seismic activity in the area where the field stations are located.

Figure 4 Geographical distribution of field stations (N.O.A. Stations)

1.2. Project Aims and Objectives

At the start of the project, some aims and objectives must be set to guide our steps in working out solutions.

1.2.1. Aims

The aims of this project are to:

1. Satisfy the need for the described interface device for scientific research;
2. Involve a wide range of expertise from the MSc course modules in the design, production and formative testing of the proposed system;
3. Build a network interconnection system up to a fully functional level;
4. Demonstrate a practical implementation of communication systems in action;
5. Comment on architectures and methods used in real-time and data-critical systems;
6. Gain valuable knowledge on the low-level design and operation of network systems and protocols as well as on the hardware and software design of communication systems.

1.2.2. Objectives

The objectives of this project are to:

1. Become familiar with the practical aspects of data communication networks;
2. Study the principles of TCP/IP networks and information interchange;
3. Understand the practical issues in dealing with networked systems on the hardware level;
4. Gain self-confidence, maturity, and self discipline as a member of a research team and be able to communicate effectively with other co-workers;
5. Enhance the level of competence in carrying out the learnt knowledge from the entire MSc course and understand the practical limitation;
6. Extend current knowledge in microcontroller based systems;
7. and be able to develop a commercially realistic application in an economic and cost effective manner.

2. Project Evolution

With hopes of independency from the N.O.A. administration, it is essential that currently established communication infrastructure is abandoned and other solutions adopted. All around the globe there is a tremendous change with classical analogue or digital proprietary telecommunication infrastructures being replaced by TCP/IP networks. The greatest of such networks is the Internet. There is a list of advantages in developing a TCP/IP implementation of the currently working system.

Moving on to a TCP/IP networking solution permits:

1. adaptive flow control without need for operator intervention;
2. adaptive routing of data packets;
3. increased data integrity assurance through the use of Cyclic Redundancy Checks;
4. lower cost compared to leased lines when data is not time-critical or there is no need for 'always-on' links;
5. use of open protocols over TCP/IP protocol stack;
6. QoS can be guaranteed if paid for at the link provider;
7. IP networks by use of different transmission mediums (telephone copper wires, optical fibres, mobile & satellite links) are becoming omni-present.

The proposed network topology is depicted in Figure 5 overleaf.

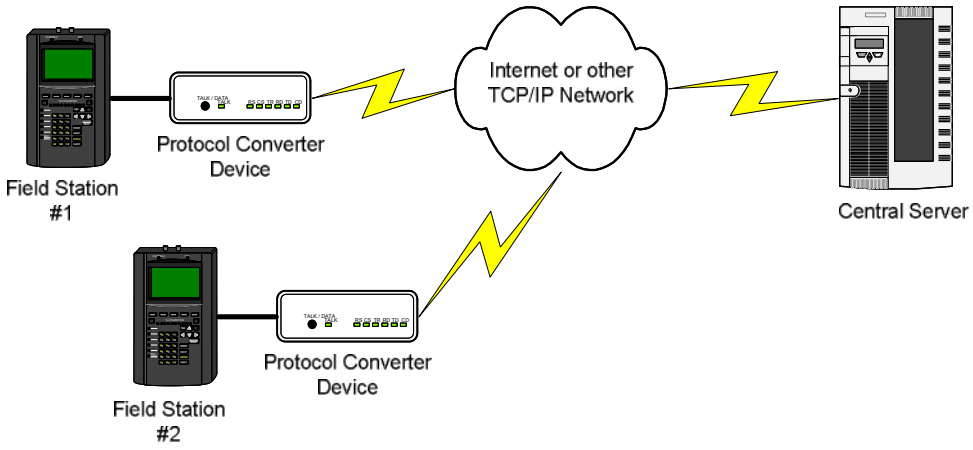

Figure 5 New network topology

In the proposed network scheme, the obsolete field stations are replaced with new ones carrying increased intelligence. A PhD student under supervision of Prof. Nomikos is currently developing a data acquisition and logging system to be used as part of Prof. Nomikos' seismic research project. Data acquired will be stored in some form of digital data storage unit and transmitted to the central station whenever it is appropriate. In this implementation of the system, the data logging equipment will be placed at various seismic hot spots across Greece, where Internet connection will be available by local research or Technological Education Institutions. This project will concentrate on the transfer of acquired data from a number of such remote stations over the Internet to a central processing server.

Each field station of the new system will be responsible for establishing a communication channel with the central station in order to transmit collected data temporarily stored in it for further storage on the central database server. At leisure time, data will be processed for feature extraction.

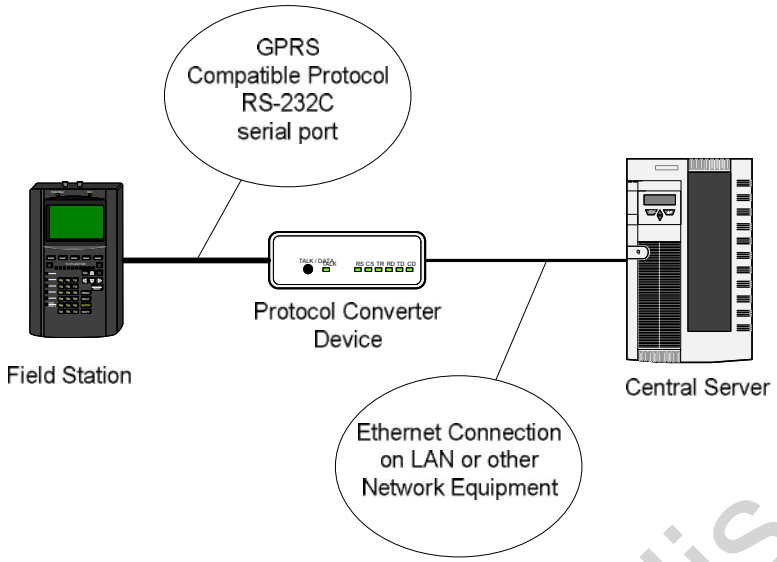

Figure 6 Protocol Converter Connections

At the time this project was starting another one was being run in parallel, for which the TCP/IP connection would be alternatively become available by use of mobile communication devices such as GPRS modems (General Packet Radio System). In order to allow a single data logging module to be used in field stations for both projects, this project needs to be able to handle the same protocols used by the GPRS module to establish connections and transfer data. The protocol used in this module incorporates a modified AT command set (similar to that used in PSTN modems) which supplies all the control logic and information required to establish a TCP/IP socket connection and transfer data over the airwaves. In this case, GSM network coverage is required in the installation area of the field station, which allows the presence of field stations even on geographical locations unreachable by wire networks.

3. Project Specifications

Though this project started with the intention of designing a "home-made" solution for the problem in question, research on available solutions in the market was essential in defining the capabilities and specifications of the device to build. Devices found available in the international market (Ref. [1]~[7]) appear to be more than fully equipped for use in the system, however none appears to cover the special needs of it. Cost, power consumption, and functionality factors are not directly addressed by these devices. Therefore, a custom solution is undeniably a one-way road to fulfilling requirements for the project.

The device to be built needs to be able to carry most of the following recommended specifications:

1. Allow a short development period to meet the deadlines set by the duration of this MSc course.
2. Low cost implementation to meet small overall budget for the project. Cost of materials and development tools needs to be considered and expenses kept at low levels.
3. Incorporate a high-speed serial link to the data acquisition equipment of the field station using standard baud rates.
4. Make use of well-established widely known communication protocols. The GPRS modem AT command set needs to be emulated by the device for increased compatibility between projects and open connectivity of the field station.

5. The software to be developed needs to incorporate "crash" or "hang" avoidance techniques to provide increased Mean-Time-Between-Failures (MTBF) and Mean-Time-Between-Repairs (MTBR) figures. This is a requirement of utter importance, since the field stations will be installed in remote locations where maintenance crew might be extremely hard to be present in case of complications with normal operation of the station.
6. The hardware of the device needs to consume as little power as possible. This is another requirement pushed by the fact that field stations will be placed in areas where electric power supply might be absent or interrupted regularly. Also in the undesired event of seismic activity, field stations should be self-reliant in respect to electric power the provision of which cannot be guaranteed. The device is required to be able to be powered by either batteries or other form of low-efficiency power sources, such as photovoltaic cells. One way to enforce this design requirement is to remove any components or circuits that unnecessarily draw current from the power source. In addition, every component must be chosen in its low power version.
7. Establish TCP/IP connections through a standard programming application interface for ease of programming and debugging. Proprietary solutions must be avoided for reasons of programming complexity.

Summarizing the above from an engineering side of view, the proposed hardware is to be designed as a microprocessor based system with sufficiently low power requirements and manufacturing cost. Two connection interfaces must be available to the user, an Ethernet 10/100Mbps port supporting most TCP/IP family protocols, and a minimal RS-232 serial port. Transfer of data must be performed using TCP datagrams in a client-server software model.

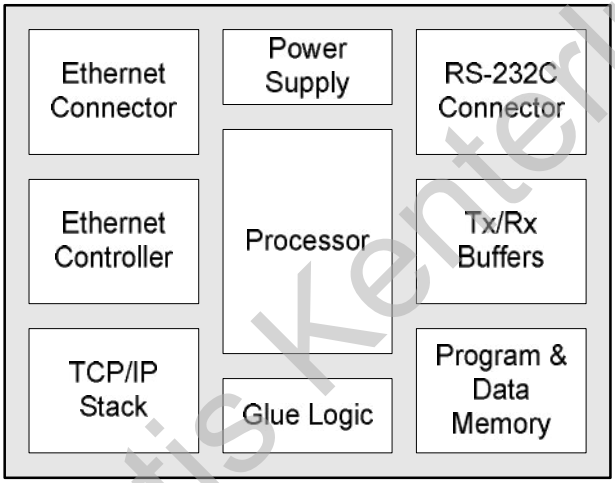

Figure 7 Draft System Schematic

3.1. Justification of Component Selections

After having drawn an initial project specifications outline, the technology behind every suggestion needs to be defined. The following pages describe how decisions were made towards finding the right parts for the project.

3.1.1. The Microcontroller

For the device to have the intelligence required to fit the purpose of this project, a data processing unit is required. Using a standard personal computer processor would add unnecessary interfacing and programming complexity because of the peripheral components required and their internal structure. Additionally the development and cost of materials would greatly exceed the budget. Using a Pentium processor for a system with such low processing power requirements would be overkill. A smaller and more practical solution is required and that can only be found in the microcontrollers' domain.

Choosing the heart and brains of the device could not be an easy task. With the writer having much experience with 8051 microcontroller derivatives, deciding to follow another path could prove to be an unnecessary risk. Nevertheless, expansion of knowledge was a tempting call. Having some years ago acquired a starter development kit for another microcontroller family, that of Atmel's AVR, but never having it put into real use was a hidden desire begging to come true. The path was now drawn and a suitable target microcontroller had to be selected.

The Atmel STK200 Development Kit

The STK200 is an ideal starter development kit for people trying to grasp the capabilities of the AVR microcontrollers while building their own applications using the facilities provided (Ref. [22]).

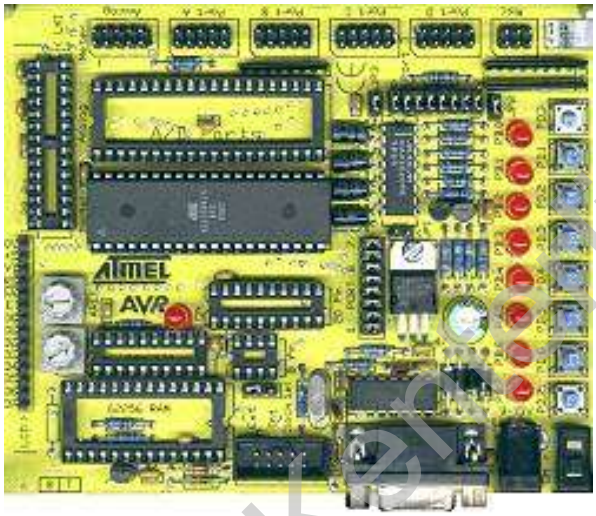

Figure 8 ATMEL STK200 Starter Development Kit

The STK200 connects to a personal computer using two of its ports. First, a standard parallel port for the dongle device with which programming of the microcontroller placed on the board is possible. Secondly, a standard serial port (RS-232C) which connects the on-chip serial port of the microcontroller with the personal computer thus allowing exchange of data and control commands from one end to the other.

The STK200 was originally designed for use with the AT90 series of AVR microcontrollers; however, it can accommodate any other compatible AVR microcontroller on DIP40, DIP28, DIP20, and DIP8 IC sockets. The STK200 is now obsolete and has been replaced by STK500.

Specifications list strictly dictated that the system would have to be as power efficient as possible. For this reason, a low power 3.3V microcontroller derivative would be required. In addition, to reduce design and implementation costs, the microcontroller would have to include as many required peripheral chips with in its package as possible. These mainly involve the program memory, data memory, and non-volatile data storage (EEPROM) that would have to be adequately large to accommodate the device's firmware. Second in order of importance, but not to be taken lightly, is the presence of an on-chip UART, which will be used for connection to the serial port of the field station's data acquisition equipment. The microcontroller to be used must also come equipped with enough I/O pins to use as address and data bus and control lines for external peripherals. The more self-contained the microcontroller and its firmware the less the external design considerations will be.

Searching on Atmel's web site for their line of microcontroller products it became clear that the AT90 product series that the development board was designed for was becoming obsolete. To avoid future problems the ATmega series would provide the candidates for the processing hardware position in the system. Compatible candidates needed to be picked up from a long list of products. Those falling inside the specifications, which are also available in the Greek electronics market, are given in Table 1 in the following page.

Device	ATmega128L	ATmega64L	ATmega32L	ATmega16L	ATmega162L	ATmega8L	ATmega8535L	ATmega8515L
Flash (Kbytes)	128	64	32	16	16	8	8	8
Vcc (V)	2.7-5.5	2.7-5.5	2.7-5.5	2.7-5.5	2.7-5.5	2.7-5.5	2.7-5.5	2.7-5.5
SRAM (bytes)	4096	4096	2048	1024	1024	1024	512	512
EEPROM (Kbytes)	4	2	1	0.5	0.5	0.5	0.5	0.5
F.max (MHz)	8	8	8	8	8	8	8	8
Max I/O Pins	53	53	32	32	35	23	32	35
Ext Interrupts	8	8	3	3	3	2	3	3
16-bit Timer	2	2	1	1	2	1	1	1
SPI	1	1	1	1	1	1	1	1
UART	2	2	1	1	2	1	1	1
8-bit Timer	2	2	2	2	2	2	2	1
Package Options	TQFP64	TQFP64	PDIP40, TQFP44, MLF44	PDIP40, TQFP44	PDIP40, TQFP44	PDIP28, TQFP32, MLF32	PDIP40, TQFP44	PDIP40, TQFP44

Table 1 Parametric Comparison Table for AVR ATmega microcontrollers

From the candidate microcontrollers of Table 1 the one that was originally chosen because of its full featured configuration was the ATmega8515 (Ref. [23]). It operates at the range of 2.7-5.5V and comes with sufficiently large memories for the firmware. It was also one of the only ones found in PDIP40 package, which could be fitted in the STK200 kit.

However, during the code development stage it became clear that the small program memory of 8Kbytes that the ATmega8515 came with would not suffice the full application code. Another microcontroller had to be used, one with more program memory. Looking back on Table 1 and searching once more on the electronic parts stores, a more viable solution was found. The microcontroller to replace the ATmega8515 was none other than the ATmega162 (Ref. [24]), which offers twice the capacity on program, data, and non-volatile memories. This increase in internal memory would prove invaluable for the execution of the microcontroller code. With larger data memory, larger communication buffers can be implemented and more data can be processed immediately. Both microcontroller devices are compatible at the instruction set level and thus replacement causes no software disarrangement (Ref. [25]).

3.1.2. The TCP/IP Stack

When considering the computer network portion of the device, it became clear there were two paths that could be followed: either implement a software TCP/IP stack or make use of existing hardware TCP/IP stack solutions. Deciding which path to walk is not an easy task, engineering knowledge and expertise has been applied. The pros and cons of each path are analysed in the following pages in the form of weighted decision matrices.

First, it is important to identify the criteria by which the selection will be justified.

		A	B	C	D	E	F	G	De-norm	Normalized
Time required for development	**A**	=	+	=	+	+	+	+	+5	**+10**
Additional hardware components required	**B**	-	=	=	=	=	=	=	-1	**+9**
Programming Complexity	**C**	=	=	=	+	+	-	-	0	**+5**
Interfacing Complexity	**D**	-	=	-	=	-	=	-	-4	**+1**
Power Consumption	**E**	-	=	-	+	=	-	-	-3	**+2**
Cost per unit manufactured	**F**	-	=	+	=	+	=	+	+2	**+7**
Development cost	**G**	-	=	+	+	+	-	=	+1	**+6**

Table 2 Device Selection Weights Matrix

Time required for the development of this project is crucial since strict deadlines have been placed beforehand from the university and must not be exceeded. Using more hardware components to the project not only adds to the cost of materials but also increases design complexity and thus requires more

money and time. Programming complexity refers to the complexity of the firmware to be placed on the device. If the firmware is too complex to build, it will require much development and debugging time. Interfacing complexity refers to the difficulty in connecting together the components. If originally the interfacing complexity is high, more hardware or software will be required to proceed with development. Power consumption as mentioned in the specifications outline should be kept as low as possible. Cost is also a very important factor. Some solutions will require financing only at the development stage, while others will extend expenses on the manufacturing stage as well.

	Weight	Description	Points	Weighted Points
Time required for development	**A (+10)**	Short Time (2-3 weeks)	+3	+30
		Medium Time (3-4 weeks)	+1.5	+15
		Long Time (more than 4 weeks)	-2	-20
Additional hardware components required	**B (+9)**	None	+3	+27
		1-2 additional components	+2	+18
		More than 2 components	-1	-9
Programming Complexity	**C (+5)**	Short functions	+3	+15
		Large functions	+1	+5
		Large functions & Hardware control	-3	-15
Interfacing Complexity	**D (+1)**	Parallel Address/Data Bus	+5	+5
		Serial Interface	+2.5	+2.5
Power Consumption	**E (+2)**	Up to 10mA	+3	+6
		Up to 20mA	+2	+4
		Up to 30mA	+1	+2
Cost per unit manufactured	**F (+7)**	Small	+3	+21
		Average	+2	+14
		Large	-1	-7
Development cost	**G (+6)**	Small	+3	+18
		Average	+2	+12
		Large	-1	-6

Table 3 Analysed Weighted Selection Criteria

Software TCP/IP stack solution is split into two categories. In the first category we find the TCP/IP software stack that is to be built by the designer of the project himself. Attempting to do so will require extreme amounts of time to successfully and fully code the TCP/IP state machine. In addition some external hardware components and their respective software drivers are required to interface the device with the Ethernet. These components are not only expensive and hard to find in the Greek electronics market but also difficult to place on a board. Implementing a commercial TCP/IP software stack although it has minimum development time, at a significant price of course, it may require paying royalties to the stack designer for each device manufactured.

Weight	Home-brew Software TCP/IP Stack	Commercial Software TCP/IP Stack	Hardware TCP/IP Stack & Ethernet Module
A	Extremely Long Development Time	Short or no development time	Short or no development time
B	More than 2 components for Ethernet PHY	More than 2 components for Ethernet PHY	None (already on module)
C	Large functions and excessive hardware control	Large functions and excessive hardware control	Short functions (hardware drivers)
D	Parallel Address/Data Bus (Ethernet PHY)	Parallel Address/Data Bus (Ethernet PHY)	Parallel Address/Data Bus (Entire module)
E	Estimated up to 10mA	Estimated up to 10mA	Estimated up to 30-40mA
F	Small or none	Depending on royalties paid to TCP/IP stack designer	Average to large depending on the cost of each module
G	Small	Average	Large

Table 4 TCP/IP Solutions

Hardware TCP/IP stack solutions offer minimum development time, without requiring any external components, since everything required is already available on a single chip or module. Most hardware solutions come with freely available driver functions to use with the target system. In this case, the time required to produce the end application code is brought to a minimum relieving the designer from tiny details and enabling him/her to deal only with the important aspects of the application. Advantages offered by use of a hardware TCP/IP stack solution are countered by increased cost of each device.

Considering the above, the following decision matrix was drawn to weigh each possible solution and discover which one covers the needs of this project.

	A	B	C	D	E	F	G	Total Points
Home-brew S/W TCP/IP Stack	-20	-9	-15	+5	+6	+21	+18	**6**
Commercial S/W TCP/IP Stack	+30	-9	-15	+5	+6	+14	+12	**43**
H/W TCP/IP Stack & Ethernet Module	+30	+27	+15	+5	+2	-7	-6	**66**

Table 5 TCP/IP Stack Solution Weighted Decision Matrix

As justified from Table 5 above the solution for the TCP/IP stack problem is to be found in the commercial hardware domain.

After careful and extensive search on the Internet on such hardware modules (Ref. [21]), the one selected for use is described in the following pages. Due to lack of space, there can be no comparison between all hardware TCP/IP stack modules found and only the winning candidate is described. The reason that led to selecting this component over others is the fact that it provides limited yet standard communication functionalities at a small cost.

The Wiznet IIM7010A module (Ref. [26]) is a combination of a hardwired TCP/IP stack, a 10/100Mbps Ethernet PHY, an Ethernet MAC Jack, and Rx/Tx buffer memory. It comes in the form of a self-contained micro-board module as seen in Figure 9 below.

Figure 9 The IIM7010A Module

Connection to the microcontroller unit is quite easy owing to the full address/data bus, which allows the module to be connected just like any other memory or peripheral chip. Its 15bit address bus will require a total address range of 32Kbytes on the microcontroller's memory map.

On the communications aspect of the module's operation there are four programmable socket channels. These channels can be programmed through a socket API as in Windows-based personal computer systems. A total of 16Kbytes of dual-port memory is equally split into two regions dedicated for the communication Tx and Rx buffers respectively. Each of these regions can be fragmented into pieces of 1, 2, 4 or 8Kbytes depending on the use of socket channels by the application software. All socket channels can be programmed for use with TCP, UDP or raw data exchange protocol, over standard IPv4 packets and Ethernet frames in the physical layer.

The most important feature of the IIM7010A module is the freely available code for rapid application development in microcontroller based systems.

4. Project Design

This project can be partitioned into two design areas, the Hardware Design and Software Design. In this chapter, both partitions will be presented and analysed as much as possible without reaching into much depth. For more in-depth information on modules and functions, please consult the datasheets that are found in the accompanying CD-ROM.

4.1. Hardware Design

For reasons of low power consumption, the entire device is internally powered by a 3.3V voltage regulator and all major components and glue logic circuits have been selected to work at this voltage. The microcontroller is clocked at 8MHz which is the highest frequency it will work at when powered at 3.3V according to its datasheet.

Glue logic refers to all the small circuits that play an important role in connecting major components together. The entire glue logic circuit consists of a single standard CMOS 8bit latch chip (74HC573).

The lower portion of the microcontroller's address bus is multiplexed with the 8bit data bus and an external latch chip is required to de-multiplex these buses. The latch is controlled by the ALE signal of the microcontroller, which latches

the lower byte of the address word on the chip. After that is done, the I/O lines serve as data bus.

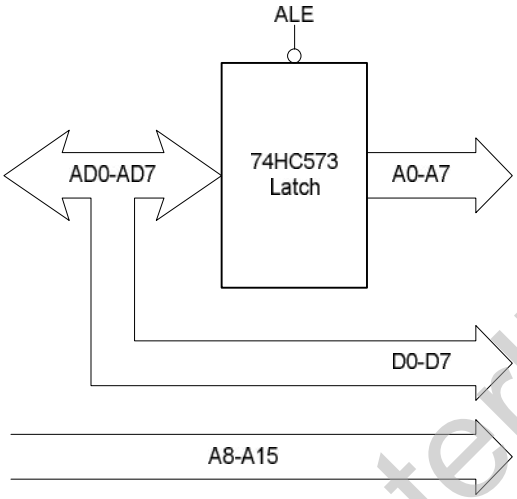

Figure 10 Address/Data bus de-multiplexing

The IIM7010A module responds to the address range of 0x8000-0xFFFF (32Kbytes). Normally at this address range the A15 pin of the address bus has a logic value of '1'. Standard address decoding circuits would require that this pin is inverted and then fed to the /CE input of the chip to enable when any address value in this range appears on the address bus. However, since the presence of an inverter gate would require an entire 74HC04 chip that carries 6 inverters there would be too much waste of materials, power, PCB space and increased cost. The solution given was to completely ignore the A15 line and connect the /CE input of the IIM7010A module to ground. The module would then be permanently enabled. This does not constitute a problem to the device since no other external peripheral chip connects to the address/data bus and no collisions will be witnessed. Considering the fact that the firmware code makes no random memory accesses the action of removing a single chip is not to cause fear. This assumption was also experimentally tested and verified.

4.2. Software Design

Though this is by name a hardware project, in reality the software portion of it is far greater and complex, thus it deserves a fair amount of attention.

4.2.1. Development Tools Used

For the development of the microcontroller's resident firmware code, the AVR-GCC compiler was used. This is a freely available open source C compiler written specifically for Atmel's AVR microcontroller family members. AVR-GCC was chosen among other commercial products primary due to its zero purchase, installation, and maintenance cost. Though AVR-GCC is publicly available without a full featured Integrated Development Environment (IDE) as other commercial products do, there are many freeware IDE applications available on the Internet that can be customised for use with AVR-GCC. As the IDE application for the purposes of this project, Programmer's Notepad was chosen (not to confuse with Windows Notepad).

The following pages demonstrate how to configure Programmer's Notepad IDE for use with the AVR-GCC compiler. See Figure 12 to Figure 14 for the procedure to add the menu items that allow compiling source files from inside the Programmer's Notepad application. This operation requires that a valid "*makefile*" file is present in the path where the source files are located on the local hard disk drive. The *makefile* file used for this project can be found in the accompanying Compact Disk and in the appendix.

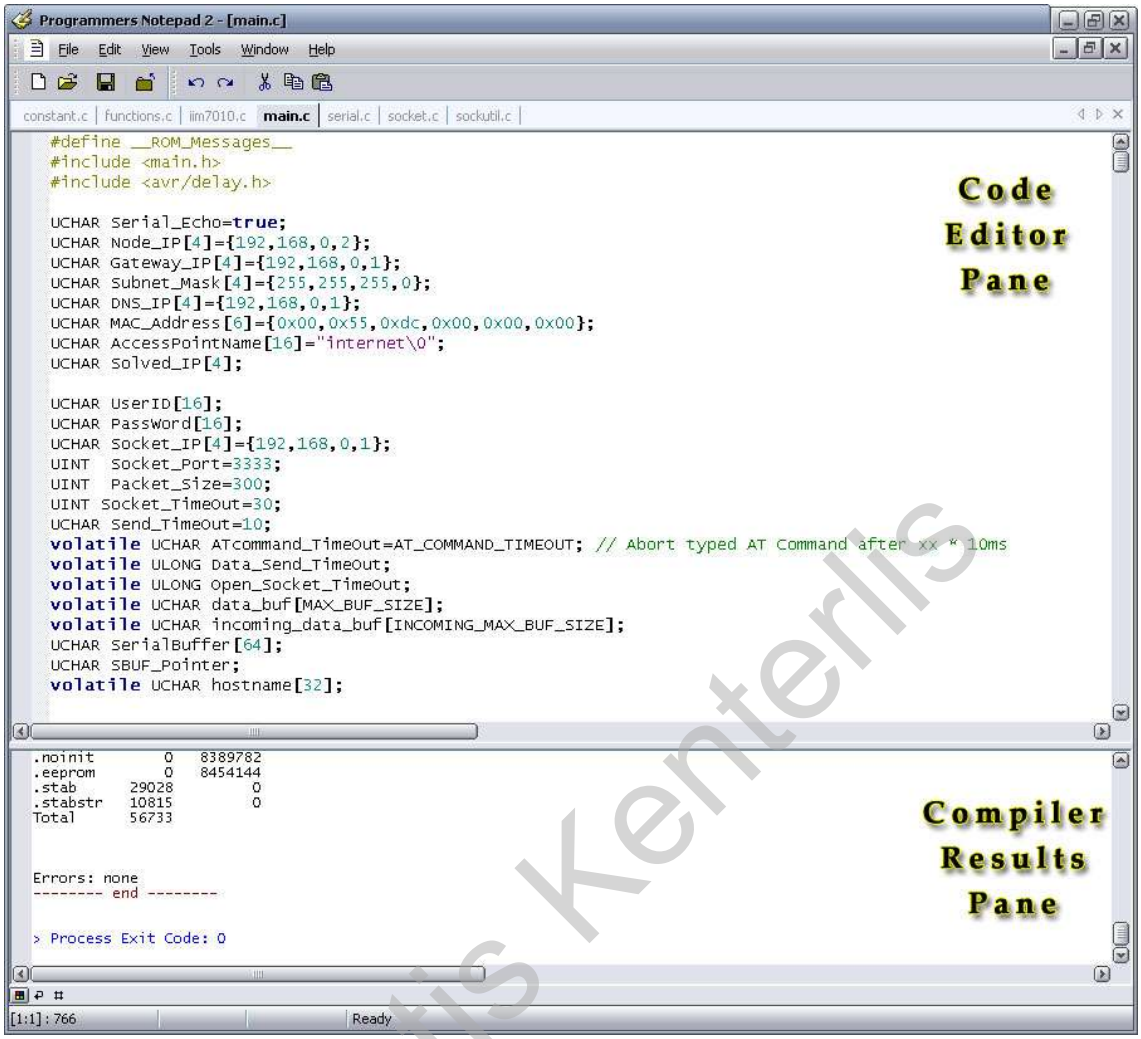

Figure 11 Programmer's Notepad Application Window

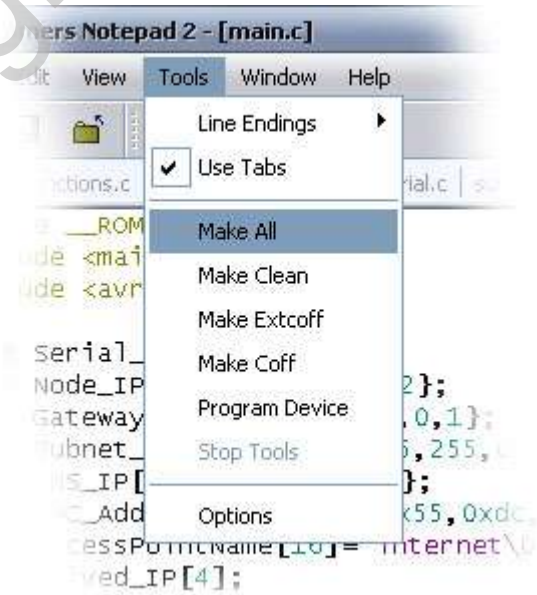

Figure 12 Tools Menu

From Tools→Options the following window appears and menu items in the Tools menu can be created or edited.

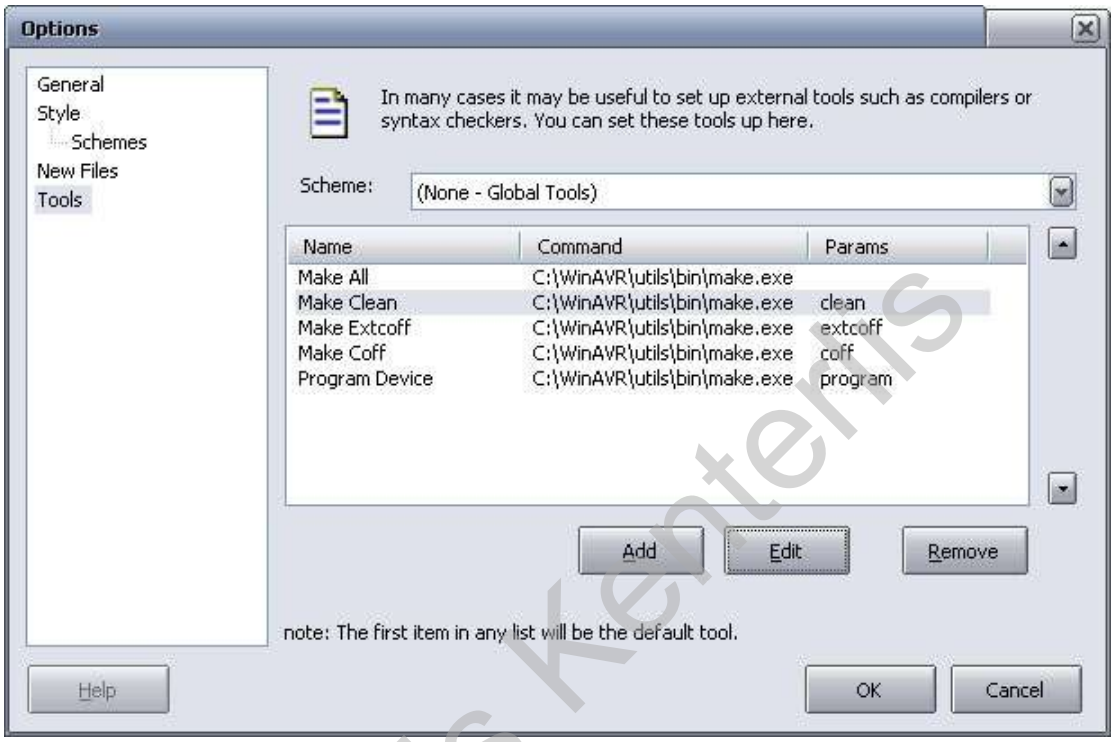

Figure 13 Configuration of Tools Menu

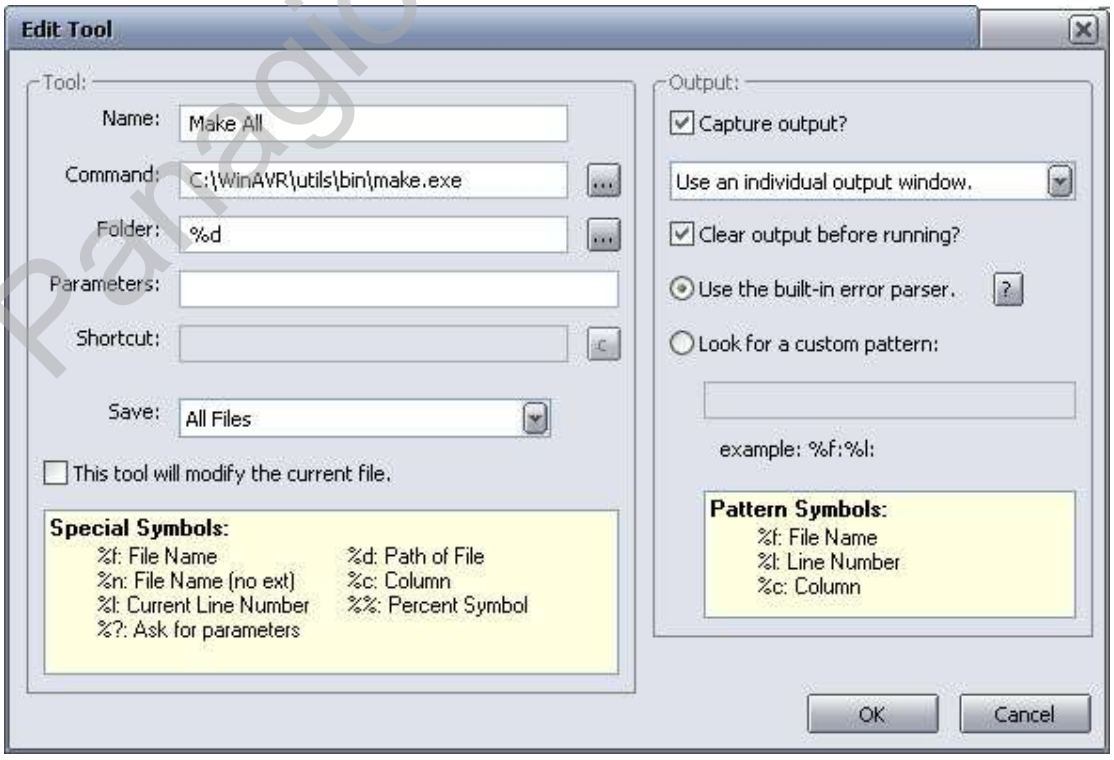

Figure 14 Menu Option Edit Window

For programming the microcontroller on the STK200 development board the PonyProg2000 freeware program was used. A capture of the main window of PonyProg2000 is displayed in Figure 15 below. In the MDI child window of the applications the contents of the Intel HEX file to be programmed on the microcontroller are presented in hexadecimal and ASCII form next to the starting address of the memory location to be written. Using this window, checking for code consistency with source files in memory locations used by constant values and interrupt vector tables became possible. Verifying the programming procedure was available and used to check for device faults or programming failures.

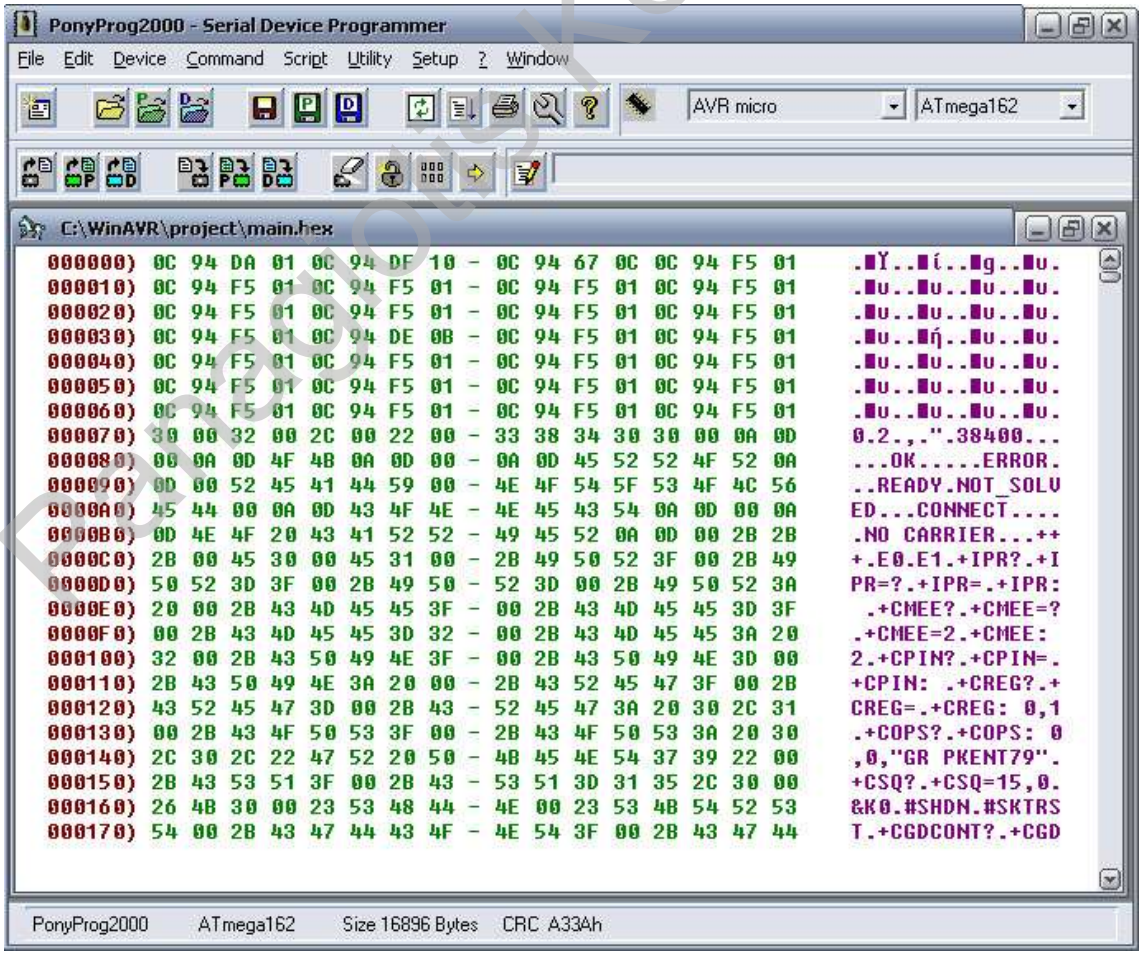

Figure 15 PonyProg2000 Work Environment

In order for PonyProg2000 to talk to the microcontroller on the STK200 development board, the former has to be configured accordingly. In Figure 16 below the Interface Setup window is displayed with the options required for the STK200 to function properly when a programming operation is requested.

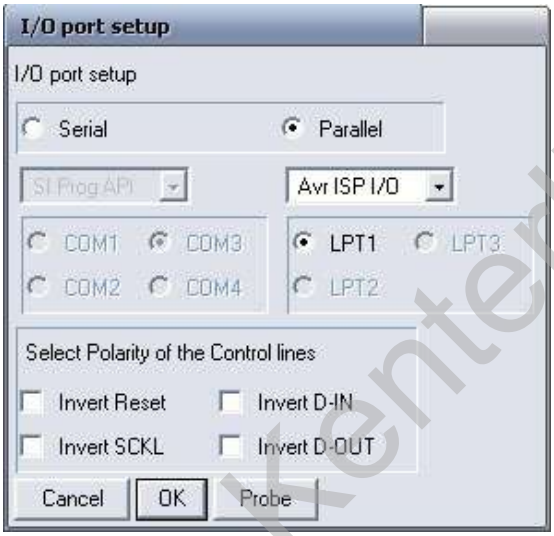

Figure 16 PonyProg2000 setup options for STK200

After having configured the programming interface that PonyProg2000 will communicate with, it is important to select the target device used on the STK200 development board. This step must not be overlooked. The programming algorithm of PonyProg2000 needs to know the exact device to program in order to generate the correct timing and control sequences for the Intel Hex file to be successfully downloaded to the device. The device selection is easily performed by selecting the correct one from the list as seen in Figure 17. After this procedure has finished, another important step is to configure the microcontroller with the programming of the configuration bits as in Figure 18. CKDIV8 needs to have a value of '0' to disable the internal prescaller and prevent the microcontroller from running at 1/8 of the external clock frequency.

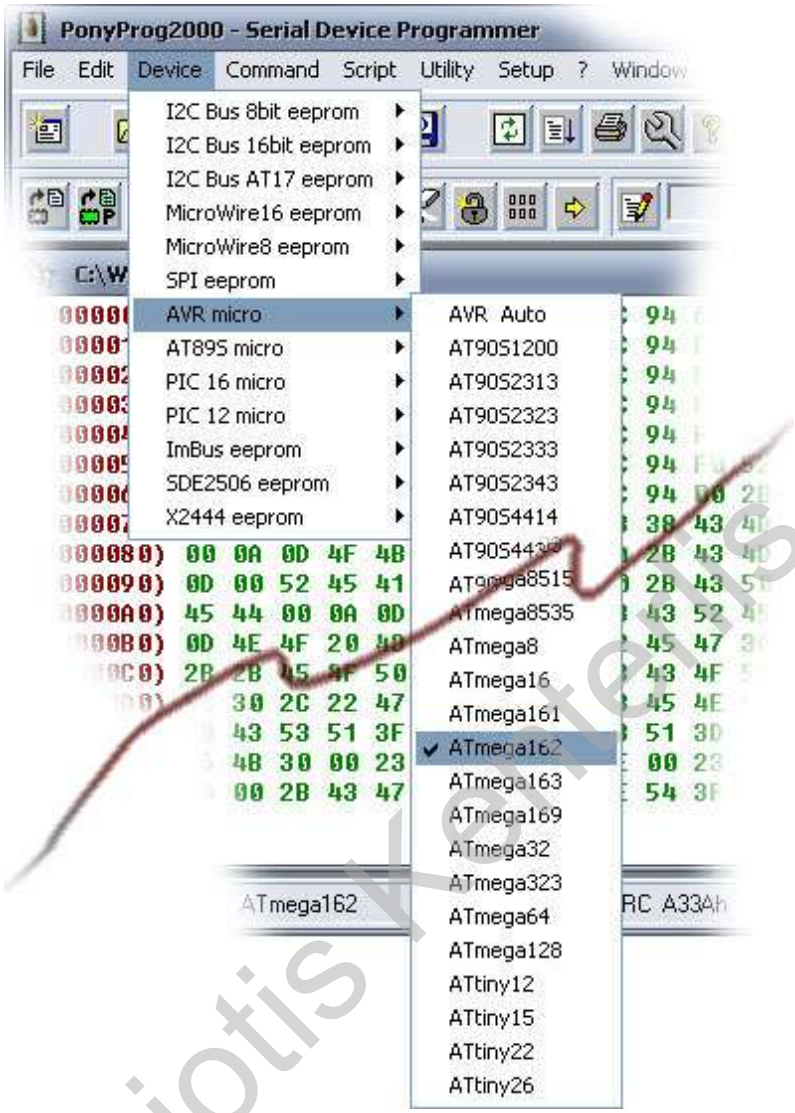

Figure 17 Setting up for use with the target microcontroller

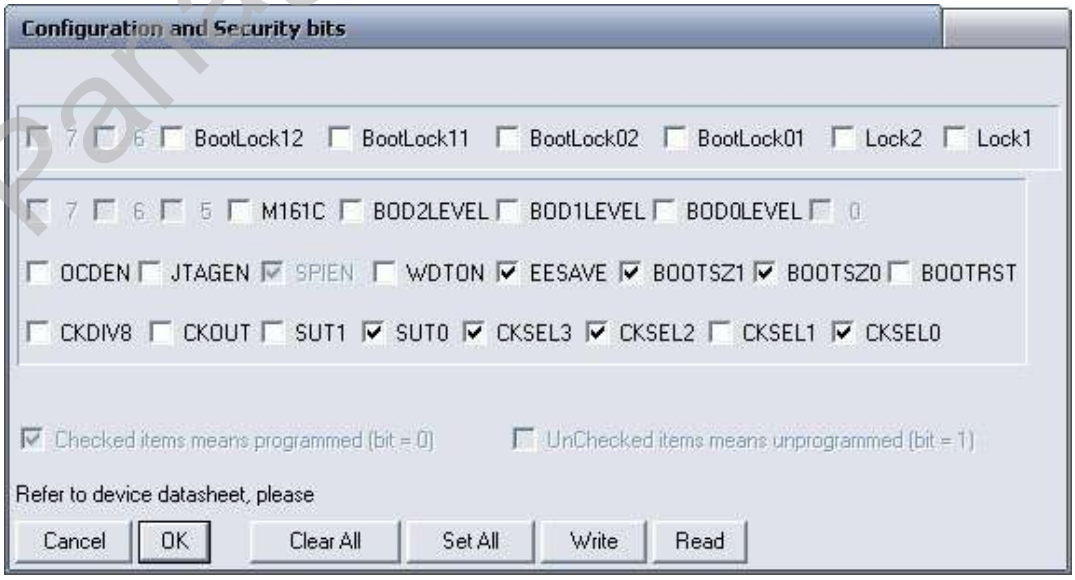

Figure 18 Settings for Configuration and Security bits of the microcontroller

4.2.2. Microcontroller Firmware Code Explained

The finalised firmware code makes use of all internal memory facilities of the microcontroller. The program memory also accommodates numeric and string constants as well as default values for configuration parameters. The data memory is heavily used for local function variables, global variables, and Tx/Rx buffers for both the serial and Ethernet communication ports. The non-volatile EEPROM memory is used to store operation parameters such as socket information, device configuration, etc. This information is kept in place even when the device is supplied with power, and so they can be recalled at any time without data loss.

The on-chip serial port is configured to work at a baud rate of 38.4kbaud, with data frames of 8bits, 1 stop bit and no parity bit. The parallel address/data bus used to connect the IIM7010A module is configured for operation with no wait-states to increase data transfer rates. The 16bit Timer/Counter T1 is programmed to provide an internal interrupt-generating clock with a period of 10ms using a prescalled input clock of the external crystal. This periodic interrupt is used at various points in the code to implement time-out counters or fixed time delays. The two external interrupt sources of the microcontroller, INT0 and INT1 are used for connection to the interrupt output of the IIM7010A module and as a wake-up signal when the device is in power down mode, respectively. The microcontroller will enter power down mode when the appropriate command has been received and by doing so, its power requirements will fall at extremely low levels.

As mentioned in paragraph 3.1.2, most of the software code required to use the TCP/IP stack module had been already provided by the manufacturing company. Though the entire collection of lines of code was written in C for the 8051 microcontroller, with a few minor modifications it became possible to use with the AVR-GCC compiler (Ref. [27]). This action greatly reduced the amount of development time that would be required had the drivers' code been written by the author from scratch.

Due to the simplicity of communications requirements on the Internet/Ethernet side, one socket channel is sufficient. As specifications dictate, there can be only one data connection path originating from the field station and terminating at the central server. When the designed software is executed making use of its full functions, both TCP and UDP sockets are used, however not simultaneously.

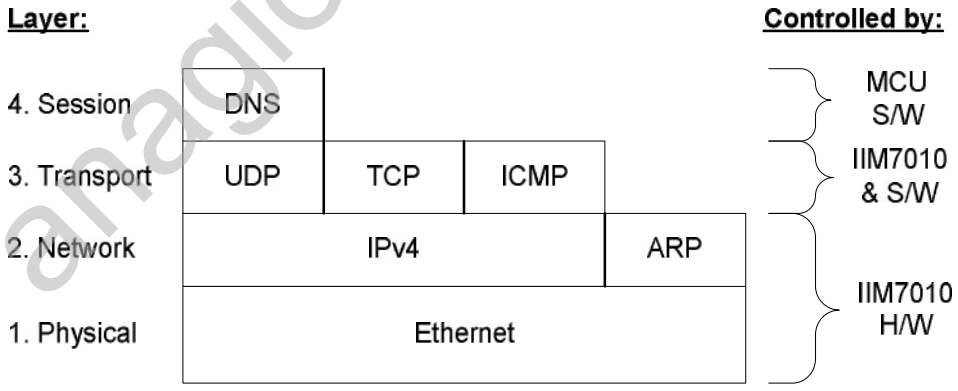

Figure 19 Protocol Stack on device Hardware & Software

TCP datagrams are used for normal data exchange between the field station and the central server. TCP is selected as the transport layer protocol because of its connection-oriented approach of communication links. Each datagram

communicated is acknowledged on the receiver and if there is loss or corruption of datagrams, then re-transmission of packets is performed. Therefore, data integrity is fortified and in combination with even a simplistic session layer protocol then it can be said that the data exchange is always error-free.

For the device to enter data-transfer mode, a computer is required to be listening to the TCP port that the device will attempt to connect. The device acts as a client and the computer as the server. The server can receive connections from many field stations simultaneously and process the incoming data immediately since all modern operating systems support multi-threading or multi-tasking applications.

UDP datagrams are used for querying the DNS server when the field station's software requests the resolution of a hostname to its IPv4 network address.

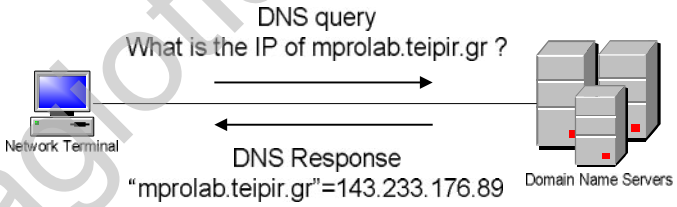

Figure 20 DNS query and response scheme

The DNS query operation is not performed auto-magically, but a series of steps is required. First, the query message must be generated and for that, the remote host's name is required. Next, the query message is encapsulated in a UDP datagram and sent to the DNS server using its IPv4 address. When the DNS server has performed its internal operations, it will respond to the device with either a list of one or more hostnames and their IPv4 addresses, or an error message indicating that the hostname could not be resolved. The

response of the DNS server will be encapsulated in a UDP datagram. For more information on the DNS protocol, see Ref. [14].

ICMP and ARP are internally used respectively to reply to ECHO packets and resolve the MAC address on the local network of the destination IP address. No access to these protocols is possible through the module's drivers.

4.2.3. Communication Control Protocol

- **Definitions**

On the following pages, some terms will be used very often and for reasons of clarity are explained here.

AT command: *"A command that forms part of a Hayes modem control language".*

		Byte Value	
Term	**Description**	**Hexadecimal**	**Decimal**
<CR>	Carriage Return	0D	13
<LF>	Line Feed	0A	10
_	Blank Space	20	32

- **GPRS AT Commands**

Since the Ethernet does not provide many of the services and control procedures that the GPRS network has, many of the following commands that will be analysed have no effect on the operation of the device when being received for execution. Instead, the device responds in a predefined manner to these commands to preserve compatibility with GPRS modems.

The original set of AT commands to implement was drawn by Mr. Grigoris Koulouras, the PhD student developing the data acquisition equipment for the field stations. The document provided by Mr. Koulouras is also included in the appendix as a proof of acknowledging his work in this part of the project. Parts of his work were reproduced in the following pages. Wherever necessary corrections and alterations have been made to reflect the way things work with the device being developed in this project in a more accurate manner.

Command Syntax: AT	
Command	**Valid Responses**
AT Note: Check if device is present on the serial port.	*<CR><LF>* **OK** *<CR><LF>* Note: Device is present and waiting for command.

Command Syntax: ATE	
Command	**Valid Responses**
ATE0 Note: Turns off local character echo. **ATE1** Note: Turns on local character echo.	*<CR><LF>* **OK** *<CR><LF>* Note: Command has been successfully processed.

Command Syntax: AT+IPR	
Command	**Valid Responses**
AT+IPR? Note: Request for the current DTE-DCE baud rate.	*<CR><LF>* **+IPR:_<rate>** *<CR><LF>* **OK** *<CR><LF>* Note: Returns the currently configured baud rate of the serial port. Always returns 38400.
AT+IPR=? Note: Request for the allowed DTE-DCE baud rates to use.	*<CR><LF>* **+IPR:_(38400)** *<CR><LF>* **OK** *<CR><LF>* Note: Returns the allowed baud rate of the serial port. Always returns 38400.
AT+IPR=<rate> Note: Request to change the current DTE-DCE baud rate.	*<CR><LF>* **OK** *<CR><LF>* Note: Dummy command. Always returns OK, but no change is performed on the baud rate of the DTE-DCE link.

Command Syntax: AT+CMEE	
Command	**Valid Responses**
AT+CMEE? Note: Requests the currently configured extended error report mode.	*<CR><LF>* **+CMEE:_<mode>** *<CR><LF>* **OK** *<CR><LF>* Note 1: Returns the current error report mode. 0: No extended error report 1: Extended error report in numeric format 2: Extended error report in verbose format Note 2: Dummy command. Always returns mode=2 (+CMEE: 2).
AT+CMEE=? Note: Requests the currently allowed extended error report modes of operation.	*<CR><LF>* **+CMEE:_ 0,1,2** *<CR><LF>* **OK** *<CR><LF>* Note 1: Returns the allowed error report mode. Note 2: Dummy command. Always returns mode=2 (+CMEE: 2).
AT+CMEE=<mode> Note: Request to change the currently allowable extended error report mode configuration to the one indicated by the <mode> value. 0: No extended error report 1: Extended error report in numeric format 2: Extended error report in verbose format	*<CR><LF>* **OK** *<CR><LF>* Note 1: Response indicates that the command has been successfully processed. Note 2: Dummy command. Always operating in mode 2 (verbose mode).

Command Syntax: AT+CPIN	
Command	**Valid Responses**
AT+CPIN? Note: Requests the current PIN status.	*<CR><LF>* **+CPIN:_<status>** *<CR><LF>* **OK** *<CR><LF>* Note 1: Returns the current PIN status. Note 2: Dummy command. Status always returns the string READY to indicate that a valid PIN has already been provided and no further action is needed to set it by software.
AT+CPIN=xxxx Note: Command to enter the 4-digit PIN code of the SIM Card to gain access to its contents and services.	*<CR><LF>* **OK** *<CR><LF>* Note: Dummy command. Always returns an OK message to indicate that the PIN provided is correct. No actual SIM card is used in the device hence this command has no effect and is used for compatibility reasons.

Command Syntax: AT+CREG	
Command	**Valid Responses**
AT+CREG? Note: Requests the current network registration status.	*<CR><LF>* **+CREG:_<mode>,<status>** *<CR><LF>* **OK** *<CR><LF>* Note 1: Response parameters indicate: <mode> 0=Disable network registration unsolicited result code. 1=Enable network registration unsolicited result code. 2=Enable network registration unsolicited result code with network. <status> 0=Not registered, Mobile Equipment is not currently searching a new operator to register to 1=Registered to home network 2=Not registered but Mobile Equipment is searching a new operator to register to 3=Registration denied 4=unknown 5=Registered to roaming network Note 3: Dummy command response. No registration is required for the device. Status always returns the string +CREG:_0,1 to indicate that it is registered to the home network operator and connectivity is available.
AT+CREG=<mode> <mode> 0=Disable network registration unsolicited result code. 1=Enable network registration unsolicited result code. 2=Enable network registration unsolicited result code with network. Note: Command to set the network registration report.	*<CR><LF>* **OK** *<CR><LF>* Note: Dummy command. Always returns an OK message to indicate that the network registration report setting has been accepted. Command has no effect on the operation of the device. No registration to a mobile network is required, and is used for compatibility reasons.

Command Syntax: AT+COPS	
Command	**Valid Responses**
AT+COPS? Note: Requests the current network operator selection.	*<CR><LF>* **+COPS:_<mode>,<format>,<oper>** *<CR><LF>* **OK** *<CR><LF>* Note 1: Response parameters indicate: <mode> 0=Automatic choice 1=Manual choice 2=Set only format <format> 0=Alphanumeric max length 16digits 1=Alphanumeric short form 2=Numeric 5 digits [Country code (3) + Network code (2)] <oper> Network operator in the format defined by the <format> parameter. Note 2: Dummy command response. No actual mobile network connection is available. Command always returns the string: +COPS: 0,0,"GR PKENT79"

Command Syntax: AT+CSQ	
Command	**Valid Responses**
AT+CSQ? Note: Requests a signal quality measurement.	*<CR><LF>* **+CSQ=<rssi>,<ber>** *<CR><LF>* **OK** *<CR><LF>* Note 1: Response parameters indicate: <rssi> Received signal strength indication 0=113dBm or less 1=111dBm 2..30=109dBm…53dBm /2dBm per step 31=51dBm or greater 99=unknown or not detectable <ber> Bit error rate % 0=less than 0.2% 1=0.2%-0.4% 2=0.4%-0.8% 3=0.8%-1.6% 4=1.6%-3.2% 5=3.2%-6.4% 7=more than 12.8% 99=unknown or not detectable Note 2: Dummy command response. No actual radio signal is measured. Command always returns the string: +CSQ=15,0

Command Syntax: AT&K	
Command	**Valid Responses**
AT&K<mode> <mode> 0=Disabled 1=only CTS active. 2=XON/XOFF. 3=RTS/CTS. Note: Controls the RS-232 flow control behavior. Example: AT&K0	*<CR><LF>* **OK** *<CR><LF>* Note: Dummy command. Result is always OK but no change in the flow control is performed.

Command Syntax: AT#SHDN	
Command	**Valid Responses**
AT#SHDN Note: Command to shutdown the device.	*<CR><LF>* **OK** *<CR><LF>* Note: After the device responds with the OK message, the microcontroller enters power down mode and no further communication is possible. To re-activate the device, the INT1 pin of the microcontroller must be brought to logic '0'. The device will wake after 500ms.

Command Syntax: AT#SKTRST	
Command	**Valid Responses**
AT#SKTRST <socket> User name=none Password=none Packet size=300 Socket inactivity time out=30 (30sec) Data sending time out=10 (1sec) Socket type=0 (TCP) Remote port=3333 Note: Command to reset socket parameters to their uninitialised values.	*<CR><LF>* **OK** *<CR><LF>* Note: Socket parameters after reset are saved in the non-volatile memory.

Command Syntax: AT#USERID	
Command	**Valid Responses**
AT#USERID? Note: Request to return the current value of the USERID parameter.	*<CR><LF>* **#USERID:_"user_name"** *<CR><LF>* **OK** *<CR><LF>* Note: Returns the current value of the USERID parameter.
AT#USERID=? Note: Request to return the max length of the USERID parameter.	*<CR><LF>* **#USERID:_(15)** *<CR><LF>* **OK** *<CR><LF>* Note: Returns the max length of the USERID parameter
AT#USERID="user_name" Note: Command to set the value of the USERID parameter.	*<CR><LF>* **OK** *<CR><LF>* Note: The USERID parameter is stored in volatile memory. To store it in non-volatile memory, use the AT#SKTSAV command.

Command Syntax: AT#PASSW	
Command	**Valid Responses**
AT#PASSW=? Note: Request to return the max length of the PASSW parameter.	*<CR><LF>* **#PASSW:_(15)** *<CR><LF>* **OK** *<CR><LF>* Note: Returns the max length of the PASSW parameter
AT#PASSW="secret_code" Note: Command to set the value of the PASSW parameter.	*<CR><LF>* **OK** *<CR><LF>* Note: The PASSW parameter is stored in volatile memory. To store it in non-volatile memory, use the AT#SKTSAV command.

Command Syntax: AT+CGDCONT	
Command	**Valid Responses**
AT+CGDCONT? Note: Request to return the PDF context.	*<CR><LF>* **#CGDCONT:_<cid>,<PDP_type>, <APN>,<PDP_addr>,<d_comp>, <h_comp>** *<CR><LF>* **OK** *<CR><LF>* Note 1: Returns the PDF context. <cid> (PDF Context Identifier) A numeric parameter that specifies a particular PDF context definition. <PDP_type> (Packet Data Protocol type) A string parameter which specifies the type of packet data protocol: "IP"=Internet Protocol "PPP"=Point to Point Protocol <APN> (Access Point Name) A string parameter which is a logical name that is used to select the GGSN or the external packet data network. <PDP_addr> A string parameter that identifies the terminal in the address space applicable to the PDF. <d_comp> A numeric parameter that controls the PDF data compression. 0=Off 1=On <h_comp> A numeric parameter that controls the PDF header compression. 0=Off 1=On Note 2: Dummy command. Information returned has no effect on the operation of the device. The value of the <APN> parameter returned is taken from the device's memory as set by the AT+CGDCONT= command. The value of the <PDP_addr> parameter is the IPv4 address of the device. The response will have a format of: #CGDCONT: 0,"IP","<access point>","<local IP>",0,0 Example: #CGDCONT: 0,"IP","lan2","192.168.0.2",0,0
AT+CGDCONT=<cid>, <PDP_type>,<APN>, <PDP_addr>,<d_comp>, <h_comp> Note: Command to define the PDF context. Example: AT#CGDCONT=0,"IP","lan2","192.168.0.2",0,0	*<CR><LF>* **OK** *<CR><LF>* Note: Dummy command. Execution of this command will only change the IPv4 address of the device and set the access point name on the device's volatile memory. The access point name has no effect on the operation of the device.

Command Syntax: AT#PKTSZ	
Command	**Valid Responses**
AT#PKTSZ? Note: Request to return the current packet size parameter in bytes.	*<CR><LF>* **#PKTSZ: <packet size>** *<CR><LF>* **OK** *<CR><LF>* Note: Returns the current packet size parameter in bytes. Example: #PKTSZ: 512
AT#PKTSZ=? Note: Request to return the valid values of the packet size parameter.	*<CR><LF>* **#PKTSZ: (0,1-512)** *<CR><LF>* **OK** *<CR><LF>* Note: Returns the valid values of the packet size parameter
AT#PKTSZ=<packet size> <packet size> 0=automatically chosen by the device (not supported) 1-512=packet size in bytes Note: Command to set the value of the packet size parameter in bytes.	*<CR><LF>* **OK** *<CR><LF>* Note: The PKTSZ parameter is stored in volatile memory. To store it in non-volatile memory, use the AT#SKTSAV command.

Command Syntax: AT#SKTTO	
Command	**Valid Responses**
AT#SKTTO? Note: Requests the current value of the socket time out interval in seconds.	*<CR><LF>* **#SKTTO: <time>** *<CR><LF>* **OK** *<CR><LF>* Note: Returns the current value of the socket time out interval in seconds. Example: #SKTTO: 30
AT#SKTTO=? Note: Request to return the valid values of the socket time out interval in seconds.	*<CR><LF>* **#SKTTO: (0,1-65535)** *<CR><LF>* **OK** *<CR><LF>* Note: Returns the valid values of the socket time out interval in seconds
AT#SKTTO=<time> <time> 0=No time out. Wait until the socket is closed by the other end node. 1-65535=time out interval in seconds Note: Command to set the value of the socket time out interval in seconds. After socket inactivity of this interval, the socket is automatically closed.	*<CR><LF>* **OK** *<CR><LF>* Note: The SKTTO parameter is stored in volatile memory. To store it in non-volatile memory, use the AT#SKTSAV command.

Command Syntax: AT#DSTO	
Command	**Valid Responses**
AT#DSTO? Note: Requests the current value of the data send time out interval in hundreds of milliseconds.	*<CR><LF>* **#DSTO: <time>** *<CR><LF>* **OK** *<CR><LF>* Note: Returns the current value of the data send time out interval in hundreds of milliseconds. Example: #DSTO: 30
AT#DSTO=? Note: Request to return the valid values of the data send time out parameter in hundreds of milliseconds.	*<CR><LF>* **#DSTO: (0,1-255)** *<CR><LF>* **OK** *<CR><LF>* Note: Returns the valid values of the data send time out interval in hundreds of milliseconds
AT#DSTO=<time> <time> 0=No time out. Wait until the data packet is full before sending it. 1-255=time out interval in hundreds of milliseconds Note: Command to set the value of the data send time out interval in hundreds of milliseconds. If no new data have been added to the data packet for this amount of time, the packet is send without waiting to be full. Example AT#DSTO=235 (time out 23.5sec)	*<CR><LF>* **OK** *<CR><LF>* Note: The DSTO parameter is stored in volatile memory. To store it in non-volatile memory, use the AT#SKTSAV command.

Command Syntax: AT#QDNS	
Command	**Valid Responses**
AT#QDNS="<hostname>" Note: Command to perform a DNS query on the provided hostname. Example: AT#QDNS="mprolab.teipir.gr"	*<CR><LF>* **#QDNS="<hostname>",<ip_address>** *<CR><LF>* **OK** *<CR><LF>* Note: Hostname has been resolved by the DNS Server and its IPv4 address returned. Example: #QDNS="mprolab.teipir.gr",143.233.176.089
	<CR><LF> **#QDNS="<hostname>",NOT_SOLVED** *<CR><LF>* **OK** *<CR><LF>* Note: Hostname could not be resolved by the DNS Server. Example: #QDNS="electra2.teipir.gr",NOT_SOLVED

Command Syntax: AT#SKTSET	
Command	**Valid Responses**
AT#SKTSET? Note: Requests the current value of the socket parameters <socket type>, <remote port>, <remote address>	*<CR><LF>* **#SKTSET:_<type>,<port>,<addr>** *<CR><LF>* **OK** *<CR><LF>* Note 1: Returns the current value of the socket parameters <socket type>, <remote port>, <remote address> Note 2: Socket type is always set to 0 for use with TCP socket connections. Example: #SKTSET:_0,3333,"143.233.176.089"
AT#SKTSET=? Note: Request to return the allowed values of the socket parameters.	*<CR><LF>* **#SKTSET: (0-1),(0-65535)** *<CR><LF>* **OK** *<CR><LF>* Note 1: Returns the allowed values for the socket parameters <socket type>, <remote port> Note 2: Socket type is a dummy value since it is always read as 0 allowing only TCP sockets.
AT#SKTSET=<type>,<port>,<addr> <type> 0=TCP socket. 1=UDP socket. <port> 1-65535=TCP port number to connect to. <addr> IPv4 address in dotted-decimal notation. Note: Command to set the value of the socket parameters on the device.	*<CR><LF>* **OK** *<CR><LF>* Note 1: The socket parameters are stored in volatile memory. To store them in non-volatile memory, use the AT#SKTSAV command. Note 2: The socket type is always TCP=0 and any attempt to set it to another value will have no effect. Example: AT#SKTSET=0,1234,"143.233.176.89"

Command Syntax: AT#SKTSAV	
Command	**Valid Responses**
AT#SKTSAV Socket parameters: User name Password Packet size Socket inactivity time out Data sending time out Socket type Remote port Note: Command to store all currently set socket parameters in the non-volatile memory of the device.	*<CR><LF>* **OK** *<CR><LF>* Note: Socket parameters have been stored in the device's non-volatile memory.

Command Syntax: AT#SKTOP	
Command	**Valid Responses**
AT#SKTOP Note: Command to establish a socket connection to a remote listening port. The device attempts to connect to the IP address and TCP port of the host given by the AT#SKTSET command.	*<CR><LF>* **CONNECT** *<CR><LF>* Note: Socket connection has been established. After this message has been returned, the device enters the data exchange mode and acts as a transparent tunnel between the serial port and the TCP socket.
	Otherwise, *<CR><LF>* **NO CARRIER** *<CR><LF>* Note: A socket connection cannot be established with the remote host. The error message is returned and the device remains in command mode.

Command Syntax: +++	
Command	**Valid Responses**
+++ Note: Character sequence to force a socket close. When this sequence of characters is encountered in the data stream the device will transmit the data it has received so far and close the socket connection. If this sequence is an actual stream of data, then an escape character must be inserted to break the sequence and then removed at the receiving station. This is done under software control of the communicating stations.	*<CR><LF>* **NO CARRIER** *<CR><LF>* Note: After the socket has been closed the device returns to command mode and can process commands.

- **Proprietary AT Command Set Extension**

The following commands are an extension to the GPRS AT command set meant for this device only. They should be used for device configuration and maintenance purposes.

Command Syntax: AT%CONF	
Command	**Valid Responses**
AT%CONF? Note: Requests the device to display its network configuration.	AT%CONF? ========== Network Configuration ========== MAC Address: 00.55.DC.00.00.00 SubNet Mask: 255.255.255.000 IP Address: 192.168.000.002 Gateway Address: 192.168.000.001 Domain Name Server: 195.170.000.002 ==========

Command Syntax: AT%IP	
Command	**Valid Responses**
AT%IP? Note: Requests the currently configured IPv4 address of the device.	*<CR><LF>* **%IP:_<ip4_addr>** *<CR><LF>* **OK** *<CR><LF>* Note: Returns the currently configured IPv4 address of the device. Example: %IP:_"192.168.000.002"
AT%IP=<ip4_addr> Note: Command to set the IPv4 address of the device. Example: AT%IP="192.168.0.12"	*<CR><LF>* **OK** *<CR><LF>*

Command Syntax: AT%GW	
Command	**Valid Responses**
AT%GW? Note: Requests the currently configured IPv4 address of the network gateway.	*<CR><LF>* **%GW:_<ip4_addr>** *<CR><LF>* **OK** *<CR><LF>* Note: Returns the currently configured IPv4 address of the network gateway. Example: %GW:_"192.168.000.001"
AT%GW=<ip4_addr> Note: Command to set the IPv4 address of the network gateway. Example: AT%GW="192.168.0.1"	*<CR><LF>* **OK** *<CR><LF>*

Command Syntax: AT%DNS	
Command	**Valid Responses**
AT%DNS? Note: Requests the currently configured IPv4 address of the network's DNS server.	*<CR><LF>* **%DNS:_<ip4_addr>** *<CR><LF>* **OK** *<CR><LF>* Note: Returns the currently configured IPv4 address of the network's DNS server. Example: %DNS:_"194.177.210.210"
AT%DNS=<ip4_addr> Note: Command to set the IPv4 address of the network's DNS server. The DNS server is used by the AT#QDNS command to resolve the IPv4 address of a host's logical name. Example: AT%DNS="194.177.210.210"	*<CR><LF>* **OK** *<CR><LF>*

Command Syntax: AT%SMASK	
Command	**Valid Responses**
AT%SMASK? Note: Requests the currently configured subnet mask of the network.	*<CR><LF>* **%SMASK:_<subnet _ mask>** *<CR><LF>* **OK** *<CR><LF>* Note: Returns the currently configured subnet mask. Example: %SMASK:_"255.255.255.000"
AT%SMASK=<subnet_mask > Note: Command to set the subnet mask of the network. Example: AT%SMASK="255.255.255.0"	*<CR><LF>* **OK** *<CR><LF>*

Command Syntax: AT%MAC	
Command	**Valid Responses**
AT%MAC? Note: Requests the currently configured MAC address of the Ethernet controller.	*<CR><LF>* **%MAC:_<mac_addr>** *<CR><LF>* **OK** *<CR><LF>* Note: Returns the currently configured MAC address of the Ethernet controller. Example: %MAC:_"00.55.DA.FC.00.00"
AT%MAC=<mac_addr> Note: Command to set the MAC address of the Ethernet controller. Example: AT%MAC="00.75.6A.3C.10.00"	*<CR><LF>* **OK** *<CR><LF>*

4.2.4. Software Execution Flow Charts

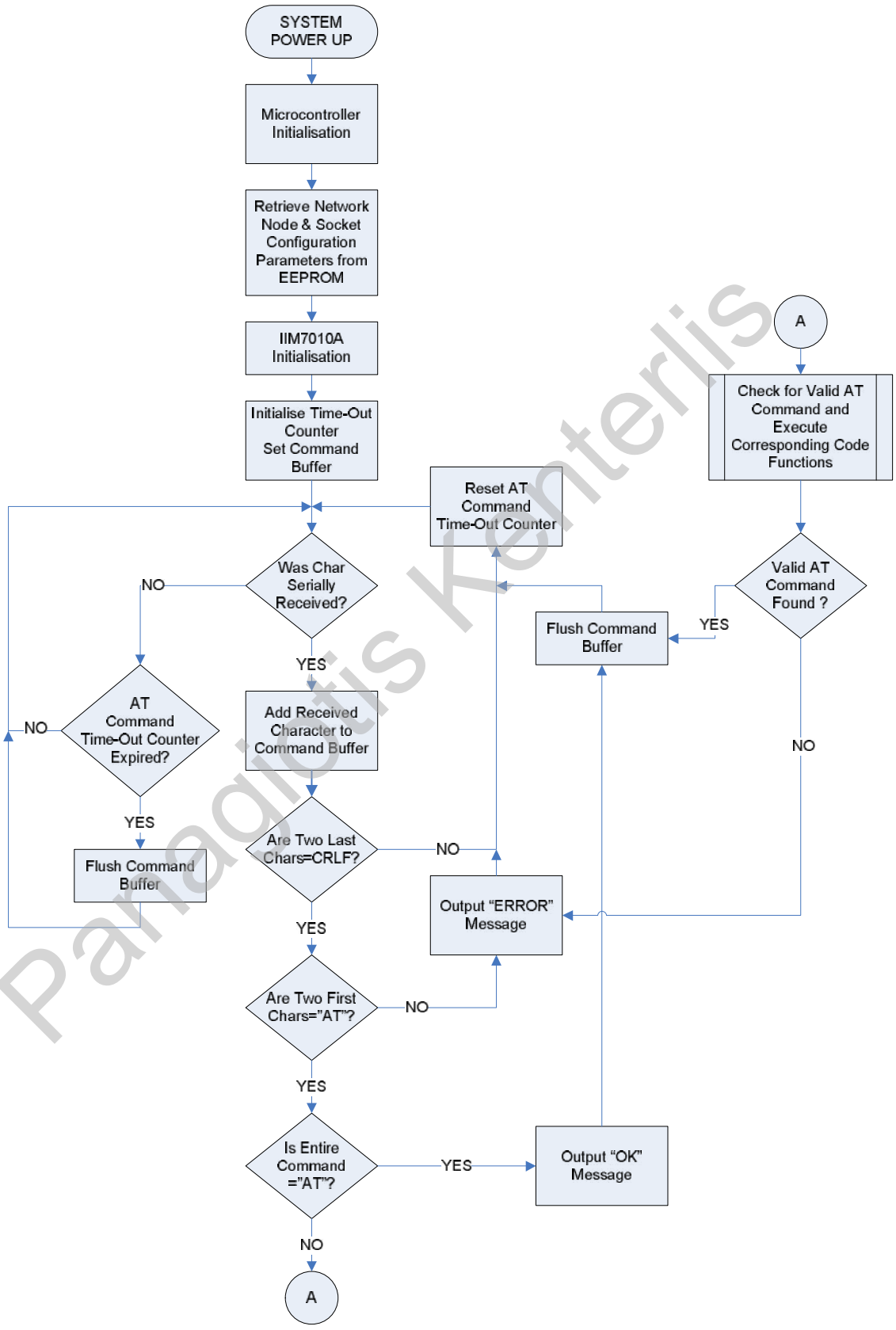

Figure 21 Main Program Execution Flow Chart

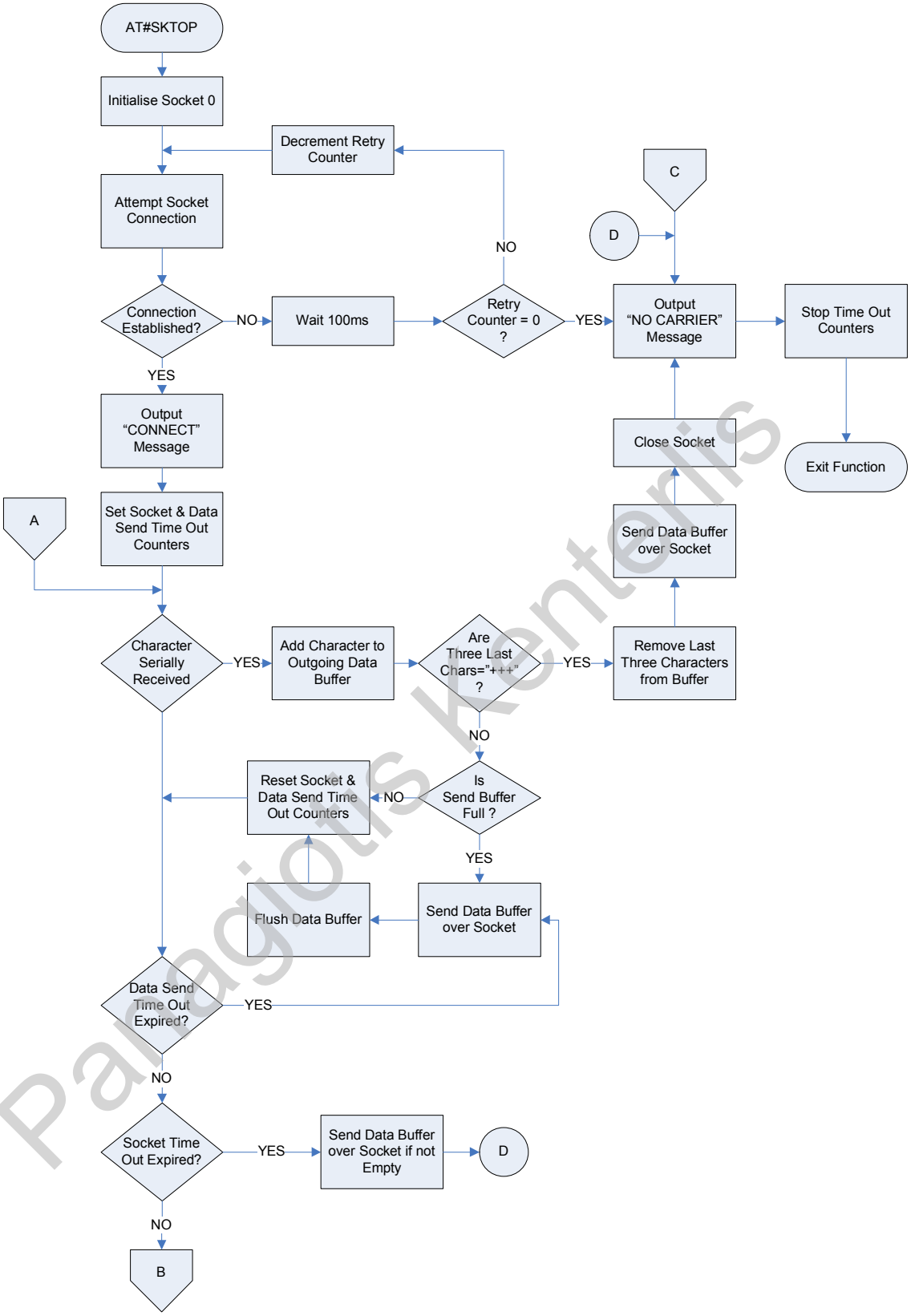

Figure 22 Data Transfer Mode Execution Flow Chart #1/2#

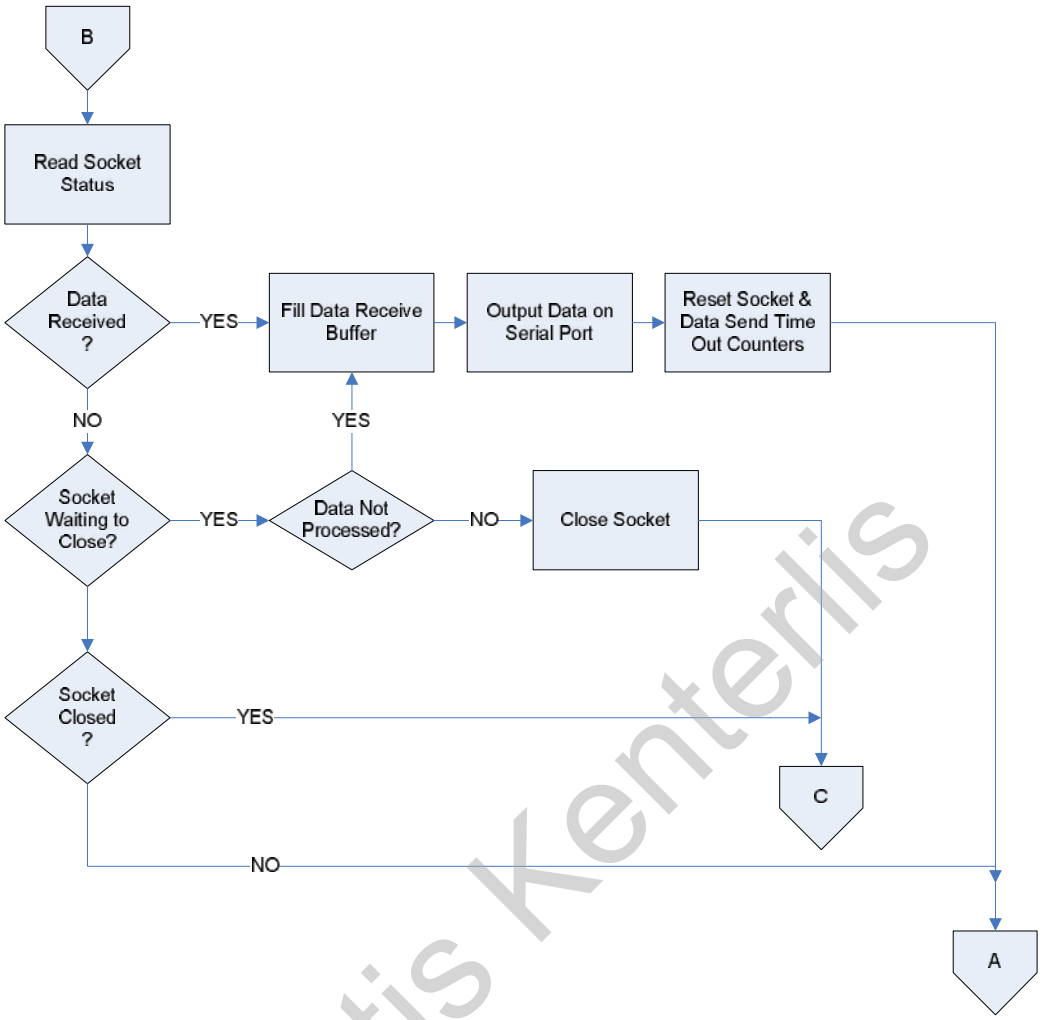

Figure 23 Data Transfer Mode Execution Flow Chart #2/2#

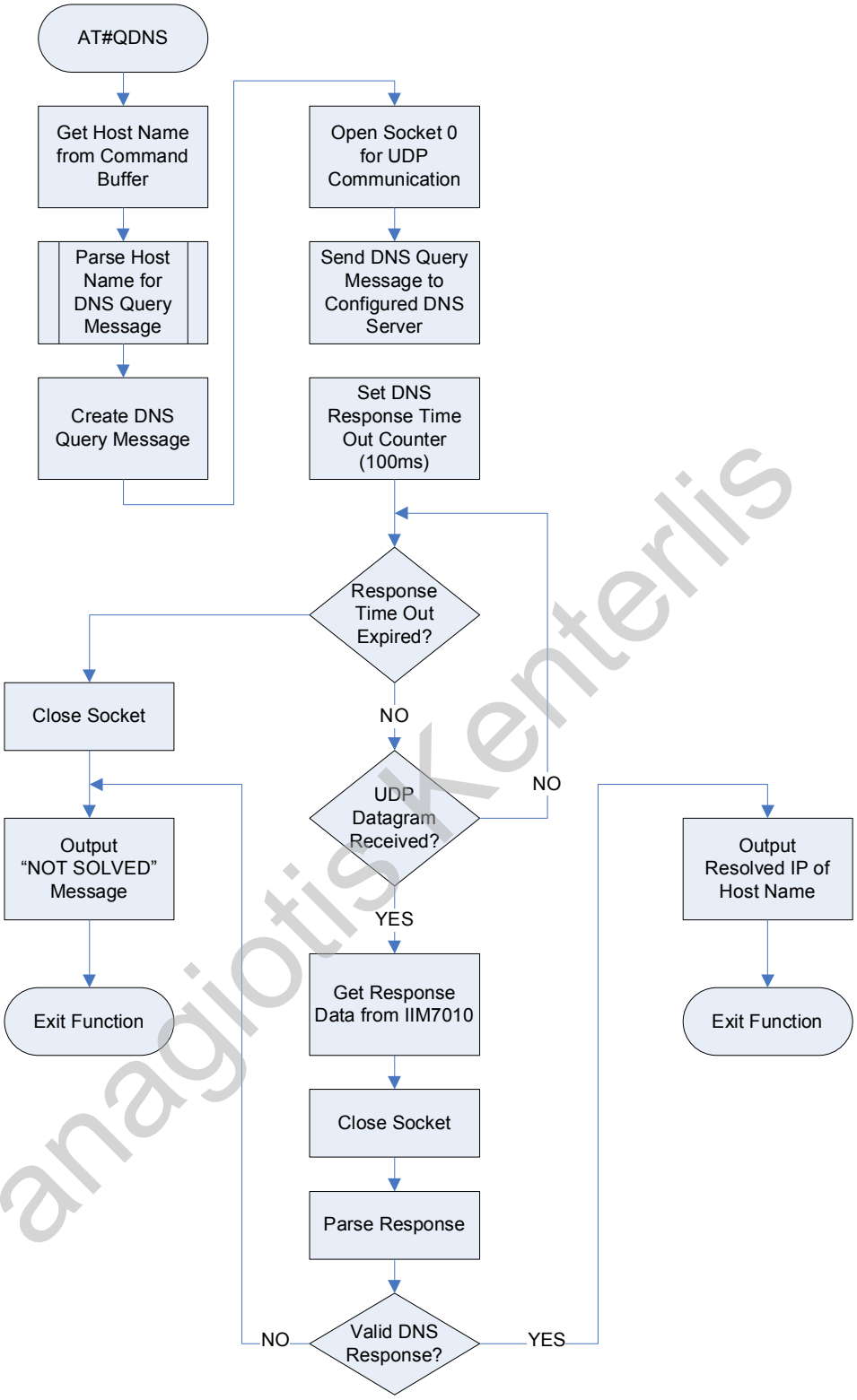

Figure 24 DNS Query Execution Flow Chart

5. Testing and Debugging

Just like every hardware & software combination project, this one as well requires some means of verification; this chapter will address the issue of testing for accordance to the specifications set at early stages of the design. It is essential to be able to test the device while still in the development stage to eliminate any malfunctions that could appear after it has been installed for full use.

5.1.1. System Debugging

System debugging is separated in two sections, those for hardware, and software. Hardware troubleshooting experienced at the early stages of development involved locating design and manufacturing errors in the prototype boards of the system device. After this had completed, identifying hidden software bugs in the code developed for the system was to be the most tedious task, since it involves both logically and practically verifying every code function written. When dealing with embedded systems, care needs to be taken so that the software is optimized to make extensive yet not greedy use of the available system resources, which are always limited.

For the development stage of the project, all of the tests and experimentations performed on the device were carried out using the experimental set-up shown in Figure 25.

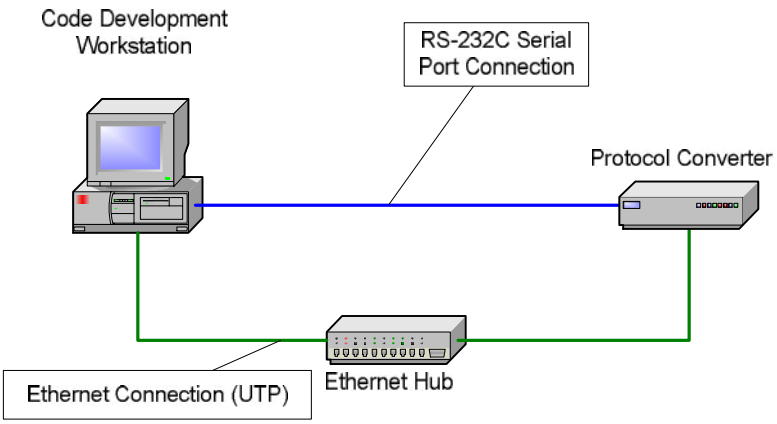

Figure 25 Experimental connections of the device

- **System Configuration**

The first step in debugging a system is to be aware of what its configuration is. This feedback can be largely performed by using the already described AT command set and its proprietary extensions. In Figure 26 below, the execution of the AT%CONF command is displayed and its response can be used to verify the configuration of the device.

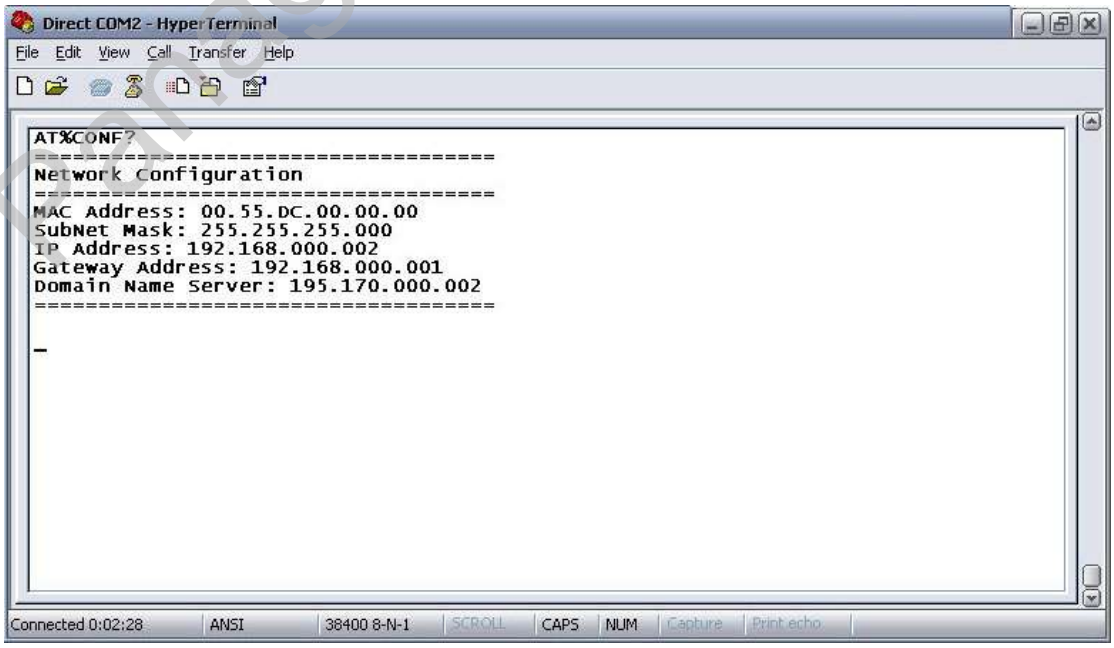

Figure 26 Obtaining the device configuration

- **Run-Time Debugging**

Having a serial connection to the computer increases the level of detectability of errors. The designer can strategically insert checkpoints or message printing function calls that verify the system status or give information on function variables during execution. Software or hardware bugs can then be easily traced and eliminated.

5.1.2. Simulation of expected use patterns

After having developed some functional piece of software code, there is need to observe the system respond to traffic patterns similar to the ones expected to appear when the project has been finalised. This will be achieved by applying loop-back tests, where test data is sent from one interface to the other and return data is compared to the original and transmission times measured. Additional tests for communication breakdown and recovery will be very helpful in gathering information to build a reliable piece of equipment. Error recovery or at least error reporting is required to assure data integrity. Packet capturing software techniques will be used to analyse packets being sent and received between the interface device and the server station.

- **Domain Name Service Query**

As already mentioned the device has the ability of resolving hostnames to their IPv4 addresses. To make this happen an external DNS server must be present. The DNS server can be configured using the AT%DNS command.

Execution of the AT#QDNS command on Windows' HyperTerminal as in Figure 27 below provides a response containing the resolved IPv4 address of the host name provided.

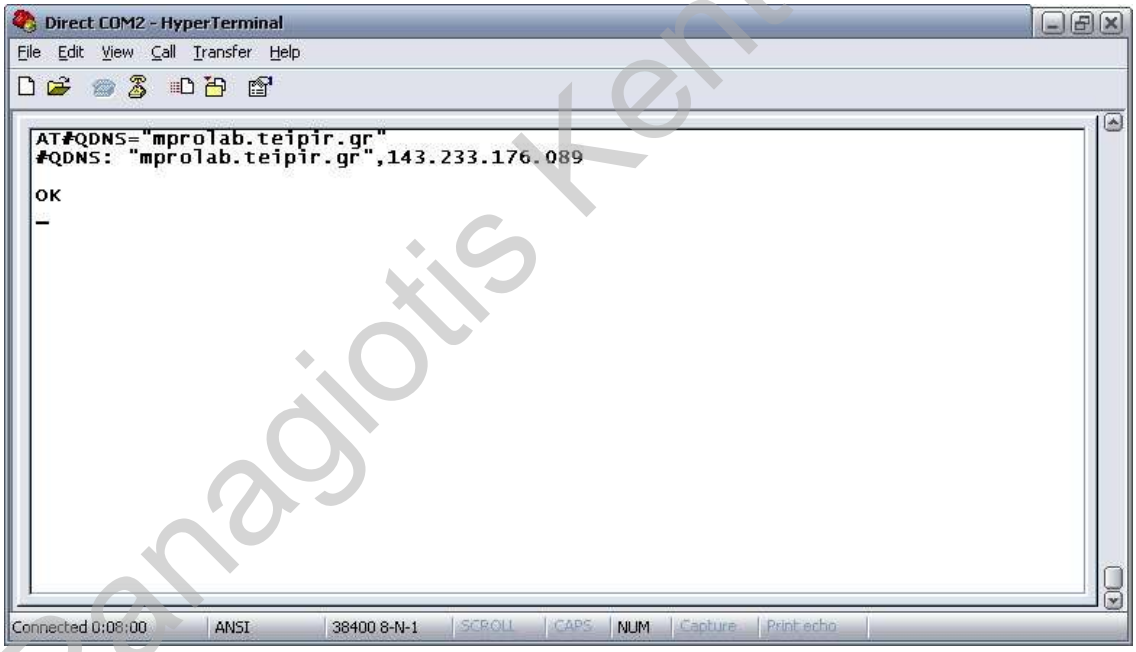

Figure 27 Querying the DNS server through the device

Using network monitoring tools such as WinDump (Ref. [29]) (execute "windump -Xvvvs 256 udp port 53") to monitor network packets exchanged with a real DNS server, valuable information was extracted and programming or protocol faults were detected and corrected. In the example shown in Figure 26 and Figure 27, the device has been requested to provide the resolved IPv4 address of the host "mprolab.teipir.gr" which is a server administrated by the writer.

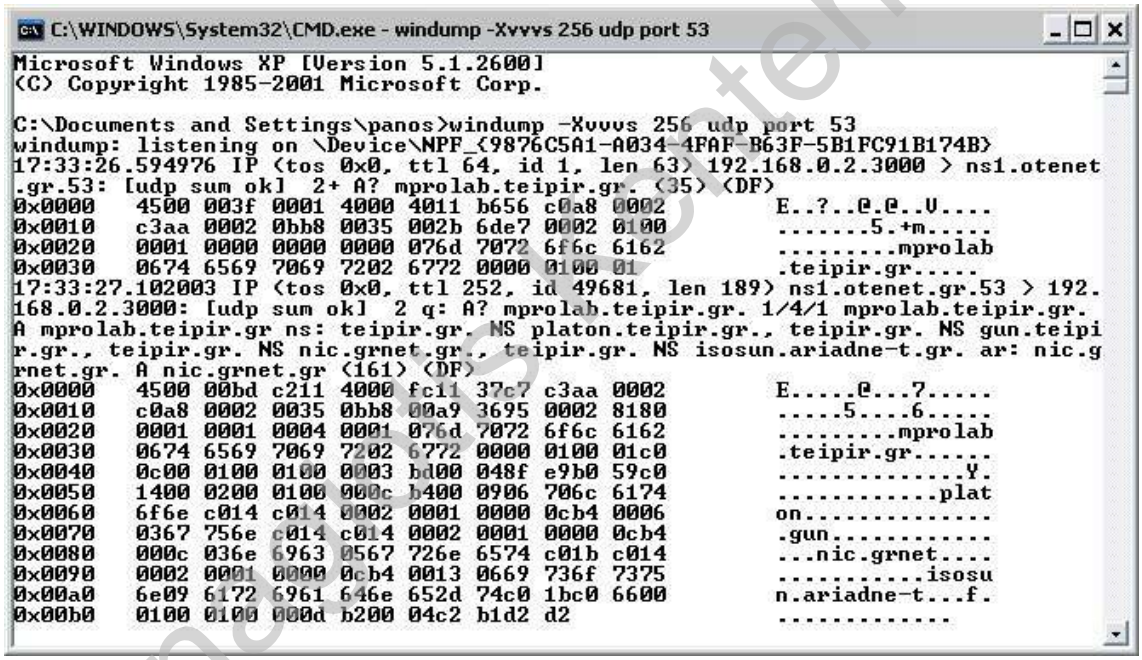

Figure 28 Windump trace of DNS query and response

- **Data Transfer Session**

The most important function of the device is undoubtedly the transfer of data from the field station to the central server. It would be unthinkable not to test the device for this function, so this is presented in the following pages.

For our testing purposes, two programs were used, PortPeeker a network port monitor running under windows and a serial terminal emulation program. Any serial terminal application is acceptable for our experimentations.

PortPeeker (Ref. [28]) will monitor control and data packets exchanged on a specified network interface and a TCP or UDP port.

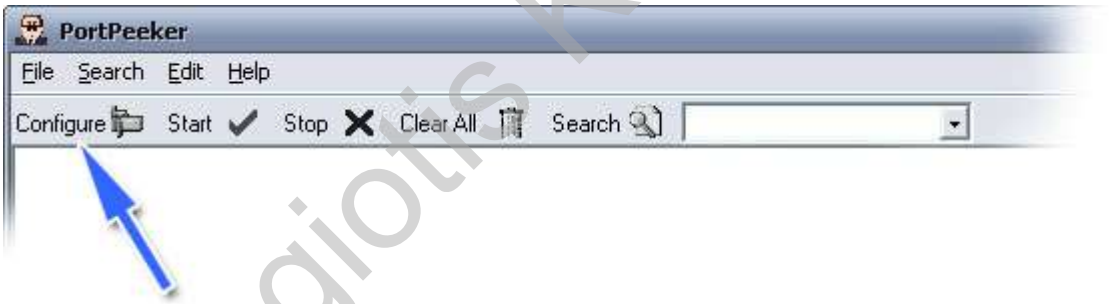

Figure 29 PortPeeker Application Window

To use PortPeeker, it must first be configured to monitor the port that the device will attempt to connect by default. For testing scenarios, this is TCP port 3000 and the listening interface of the personal computer is 192.168.0.1. The device is set to have the IPv4 address 192.168.0.2. PortPeeker is configured by pressing the Configuration button in the main application window (Figure 29). The window of Figure 30 will appear and network monitor parameters were set as displayed.

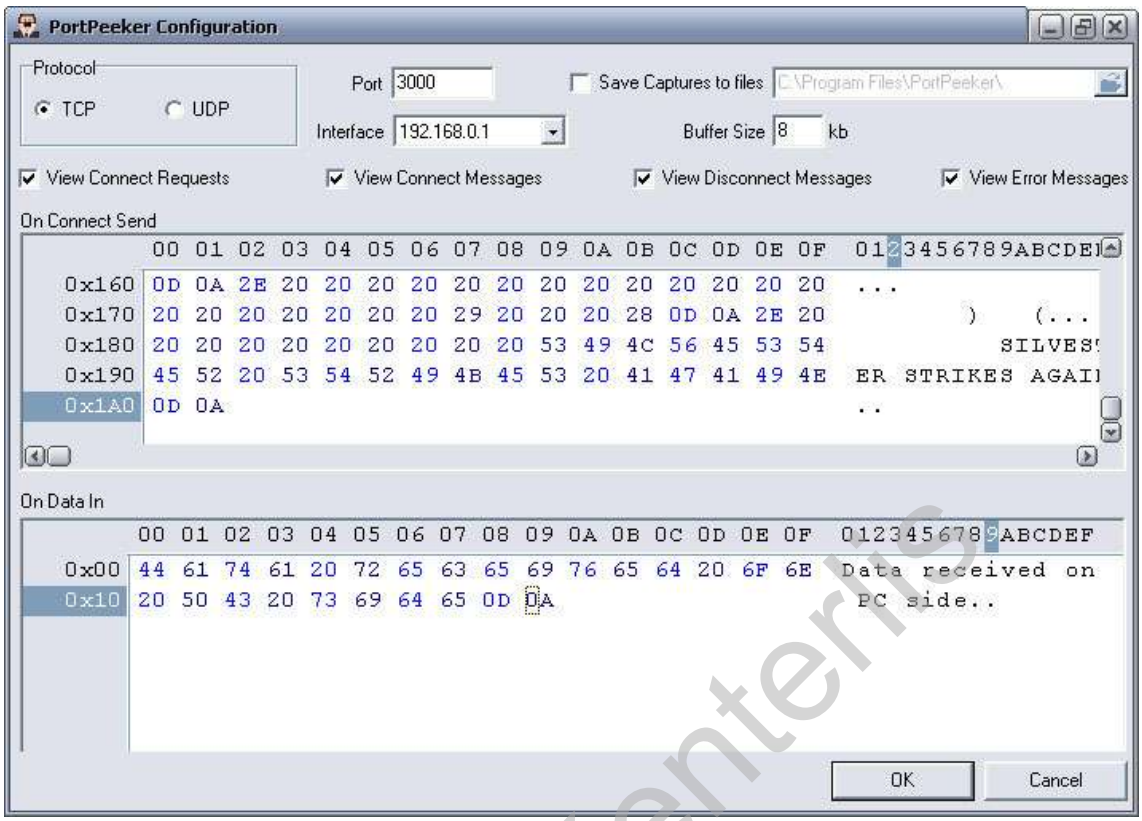

Figure 30 PortPeeker Configuration Window

The settings on this window involve two message spaces. The first one contains a message that will be sent to the device when it will establish a socket connection with PortPeeker. The second message will be sent as a confirmation message to the device when any piece of data is received from it.

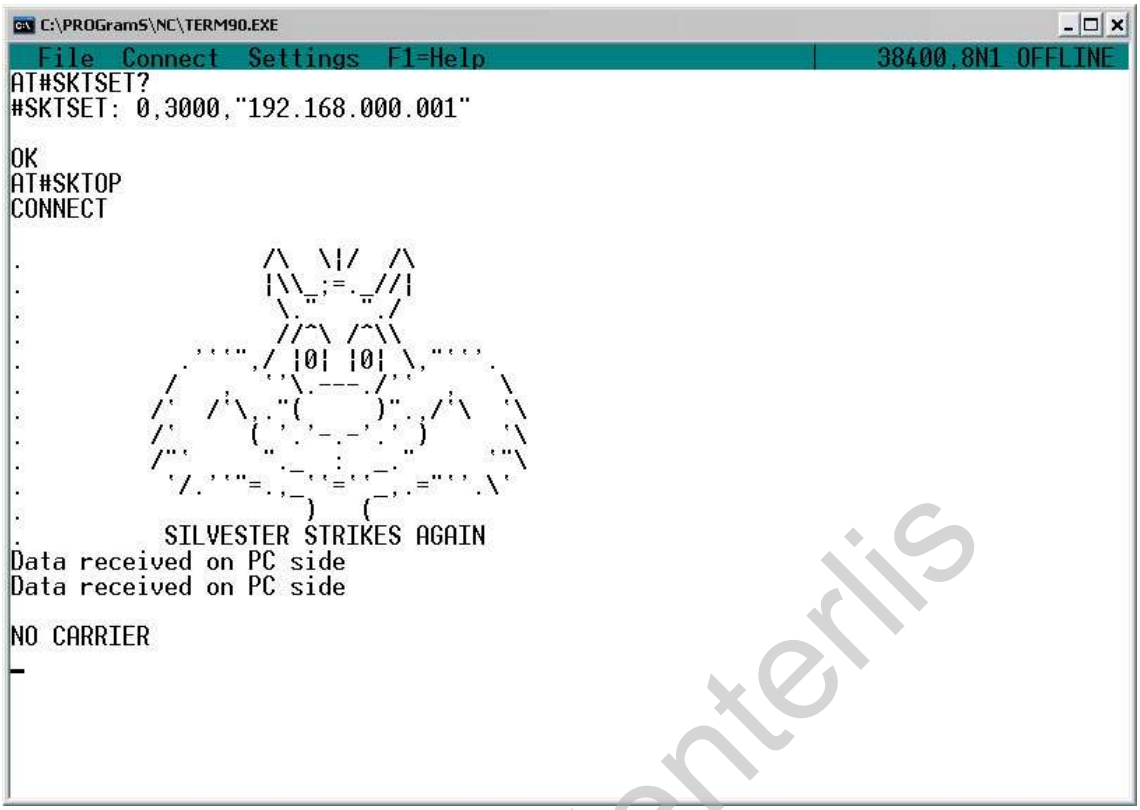

Figure 31 Terminal Screen Capture

From Figure 31 above, the reader can observe the execution of two commands; the first command (AT#SKTSET?) will display the currently configured socket parameters. The second command will attempt to establish a socket connection with PortPeeker which is listening on TCP port 3000.

As seen in Figure 31 above and Figure 32 below, when the connection is established the computer will send its, somewhat cute, user-defined "connection established" message. Typing some messages to emulate incoming serial data on the device, when the data send time out expires, the message is sent to the computer. The computer then responds with its user-defined "data received" message.

The connection is finally closed from the side of the device by typing the socket termination string "+++".

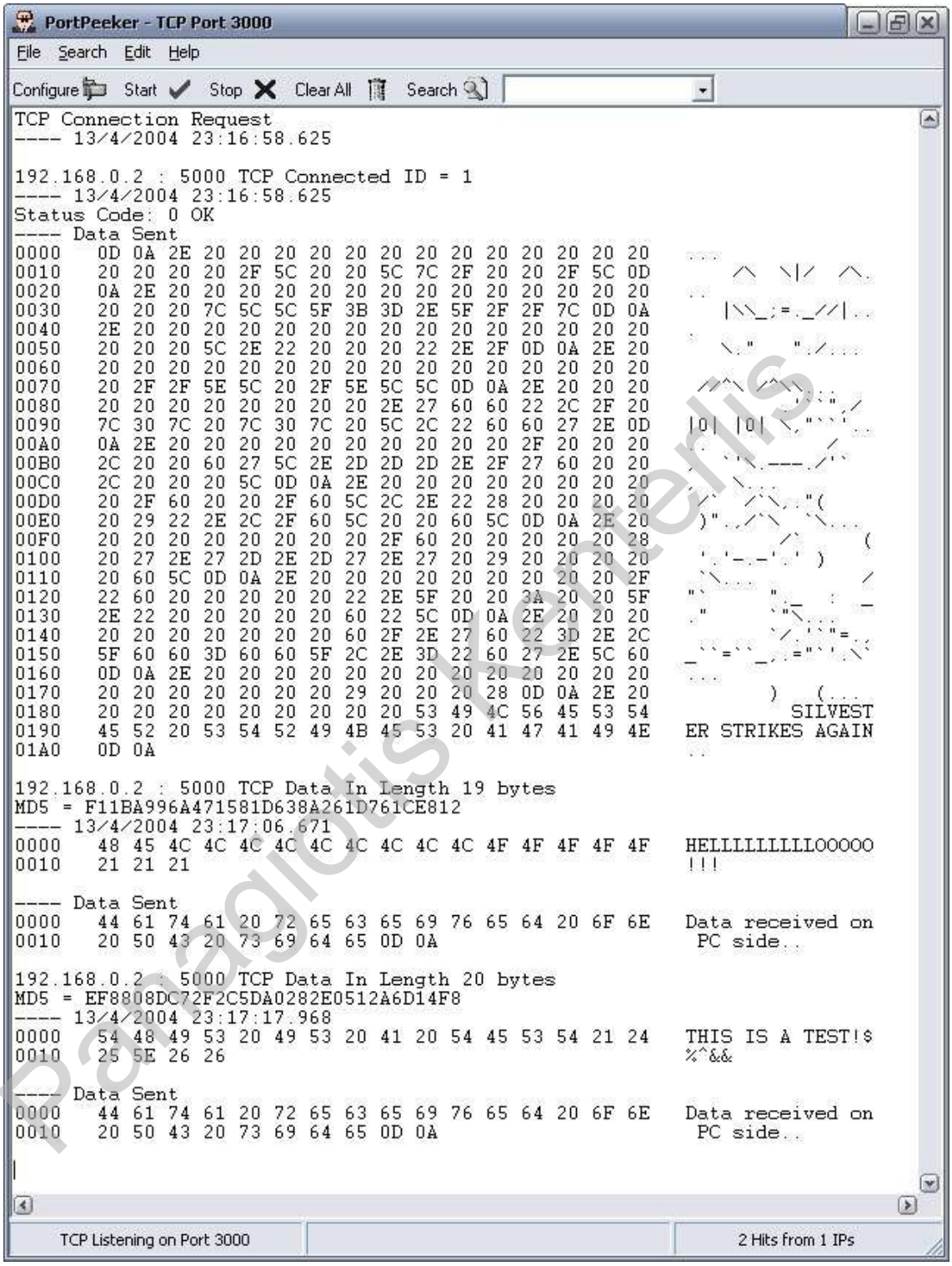

Figure 32 PortPeeker Window Capture of Data Transfer

The captured data transfer activity can also be found in clear-text form in the appendix.

5.1.3. In-field testing

Plans for the future involve testing the system out in the field. This requires that both the data acquisition equipment of the field station and the developed communication device of this project shall be placed in proper installations for carrying out real experiments. The device will be monitored and any deviation from the expected work plan needs to be processed and possible sources of problems in hardware or software be eliminated.

6 . Conclusions

Concluding this project, we take a step back and consider how it started and what has been completed. The entire specifications list has been successfully implemented and our aims reached.

The wide range of knowledge obtained from the taught modules of this course and their respective laboratory sessions has acted catalytically in favor of finishing this project. The device in question has been designed and prototype printed circuit boards produced to a fully functional level. The prototype device has been tested and verified in laboratory conditions while testing at a real field station remains as planned not to happen before the deadline of this project.

Managing such a project, although difficult at some points, it proved to be a very interesting and invaluable experience. Much knowledge in the field of computer networks and embedded systems programming has been gained. Sources of this gain have been the logical reflections occurring at every moment, as many aspects of the problems being faced with needed to be considered in order to decide on the best possible solution.

7. Future Work

Future work on this project might involve the support for new firmware releases or hardware additions to allow expansion of the seismological research project. At this moment, there is no knowledge on potential future changes. Requirements and emerging ideas are the fuel to keep the fire of this project burning.

8. Personal Thoughts

Having finished this project, I feel more confident about my abilities in handling projects of such magnitude in real life.

During both research and development periods, I had the opportunity to study on real-time embedded systems and expand my knowledge in microcontroller and microprocessor systems design, which is my main field of interest. In addition, it was possible to apply engineering knowledge on a very important field of computer networks.

In conclusion, as expected, this project proved to be the most exciting and challenging part of the MSc course.

9. Project Management

For this project to achieve its aims the entire period required to complete this project had to be divided in three stages, Research, Design & Development, and Testing.

Research:

- Study the TCP/IP protocol suite.
- Find out what other solutions are available currently on the market.
- Draw the primary system specifications.
- Set the achievable goals according to time and budget available for the project.
- Find, compare, and decide which parts should be used for the system.

Design & Development:

- Study datasheets of parts to be used.
- Draw hardware interconnection diagrams according to specifications.
- Produce prototype boards.
- Produce software code.

Testing

- Debug code and system using simulation scenarios of expected working conditions.

The above stages are better observed in the Gantt chart for this project given in the appendix.

10. Bibliography

[1] Freeman, Roger L. – Telecommunication System Engineering. – 3rd ed. – New York; John Wiley & Sons, Inc, 1996. – ISBN: 0471133027

[2] Green, D. C. (Derek Charles), 1931-. - Data communication. - Harlow: Longman Scientific & Technical, 1991. – ISBN: 0582060052

[3] Halsall, Fred. - Data communications, computer networks, and open systems. - 3rd ed. - Wokingham: Addison-Wesley, 1992. – ISBN: 0201565064

[4] Seidler, J. - Principles of computer communication network design / translation editor R.J. - Chichester: Horwood, 1983. - (Ellis Horwood series in electrical and electronic engineering). – ISBN: 0853122415

[5] Stallings, William. - Data and computer communications. - 3rd ed. - New York; Oxford: Maxwell Macmillan, 1991. – ISBN: 0024154547

[6] Donahoo, M., Calvert K., - TCP/IP Sockets in C: Practical Guide for Programmers (The Practical Guides Series) - Morgan Kaufmann, 2000. - ISBN: 1558608265

11. References & Other Documents

11.1. Ready-to-Use Solutions

[1] Precidia Technologies
URL: http://www.precidia.com/products/product1.html
Last Accessed: 01/02/2004

[2] ARC Electronics
URL: http://www.arcelect.com/LS-101_RS232_to_TCP-IP_ICMP_HTTP_DHCP_converter.htm
Last Accessed: 01/02/2004

[3] MOXA
URL: http://www.moxa.com/product/Serial_Device_Servers/8_16_Port/NPort_DE_308_303.htm
Last Accessed: 01/02/2004

[4] Arcom
URL: http://www.arcom.com/products/pcp/Gateways/ESS/ESS1.htm
Last Accessed: 01/02/2004

[5] Telecom Design Communications Ltd.
URL: http://www.tdc.co.uk/modems/modem_tcpip_connect1.htm
Last Accessed: 01/02/2004

[6] Kanda Systems
URL: https://www.kanda.com/s_embedded.html
Last Accessed: 01/02/2004

[7] VOX Technologies
URL: http://www.voxtechnologies.com/MoxaIndex.htm
Last Accessed: 01/02/2004

11.2. Articles & Other Study Material

[8] Sensors Magazine, Article by Thomas A. Lutz: "Using TCP/IP As an Instrument Interface"
URL: http://www.sensorsmag.com/articles/0798/tcp0798/main.shtml
Last Accessed: 01/02/2004

[9] Internet Society (ISOC) All About The Internet: History of the Internet
URL: http://www.isoc.org/Internet/history/
Last Accessed: 18/04/2004

[10] RFC 1180 - TCP/IP tutorial
URL: http://www.faqs.org/rfcs/rfc1180.html
Last Accessed: 18/04/2004

[11] RFC 791 - Internet Protocol
URL: http://www.faqs.org/rfcs/rfc791.html
Last Accessed: 18/04/2004

[12] RFC 793 - Transmission Control Protocol
URL: http://www.faqs.org/rfcs/rfc793.html
Last Accessed: 18/04/2004

[13] RFC 768 - User Datagram Protocol
URL: http://www.faqs.org/rfcs/rfc768.html
Last Accessed: 18/04/2004

[14] DNS Related RFCs (Request For Comments): rfc1035, rfc2929.
URL #1: http://www.faqs.org/rfcs/rfc1035.html
URL #2: http://www.faqs.org/rfcs/rfc2929.html
Last Accessed: 01/04/2004

[15] Thomas P. Kelliher, Carnegie Mellon (US), "Introduction to Socket API"
URL: http://www.andrew.cmu.edu/~kevinm/sockets.html
Last Accessed: 18/04/2004

[16] Brian "Beej" Hall – "Beej's Guide to Network Programming"
URL: http://www.ecst.csuchico.edu/~beej/guide/net/
Last Accessed: 18/04/2004

[17] Lakeview Research: "Serial Links using RS-232 and RS-485"
URL: http://www.lvr.com/serport.htm
Last Accessed: 18/04/2004

[18] totse.com: "Practical guide to RS- 232 interfacing"
http://www.totse.com/en/technology/telecommunications/rs232.html
Last Accessed: 18/04/2004

[19] The online Industrial Ethernet Book, Article by David Evans:
"Interfacing Serial to Ethernet"
URL: http://ethernet.industrial-networking.com/articles/i11serial.asp
Last Accessed: 01/02/2004

[20] Electrotek Concepts Inc., Article: "Data Acquisition System Description"
URL: http://www.electrotek.com/DOE/diagram.htm
Last Accessed: 01/02/2004

[21] Beyond Logic, Article by Craig Peacock: "Ethernet & TCP/IP Interfaces"
URL: http://www.beyondlogic.org/etherip/ip.htm
Last Accessed: 01/02/2004

11.3. Hardware Components Used

[22] Kanda STK200 AVR Microcontroller Starter Kit
URL: https://www.kanda.com/shopnav/shop.php3?bc=direct&bw=/browse.php3?node=30&semisupport=
Last Accessed: 18/04/2004

[23] Atmel AVR ATmega8515L
URL: http://www.atmel.com/dyn/products/product_card.asp?part_id=2007
Last Accessed: 01/02/2004

[24] Atmel AVR ATmega162L
URL: http://www.atmel.com/dyn/products/product_card.asp?part_id=2024
Last Accessed: 01/02/2004

[25] AVR Instruction Set
URL: http://www.atmel.com/dyn/resources/prod_documents/DOC0856.PDF
Last Accessed: 01/02/2004

[26] WIZnet iinchip IIM7010A
URL: http://www.iinchip.com/e_iinchip/product_module_iim7010A.htm
Last Accessed: 01/02/2004

11.4. Software Tools Used

[27] AVR-GCC Compiler
URL: http://www.avrfreaks.net/AVRGCC/
Last Accessed: 01/02/2004

[28] PortPeeker
URL: http://www.linklogger.com/portpeeker.htm
Last Accessed: 01/04/2004

[29] WinDump: tcpdump for windows
URL: http://windump.polito.it/
Last Accessed: 01/02/2004

12. Appendix

Makefile sample

```
# WinAVR Sample makefile written by Eric B. Weddington, Jørg Wunsch,
et al.
# Released to the Public Domain
# Please read the make user manual!
#
# Additional material for this makefile was submitted by:
#  Tim Henigan
#  Peter Fleury
#  Reiner Patommel
#  Sander Pool
#  Frederik Rouleau
#  Markus Pfaff
#
# On command line:
#
# make all = Make software.
#
# make clean = Clean out built project files.
#
# make coff = Convert ELF to AVR COFF (for use with AVR Studio 3.x
or VMLAB).
#
# make extcoff = Convert ELF to AVR Extended COFF (for use with AVR
Studio
#                4.07 or greater).
#
# make program = Download the hex file to the device, using avrdude.
Please
#                customize the avrdude settings below first!
#
# make filename.s = Just compile filename.c into the assembler code
only
#
# To rebuild project do "make clean" then "make all".
#


# MCU name
MCU = atmega162

# Output format. (can be srec, ihex, binary)
FORMAT = ihex

# Target file name (without extension).
TARGET = main

# Optimization level, can be [0, 1, 2, 3, s]. 0 turns off
optimization.
# (Note: 3 is not always the best optimization level. See avr-libc
FAQ.)
OPT = 2


# List C source files here. (C dependencies are automatically
generated.)
SRC = $(TARGET).c
```

Makefile sample

```
# If there is more than one source file, append them above, or
modify and
# uncomment the following:
SRC += serial.c functions.c socket.c sockutil.c constant.c

# You can also wrap lines by appending a backslash to the end of the
line:
#SRC += baz.c \
#xyzzy.c



# List Assembler source files here.
# Make them always end in a capital .S.  Files ending in a lowercase
.s
# will not be considered source files but generated files (assembler
# output from the compiler), and will be deleted upon "make clean"!
# Even though the DOS/Win* filesystem matches both .s and .S the
same,
# it will preserve the spelling of the filenames, and gcc itself
does
# care about how the name is spelled on its command-line.
ASRC =


# List any extra directories to look for include files here.
#     Each directory must be seperated by a space.
EXTRAINCDIRS =


# Optional compiler flags.
#  -g:        generate debugging information (for GDB, or for COFF
conversion)
#  -O*:       optimization level
#  -f...:     tuning, see gcc manual and avr-libc documentation
#  -Wall...:  warning level
#  -Wa,...:   tell GCC to pass this to the assembler.
#    -ahlms:  create assembler listing
CFLAGS = -n -g -O$(OPT) \
-funsigned-char -funsigned-bitfields -fpack-struct -fshort-enums \
-Wall -Wstrict-prototypes \
-Wa,-adhlns=$(<:.c=.lst) \
$(patsubst %,-I%,$(EXTRAINCDIRS))


# Set a "language standard" compiler flag.
#   Unremark just one line below to set the language standard to
use.
#   gnu99 = C99 + GNU extensions. See GCC manual for more
information.
#CFLAGS += -std=c89
#CFLAGS += -std=gnu89
#CFLAGS += -std=c99
CFLAGS += -std=gnu99



# Optional assembler flags.
#  -Wa,...:   tell GCC to pass this to the assembler.
#  -ahlms:    create listing
```

Makefile sample

```
#  -gstabs:   have the assembler create line number information;
note that
#             for use in COFF files, additional information about
filenames
#             and function names needs to be present in the
assembler source
#             files -- see avr-libc docs [FIXME: not yet described
there]
ASFLAGS = -Wa,-adhlns=$(<:.S=.lst),-gstabs



# Optional linker flags.
#  -Wl,...:   tell GCC to pass this to linker.
#  -Map:      create map file
#  --cref:    add cross reference to  map file
LDFLAGS = -Wl,-Map=$(TARGET).map,--cref



# Additional libraries

# Minimalistic printf version
#LDFLAGS += -Wl,-u,vfprintf -lprintf_min

# Floating point printf version (requires -lm below)
#LDFLAGS += -Wl,-u,vfprintf -lprintf_flt

# -lm = math library
#LDFLAGS += -lm




# Programming support using avrdude. Settings and variables.

# Programming hardware: alf avr910 avrisp bascom bsd
# dt006 pavr picoweb pony-stk200 sp12 stk200 stk500
#
# Type: avrdude -c ?
# to get a full listing.
#
AVRDUDE_PROGRAMMER = stk200

# programmer connected to serial device
#AVRDUDE_PORT = com1
# programmer connected to parallel port
AVRDUDE_PORT = lpt1


AVRDUDE_WRITE_FLASH = -U flash:w:$(TARGET).hex
#AVRDUDE_WRITE_EEPROM = -U eeprom:w:$(TARGET).eep

AVRDUDE_FLAGS = -p $(MCU) -P $(AVRDUDE_PORT) -c
$(AVRDUDE_PROGRAMMER)

# Uncomment the following if you want avrdude's erase cycle counter.
# Note that this counter needs to be initialized first using -Yn,
# see avrdude manual.
#AVRDUDE_ERASE += -y
```

Makefile sample

```
# Uncomment the following if you do /not/ wish a verification to be
# performed after programming the device.
#AVRDUDE_FLAGS += -V

# Increase verbosity level.  Please use this when submitting bug
# reports about avrdude. See
<http://savannah.nongnu.org/projects/avrdude>
# to submit bug reports.
#AVRDUDE_FLAGS += -v -v



# ------------------------------------------------------------------
---------

# Define directories, if needed.
DIRAVR = c:/winavr
DIRAVRBIN = $(DIRAVR)/bin
DIRAVRUTILS = $(DIRAVR)/utils/bin
DIRINC = .
DIRLIB = $(DIRAVR)/avr/lib


# Define programs and commands.
SHELL = $(DIRAVRUTILS)/sh

CC = $(DIRAVRBIN)/avr-gcc

OBJCOPY = $(DIRAVRBIN)/avr-objcopy
OBJDUMP = $(DIRAVRBIN)/avr-objdump
SIZE = $(DIRAVRBIN)/avr-size


# Programming support using avrdude.
AVRDUDE = $(DIRAVRBIN)/avrdude


REMOVE = rm -f
COPY = cp

HEXSIZE = $(SIZE) --target=$(FORMAT) $(TARGET).hex
ELFSIZE = $(SIZE) -A $(TARGET).elf



# Define Messages
# English
MSG_ERRORS_NONE = Errors: none
MSG_BEGIN = -------- begin --------
MSG_END = --------  end  --------
MSG_SIZE_BEFORE = Size before:
MSG_SIZE_AFTER = Size after:
MSG_COFF = Converting to AVR COFF:
MSG_EXTENDED_COFF = Converting to AVR Extended COFF:
MSG_FLASH = Creating load file for Flash:
MSG_EEPROM = Creating load file for EEPROM:
MSG_EXTENDED_LISTING = Creating Extended Listing:
MSG_SYMBOL_TABLE = Creating Symbol Table:
MSG_LINKING = Linking:
MSG_COMPILING = Compiling:
```

Makefile sample

```
MSG_ASSEMBLING = Assembling:
MSG_CLEANING = Cleaning project:



# Define all object files.
OBJ = $(SRC:.c=.o) $(ASRC:.S=.o)

# Define all listing files.
LST = $(ASRC:.S=.lst) $(SRC:.c=.lst)

# Combine all necessary flags and optional flags.
# Add target processor to flags.
ALL_CFLAGS = -mmcu=$(MCU) -I. $(CFLAGS)
ALL_ASFLAGS = -mmcu=$(MCU) -I. -x assembler-with-cpp $(ASFLAGS)



# Default target.
all: begin gccversion sizebefore $(TARGET).elf $(TARGET).hex
$(TARGET).eep \
      $(TARGET).lss $(TARGET).sym sizeafter finished end


# Eye candy.
# AVR Studio 3.x does not check make's exit code but relies on
# the following magic strings to be generated by the compile job.
begin:
      @echo
      @echo $(MSG_BEGIN)

finished:
      @echo $(MSG_ERRORS_NONE)

end:
      @echo $(MSG_END)
      @echo


# Display size of file.
sizebefore:
      @if [ -f $(TARGET).elf ]; then echo; echo $(MSG_SIZE_BEFORE);
$(ELFSIZE); echo; fi

sizeafter:
      @if [ -f $(TARGET).elf ]; then echo; echo $(MSG_SIZE_AFTER);
$(ELFSIZE); echo; fi



# Display compiler version information.
gccversion :
      @$(CC) --version




# Convert ELF to COFF for use in debugging / simulating in
# AVR Studio or VMLAB.
COFFCONVERT=$(OBJCOPY) --debugging \
```

Makefile sample

```
      --change-section-address .data-0x800000 \
      --change-section-address .bss-0x800000 \
      --change-section-address .noinit-0x800000 \
      --change-section-address .eeprom-0x810000


coff: $(TARGET).elf
      @echo
      @echo $(MSG_COFF) $(TARGET).cof
      $(COFFCONVERT) -O coff-avr $< $(TARGET).cof


extcoff: $(TARGET).elf
      @echo
      @echo $(MSG_EXTENDED_COFF) $(TARGET).cof
      $(COFFCONVERT) -O coff-ext-avr $< $(TARGET).cof




# Program the device.
program: $(TARGET).hex $(TARGET).eep
      $(AVRDUDE) $(AVRDUDE_FLAGS) $(AVRDUDE_WRITE_FLASH)
$(AVRDUDE_WRITE_EEPROM)




# Create final output files (.hex, .eep) from ELF output file.
%.hex: %.elf
      @echo
      @echo $(MSG_FLASH) $@
      $(OBJCOPY) -O $(FORMAT) -R .eeprom $< $@

%.eep: %.elf
      @echo
      @echo $(MSG_EEPROM) $@
      -$(OBJCOPY) -j .eeprom --set-section-
flags=.eeprom="alloc,load" \
      --change-section-lma .eeprom=0 -O $(FORMAT) $< $@

# Create extended listing file from ELF output file.
%.lss: %.elf
      @echo
      @echo $(MSG_EXTENDED_LISTING) $@
      $(OBJDUMP) -h -S $< > $@

# Create a symbol table from ELF output file.
%.sym: %.elf
      @echo
      @echo $(MSG_SYMBOL_TABLE) $@
      $(DIRAVRBIN)/avr-nm -n $< > $@



# Link: create ELF output file from object files.
.SECONDARY : $(TARGET).elf
.PRECIOUS : $(OBJ)
%.elf: $(OBJ)
      @echo
      @echo $(MSG_LINKING) $@
```

Makefile sample

```
      $(CC) $(ALL_CFLAGS) $(OBJ) --output $@ $(LDFLAGS)


# Compile: create object files from C source files.
%.o : %.c
      @echo
      @echo $(MSG_COMPILING) $<
      $(CC) -c $(ALL_CFLAGS) $< -o $@


# Compile: create assembler files from C source files.
%.s : %.c
      $(CC) -S $(ALL_CFLAGS) $< -o $@


# Assemble: create object files from assembler source files.
%.o : %.S
      @echo
      @echo $(MSG_ASSEMBLING) $<
      $(CC) -c $(ALL_ASFLAGS) $< -o $@




# Target: clean project.
clean: begin clean_list finished end

clean_list :
      @echo
      @echo $(MSG_CLEANING)
      $(REMOVE) $(TARGET).hex
      $(REMOVE) $(TARGET).eep
      $(REMOVE) $(TARGET).obj
      $(REMOVE) $(TARGET).cof
      $(REMOVE) $(TARGET).elf
      $(REMOVE) $(TARGET).map
      $(REMOVE) $(TARGET).obj
      $(REMOVE) $(TARGET).a90
      $(REMOVE) $(TARGET).sym
      $(REMOVE) $(TARGET).lnk
      $(REMOVE) $(TARGET).lss
      $(REMOVE) $(OBJ)
      $(REMOVE) $(LST)
      $(REMOVE) $(SRC:.c=.s)
      $(REMOVE) $(SRC:.c=.d)


# Automatically generate C source code dependencies.
# (Code originally taken from the GNU make user manual and modified
# (See README.txt Credits).)
#
# Note that this will work with sh (bash) and sed that is shipped
with WinAVR
# (see the SHELL variable defined above).
# This may not work with other shells or other seds.
#
%.d: %.c
      set -e; $(CC) -MM $(ALL_CFLAGS) $< \
      | $(DIRAVRUTILS)/sed 's,\(.*\)\.o[ :]*,\1.o \1.d : ,g' > $@; \
      [ -s $@ ] || rm -f $@
```

Makefile sample

```
# Remove the '-' if you want to see the dependency files generated.
-include $(SRC:.c=.d)



# Listing of phony targets.
.PHONY : all begin finish end sizebefore sizeafter gccversion coff
extcoff \
      clean clean_list program
```

Data Transfer Session Capture

```
TCP Connection Request
---- 13/4/2004 23:16:58.625

192.168.0.2 : 5000 TCP Connected ID = 1
---- 13/4/2004 23:16:58.625
Status Code: 0 OK
---- Data Sent
0000   0D 0A 2E 20 20 20 20 20 20 20 20 20 20 20 20 20   ...
0010   20 20 20 20 2F 5C 20 20 5C 7C 2F 20 20 2F 5C 0D       /\  \|/  /\.
0020   0A 2E 20 20 20 20 20 20 20 20 20 20 20 20 20 20   ..
0030   20 20 20 7C 5C 5C 5F 3B 3D 2E 5F 2F 2F 7C 0D 0A      |\\_;=._//|..
0040   2E 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20   .
0050   20 20 20 5C 2E 22 20 20 20 22 2E 2F 0D 0A 2E 20      \."   "./...
0060   20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
0070   20 2F 2F 5E 5C 20 2F 5E 5C 5C 0D 0A 2E 20 20 20    //^\ /^\\...
0080   20 20 20 20 20 20 20 20 2E 27 60 60 22 2C 2F 20          .'``",/
0090   7C 30 7C 20 7C 30 7C 20 5C 2C 22 60 60 27 2E 0D   |0| |0| \,"``'..
00A0   0A 2E 20 20 20 20 20 20 20 20 20 20 2F 20 20 20   ..          /
00B0   2C 20 20 60 27 5C 2E 2D 2D 2D 2E 2F 27 60 20 20   ,   '\.---./'`
00C0   2C 20 20 20 5C 0D 0A 2E 20 20 20 20 20 20 20 20   ,   \...
00D0   20 2F 60 20 20 2F 60 5C 2C 2E 22 28 20 20 20 20    /`  /`\,."(
00E0   20 29 22 2E 2C 2F 60 5C 20 20 60 5C 0D 0A 2E 20    )".,/`\  `\...
00F0   20 20 20 20 20 20 20 20 2F 60 20 20 20 20 20 28          /`     (
0100   20 27 2E 27 2D 2E 2D 27 2E 27 20 29 20 20 20 20    '.'-.-'.' )
0110   20 60 5C 0D 0A 2E 20 20 20 20 20 20 20 20 20 2F    `\...          /
0120   22 60 20 20 20 20 20 22 2E 5F 20 20 3A 20 20 5F   "`     "._  :  _
0130   2E 22 20 20 20 20 20 60 22 5C 0D 0A 2E 20 20 20   ."     `"\...
0140   20 20 20 20 20 20 20 60 2F 2E 27 60 22 3D 2E 2C          `/.'`"=.,
0150   5F 60 60 3D 60 60 5F 2C 2E 3D 22 60 27 2E 5C 60   _``=``_,.="`'.\`
0160   0D 0A 2E 20 20 20 20 20 20 20 20 20 20 20 20 20   ...
0170   20 20 20 20 20 20 20 29 20 20 20 28 0D 0A 2E 20          )   (...
0180   20 20 20 20 20 20 20 20 20 53 49 4C 56 45 53 54            SILVEST
0190   45 52 20 53 54 52 49 4B 45 53 20 41 47 41 49 4E   ER STRIKES AGAIN
01A0   0D 0A                                             ..

192.168.0.2 : 5000 TCP Data In Length 19 bytes
MD5 = F11BA996A471581D638A261D761CE812
---- 13/4/2004 23:17:06.671
0000   48 45 4C 4C 4C 4C 4C 4C 4C 4F 4F 4F 4F 4F   HELLLLLLLLOOOOO
0010   21 21 21                                    !!!

---- Data Sent
0000   44 61 74 61 20 72 65 63 65 69 76 65 64 20 6F 6E   Data received on
0010   20 50 43 20 73 69 64 65 0D 0A                      PC side..

192.168.0.2 : 5000 TCP Data In Length 20 bytes
MD5 = EF8808DC72F2C5DA0282E0512A6D14F8
---- 13/4/2004 23:17:17.968
0000   54 48 49 53 20 49 53 20 41 20 54 45 53 54 21 24   THIS IS A TEST!$
0010   25 5E 26 26                                       %^&&

---- Data Sent
0000   44 61 74 61 20 72 65 63 65 69 76 65 64 20 6F 6E   Data received on
0010   20 50 43 20 73 69 64 65 0D 0A                      PC side..
```