Implementation of modern communication interfaces in systems with existing serial interface

EMBEDDED SYSTEMS, OIL MONITORING

JAMIL NASSER
Abstract

Today’s robotics technology is often extremely user and location dependable, meaning that only a specific user using a specific hardware or software interface in a specific location can access that technique and alter it. This thesis focuses on an oil-monitoring system called Orilink, with those exact constraints. Orilink is only accessible through a specific computer using a serial port located in a special place. It is neither accessible from elsewhere nor through the cloud. This thesis removes that constraint and enables the access to Orilink independently of the user’s location or software / hardware interface.

Keywords
Embedded system, Ethernet, USB, Serial RS-232, Serial RS-485
Table of Contents

1 Introduction ........................................................................................................... 3
1.1 Background ........................................................................................................ 3
1.2 Problem .............................................................................................................. 3
1.3 Purpose ............................................................................................................... 3
1.4 Goal ................................................................................................................... 4
1.4.1 Benefits and problems ................................................................................... 4
1.5 Delimitations ..................................................................................................... 5
1.6 Outline (Disposition) ......................................................................................... 5

2 The OriLink system hardware and software ......................................................... 6
2.1 The Alentec & Orion AB oil monitoring system .................................................. 6
2.2 OriLink oil managing system internal communication ..................................... 10
2.3 Hardware .......................................................................................................... 11
2.3.1 Component description ................................................................................ 11
2.3.2 Reasoning behind the choice of prototype board components .................. 12
2.3.3 Electronic component connections ................................................................. 13
2.4 OriLink PC-software ......................................................................................... 13

3 Initial Design ....................................................................................................... 14
3.1 Studies about similar switches, data structures and algorithms ...................... 15
3.2 Software architecture ....................................................................................... 16
3.3 Main event loop ................................................................................................ 19
3.4 Programming environment .............................................................................. 21
3.4.1 Appropriate programming language for the purpose .................................. 22
3.4.2 Existing microcontroller libraries ................................................................. 22

4 Detailed design .................................................................................................. 23
4.1 Hardware .......................................................................................................... 23
4.2 Software .......................................................................................................... 24
4.3 Boot loader ........................................................................................................ 26
4.3.1 Microcontroller configuration .................................................................... 27
4.4 Kernel ............................................................................................................... 27
4.4.1 Controller initialization .............................................................................. 28
4.4.2 Development, debugging and testing controller application ...................... 29
4.4.3 Main function loop ..................................................................................... 32

5 Result ................................................................................................................... 36

6 Conclusions ....................................................................................................... 37
6.1 Future work ..................................................................................................... 37

References ............................................................................................................. 38

Appendix A ............................................................................................................. 41
Appendix B ............................................................................................................. 44
Appendix C ............................................................................................................. 45
Table of Figures

Figure 1. Design of the current system ................................................................. 3
Figure 2. Design of the new system ........................................................................ 4
Figure 3. An example of many possible connection layouts of the system .......... 6
Figure 4. MPDM ..................................................................................................... 7
Figure 5. Key Pad .................................................................................................... 7
Figure 6. Printer Module ......................................................................................... 8
Figure 7. PC-interface with null modem cable .......................................................... 8
Figure 8. LED Display ............................................................................................ 9
Figure 9. Tank Module ........................................................................................... 9
Figure 10. OriLink communication protocol example ............................................. 10
Figure 11. Prototype board for OriLink system ...................................................... 11
Figure 12. SIO2 schematic .................................................................................... 41
Figure 13. PIC18F8722 pin layout ......................................................................... 42

Table of Tables

Table 1. Pin description ........................................................................................ 42
1 Introduction

This chapter introduces a background to the problem, in order to understand the reasoning behind implementation of packet based protocol for industries. Background, the problem, specification of goals and an outline of the thesis are summarized in this chapter.

1.1 Background

Alentec & Orion AB[3] is a Swedish based global corporation focusing on the development of hydraulic pumps. Over the years they have developed an oil monitoring system called Orilink[4], and as for now, Orilink can only be accessed through a special terminal connected to the Orilink hardware via an RS-232 serial cable. For the everyday use, such as oil monitoring this doesn’t pose a problem.

1.2 Problem

Once the system needs an update or debugging, the system access obstructs the work because of the connection constraints. In order to debug or update the system, a technician needs to be physically present at the location of the Orilink hardware or software. In case of failure or emergency update, the system is then put on hold until a suitable technician is relocated to the system itself. This is a particular issue when a company using multiple Orilink systems needs to perform system updates. The updates must be performed manually for each and every system which is not very time effective. This leads us to the limits opposed by the Orilink software that are defined by the connection through serial RS-232, the physical linkage to a technician on spot, and the fact that the terminal connected to the software needs to be a computer, Figure 1.

![Figure 1. Design of the current system](image)

1.3 Purpose

The world of robotics is in the process of evolving but the outside world has already evolved. An example illustrating this is the communication through the cloud. It has long been a standard in our daily life, we practically breathe wireless communications, but in the world of robotics it is still considered as high-tech.
In order to remove the constraints opposed by the system, OriLinks availability needs to improve drastically, thus meaning extending its accessibility from anywhere around the globe through various connection types. The vision is to alter Orilink and make it accessible through not only serial connections, but even through USB, Ethernet, the cloud and using a portable device, Figure 2. This will not only transform the system and enhance the user friendliness and flexibility, but will even make it more accessible and susceptible for up to date communication types.

1.4 Goal

What makes this vision so interesting is that it doesn’t stop in being just a vision; it has the possibility of evolving into a real product. Since this subject is new to oil monitoring systems, the project requires careful planning. By the time this report has reached the conclusion, Ethernet, USB and serial communication protocols will be implemented in the new OriLink module and accessible from OriLink PC software.

1.4.1 Benefits and problems

From security perspective, it is not always good to open up the system to the outside world, as is done using Ethernet. The system’s great advantage is also the system's weak point. Today’s modern operating systems have highly advanced security systems to counter hacking and thus weaken network system. Those advanced networking security features are currently not implemented in to the oil monitoring systems hardware, which makes it easier to force the network through the oil monitoring system.
1.5 Delimitations
This thesis is about developing embedded software for the existing Alentec & Orion prototype card. Designing the prototype board is outside the scope of this project. Because the finished prototype hardware exists, it puts limits on the architecture of the embedded software.

Similarly, the requirement of use of the existing Alentec & Orion development libraries makes its mark in the design of the embedded software. This work will relate only to embedded software and light debugging PC software. The development of Orilink PC software is outside the scope of this thesis.

1.6 Outline (Disposition)
In chapter 2, we discuss the OriLink system, with the intend to familize the reader with OriLink oil monitoring system. Studies about a fitting algorithm for the OriLink module embedded software, embedded software design and presentation of the development environment are presented in chapter 3. Chapter 4 describes the design achieved in chapter 3, and implementation of the embedded software model in to OriLink module. Chapter 5 presents the results and the conclusions are discussed in chapter 6.
2 The OriLink system hardware and software

This chapter describes the OriLink oil monitoring system hardware and software. Existing Alentec & Orion prototype board for which embedded software was developed to, through this thesis, is the main goal of this chapter. The prototype board is meant to become part of Alentec & Orion oil monitoring system.

The Alentec & Orion new module (prototype board), was developed in respect of other modules in the system. One has to familiarize with the whole Alentec & Orion oil monitoring system, in order to understand the design of the current hardware.

2.1 The Alentec & Orion AB oil monitoring system

Alentec & Orion oil managing system is called OriLink. It’s a modern and advanced oil management system consisting of several modules. Each module is a part of multi-master network, which can be explained as a network of modules, where all modules have the same priority in the network [5]. OriLink pc software makes it possible to overview the workshop oil system layout on the computer screen where all activities are graphically visible in real time. Figure 3 illustrates an example of many possible connection layouts of the system. This section describes only OriLink modules in Figure 3. The ticket printer, reels, oil tanks etc in Figure 3, are outside the scope of this thesis and will not be explained. Modules according to Figure 3 numbers are explained below.

![Diagram of OriLink system](image)

Figure 3. An example of many possible connection layouts of the system
OriLink oil managing system consists of following modules:

1. Main Module (MPDM - Multi Point Dispense Module)
   - MPDM, Figure 4, is used to control solenoid valves for operating reels, to dispense oil. Each MPDM has the power to control up to four solenoid valves.

   ![Figure 4. MPDM](image)

2. Key Pad
   - The keypad, Figure 5, is used to give the operator access to the system using a personal ID-code. It can even be equipped with a serial port kit for external readers.

   ![Figure 5. Key Pad](image)
3. Printer Module
- The printer module, Figure 6, enables a connection to the oil managing systems printer. The transaction information is stored in the printer module should the system not be connected to a PC.

![Figure 6. Printer Module](image)

4. PC-interface
- The PC-interface, in Figure 7, enables a connection from the oil managing system to a PC. It can also be used as a line amplifier when the communication cable is longer than 2000 meters. This thesis is about replacing the existing old serial PC-interface with the existing Alentec & Orion prototype board (new OriLink PC interface). The new PC-interface will not only connect OriLink to a PC, but even to WAN and LAN. Some parts of the existing OriLink PC software functions will be implemented in the new PC-interface embedded software, thus making it possible to connect the system directly to cloud and servers.

![Figure 7. PC-interface with null modem cable](image)
5. LED Display

- The LED Display, Figure 8, displays the dispensed volume. It shows the actual time when the system is not in use. This is only possibly if the LED Display is equipped with a clock module or connected to a PC.

![Figure 8. LED Display](image)

9. Tank Module

- Figure 9 illustrates the OriLink tank module. Which is used for operating pumps and sensors that are connected to oil tanks. It has four connections for controlling air solenoid valves to turn the pumps on or off. The tank module can also work together with a low level or low and re-order sensor. It even controls waste oil solenoid valves and sends re-order warning messages to the OriLink PC software.

![Figure 9. Tank Module](image)
2.2 OriLink oil managing system internal communication

OriLink modules communicate through the RS-485 serial protocol when connected to each other, creating a multi-master network. The PC-interface module used to connect the OriLink system to a PC, communicates with a PC using the RS-232 serial protocol. It also communicates with connected modules using the RS-485 serial protocol. The RS-485 serial communication protocol transmits at 4800 bits per second, whilst the RS-232 serial communication protocol transmits at 19200 bits per second.

OriLink network addresses data as bytes. The protocol language is in hexadecimal digits, where one byte can hold two digits.

Every data package starts with the hexadecimal letters FF FF FF and ends with a CRC-checksum for the package. The only exception resides in the module properties data package sent from a PC, that start with the module address that OriLink PC software wants to communicate with.

An example from a system communication:

The Free Serial Port Monitor version 3.31 was used to monitor the communication between the OriLink system, and the OriLink PC software “Engine Pro”, version 1.010RC7.

Figure 11 shows the communication between the OriLink PC software and the OriLink oil managing system.

This particular system consisted of five modules: one MPDM module, two Printer modules, one keypad and one LED module.

PC communication is marked in red, Figure 10, and the OriLink hardware communication is marked in black. The PC software initiates the communication by sending the message “FF FF FF”. This message is then broadcasted by the PC-interface to all active ports, even back to the PC. Consequently, the PC-software sends a standard update request “DF 00 04 E3” that is acknowledged by every module by sending back their system addresses to PC. This enables the PC software to identify the modules and their configuration in the system.

Figure 10 illustrates the answer from a MPDM at address 1100, towards a PC with the software version 10rc55, module specific properties and module settings.
2.3 Hardware

A prototype board specially designed for OriLink oil managing system had to be created by Alentec & Orion, due to technical specifications unavailable on already existing boards. For the board to be created, the existing PC interface was taken and used as a platform. Hardware components were later replaced and added in order to fit the specification criteria. The detailed hardware design of the new PC interface is outside the scope of this thesis. The prototype board is illustrated in Figure 11.

![Figure 11. Prototype board for OriLink system](image)

2.3.1 Component description

One of the main and important components of an electronic board with built in intelligence is the main central processing unit (CPU). Microchip PIC18F8722, is a microcontroller that is used as CPU in the new OriLink PC-interface. The OriLink PC-interface will hereafter be referred to as SIO2. PIC 18F8722 is an 8-bit microcontroller and has a working speed of 25 MHz that is determined by the external 25Mhz crystal on the SIO2 electronic board.

It is hard and not cost efficient to implement all the functionality directly into the microcontroller, that is where external, standalone chips, also called controllers, comes in to the picture. A controller makes it easier to program and design the electronic board. Most commonly used controllers are communication controllers. All communication on the SIO2 is done via controllers.

Following main electronic components are found on the OriLink SIO2 electronic board:

- **Main processor (CPU):** Microchip PIC 18F8722, Figure 13.
- **External crystal:** 25Mhz
- **Ethernet communication controller:** ENC28J60
- **USB communication controller:** MAX3420
- **Serial RS232 communication controller:** MAX3100 and MAX232
Serial RS485 communication controller: two MAX485 controllers

2.3.2 Reasoning behind the choice of prototype board components

This section describes the components Alentec & Orion used in the creation of the prototype board, which is a necessity in the realization of the OriLink new PC-interface. Common denominator of all components is cost.

Microcontroller, PIC18F8722

Choice of microcontroller determined easily with respect to firstly minimum requirements of the system and secondly the cost of the microcontroller.

PIC18F8722[6] microcontroller has two Master Synchronous Serial Port (MSSP) modules supporting 3-Wire serial peripheral interface (SPI) and two EUSART modules, serial communication modules, see Figure 21. Which simply are interfaces used for communication between microcontroller and peripheral in embedded system. SPI is used for communication to Ethernet, USB and serial RS-232 controllers in the OriLink new PC-interface, also called SIO2. The microcontroller has also three programmable external inputs and four input change interrupts for various communication controllers, like RS-485 controllers. PIC18F8722 microcontroller also has enhanced addressable USART modules supporting RS-485 and RS-232.

Ethernet controller, ENC28J60

ENC28J60[7] Ethernet controller is a 28-pin, 10BASE-T stand-alone Ethernet controller. It has a SPI serial interface and an on board MAC & PHY with 8 Kbytes of Buffer RAM. The advantage of using this controller is the SPI interface for communication and on board MAC.

USB controller, MAX3420


Serial controller, MAX3100

Max3100[9] Using an SPI interface for communication with the host microcontroller (µC).

Serial controller, MAX232

The MAX232[10] controller converts signals from an RS-232 serial port to signals suitable for use in TTL compatible digital logic circuits. The MAX 232 is a dual controller/receiver and typically converts the RX, TX, CTS and RTS signals. Advantage of this controller is that it can provide RS-232 voltage level outputs (approx. 7.5V) from a single +5V supply.

Serial controller, MAX485
MAX485[11] controllers are compatible with Enhanced Universal Asynchronous Receiver Transmitter (EUSART), which is supported by PIC 18F8722 microcontroller.

2.3.3 Electronic component connections

Microcontrollers’ physical connection to the current communication controllers has to be known prior, in order to be able to create application for a microcontroller. The connection of the electronic components is not easy task. All connections of the prototype board is made in view of electromagnetic compatibility (EMC[12]) and CE[13] requirements.

See Appendix A for further, detailed information about the prototype board hardware.

2.4 OriLink PC-software

This section describes OriLink real time PC services. OriLink plug and play array, which registers OriLink modules in the system and HWFlash, OriLink software that is used to reprogram OriLink modules are real time PC services. Real time PC services have to be moved to the new PC interface, in order to eliminate the need of PC.

Figure 17 illustrates OriLink PC software called “Engine”. Engine has to be executed in order to establish communication with the OriLink hardware.

![OriLink Engine Pro](image)

**Figure 12 – OriLink pc software, Engine**

At startup, Engine initiates a plug and play array for connected OriLink modules. The plug and play array detects all connected modules, their specific software version, address and configurations, Figure 18.
HWFlash, Figure 19, is a PC based software, used for upgrading, changing and reinstalling OriLink modules software, including languages. It uses binary files for programming of module specific software, which is included in the installation directory of the OriLink pc software.

Figure 14 – HWFlash, flashing software

3 Initial Design

Studies about a fitting algorithm for the SIO2 embedded software, embedded software design and presentation of the development environment are presented in this chapter.

A high level structure of a software system is needed in order to get a good overview over software design. The high level specification of software artifact
helps to find flaws and to make it easier to optimize the end result of the software.

The criteria for embedded system software for the microcontroller are, it should be small and efficient and be able to fit in the target microcontrollers program memory area. OriLink PC-interface has to be also quality-driven, which means that OriLink PC-interface should be able to receive and forward data packages without corrupting them, because of it handling total communication of the OriLink oil management system.

3.1 Studies about similar switches, data structures and algorithms

According to George Karypis[14], a Professor at the Department of Computer Science & Engineering at the University of Minnesota, The most important criterion for routing is where and when the routing function is determined.

Routing and switching communication problem can be broken into several sub categories; according to the number of source nodes, destination nodes, and the way the transferred information is handled.

Pavel Tvrdik[15] has divided routing algorithms in three major classes;

**One-to-one communication:**
Information is exchanged between one or several pairs of nodes, node to node, information is not duplicated.

*single-pair communication:*
From one node to another node, just one isolated communicating pair.

*many one-to-one communications:*
Still one to one but several pairs of nodes exchange information simultaneously.

*permutation routing:*
A node can send and receive messages from different nodes at the same time. Every node is the source of one message and the destination of another message at the same time.

**One-to-many communication (collective communication):**
One node is the sender and several other nodes, or all nodes, are the receivers.

*Broadcast*
When the same information is disseminated into multiple copies among the other nodes, it is called a multicast and a one-to-all broadcast communication operation.
**Scatter**
If the information sent to each destination node is different, the pattern is called a one-to-all scatter.

**All-to-all communication (collective communication):**
Every node can send and receive information from every other available nodes where it can be an all-to-all broadcast or a scatter

The communication type for the OriLink system network is collective, because the OriLink network uses serial communication in between modules. Furthermore, all modules has to be quiet and listen when a module transmits, defining the serial network type to one-to-many communication.

With that in mind, One-to-many communication must be implemented in to the SIO2 embedded software, in order for it to become a part of the OriLink network.

### 3.2 Software architecture

One of the main tasks of SIO2 module is to register Orilink modules in the network. Since Orilink modules are of the hot swap type, the list of registered modules is called “plug and play array”, abbreviated to PNP-array.

When the board is connected to the appropriate power supply, SIO2 module should start its operations as follows;

Below is the set of structures needed to reason about the SIO2 embedded software system. Flow chart of the SIO2 software is illustrated in Figure 14.
1. **Initialize module**

When SIO2 is connected to appropriate power supply and the microcontroller successfully manages to stabilize, it looks up for microcontroller software and starts to execute it. Initialize module is the first paragraph of the SIO2 module that is executed.

- Define microcontroller type and setting up the I/O pins.
- Configure timers
- Configure all communication controllers connected to the microcontroller
- Read module address from memory configuration part
- Read configuration data, module specific address
- Initialize controller specific communication methods
2. **Check for connected devices**

   It’s time for the SIO2 to find active communication ports, after Initialize module has successfully completed.

   - Check interrupt from communication controller
   - Put them on active port list, if there are any

3. **Find connected OriLink modules**

   If we find any active ports, that means that there are connected modules, send them a request message to identify them.

   - Send an standard acknowledgment request on RS-485 serial ports

4. **Register modules**

   The modules will answer with type of module and module specific address. Save the address according to type of module from the connected modules on special dedicated memory area on chip.

   - Wait for modules to replay with acknowledgment plus module addresses
   - Register and save module addresses

5. **Notify OriLink PC software**

   OriLink PC software has to first register modules, in order to send the information about the connected modules to PC.

   - Send SIO2 configurations and address to PC
   - Send connected module name and addresses

6. **Start with the main chores**

   Start a main loop, where the following should be done:

   - Flash with power LED on the board (alive)
   - Handle data communication
7. **Update PNP-array**

OriLink modules and the PC connection are hot plugged in. More specific, SIO2 should be able to handle connecting and disconnecting connection to the board without shutting down the system.

All OriLink modules send a connection request at power on or first time connection but not if they are unplugged from the system. To detect unplugged modules from the system, it has to be able to update the system list every 10 sec.

### 3.3 Main event loop

Generally, a main loop is used in an event-driven[24] application. The main loop of an event-driven application listens for events and then triggers a callback function, when an event is detected. In embedded system, one usually don’t implement a constantly running loop, instead, the same may be achieved by using hardware interrupts.

There are no strict rules for how embedded system main loop should be. A suitable main loop for purpose is the best solution. Event loop is a specific implementation technique of systems that use message passing and forms the central control flow construct of a program. An event loop is one of the methods for implementing inter-process communication.

Functions in this section are most important parts of intercommunication design of SIO2. SIO2 main loop design is illustrated in Figure 15.
Figure 16. Main event loop for SIO2

- **Enable interrupts**
  Interrupts disables when an event is taken care of, needs to be enabled

- **Clear watch dog timer**
  A watch dog timer is a must in embedded system software to recover hardware or software malfunction. The watch dog timer is cleared every loop sequence for preventing it to time out.

- **Check communication controllers for incoming data**
  Constantly checking interrupt handlers routines for incoming data.

- **If data received**
  Incoming data should be copied to a common buffer that is checked every loop cycle.
    - *if the message and the module address matches*
      The message is intended for the OriLink PC-interface, when the received message header in the buffer matches OriLink module address. The intended messages could be of two types:
- **Switch PNP update**  
  First type, where the module sends an alive message to the PC.

- **Switch ON menu update**  
  Second type, where the message is intended for the internal menu, for example changing OriLink PC-interface module address
  - *Forward the message to all active ports*  
    Received packet was not intended for the OriLink PC-interface module, broadcast it on all active ports.

- **When the timer counts to 100ms**  
  To indicate that module is alive and not malfunction, the system will
  - Flash on board LED light
  - Check if there are any IP activity

- **When the timer counts to 1000ms**  
  OriLink PC-interface will periodically, every 10 seconds, check for any changes in the network.
  - *Send PNP update request to all connected modules and PC*

- **Break from the loop and go to start**  
  Collect information about modules and send them to PC

Embedded software for the microcontroller should be small and efficient and be able to fit in the program memory area.

SIO2 has to transmit and receive data from OriLink software throw Ethernet, serial port and USB, unlike the old OriLink PC-interface that is using only serial port. That puts different demands on embedded software for OriLink PC-interface microcontroller.

### 3.4 Programming environment

It is extremely important to set the right environment for the software development before the development starts. Right environment gets particularly important when the product to be developed, must fit into already existing system, but also has to be easily modified in the future.

Development of software for embedded system requires more careful planning, because of the limited debugging ability of the hardware, compared
with debugging PC software. Microcontroller requires its specific programmers and devices in many cases require special library to be included.

3.4.1 Appropriate programming language for the purpose
Microcontrollers are a part of the low- and mid-level hardware[16]. The Low- or mid-level hardware is only programmable by suitable low- and mid-level programming languages.

A microcontroller [17] is often programmed in low-level programming language, Assembler. "A low-level programming language provides little or no abstraction from the microcontroller's instruction set architecture" [18].

Several high-level programming languages are also common use to target microcontrollers. These languages are either designed especially for a special purpose, or for more common languages such as the C programming language. Some microcontrollers have environments aiding the development of certain types of applications. Microcontroller vendors often give out tools freely to facilitate the adoption of their hardware.

The microchips MPLAB XC compiler [19] is used to program microchips range of microcontrollers, PICs, with high-level C and C++ programming languages. It has enough environments to assist in developing code for the PIC microcontroller in C and C++ programming languages.

3.4.2 Existing microcontroller libraries
Following are the latest relevant libraries used for developing SIO2 embedded software.

The Microchip MPLAB X IDE [20] is the microchips latest software tool. It provides supporting libraries for developing C and C++ programming language code for PIC series of microcontrollers. MPLAB X IDE was unfortunately not used in this thesis, because of compatibility issues with Windows XP, Hitech PICC-18 was used instead, for Windows XP.

A TCP/IP stack is the software used when driving the Ethernet device peripheral hardware on a microcontroller. Typically, the TCP/IP peripheral hardware only supports the transaction level and below the TCP/IP protocol. Enumerations and transfers are left to the firmware or software to implement.

Microchip developed a TCP/IP stack library[21] for PIC microcontrollers. The microchip microcontroller TCP/IP stack, along with library files for serial communication, is found in the Microchip Libraries for Applications (MLA)[22].

A USB stack is needed in the implementation of the USB functionality, like the TCP/IP stack.
M-Stack[23] is a free and open source implementation of a USB stack for Microchip PIC platforms.

4 Detailed design

This chapter describes the design and implementation of the OriLink SIO2 software. The model implemented in this chapter is completed in chapter 3.

The OriLink system design holds a communication and configuration library specifically designed for OriLink. The SIO2 module was designed and created in that same environment in order to take advantage of those libraries.

The SIO2 embedded software can be divided into two parts, boot loader and kernel:

- Boot loader is a program that loads kernel software in to the module. It should be programmed into the SIO2 board in conjunction with the production of the board. The advantage of using the boot loader is that it doesn't require any development environment or any hardware programming tool for programming SIO2 module.

  The SIO2 board can be shipped directly to the customer where the module is later programmed with the desired software version, language and configurations.

- The kernel is the part of the embedded software that has all the SIO2 module software logic functions. The kernel makes the module function as it is defined.

4.1 Hardware

The physical tools available for software development of SIO2 at Alentec facilities were the following:

ICD2 - Microchips programming hardware interface, for programming PIC microcontroller.

An OriLink oil monitoring system - consisting of OriLink modules and an old OriLink PC interface.

The Microchip programming hardware, ICD2, is limited to Windows XP as a development environment. Mplap X or Environments newer then Windows XP does not provide any support or controllers for the ICD2 programming hardware.
4.2 Software

The following software development tools are relevant for the OriLink development environment:

- **Operating system:** Windows 7 Professional, 64bit
  Windows XP SP2, 32bit
- **Virtual machine:** VMware Player 7.0.0
- **Compilers:**
  - Microsoft nmake 6.00.9782.0
  - Microchip Mplab IDE 8.92
  - HI-TECH PICC-18 STD 9.52
  - Microsoft Visual Studio 2010
- **Tools:**
  - OriLink Engine Pro 1.010RC7
  - Free Serial Port Monitor 3.31

SIO2 board was programmed using ICD2. Since Windows XP is the only operating system containing the required support and controllers for that type of programming and I was currently sitting on a Windows 7 environment, a virtual machine was used to access Windows XP.

Programming environment, containing OriLink specific library files, were copied from Alentec backup server to newly installed Windows XP operating system on virtual machine.

A folder named “Sio” was created for development of SIO2 in the Orilink18 folder. This is further illustrated in Figure 16. Where the programming environment for SIO2 were inserted.
“Orilink 18” folder contains development software for all OriLink hardware that has been developed, using PIC18 microchip as a main microcontroller. Library files for Microchip PIC18-based OriLink modules are stored in a separate folder, called “Lib”. The “Lib” folder is embedded in the blue frame in Figure 16.

A bat file is PC instructions stored in a file. setenv951plo.bat was used for setting up OriLink specific environment variables for the operating system Windows XP, Figure 16.
4.3 Boot loader

Boot loader is the only preinstalled software in an OriLink modules default. OriLink boot loader makes it possible to reinstall module software if the module kernel malfunctions or a corrupted.

When connected to the PC through OriLink pc interface, Engine detects the blank module and registers it at plug and play array using the default address DFF1. HWFlash is then used to program and flash the new module with the appropriate language and software.

A boot loader was created in order to connect and register the new SIO2 module to the Engine PC software. This way, the module become part of the OriLink oil monitoring system.

The general function flow of the OriLink SIO2 boot loader is demonstrated in Figure 20.

Figure 18. SIO2 boot loader flow chart

Already created OriLink library files that were used, beside Microchip and HITECH library files, to create the boot loader are following:

- **NetA.c**
  Holds all the OriLink Verbs communication functions and configurations for RS-485. Such as packet length, headers, CRC-checksum and so on.

- **TimerA.c**
  Timer functions for the module and also for synchronizing OriLink modules in between and with pc.
MenuA.c
OriLink Verbs decoder, all the functions that are needed after a packet is received by a module. For example, decoding headers and messages and also creating answers for requests.

### 4.3.1 Microcontroller configuration

In order for the boot loader to function properly the microcontroller had to be configured. The microcontroller has to be aware of following configurations, Clock source and clock speed of the crystal, which in our case is external, high speed oscillator at 25Mhz.

In order to get boot loader working, it is sufficient to configure one of RS-485 channels, channel one in this case. The pin names were specified according to the OriLink predefined library: NetA.c and Net.h.

### 4.4 Kernel

In the domain of embedded systems, the kernel is a central piece in this technical puzzle and is often referred to as the heart. The OriLink SIO2 Kernel is the software that controls the entire system by managing every function, providing functionality to the communication network and handling the embedded system functionalities. Its reliability is therefore of high importance and is determined by which functions it’s expected to deliver in contradiction to what’s actually delivered.

As mentioned in section 4.1.3, the boot loader is essential and plays a key role in the uploading of the kernel into OriLink. The kernel and the boot loader share the same embedded software base, with the only difference that the kernel also implements all on board controllers and all needed functionality for a well-functioning module.

The Embedded SIO2 software consists of various major parts, they are as followed:

- Initialize and configure the microcontrollers internal units
- Initialize and configure controllers connected to the microcontroller
- Read the customized configuration such as module specific serials and MAC addresses.
- Initialize software (timers, buffers, flags etc.)
- Begin the main loop

Prior to the main loop to be architected for standard OriLink calls, standard procedures must be implemented and followed. Which will ensure that SIO2 will function as a part of OriLink system.
4.4.1 Controller initialization
Almost all microcontrollers on the market have embedded communication units, which makes easy connecting external peripherals to it, from programmer point of view.

Before achieving a successful communication to external peripheral, microcontroller’s internal units has to be configured to pass the connected peripheral.

PIC18F8722 microcontroller, has following modules for communicating with external communication;

- two MSSP, Master Synchronous Serial Port
  * Is a serial interface, for communicating with other controllers or microcontroller devices.

- two EUSART, Enhanced Universal Synchronous Receiver Transmitter
  * translates data between serial and parallel forms. communication standards such as RS-232, RS-422 or RS-485 are commonly used in conjunction with UARTs.

![Figure 19. PIC18F8722 embedded modules](image)

ENC28J60 (Ethernet), MAX3100 (RS-232) and MAX3420 (USB), are all connected to the MSSP1. While RS-485 channel one is connected to EUSART1 and RS-485 channel two is connected to EUSART2, fig xxx.

**MSSP configuration**
MSSP1 module is connected to three following I/O pins; RC3, RC4 and RC5. Which for Serial Peripheral Interface, SPI, are defined; RC5 as serial data output, RC4 as serial data input and RC3 as serial clock pin.
Additional pins that’s needed except for MSSP pins are for interrupt, chip select, when there is more than one connected controller, and device reset, seen in Table 1.

**EUSART configuration**
EUSART is configured as asynchronous, means that data can be transmitted intermittently rather than in a steady stream, with auto-baud calibration. Just two I/O pins are used for EUSART, because it uses asynchronous communication, where serial clock pin is leftover.
EUSART1 uses RC6 as transmit and RC7 as receive pin. Where EUSART1 uses RG1 as transmit and RG2 as receive pin.

External controllers connected to microcontroller have to be configured as well, in order to make SIO2 work

MAX3100 controller, Serial RS-232, needed to be configured with appropriate baud rate for the OriLink interface, which is 19200b/s. MAX3420, USB, needed configuration for the SPI to be configured as master.

4.4.2 Development, debugging and testing controller application

Serial RS-485

RS-485 had to be the first communication chip to be working, in order to make boot loader work. The boot loader was compiled with nmake and HEX file programmed in the SIO2 by ICD 2 programmer. Functionality of RS-485 was verified as soon as OriLink HWFlash found SIO2 boot loader.

Boot loader and RS-485 communication is the backbone of the SIO2 binary file. In order to make the SIO2 function as the old SIO, RS-232 needed to be activated, which is standard SPI output and input procedure, as following:
Checking interrupt by reading status of external interrupt pin, RB2.
Selecting the MAX3100 controller by resetting chip select (RD4) pin, and then transmitting and receiving characters by writing/reading to MSSP1 Receive Buffer/Transmit Register. Where MSSP1 interrupt flag bit indicates when transmission/reciption is complete. (SSP1IF)

Developing and debugging serial port, RS-232.
The serial port, RS-232 on the SIO2 was connected to PC, while RS-485 was connected to the OriLink system. Serial Port Monitor and Windows Hyper Terminal were used for debugging proposes.

Developing and debugging Ethernet, ENC28J60
Ethernet controller, which is an important point in this thesis, was developed almost exclusively with Microchips TCP/IP stack library. Microchip TCP / IP stack operates as follows;

The source file, “StackTsk.c”, of Microchip TCP/IP implements a special application module known as “StackTask”. When given processing time, StackTask polls the MAC layer for valid data packets. When one is received, it decodes it and routes it to the main function for further processing.
A temporarily MAC address of 001122334455 was given to the SIO2. Dynamic Host Configuration Protocol, DHCP, was also enabled in order to receive dynamically IP address.

DHCP is used for dynamically distributing network configuration parameters, such as IP addresses. SIO2 module request IP addresses and networking parameters automatically with DHCP, from a DHCP server.

Also, NBNS was enabled in order to decode the IP addresses with name. NBNSTask, provided in Microchip TCP/IP library is used to for this purpose, see Figure 23.

Then, running TickInit() and StackInit(), provided in Microchip TCP/IP library, is all that requires to make the broadcast on Ethernet. If a connection request is made on broadcast, a connection is established.
For development and debugging purposes of Microchip TCP/IP stack implementation in SIO2 board, two C#.NET software was developed. One for detecting the SIO2 in the network, and another for connecting to it.

**Detecting SIO2 IP address**

For managing and adding the SIO2 to the network, it is important and crucial point to know its IP address. A C#.NET application was created, Application1, for this propose. It broadcasts a message, "finger print", to all addresses (255.255.255.255) and port 992, the one that SIO2 listens to, using UDP layer. The IP address and software version registers, when SIO2 replays the answer, see Figure 24.

![Figure 22 – IP recorder](image)

Then, a new C #software, Application2, was created using .NET sockets to connect to SIO2 by IP address and telnet port 23, retrieved by application 1.

To concentrate just for developing of SIO2 software, comocom software, which emulates serial port, was used. Application 2 outputs data to virtual com port where OriLink Engine reads from the same virtual com port. Because Engine still uses its com port connection, there were no need for modification in OriLink Engine software for it.

**USB, MAX3420**

Uses same standard SPI as mentioned above for reading and writing.

USB stands for Universal Serial Bus device class. The class may include more than one interface, such as a custom control interface, data interface, audio, or mass storage related interface. Communication Device Class, CDC-ACM, is the class suitable for applications like OriLink SIO2. CDC-ACM class makes it possible to use USB as if it was RS-232.

The advantages are for CDC-ACM:
* no need to modify the OriLink pc software, for USB connection.
* few hardware and software embedded modifications

Figure 25 shows similarities between CDC-ACM and RS-232.
To accomplish successful USB device implementation, c code example from Maximintegrated, AN 3690[25] was modified and implemented into the SIO2.

AN 3690 demonstrates how to use USB as standard HID class. The code was then modified according to an example code[26] for Atmel microcontroller to achieve CDC-ACM class.

4.4.3 Main function loop
Main function loop combines all the communication elements to function as one communication unit. The main idea of this main loop is that it should check every communication interface specific main functions, for received data.

The flowchart for each communication controller in the SIO2 looks like in the Figure 26;
Copy the data to the global buffer, if there is any.
The status of the buffer is checked in the main function, below. Before entering the main loop, some specific configurations have to be done.

- Register SIO2 module boot reason.
  One of great advantages of PIC18F8722 is that it can detect boot reason, by reading the status of the onboard reset button and status of RCON register.

- Initializing timers
  By setting Presale value to 1:16 at 25 MHz in T0CON, Timer 0 Control Register, which are done by OriLink timer library file.

- Set the module address to one defined by user.
  OriLink modules are identified by their unique address given by user. Set default address, which is 900, if user has not defined one.

The SIO2 main loop has also must have support for the OriLink message processing system. It has to have answer to every combination of requests from OriLink pc software.

**SIO2 main function structure**

Following is a detailed description of SIO2 main functions. See figure 14 and figure 15 for flow chart.

- Clear watchdog timer
A watchdog timer is an electronic timer that is used to detect and recover from computer malfunctions. The timer will elapse and generate a timeout signal, if not cleared.

- Enable all I/O pins
  Reset all pins to I/O pins, regardless of previous state.
- Disable comparators
  Turn off the comparators, regardless of previous state
- Set I/O pins as digital
- Turn off system LED
  System LED light
- Turn off all SPI
- Register SIO2 module boot reason
- Initialize timer
  Set global timers
- Initialize Net
  Initializes OriLink communication protocol and RS-485 controllers
- Set unit address to 900
- Initialize Ethernet
- Initialize Telnet
  For TCP/IP communication on port 23
- Initialize RS-232
- Initialize USB
- Enable interrupt priority

MAINLOOP:

- Enable high and low priority interrupts
- Clear watchdog timer
- Ethernet main
- Telnet main
- RS-232main
- USB main
- Command 0100 main
  For HEX communication
- Message received
  - Does the message & module address match
    - Switch buffer [3]
      If the message is addressed to the module, 4th word is a command, below
      - Case 20, update pnp
        Send the PNP array
- Case F0, menu dispatch
  SIO2 menu configuration
  - Is address DF00
    Standard, broadcast address
    - Update pnp
      Send module address as answer to request
- Timer 100ms
  - Flip system LED
    Blink with LED light
  - Update IP AP 100ms
    Runs TickUpdate(), Microchip TCP/IP Stack
  - MAX3100 timer
- Timer 1000ms
  - Timer uptime
    - Pnp update
- PNP update 20
  - Send address
  - Chip version , chip subversion, timer uptime, timer boot reason
- Broadcast the message
- goto MAINLOOP

**Message decoding**
Header of the message is decoded first, to check if it is a request to SIO2. The message is then total decoded, if the message is addressed to SIO2, a corresponding answer is sent back to the OriLink pc software.

If the received message is an update request, then the PNP array with all modules is sent.

Broadcast the message otherwise, broadcasting will be changed to addressable messages in the future.

Check timers with 100ms and 1 second’s interval. Update pnp array to pc with 10sec interval.
5 Result

The new OriLink module, SIO2, works already as a demo in Alentec & Orion AB facilities, where the interest in it has been great. Alentec & Orion AB has already developed future plans for the oil monitoring, where SIO2 module is the main piece of the design. SIO2 has opened a door for OriLink oil monitoring system to whole new world. OriLink oil monitoring system can now be connected to LAN using new OriLink pc interface, SIO2. One can now use the system in areas that previously have been almost impossible, because of the old interface. USB and serial RS-232 ports, on SIO2, can also be used for connecting the OriLink system to PC. Big positive effects are noticed in sales and marketing. Sales competition has become easier, because of connectivity possibilities. Demand for system and communication functionality has increased.
6 Conclusions

Today’s huge demand for gadgets and connectivity has meant that consumers opt out gadgets that cannot be accessed remotely. As the goal of this thesis is completed with the introduction of Ethernet and USB to oil monitoring system. SIO2 module strengthens OriLink position in the market and weakens the competition against similar systems. SIO2 module enables and increases the need for network-based features for OriLink system. One of the features that is in demand and is under development is: tank monitoring systems that can send status messages over the Internet.

Opening the door to the outside world brings not only positive things. The negative aspect of connectivity is that the system that has been totally cut off from the outside world suddenly becomes vulnerable to outside intrusion. It means that something that has not previously been taken in mind to, namely intrusion prevention and security level implementation.

The preliminary study did not corresponded completely with the work due to the preliminary study was made of the current development environment while development took place on the dated development environment, it resulted that the time for carrying out of the project was slightly prolonged than expected. The main obstacle in the work was not being able to debug the SIO2 embedded software according to the modern technology, that, because of the use of legacy programming environment used for OriLink.

What you should keep in mind is that OriLink systems are developed based on serial communication standard. Which means that the communication is disturb sensitive than packet-based. Where serial communication cannot confirm receipt of the packet. With that in mind, the internal communication is not adapted to today’s packet-based communications. OriLink system communication should undergo an update and modification of the existing communication protocol to meet and keep up with future updates and modifications.

6.1 Future work

The future work for the module would include increasing network security. Security and privacy are of great importance in now days society, especially in industries. Implementation of better safety layer directly in the Microchip TCP/IP stack would facilitate the movement of the software. Another aspect of that is being able to lock the communication module to a fixed server MAC address. This way, SIO2 module would know if another device is interfering.
References


Appendix A

SIO2 schematic, component connections for the SIO2 board, is illustrated in Figure 12.

Figure 25. SIO2 schematic

Figure 13 shows a breakdown of the pins for pic18f8722 microcontroller. Used for easier overview of pins properties.
Table 1 shows the connection between the peripheral equipment and the microcontroller, concerning pins characteristics.

<table>
<thead>
<tr>
<th>Controller</th>
<th>Operation</th>
<th>Microcontroller pin</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENC28J60 [1]</td>
<td>Data output</td>
<td>RC5/SDO1</td>
<td>SPI data out</td>
</tr>
<tr>
<td></td>
<td>Data input</td>
<td>RC4/SDI1/SDA1</td>
<td>SPI data in</td>
</tr>
<tr>
<td></td>
<td>Clock in pin for SPI interface</td>
<td>RC3/SCK1/SCL1</td>
<td>Synchronous serial clock input</td>
</tr>
<tr>
<td></td>
<td>Chip select</td>
<td>RD0/AD0/PSP0</td>
<td>Digital I/O</td>
</tr>
<tr>
<td></td>
<td>Interrupt</td>
<td>RB4/KBI0</td>
<td>Interrupt-on-change</td>
</tr>
<tr>
<td></td>
<td>Active-low device Reset input</td>
<td>RD1/AD1/PSP1</td>
<td>= Digital I/O</td>
</tr>
<tr>
<td>MAX3100 [2]</td>
<td>Data output</td>
<td>RC5/SDO1</td>
<td>SPI data out</td>
</tr>
<tr>
<td></td>
<td>Data input</td>
<td>RC4/SDI1/SDA1</td>
<td>SPI data in</td>
</tr>
<tr>
<td></td>
<td>Clock in pin for SPI interface</td>
<td>RC3/SCK1/SCL1</td>
<td>Synchronous serial clock input</td>
</tr>
<tr>
<td></td>
<td>Chip select</td>
<td>RD2/AD2/PSP2</td>
<td>Digital I/O</td>
</tr>
<tr>
<td></td>
<td>Interrupt</td>
<td>RB2/INT2</td>
<td>External interrupt 2</td>
</tr>
<tr>
<td></td>
<td>Active-low device Reset input</td>
<td>RD3/AD3/PSP3</td>
<td>= Digital I/O</td>
</tr>
<tr>
<td>----------------</td>
<td>-------------</td>
<td>-------------</td>
<td>----------</td>
</tr>
<tr>
<td>MAX485 (1) [4]</td>
<td>Interrupt</td>
<td><em>RB0/INT0/FLT0</em></td>
<td>External interrupt 0</td>
</tr>
</tbody>
</table>

Table 1. Pin description
Appendix B

In order to use OriLink predefined libraries, module specific hardware had to be defined according to OriLink programming standard: ASM defines the pin for using with assembler programming language and CPP with C programming language. See Appendix A, NetA.c for further definitions.

\[
\begin{align*}
\text{ASM\_NETRC} &= \_\text{PORTC,7} \\
\text{CPP\_NETRC} &= \text{RC7} \\
\text{ASM\_NETRCTRIS} &= \_\text{TRISC,7} \\
\text{CPP\_NETRCTRIS} &= \text{TRISC7} \\
\text{ASM\_NETTX} &= \_\text{PORTC,6} \\
\text{CPP\_NETTX} &= \text{RC6} \\
\text{ASM\_NETTXTRIS} &= \_\text{TRISC,6} \\
\text{CPP\_NETTXTRIS} &= \text{TRISC6} \\
\text{ASM\_NETMONITOR} &= \_\text{PORTB,0} \\
\text{CPP\_NETMONITOR} &= \text{RB0} \\
\text{ASM\_NETMONITORTRIS} &= \_\text{TRISB,0} \\
\text{CPP\_NETMONITORTRIS} &= \text{TRISB0} \\
\text{ASM\_NETCONTROLLERENABLE} &= \_\text{PORTC,2} \\
\text{CPP\_NETCONTROLLERENABLE} &= \text{RC2} \\
\text{ASM\_NETCONTROLLERENABLETRIS} &= \_\text{TRISC,2} \\
\text{CPP\_NETCONTROLLERENABLETRIS} &= \text{TRISC2} \\
\text{ASM\_SYSTEMLED} &= \_\text{PORTC,0} \\
\text{CPP\_SYSTEMLED} &= \text{RC0} \\
\text{ASM\_SYSTEMLEDTRIS} &= \_\text{TRISC,0} \\
\text{CPP\_SYSTEMLEDTRIS} &= \text{TRISCO}
\end{align*}
\]

Configure all remaining microcontroller pins as outputs.

Enable all global and peripheral microcontroller interrupts.  
GIE=1; PEIE=1;  
Set finally the module address to DFF1.
Appendix C

Below is a code snip for the SIO2 main function.

```c
void main()
{
    /* Clear watchdog timer */
    asm("clrwdt");

    /* Enable I/O ports. */
    EBDIS=1;

    /* Comparators Off */
    CMCON=0x07;

    /* Configure the A/D CONTROL REGISTER to Digital I/O */
    ADCON1=0x0F;

    /* Configure the onboard LED */
    CPP_SYSTEMLEDTRIS=0;

    /* Turn off all SPI stuff */
    // ENC28J60 -- RESETPIN
    TRISD1=0;
    LATD1=1;
    // ENC28J60 -- CSPIN
    TRISD0=0;
    LATD0=1;
    // MAX3100 -- SHDNPIN
    TRISD3=0;
    LATD3=0;
    // MAX3100 -- CSPIN
    TRISD2=0;
    LATD2=1;
    // MAX3420 -- RESETPIN
    TRISD5=0;
    LATD5=0;
    // MAX3420 -- SSPIN
    TRISD4=0;
    LATD4=1;

    /* Initialize timers */
    TimerInit();

    /* Initialize RS-485 communication */
    NetInit();

    /* Set unit address */
    NETADDRESSUINT=0x0900;

    /* Initialize IP */
    IpAppInit();
    TelnetInit();
    MAX3100init();
    MAX3420init();

    /* Initialize internal net, commands */
    Command0100Init();

    /* = Enable priority levels on interrupts */
    IPEN=1;
}

MAINLOOP:
```
/* Enable high and low priority interrupt */
GIEH=GIEL=1;

/* Clear watchdog timer */
asm("clrwdt");

IpAppMain();
TelnetMain();
MAX3100Main();
MAX3420Main();

/* Commands Main COOP OS */
Command0100Main();

/* if a packet is received */
if(NetCallFuncNetIn)
{
    NetCallFuncNetIn=0;

    /* Is the packet aimed for the SIO2? */
    if(TBUFAADDRESS(NetAddress[1],NetAddress[0]))
    {
        switch(NetBuffert[3])
        {
            HANDLER(0x20,pnpupdate.u.uc=NetBuffert[4])
            HANDLER(0xF0,OnMenuDispatch())
        }
    }
    else
    
    }

    } /* if 100ms has passed */
if(TimerCall100mS)
{
    TimerCall100mS=0;
    CPP_SYSTEMLED ^=1;
    IpApp100ms();
    MAX3100Timer100ms();
    MAX3420Timer100ms();
}

/* If one second has passed */
else
if(TimerCall1000mS)
{
    TimerCall1000mS=0;

    if(TimerUpTime==1)
    {
        /* After reboot -- update verbs */
        pnpupdate.u.uc=0xFF;
    }
    else
    
    }

}

/* Enable high and low priority interrupt */
GIEH=GIEL=1;

/* Clear watchdog timer */
asm("clrwdt");

IpAppMain();
TelnetMain();
MAX3100Main();
MAX3420Main();

/* Commands Main COOP OS */
Command0100Main();

/* if a packet is received */
if(NetCallFuncNetIn)
{
    NetCallFuncNetIn=0;

    /* Is the packet aimed for the SIO2? */
    if(TBUFAADDRESS(NetAddress[1],NetAddress[0]))
    {
        switch(NetBuffert[3])
        {
            HANDLER(0x20,pnpupdate.u.uc=NetBuffert[4])
            HANDLER(0xF0,OnMenuDispatch())
        }
    }
    else
    
    }

    } /* if 100ms has passed */
if(TimerCall100mS)
{
    TimerCall100mS=0;
    CPP_SYSTEMLED ^=1;
    IpApp100ms();
    MAX3100Timer100ms();
    MAX3420Timer100ms();
}

/* If one second has passed */
else
if(TimerCall1000mS)
{
    TimerCall1000mS=0;

    if(TimerUpTime==1)
    {
        /* After reboot -- update verbs */
        pnpupdate.u.uc=0xFF;
    }
    else
    
    }

}
/* if there is request for PNP update and we are able to allocate the net */
if(pnpupdate.u.uc && NetAlloc(38 + (NetAddress[0] & 0x03)) )
{
    pnpupdate.u.uc &= 0x03;
    NetSendAddress(0xDF02); //2
    pnpupdate.u.b.address=0; //Always
    if(pnpupdate.u.b.mainstatus)
    {
        UCHAR ts;
        ts = GetTerminatorStatus();

        pnpupdate.u.b.mainstatus = 0;

        NetSendChar(2 + 1 + 7 + CHIPVERSIONSZ + CHIPSUBVERSIONSZ + 5 + 12 + 5 + 12 + 12 + 12 + 12 + 12 + 12 + 12 + 12 + 1 + 20); //1
        NetPut(";t=%s;v=%s", CHIPVERSION, CHIPSUBVERSION); //1
        NetPut(";ts=%u", ts); //5
        NetPut(";ut=%lu", TimerUpTime); //12
        NetPut(";br=%u", TimerBootReason); //5
        NetSendChecksumStopController();
    }
    else
    {
        NetSendChar(2 + 1 + 7 + 1); //1
        NetPrintFaddr(); //7
        goto MAINLOOP;
    }
}

The code below is for the IP main.

```c
void IpAppMain()
{
    /* manages communication over Ethernet controller ENC28J60, according to the MAC address */
    StackTask();

    /* translate host names on local network to local IP addresses */
    NBNSTask();

    /* Broadcast IP address */
    BroadcastMain();
}
```

The code below is for the Telnet main connection.

```c
void TelnetMain()
{
    UCHAR n;
    UCHAR c;
    for(n=0;n<TCPTRANSPORTCOUNT;n++)
    {
        putchtarget=252;
        tcpputtarget=tcpt[n].socket;

        switch(tcpt[n].state)
        {
            case TCPTTRANSPORT_LISTEN:
                if((tcpt[n].socket=TCPListen(23))!=INVALID_SOCKET)
                    tcpt[n].state=TCPTTRANSPORT_LISTEN_WAIT;
                break;

            case TCPTTRANSPORT_LISTEN_WAIT:
                if(TCPIsConnected(tcpt[n].socket))
                    tcpt[n].state=TCPTTRANSPORT_CONNECTED_WELCOME;
                break;

            case TCPTTRANSPORT_CONNECTED_WELCOME:
                if(TCPIsConnected(tcpt[n].socket))
                {
                    if(TCPIsPutReady(tcpt[n].socket))
                    {
                        TCPFormat("Welcome to OriLink PC interface\n");
                        TelnetPrompt();
                        TCPIFlush(tcpt[n].socket);
                        tcpt[n].state=TCPTTRANSPORT_GET;
                        tcpt[n].charbuf[0]=0;
                        tcpt[n].charbufpoint=0;
                    }
                    /* Show on all other TCP sockets -- Async */
                    UCHAR m;
                    for(m=0;m<TCPTRANSPORTCOUNT;m++)
                    {
                        if(m!=n)
                        {
                            if(TCPIsConnected(tcpt[m].socket))
                            {

                            }
                        }
                    }

                    tcpputtarget=tcpt[m].socket;
                }
    }
    UCHAR saddr[16],smac[18];
```
SOCKET_INFO *si;

si=TCPGetRemoteInfo(tcpt[n].socket);

lpAppAddress2Str(saddr,&si->remote.IPAddr);

IpAppMAC2Str(smac,&si->remote.MACAddr);

printf("New connection made on socket[%02u] %s:%d -- %s\n",n+1,saddr,si->remotePort.Val,smac);

}

tcpputarget=tcpt[n].socket;

}

}

}

}

else
{
    tcpt[n].state=TCPTRANSPORT_DISCONNECT;
}

break;

case TCPTRANSPORT_GET:
    if(TCPIsConnected(tcpt[n].socket))
    {
        UINT sz=TCPIsGetReady(tcpt[n].socket);
        if((sz+tcpt[n].charbufpoint)>TCPTRANSPORTBUFFERSIZE)
            sz=TCPTRANSPORTBUFFERSIZE-
tcpt[n].charbufpoint;

tcpt[n].charbufpoint=TCPGetArray(tcpt[n].socket,&tcpt[n].charbuf[tcpt[n].charbufpoint],sz);
        if(tcpt[n].charbufpoint<TCPTRANSPORTBUFFERSIZE)
            tcpt[n].charbuf[tcpt[n].charbufpoint]=0;
    TCPDiscard(tcpt[n].socket);

    
    
    
    
    
    
    
    }

    if(strchr(tcpt[n].charbuf,\n'))
    {
        //scanf(tcpt[n].charbuf,"%s %s %s",sa,sc,se);
        getcmdindex(tcpt[n].charbuf,0,sa);
        getcmdindex(tcpt[n].charbuf,1,sc);
        getcmdindex(tcpt[n].charbuf,2,se);
        getcmdindex(tcpt[n].charbuf,3,sc);
        getcmdindex(tcpt[n].charbuf,4,se);
        getcmdindex(tcpt[n].charbuf,5,sc);
        TCPFormat("sa[%s] sc[%s] sc[%s] sd[%s] sd[%s] sf[sf];

        OnTelnetCommand(n);
        TCPFlush(tcpt[n].socket);
        tcpt[n].charbuf[0]=0;
        tcpt[n].charbufpoint=0;
Main routine for serial interface, RS-232

```c
void MAX3100Main()
{
    UCHAR irq;
    irq=MAX3100GetIrq();
    if(irq==0)
    {
        MAX3100READDATA rd;
        rd.uint=_SPIVX16(0);
        if(rd.b.R)
        {
            switch(rd.b.data)
            {
                case '\r':
                    break;
                case '\n':
                {
                    putchargetarget=251;
                    MAX3100OnCommand(st.charbuf);
                    
                    st.charbuf[0]=0;
                    st.charbufpoint=0;
                }
                break;
                case '3':
                {
                    _SPIVX16(0x8200+['
                    _SPIVX16(0x8200+['
                    
                    UCHAR n;
                    for(n=0;st.charbuf[n];n++)
                    
                    _SPIVX16(0x8200+st.charbuf[n]);
                }
                break;
                default:
                {
                    /* handles OriLink communication */
                    MAX3100OnPCCommand(st.charbuf);
                    
                    st.charbuf[st.charbufpoint++]=rd.b.data;
                    st.charbuf[st.charbufpoint]=0;
                }
            }
        }
    }
}
```
Main routine for USB interface.

```c
void MAX3420Main()
{
    if(Suspended)
        check_for_resume();

    if (MAX_Int_Pending())
        service_irqs();

    msec_timer++;

    if(msec_timer==TWENTY_MSEC)
    {
        msec_timer=0;
        /* check for data on USB port */
        if(incData())
        {
            /* Put the data in the buffert */
            NetBufferData = 1;
            /* turn the communication LED for USB */
            L0_ON
        }
    }
}
```