LULEÅ TEKNISKA UNIVERSITET

2001:304

MASTER'S THESIS

Software Modem for the Munin nano Satellite

Johan Axelsson-Åhl

Civilingenjörsprogrammet Datateknik

Institutionen för Systemteknik
Avdelningen för Programvaruteknik

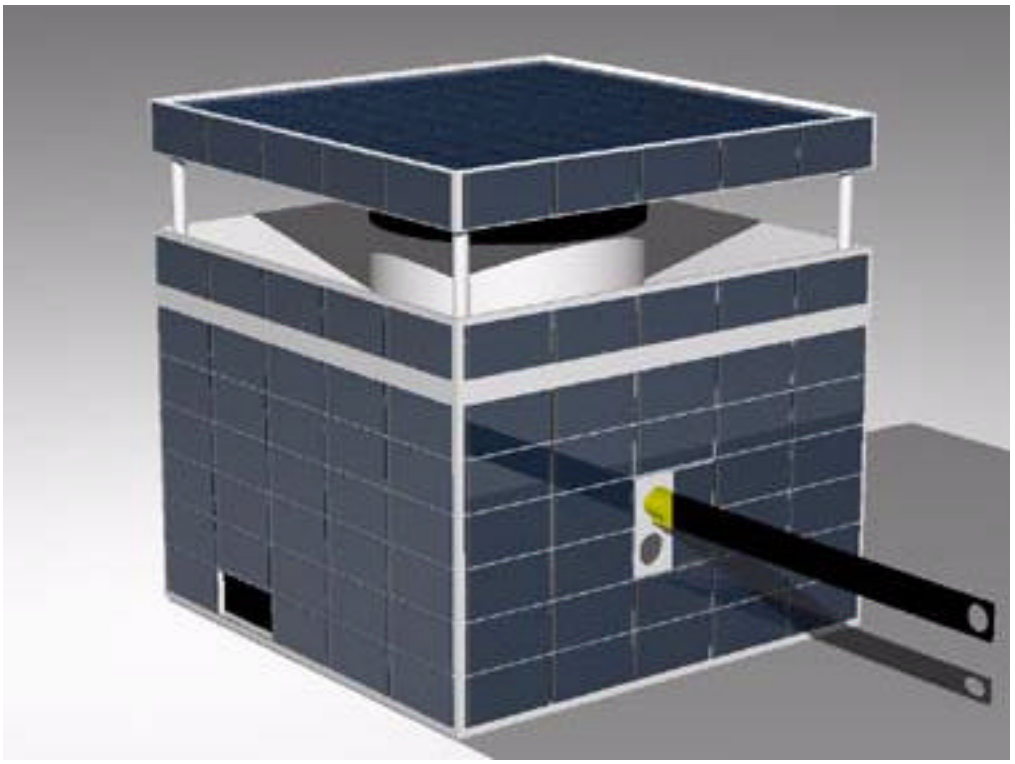
2001:304 • ISSN: 1402-1617 • ISRN: LTU-EX--01/304--SE

Software Modem for the Munin nano Satellite

A Master Thesis in Computer Science

By:

Johan Axelsson-Åhl



Luleå University of Technology
October 1998

1 Abstract

At the Swedish Institute for Space Physics a very small satellite, a nano satellite, is being constructed (1997-1998). The satellite, called Munin, is built with standard components, but also carries state of the art scientific instruments. A part of the project is to obtain a public outreach and to use it as a educational tool, for these reason students are taking part in each step of the construction and implementation process.

This thesis work describes the implementation of the ground stations communication equipment. The reception mechanism consists of a modem, and a PC controlling the modem. The modem's main equipment is a digital signal processor. All the logic for the modem is made in software. The modem controller is made as an device driver under the Linux operating system on a standard PC. The modem and the controller communicates over a standard RS232 interface.

The modem achieves acceptable bit error rates at realistic low field strength levels for the signal to be able to receive the necessary amount of data from the satellite.

2 Preface

This Master Thesis is the final assignment for the Master of Computer Science and Software Engineering educational program at Luleå University of Technology.

This thesis work was initiated and carried out at the Swedish Institute of Space Physics (IRF) in Kiruna. It was made within the Munin satellite project.

I would like to thank my supervisor at IRF, Walter Puccio. He has been a great help for me during the practical implementation of the assignment, and who without this thesis perhaps never would have been finished.

3 Table of contents

4 Introduction

This Master thesis was carried out at The Swedish Institute of Space Physics (IRF), at the Kiruna division. The main purpose of the thesis work was the construction of the ground station modem for the Munin satellite system in software on a Digital Signal Processor (DSP). The assignment also included the construction of the software controlling the modem from a remote PC computer.

4.1 Background

4.1.1 The Swedish Institute of Space Physics

IRF, founded in 1957 by the Swedish Royal Science Academy, is since 1973 a governmental research institute. The primary tasks for the organisation is to carry out fundamental research, serve as an educational resource, and to perform associated observatory activities in the space physics area.

IRF has about 120 employees, divided in four divisions: The Kiruna Division (IRF-K), where the main office is located, the Umeå Division (IRF-Um), the Uppsala Division (IRF-U), and the new Solar Terrestrial Physics, Lund Division (IRF-STL).

The main research carried out by IRF is experimental and theoretical fundamental research in space and atmospheric physics. This includes among others magnetosperical and ionospherical physics. The tools used for measurements are satellites, probe rockets, stratosperical balloons and ground based equipment like radar and cameras.

4.1.2 The Munin Project

Sweden has a successful tradition of building very small satellites, micro satellites, for space research. IRF has been very active in these projects, in the construction of the satellites and by providing scientific equipment which experiments are carried out on. Examples are the satellites Astrid and Freja.
In the autumn of 1996 a project aiming to construct an even smaller satellite, a nano satellite, was started at IRF. The satellite was going to be low cost as it would made out of standard 'of the shelf' components, and much of the construction would be made by students in thesis work and in student projects. The satellite, named Munin, was designed together with students from the Space Engineering Program of Umeå University.

Although the satellite is low cost, it has clear scientific goals. The scientific objective with Munin is to collect data on the auroral activity on both the northern and southern hemisphere, so that a global picture of the current state of auroral activity can be made. The data collected by the satellite will be published on-line through the Munin World Wide Web server for everyone to use. The data collected will mainly serve as an input to the prediction of space weather.

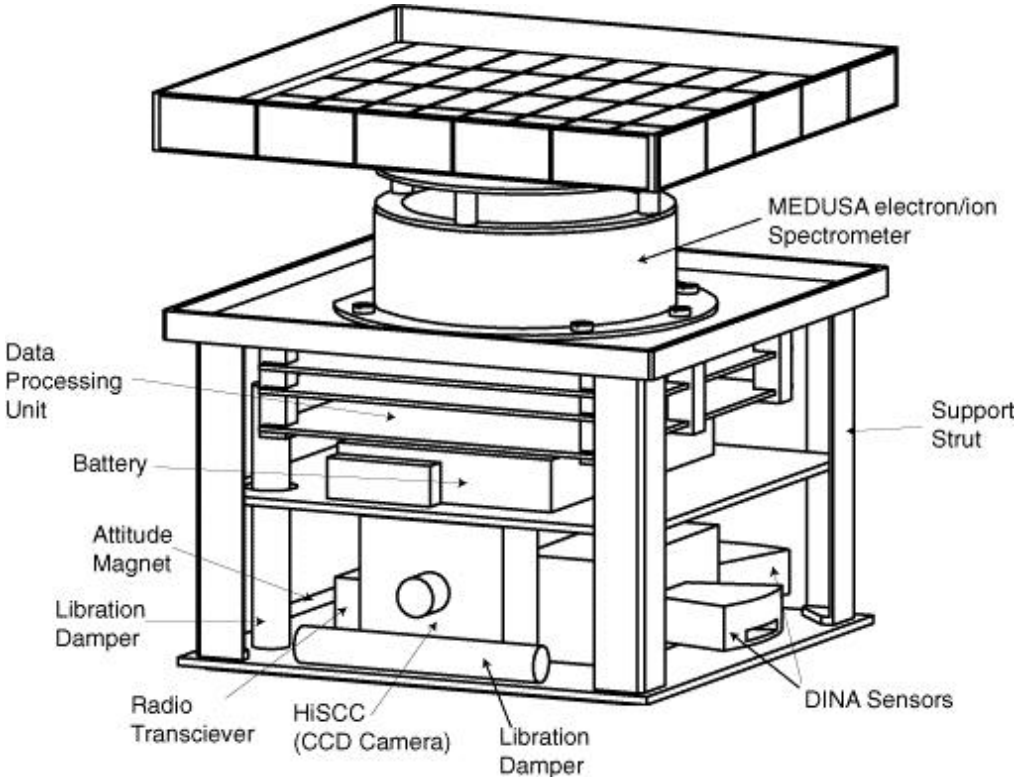


Figure 1: The Munin satellite

The very low mass satellite, weighing only 5kg, is equipped with several highly modern scientific instruments to achieve its goals. It is equipped with a combined electron and ion spectrometer called Medusa, constructed by Southwest Research Institute (SwRI). The Medusa is also flown on the Swedish Astrid-2 satellite. The satellite will measure high-energy particles with a solid-state detector (DINA sensors). Also a miniature CCD camera is carried by the satellite, taking pictures of the auroral activity. All information from these instruments will be published on-line on the Munin web site.

The satellite has a passive attitude control system using magnet and oscillation dampers. Silicon solar cells and a Li-Ion battery provide the needed power. The satellite uses the UHF-band for the up- and down link to the ground station. Digital Signal Processors perform instrument control, data compression and telemetry formatting, as well as serving as a software modem. The Mechanical Engineering department at Luleå University of Technology (LTU) has developed the launch rocket – satellite separation system. The satellite was built during 1997/98 and will be ready for launch now in the fourth quarter of 1998.

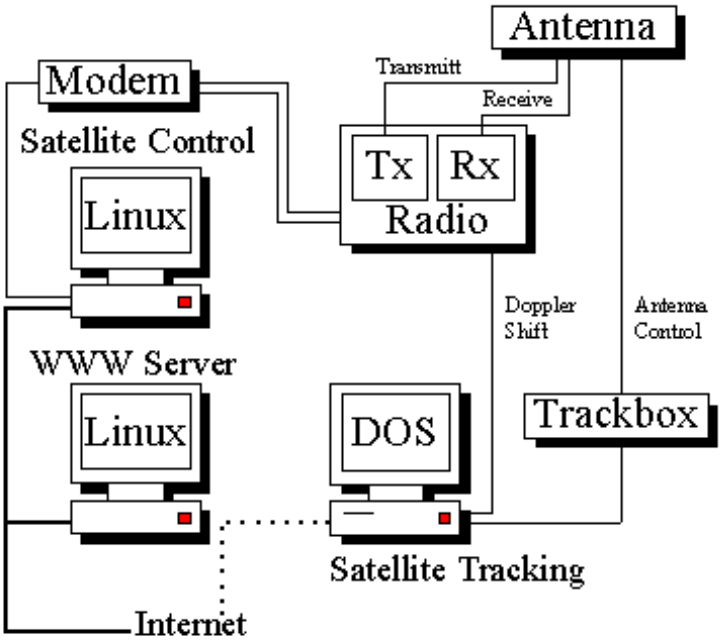


Figure 2: Layout of the ground station

The ground station for the satellite system is situated in a building called “Snickeboa“ near the IRF office complex outside Kiruna. The stations main tasks are to receive scientific data and to transmit control commands to the satellite, and also up-load new executable code for the onboard computer. It will be manned and controlled by students from Umeå University. The ground station will be equipped with a “trackbox“ working in conjunction with a dedicated satellite-tracking computer. The tracking computer will use Doppler shift positioning technique to control the antenna attitude towards the satellite so it can obtain optimal receiving conditions. The signals received are handled by the software modem running on a DSP system, controlled by a Linux driven computer. Finally there is a WWW server, hooked on to the Internet, doing the publication of the received scientific data.

4.2 Purpose of this thesis

The purpose of this thesis was to design and implement a ground station signal modulator/demodulator (modem) for the Munin satellite. The assignment was to implement the modem in software on a DSP signal processor. A Linux device driver, connected to the DSP via an Universal Asynchronous Receiver Transmitter (UART) serial interface should control the modem. The device driver should act as an ordinary Linux device file, formatting the output to an easy to read format and to simplify the control of the modem.
The modem should be able to receive all data from the satellite gathered during one orbit around earth. It should also have the ability to control the satellite by uploading control messages and new binary code.

4.3 Scope

This work limits to construct the ground station modem on a TMS32050 DSP, the Linux device driver controlling the modem, and a loader utility in Linux for easy downloading of executable code to the ground station modem DSP. A phase correcting filter optimal for the

modem constellation, placed in the DSP modem on the satellite for implementation reasons, should also be constructed. The assignment also includes the implementation of a protocol for the uploading of commands to the satellite.

The modem carried by the satellite is not considered in this thesis, although much of the work made on the ground station can be used for the satellite implementation.

5 Analysis

This thesis work was very practical in its nature. The demands on the system, and the hardware to solve the problem were already decided when I started the work.

5.1 Requirements on the System

The demands on the system is that under normal atmospheric conditions it shall be able to transfer all scientific data collected by the satellite during one orbit around earth with only a low number of faulty blocks,. If the atmospheric conditions are poor and the ground station modem are producing to many faulty blocks, the operator shall have the ability to adjust the transfer speed to get more correct packets, actively lowering the bit rate.

The operator shall be able to transfer control commands and new binary code up to the satellite.

5.2 System Model

5.2.1 Overview

The system (Figure 3) consists of the Munin satellite orbiting earth, and a ground station situated near the IRF office complex outside Kiruna. Munin will orbit earth at an altitude of 1000km and passes over the ground station 9-11 times a day.
In the satellite there is a DSP modem communicating with the ground station via a FM-transceiver. The signal is modulated with binary Pulse Amplitude Modulation (PAM) on the FM signal and transmitted to the ground station on the 400MHz band. The ground station has a transceiver receiving the signal, and a DSP modem implemented in a TMS320C50 device processing the signal. The DSP communicates over an UART serial interface with a device driver in a Linux driven standard PC, transferring the received signal. The device driver has some of the functionality found in a ‘normal’ Linux character device driver. A process can read and write from/to it using Linux standard libraries (**read()** and **write()**). The output from the device driver is formatted to ease reading, and input to it has to be formatted obeying a set of rules to ensure security and the integrity of the satellite.

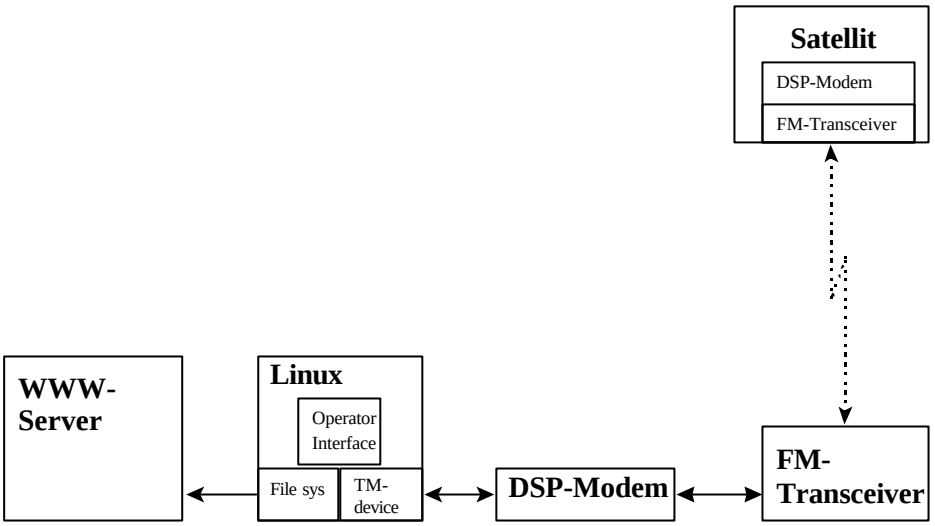


Figure 3: Overview of the communication system for the Munin nano satellite.

5.2.2 Information Flow

Just before Munin rises over the horizon relative to the IRF ground station, it starts to transmit scientific data gathered over earth in its previous lap, which have been stored local in the satellites RAM modules. The data have been compressed and placed in blocks of 512 bytes. The data is transmitted over the FM transceiver on Munin to the ground station. The analogue signal is transferred to the DSP modem, which synchronises to the signal and starts to decode it. The modem produces a bit stream and transfers it to the Linux device driver over a RS232 serial interface. The Linux device listens to the bit stream and searches for a pre-defined bit pattern identifying that a valid data packet is received. Valid packets are stored in Linux kernel memory space in a circular buffer containing up to 32 packets. Processes reading from the device get one packet of 512 bytes at a time. Figure 4 show the signal is adapted through the system.
There could also be service information about the satellite arriving, in the same manner as ordinary data. The device driver does not recognise the difference between data and service information, it is the responsibility of the reading process to make the distinction.

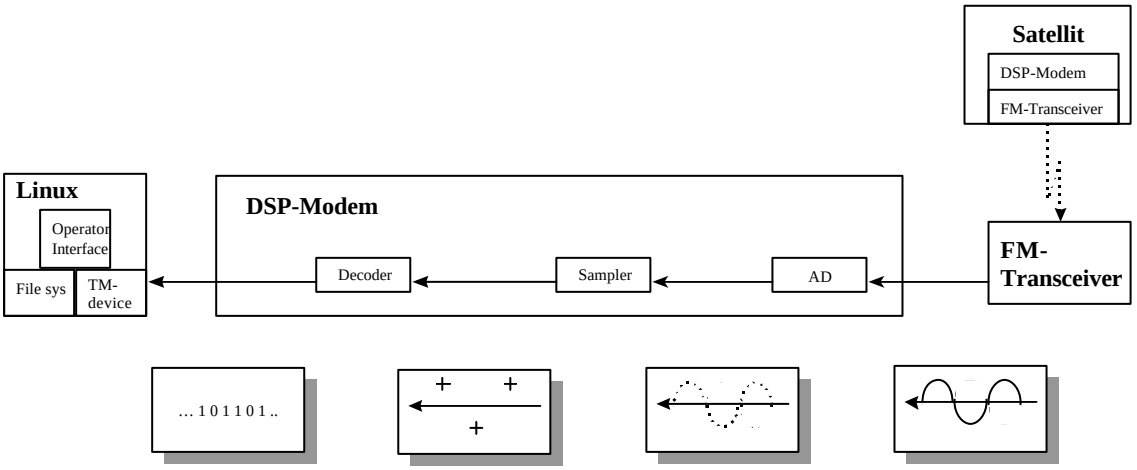


Figure 4: Signal manipulation by modem

Transmission up-link to the satellite either contains control commands or upload of new executable code. It is carried out in the reversed order to the reception previously described. Only one process at a time can have the device opened for writing.
When the ground station is transmitting data the reception is shut of because of the radio interference.

5.3 Data Packet Format

The data is placed in 512 bytes packets. Each block has a header containing a four-byte keyword leaving 508 bytes for the payload (figure 5). The communication system does not analyse the payload and it is free to put any kind of data here, it is up to the receiver to analyse and evaluate the contained information. No error detection or error correction is made on the packet; it is left to the user of the protocol.

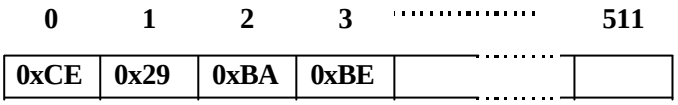


Figure 5: Data packet with keyword.

The sender and the receiver have to keep track of the number of valid bytes in the packet if the packet is not totally filled.

5.4 The Ground Station DSP Modem

5.4.1 Overview

In figure 6 the DSP modem board is shown when the board is used for receiving, and in figure 7 when it is used for transmitting. Some of the physical circuitry is used in both cases, while other is dedicated for its case.

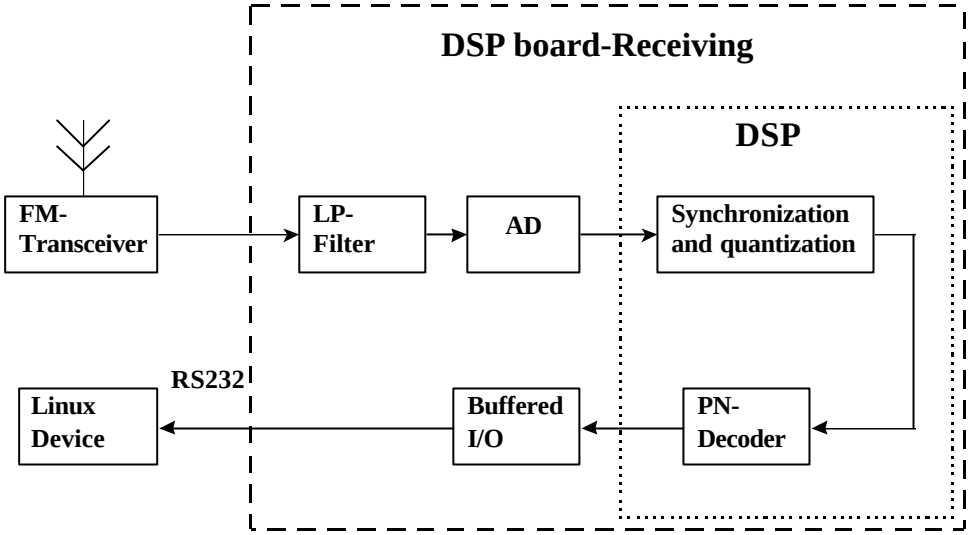


Figure 6: DSP modem board and connections when receiving data.

Figure 6 shows how the modem is configured to receive data. The signal from the transceiver is run through an analogue low-pass filter with cut of frequency adjust to remove high frequency noise. Then the 12 bit A/D converting the signal to the digital domain quantifies the signal. The signal, now represented as an integral value, is quantified and analysed. The DSP tries to synchronise with the signal and to find valid data bits in the stream. When synchronised the DSP enters a phase-locked loop to retain the sync. The now hopefully valid bits are delivered to the Pseudo-Noise (PN) decoder converting the coded stream into its original shape. The bits are shifted into a temporal buffer until a whole byte is received. The byte is then transferred to the I/O interface, which sends it to the Linux device driver over standard RS232 interface.

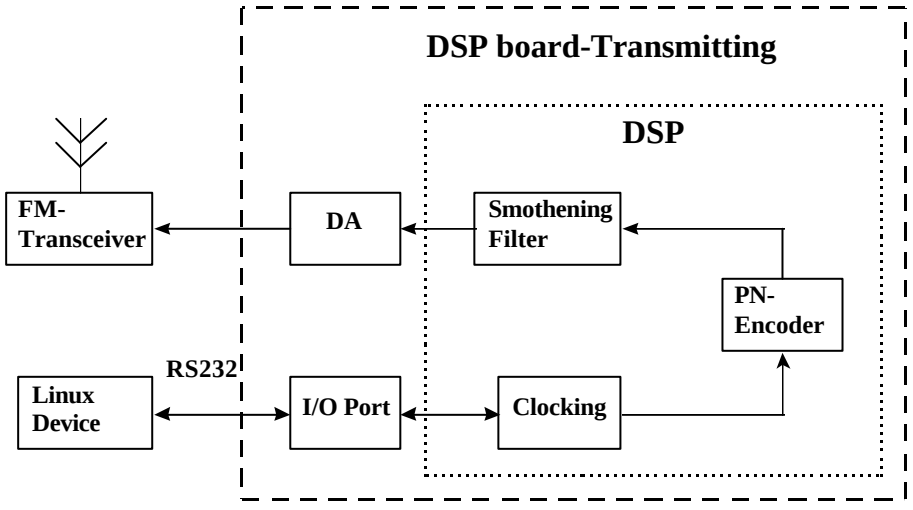


Figure 7: DSP modem board and connections when transmitting.

In figure 7 the transmission procedure is shown. The Linux device transfers bytes set for transmission to the satellite over the RS232 interface to the I/O port on the DSP board. The DSP reads from the I/O port when the clock ticks according to the transmission baud rate. It signals the Linux device when it is ready to receive next byte. The byte is shifted to access the bits in transmission order. The bits are run through a PN-encoder to get them into a format suitable for transmission. After quantification and over sampling of the bits they are

run through a smoothening filter that makes the square wave sinusoidal. The bit stream is then sent to the D/A, which makes an analogue signal of the integral values. The stream is transferred to the transceiver that transmit the coded bytes to the satellite.

The system on the DSP is a time critical system, and it is totally interrupt driven. It uses the system clock interrupt to achieve the right bit rate when receiving and transmitting. It sleeps until it gets an interrupt, it analyses the interrupt and starts to execute the appropriate code. However, when the DSP is decoding the incoming signal at highest speed, almost all processor time is used up for program execution.

All code for the DSP was written in Assembler.

5.4.2 Receiving Data

The receiving process is shown in figure 7 above.
The modem is constructed to receive data at four different bit rates. 4800, 9600, 14400, and 19200bps. Special channel adjusting digital filters are included in the signal path customised for these bit rates. The system switches between the filters depending on the current bit rate. These filters are placed in the satellites modem for implementation reasons, there are not enough clock cycles left for the ground stations modem to run the filter when the highest bit rate is used.

5.4.2.1 Locking DSP to incoming signal

The synchronisation functionality between the signal and the modem lies in the signal it self. Because the data bits are coded with binary PAM, zero crossings for the signal occurs whenever there is a shift in bit value from high to low value. This fact is used to manage the synchronisation of the modem to the incoming signal.

The fact that the bits are PN-scrambled guarantees that shifts in bit values occur relatively frequent.

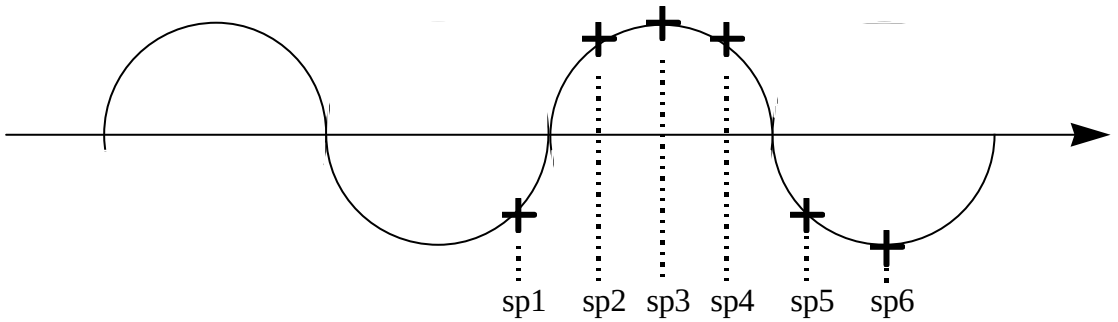


Figure 8: Modem sampling of incoming signal from radio.

The signal from the radio is constantly analysed by the reading process. It is sampled six times during one period according to the current bit rate setting (two data bits per period). The DSP timer interrupt govern when the samples are taken, one sample per interrupt period.

Sample1 and sample2 are taken as a synchronisation samples. If they have different signs a zero crossing has occurred between them, and the modem tries to synchronise to the signal. When synchronising to the signal the absolute value of the two samples are compared. If the absolute value of sample1 is larger than sample2 the modem takes the data sample to early. To adjust the modem the next sample, sample3, is delayed a few time units to get the data

sample point closer to the extreme of that bit. If the absolute value of sample1 is smaller than sample2, sample3 is taken earlier.

Then sample3 is taken as a data sample.

Again sample4 and sample5 are synchronising samples.

5.4.2.2 Decoding data

As shown in figure 8 above, every third sample is taken as a data sample. If the value of the sample is positive, it is decoded as a one received. If it is negative it is decoded as zero.

5.4.2.3 Sending to Linux device

As the bits are decoded they are assembled into bytes and transferred to the Linux device over the serial interface. The modem does not try to analyse if it really is synchronised to any signal, it is up to the Linux device driver to decide if valid data bits arrives. The Linux device makes this by scanning the bit stream for the 4 byte wide code word that is first in every valid data packet. If the device finds this code word it knows that the following 508 bytes are valid data bytes.

5.4.3 Transmitting Data

The transmission process is shown in figure 8 above.
The modem can only transmit data at 600bps. This relatively slow transmission rate is sufficient because only a few bits are necessary to transmit when sending commands to the satellite. The upload of new executable code is not foreseen to occur often. By using such a low bit rate the algorithm for receiving the signal in the satellite gets less complex.

5.4.3.1 Receiving data from Linux device

The Linux device driver sends a byte via the serial interface, and the modem receives an interrupt from the UART interface, which tells that new data is available in its registers. The modem then enters its “transmitting mode“ and stops decoding incoming bytes. It is a protocol between the Linux device and the modem managing transmission. In this protocol the Linux device gives the number of bytes that the message consists of.
The modem transmits the byte to the satellite, and signals to the Linux device to transfer the next byte for transmission by setting the MCR signal in the UART interface. When the indicated number of bytes has been transmitted the modem goes back into listening mode.

5.4.3.2 Preparing data for transmission

Before the data bits are transmitted they are PN-scrambled by the modem. To make it possible for the DA to construct the out signal the bit values are first given a numerical value, the maximum and minimum values that the D/A can handle. The bits are over sampled to make it possible to construct a smooth signal. Then they are run through a smoothening FIR filter to get rid of the sharp edges between ones and zeros.

5.4.3.3 Transmission

When a transmission is initiated the data bits cannot be transmitted at once. The satellite signal reception mechanism must get a chance to get synchronised with signal. So a dummy vector of bits are sent first, about 300 bits. The data is transmitted, and then again the dummy vector is transmitted to push out the valid bits from the digital filters on the modem.

5.4.4 Adjusting Modem Properties

It is possible to adjust some of the modem settings while code is actually executing on the DSP. Via the Linux device driver all program memory onboard the DSP can be accessed and

manipulated. This is a very useful feature because it enables the operator of the Linux device driver to have full control of the modem.

Actually, only a few memory positions are really interesting to manipulate. In the Linux environment these are pre defined in a header file to simplify for the device driver user.

The transmission protocol between the Linux device and the modem shows if the bits received from the Linux device are data for transmission to the satellite, or if it is control commands to the ground station modem.

5.4.4.1 Variables interesting to change

The modem variables interesting to change are quite few:

- The variable containing the timer interrupt interval to change the receiving bit rate.
- The variable containing the adjustment step that the timer variable is inflicted with when the modem tries to synch to the incoming signal, to tune the synching mechanism.
- The variable containing the "mean value" of the incoming signal that is subtracted from the incoming signal to get zero passes. This makes it possible to tune the modem performance in case of temporal constant disturbances.

5.4.5 Internal States

The modem can be in three different states.

- The "receiving" state, which is the "normal" state, when the modem constantly tries to synch to a signal and decodes bits from the signal.
- The "transmission" state, transmitting data.
- The "transmission cool down" state a while after it transmitted data to the Satellite. This state is necessary to speed up the transmission of messages in sequence. Because the satellite already is synchronised to the signal no synchronisation period is necessary before the data bits are transmitted.

5.4.6 Channel Adaptive Filter

The transmission channel, the disturbances inflicted on the signal from the transmitting process to the receiving process, consists of both linear and non-linear sources. The sources are for example phase shifts due to analogue filters, quantification errors in the A/D and D/A and so on. Also the media that the signal is transmitted over, the atmosphere deprecates the signal. Some of these disturbances, like those caused by electrical components, are however relatively constant and can be estimated. By inserting a channel adaptive filter into the transmission channel in the system construction phase at least the constant disturbances can be eliminated. A simple algorithm uses a FIR filter as the correcting factor and is described below.

When the correcting filter is constructed a signal is transmitted through the transmission channel. The deprecated signal is received by the DSP and analysed. The value measured at the data sampling point is examined. A guess is made. If the value is larger than zero the algorithm guesses that it should be a one that is received, and assigns the sample the extreme positive value of the signal. If the value is negative the guess that the value should be a zero is made, and the sample is assigned the extreme negative value. The sample is then compared with the output of the filter (figure 9). The difference between the filter output and the guess serves as input to the recalculation of the correcting filters coefficients.

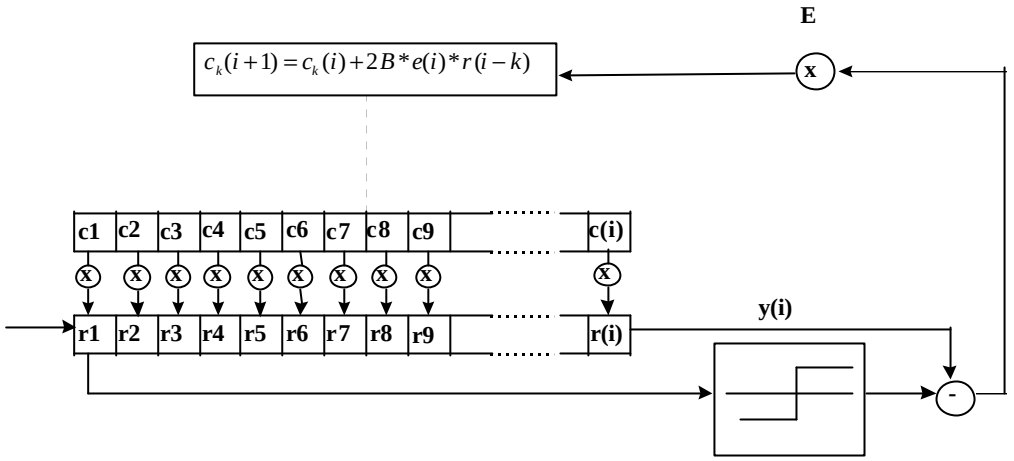


Figure 9: Channel adaptive filter

When this algorithm is run, the parameters slowly converge toward their accurate values.

The algorithm in figure 9, shown closer below, is trimmed by setting the constant *B*. With a large *B* a faster but less accurate result is achieved.

$$c_k(i+1) = c_k(i) + 2B * e(i) * r(i-k)$$

$$e(i) = r(i) - y(i)$$

$$y(i) = \sum_{k=0}^{N-1} c_k * r(i-k)$$

5.4.7 Pseudo Noise Scrambling

In the Munin communication system the data transferred between the satellite and the ground station is, as mentioned above, modulated by Pulse Amplitude Modulation. The receiver gets synchronised with the received signal by recognising shifts between high and low levels in the signal, e.g. ones and zeros. For the receiver to get and retain this synchronisation the signal has to do this shift ever so often. But binary data can contain monotone sequences of either ones or zeros. When this occur no shifts between high and low signal values take place, and the demodulator cannot lock to the signal. To get around this problem a pseudo noise generator, forcing shifts to occur, scrambles the data transmitted and the possibility for the receiving demodulator to get in synch with the received bit stream increases.

The technique used to achieve the PN scrambling of the data, is to shift the bits for transmission into a register as showed in figure 10. Then XOR'ing together three pre-defined bits from the shift register. The result of the XOR operation is then transmitted. Next data bit queued for transmission is then shifted into the register, and the XOR operation takes place again for the next transmission operation.

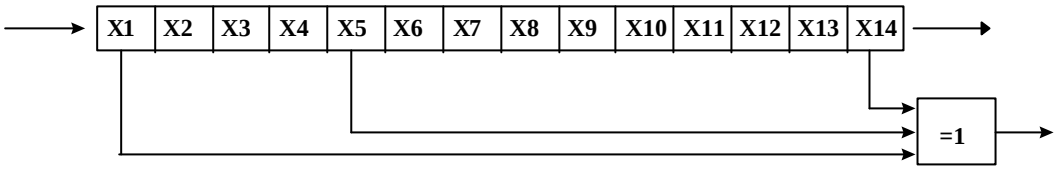


Figure 10: Pseudo Noise scrambling

At the receiving end the inverse operation is performed. The incoming bit stream is in the same manner put into a shift register and the same bits as on the transmitting side are XOR'ed together, producing the originally transmitted bit.

This is a very low cost operation, counting in processor instruction cycles, which produces a data stream where it is possible to synch with the amplitude. The down side of this method is that one transmitted bit that is received corrupted, can corrupt three data bits.

5.5 The Linux Device Driver

5.5.1 Overview

The device driver controls the modem board via the UART serial interface. It can load data to the modem set for transmission to the satellite, or control commands to the ground station modem. To use the device driver as a writer a certain protocol is followed. A specific code word must be first in every command or data vector if the driver is going to allow it to be sent over the modem, this to protect the satellite from unintended commands. Only one Linux process can write to the driver at a time.
The driver also acts as a reader from the modem, listening to the incoming bit stream and analysing it for valid data packets. The driver collects bits to bytes, and the bytes are buffered. Reading processes can receive single bytes up to a whole buffer size worth of data, 512 bytes, at a time. There is no restriction up on the number of listening processes that can use the driver simultaneously.

The device driver is written in the program language C, and it is an ordinary Linux character device. It is accessed via ordinary device system calls, like the system calls **read()** and **write()**.

The device driver is partly interrupt driven. It reserves the interrupt own by the UART interfacing the modem. When the UART fires an interrupt the driver questions the UART for the cause, and acts accordingly.

The computer that the driver is loaded in is connected to the modem board through an RS232 interface. An UART16550A serial card interfaces the modem board. The UART has a 16 level FIFO queue for both transmission and reception.

The device driver is written as a loadable Linux kernel module. This means that the module is only loaded into the kernel when it is needed, saving kernel memory space when not used.

5.5.2 Loading the Device Driver Into Linux Kernel

The device driver can be loaded at any time, either at boot time or dynamically when it is needed. This is a functionality that Linux provides. When the driver is loaded into kernel space some resources must be reserved. The interrupt for the UART and the UARTs I/O port address is allocated. If another process already reserves these resources the loading will fail.

5.5.3 Transmitting Data to the Modem

A writing process accesses the device through the Linux device control system, the file system (figure 11). For the writing process the device acts as an ordinary file.
Because the I/O interface towards the DSP is much slower, there is a circular buffer for temporary storage. Two pointers access this buffer. One managing where the next write will take place and the other the next byte that will be transmitted to the DSP. If there is risk for buffer overrun the writing process is blocked until there is some free space in the buffer again.

The device sends data to the modem via the serial interface. It sends one byte at a time. Between byte transmissions a signal is excepted over the UARTs MSR line. When this signal is received the UART fires its interrupt and indicates the received signal. The device driver then gets the next byte from the buffer and sends it to the UART.

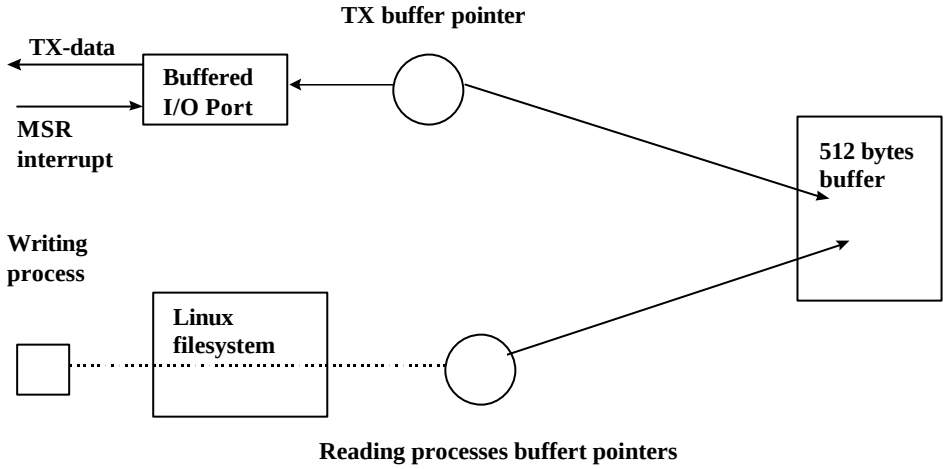


Figure 11: Transmitting process

The device driver controls that only one Linux process at a time can have the device opened for writing.

5.5.4 Receiving Data from DSP

The reading from the I/O port takes place continuously. To make the system as efficient as possible it buffers a couple of bytes before it fires its interrupt, because of the overhead an interrupt inflicts on the Linux system. The device driver (figure 12) has a set of 32*512 bytes buffers to place incoming data into. These buffers are accessed in a circular manner. The reading process, as in the writing case, accesses the device driver through Linux file interface. When a reading process starts to read it will not get data until a transmission buffer has been filled.

Several reading processes can be active simultaneously, each having a dedicated reading pointer into the drivers buffer space.

The device can be opened both for buffered and un-buffered read.

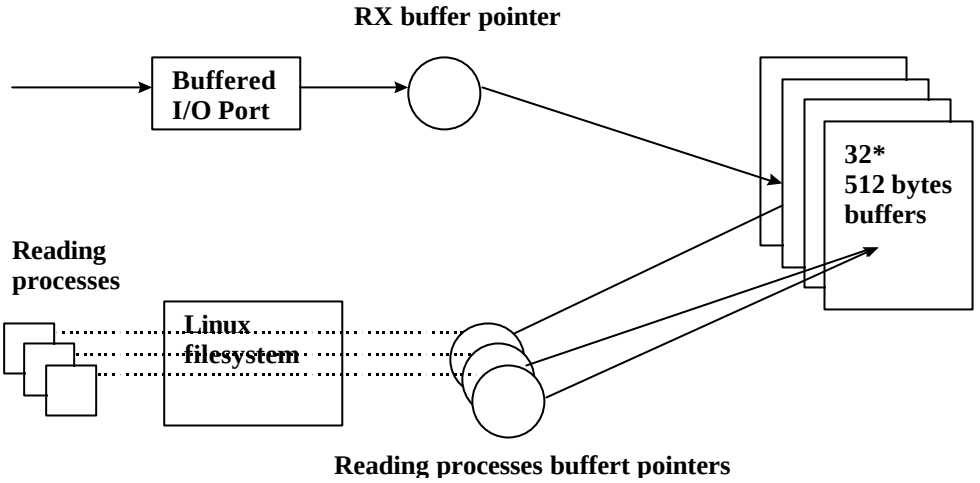


Figure 12: Reading process

5.6 Utility Programs

A couple of utility programs have also been constructed. Some as part of the assignment, other as help programs in the modem construction process.

5.6.1 Linux DSP Loader

There is no Linux loader available for the DSP used in the modem. From Texas Instruments FTP base source code for a DOS environment loader is available. After some reprogramming of the I/O parts it worked fine under Linux.

This program proved very helpful during the testing phase of the DSP code, and it shall be used by the Munin project as modem boot tool.

5.6.2 Adaptive Filter Calculator

It is very difficult to calculate the parameters for the channel-correcting filter used in the transmission channel theoretical. Quite a few parameters must be weighted and an accurate knowledge of these parameters is essential. Instead a program that calculated the parameters numerically was constructed.

The idea is that the modem sends a pre-defined signal, passes it trough all the components of the transmission system and a simulated transmission channel, and then receives the signal it self. The program compares the sent value with the received and adjusts parameters of a digital FIR filter acting in the transmission system accordingly.
The program is simply run for a minute or so until the parameters of the filter has stabilised, and the filter parameters have been calculated.

The filter constructed by this program lowered the Bit Error Rate (BER) at low signal field strengths.

5.6.3 Osc.

A program called “osc“ measures exactly what the DSP receives from the AD converter. It sends the values to a DOS program that displays them graphically.

This program proved useful in the debugging process of the modem code.

5.6.4 Usage Example Functions for the Linux Device Driver

To make it easier for the one that eventually will use the program and design a graphical command interface for the modem, a couple of example programs were constructed. These

programs show how to read from the device, appropriate system calls, and how to write to the device in a secure way.

6 Conclusion

6.1 Result

Tests made shows that the modem achieves to receive all data from the satellite from one journey around earth. It achieves acceptable BER with low enough field strengths from the satellite transmission.

6.1.1 Tests

The tests were made on a channel simulator. The signal was generated by the modem that will be placed in the satellite, passed through the channel simulator, and received by the ground station modem. The signal transferred was known, and all errors at reception were counted.

Some parts of the system were not used during these tests. The antenna and the pre-amp were not used (due to practical reasons). When these components are incorporated in the system approximately another -20 dB lower filed strength will be achievable according to my supervisor at IRF, Mr W. Puccio.

The demand on the tested system was to reach a BER around 1E-4 with field strength around -91 dB.

Baud Rate	Field Strength	BER	SNR
9600	-86	5.0E-7	46
	-88	1.0E-5	35
	-90	1.7E-4	26
	-91	4.9E-4	22
19200	-86	2.0E-6	42
	-87	5.0E-6	39
	-89	4.1E-5	30
	-90	2.0E-4	24
	-91	1.2E-2	9.6

The Signal to Noise Rate (SNR) above is calculated by:

$$P_b = Q\left(\sqrt{\frac{SNR}{2}}\right)$$

$$SNR = 2 * Q^{-1}\left(P_b\right)$$

Which is valid for binary PAM.

7 References

7.1 Literature

Stevens, Richard W. 1996. *Advanced Programming in the UNIX Environment*. Addison-Wesley Publishing Company.

Proakis, Salehi. 1994. *Communication Systems Engineering*. Prentice Hall International.

Kernighan, Brian W. & Ritchie, Dennis M. 1989. *The C Programming Language*. Prentice Hall International.

Dr. Steven A. Et al. 1997. *V. 34 Transmitter and Receiver Implementation on the TMS320C50 DSP*. Texas Instruments.

7.2 Web Sites

IRF: *The Munin home page.* URL: http://munin.irf.se/ (1998-10-16)

Southwest Research Institute URL: http://www.swri.edu/ (1998-10-16)

IRF: *IRF home page.* URL: Http://www.irf.se/ (1998-10-16)

The LinuxHQ Project: *LinuxHQ home page* URL: http://www.linuxhq.com/ (1998-10-16)

Texas Instruments: *Texas instruments home page* URL: http://www.ti.com (1998-10-16)

EXAR: *ST16C550* URL: http://www.exar.com/products/st16c550.html (1998-10-16)

Chalmers Technical Institute URL: http://www.cs.chalmers.se/Cs/Grundutb/Kurser/xjobb/Doc/AttDokExjobb-1.html (1998-10-16)

8 Acronyms

A/D	Analogue to Digital
BER	Bit Error Rate
bps	bits per second
D/A	Digital to Analogue
DINA	Detector of Ions and Neutral Atoms
DSP	Digital Signal Processor
FIFO	First In First Out
FIR	Finite Impulse Response
FM	Frequency Modulation
MEDUSA	Miniaturised Electrostatic DUal-tophat Spherical Analyser
Modem	Modulator/Demodulator
PAM	Pulse Amplitude Modulation
PLL	Phase Locked Loop
PN	Pseudo Noise
SNR	Signal to Noise Ratio
TM	TeleMetric
UART	Universal Asynchronous Receiver Transmitter