SECONDARY SCHOOL OF COMMUNICATION TECHNOLOGY

110 00 Prague 1, Panská 856/3URL : www.panska.cz

☎ 221 002 111, 221 002 666e-Mail : sekretariat@panska.cz

MATRICULATION EXAM

PRACTICAL EXAMINATION IN VOCATIONAL SUBJECTS

Hardware Omega Server

Field of study:	**26-45-M/004** **Digital telecommunications technology**	
Class:	**4.**	**AJan Kapic, Václav Košař**
School year:	**2006/2007**	name and surname of the author

1 ANNOTATION:

The aim of our work was to replace the Xapi server program, designed for remote programming of the Ateus Omega control panel via the Internet, with a module that will be built directly into the control panel, thus eliminating the problem of dependence on a switched-on computer. Basically, it is a simple passive network card, which also communicates with the control panel (Serial port).

2 ANNOTATION:

The aim of our work has ordered us to substitute the Xapi server application by a HW module, directly built in the phone exchange, which removes the problem of the dependence on switched PC. The Xapi server is designated for distant programming of the phone exchange Ateus Omega via Internet network. It goes in general about a simple passive network card while this card communicates further with the phone exchange (Serial port).

"We declare that we have prepared this thesis independently and have used the literature sources and information cited and listed in the list of literature and information sources used."

Prague, on ....

Signature

3 Home

The goal of our long-term graduation thesis was to design and build a "hardware Xapi server". The Xapi server is a software utility that allows the control panel to communicate
with other computers on the Internet. The disadvantage of this method lies in the nature of the whole solution to the problem- the Xapi server is a program and therefore depends on the running computer. We have tried to eliminate this disadvantage by replacing the Xapi server with our module, which will be built directly into the control panel. The basic questions of the whole work were:

1. How to connect DsPIC with Ethernet and how to work with its protocols?
2. How to connect the DsPIC to the control panel via serial port (connector wiring, voltage levels) ?
3. What changes in data occur during the Ethernet to Serial port transition?

The first question was solved by using the ENC28J60 integrated circuit, which works as an Ethernet controller operating on layers 1 and 2 of the OSI model, studying the relevant issues and building a package of programs for DsPIC capable of processing data of higher protocols. Which was programmatically and time consuming, but possible with complete documentation for the future.

The second question was solved by measuring the control panel connector for communication modules and redrawing the wiring of the existing module for RS232 communication.

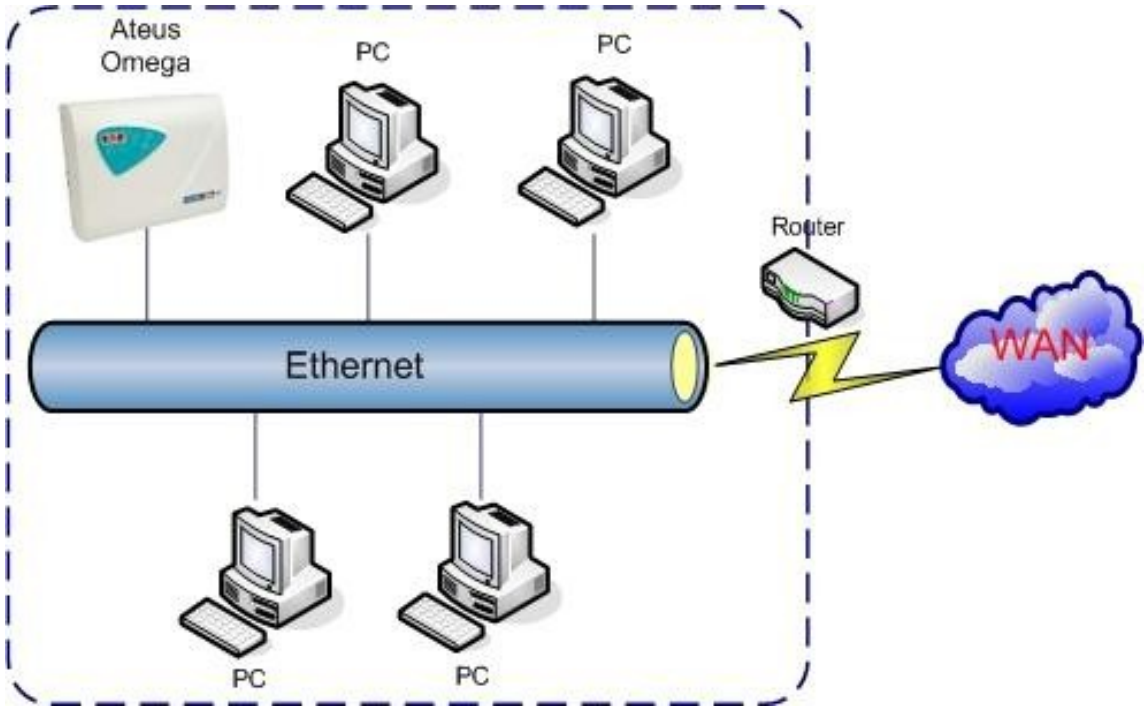
The third problem was significantly more complicated than the others. We attempted to eavesdrop on the Xapi server's communication with both interfaces, but we were unable to detect any major continuities in the output data. It was only by comparing the data in the TCP packets when programming with the Xapi server and the data sent by the computer when "normally" programming the control panel directly using the Omega program that we began to discover connections. Unfortunately this was not enough, so we were forced to ask for the original documentation for the serial communication of the control panel from the manufacturer - 2N. With the catalogue sheets we received, we could finally work out the first ideas of the whole solution. However, the CRC32 counting was still an issue.

This was followed by a progression in hardware and software design. Thanks to the training board we made the first DsPIC - ENC connection. The first programs for serial

communication were created. The problem with CRC32 was solved just before the creation of the final PCB (MEDEA MODULE) thanks to the great commitment of Professor Kubalik (more about CRC32 in the relevant chapter). Unfortunately, although the program for communication via TCP/IP was

almost complete, we were forced to choose a simpler variant due to time constraints, and therefore the control panel is programmable only via UDP.

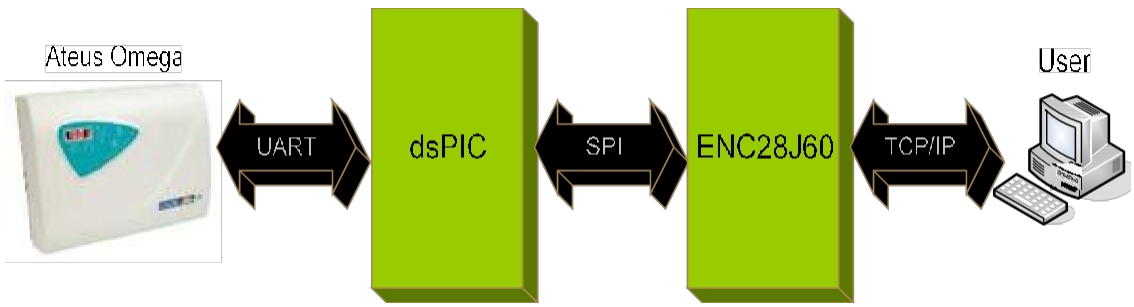
Nevertheless, the Médea module fulfils its function - our work was therefore successful, and in the full version of the assignment.



Using the control panel in an Ethernet network

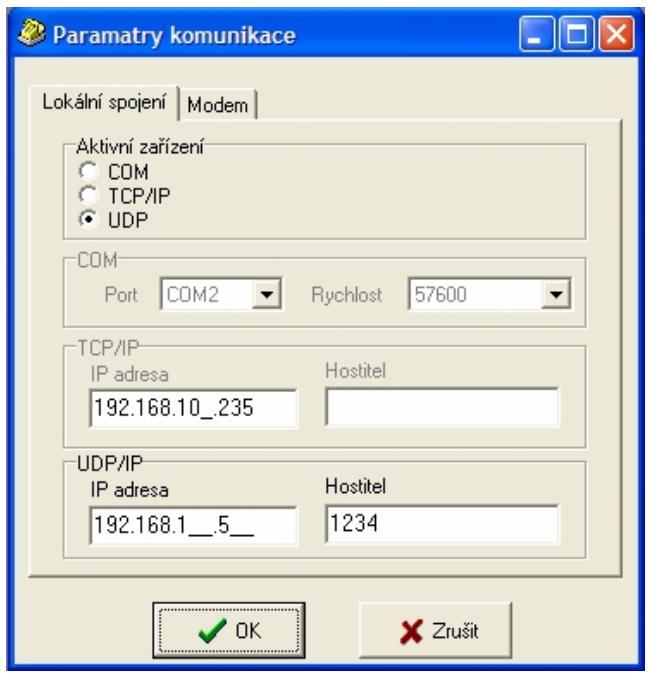
4 Monitoring logs

The figure below shows the protocols through which the individual parts of the module communicate with their environment.



Brief outline of communication protocols

The Omega program allows three ways of communication with the control panel: over the serial line (COM), over UDP and TCP/IP. We are interested in the last two methods implemented on the TCP/IP network.



Preview of the configuration window in the Omega program

4.1 Serial data transfer

A serial transmission is one in which the signal elements of the same data stream are transmitted in series. On the other hand, in parallel transmission, a certain number of signal elements (e.g. bits) are transmitted simultaneously.

4.1.1 Synchronous and asynchronous serial transmission

When transmitting information (serial and parallel), we must ensure that the receiver correctly evaluates the validity of the individual tags generated by the transmitter. Therefore, the transmitter and receiver must be s y n c h r o n i z e d i n time in some way. It is according to the type of synchronization that we distinguish serial transmission into synchronous and asynchronous.

Synchronous transmission is done by means of an isochronous signal, i.e. one where the spacing of two arbitrary characteristic moments (e.g. the start and end of individual markers) is an integral multiple of some (a priori given) unit interval.
The communication channel is therefore clocked by a common clock signal (carried separately or contained in the data signal), which defines the validity intervals of the individual tags. Synchronous transmission is most commonly used in bit-oriented protocols where information is grouped into frames. In data communications, it is mainly used for the transmission of large volumes of data.

Asynchronous (more correctly arrhythmic) transmission - the transmitter and receiver do not have a common clock signal that would define the intervals of validity of the markers. Instead, both sides have their own clocks, accurate enough to be considered isochronous for several mark intervals after phase synchronization. Since the clock needs to be synchronized periodically, this method is most often used for the transmission of short bit sequences, characters (5,6,7 or 8 bits). The synchronization takes place before each character, i, where the first meaning bit is always preceded by a so-called startbit, which is represented by the opposite signal value to the rest level on line i and lasts the same amount of time as the following bit intervals. Arrhythmic transmission is inherently suitable, for example, for communication with a computer via a terminal.

The control panel communicates using asynchronous serial transmission!!!

4.1.2 Serial communication with ATEUS-Omega control panel

Serial communication is used for diagnostics of the control panel and its parts, parameter programming, transfer of accounting data, and partly for control of the control panel operation. Connection to the control panel is possible via a standard serial line at speeds from 9k6 to 57k6 with autodetection by the control panel. The communication is carried out by semi-duplex packet sending.

Hardware connection

The physical connection to the control panel is realized by means of a galvanically separated serial port, using a 4-wire cable with Rx,Tx,GND signals. The data flow is not hardware controlled, so the PC application must be able to receive responses from the control panel without interruption at the selected communication speed.

Transmission parameters

The setting of the communication circuit is always 8N1 (without parity). The transmission speed is automatically set by the control panel in the range of 9600-57600bps when the PREFIX character is received. Higher speeds can be used on computers with 16550 (UART with FIFO).

Serial communication protocol

A new serial communication protocol is implemented in ATEUS Omega control panels. This protocol can be used to communicate over a serial link even with the help of a modem. It is a secure semi-duplex UDP packet transmission. The control panel is at all times a passive participant in the communication, i.e. it never starts transmitting spontaneously. The communication is initiated by the application, which sends a command packet and the control panel responds after its correct reception with a reply packet, in which the value of the NUM identifier is preserved. The control panel does not wait for a positive acknowledgement of the packet reception by the application, if the application does not receive a reply or the reply is invalidated by the transmission it can send a new request. The response time of the control panel depends on the traffic load and ranges from 5-300ms. During the response time the PBX ignores any further reception.

During the subsequent response transmission, any reception is considered STOP and cancels the transmission. To speed up the treatment of communication errors

The channel has full duplex 1byte non-batch responses (NAK), which the PBX sends after receiving each byte that does not fit into the protocol.

A. UDP packet structure:

PREFIX	START	LEN_L	LEN_H	NUM	DATA	CRC32
1 Byte	1 Byte	1 Byte	1 Byte	1 Byte	max. 509 Byte	4 Byte

PREFIX	znak EEh
START	znak 23h
LEN	délka NUM + DATA
NUM	identifikátor paketů (párování povel/odpověď)
DATA	posílaná data (viz formát povelů ústředny)
CRC-32	zabezpečení podle tabulky CRC32 (big endian).

Note..:

1) If the character EEh (PREFIX) occurs anywhere after the START character, it is replaced by **two** EEh characters. On the receiving side, one of the characters is omitted and is of course not counted in the CRC.

2) The CRC is provided by NUM and DATA and is sent as a big endian (LSB first).

3) The length of the DATA field depends on the size of the allocated memory in the control panel and may increase in future versions. The 509B value is chosen for compatibility with older versions and allows sending all previously used structures as well as new data for software upgrades.

B. Asynchronous responses:

Asynchronous 1-byte responses are newly introduced to speed up recovery from communication errors. Protocol violations can occur in the following cases:

Stav rozhranní	Chybný příjem	Reakce	Pozn.
KLID	znak<>PREFIX	NAK	vždy na 57600 bps
PŘÍJEM	PREFIX+START	-	restart příjmu bez odpovědi
PŘÍJEM	PREFIX bez opakování	NAK+KLID	na rychlosti PC
PŘÍJEM	přetečení délky	NAK+KLID	
PŘÍJEM	chybný CRC	NAK+KLID	CRC je porovnáván po bytech
PŘÍJEM	-	-	nedostatečný počet přijatých znaků
REAKCE	cokoli	-	příjem je ignorován
VYSÍLÁNÍ	cokoli	KLID	příjem ruší vysílání (data ztracena)

Status of the control panel communication interface:

KLID expectation of receiving the 1st character (PREFIX)

RECEIVING continued reception after receiving at least 1 character **REACTION** time for processing an error-free received **SENDING** state when the response packet is sent

When receiving an insufficient number of characters (e.g. when the cable is broken in the middle of reception), the control panel remains in the RECEIVE state for an indefinitely long time, which is not detrimental, b e c a u s e the application sends a repeated command after its own time monitoring and the control panel is able to restart reception at any time after receiving the PREFIX+START sequence. If the application changes the communication rate in the meantime, the 1st packet is received as invalid and the control panel responds by sending a NAK at the previous rate. However, the repeated packet is already received correctly.

The response **NAK = 0E0h** is intended to speed up the recovery from a communication error and to detect channel throughput. The NAK character is chosen so that it is detectable by the application at all permissible receive rates even if the 1st character of the packet (PREFIX) is corrupted, in which case the PBX sends the NAK at 57600 bps. Therefore, the application only tests the reception during its transmission and can restart the transmission immediately if any byte is received.

If an error occurs while transmitting from the application, the control panel will immediately respond to the NAK and the application should be able to restart the transmission immediately. If the application detects a receive error, it can stop transmitting by sending a STOP (any character) and immediately retransmit the request after flushing its receive buffer (watch out for Windows and 16550).
If an error occurs due to a loss of characters on the line or a failure in one of the communication directions, the application must respond by time monitoring and retrying the request. If the retry request fails several times, the cause must be displayed and, if necessary, channel throughput detection must be initiated,
i.e. cyclically send a correct packet (e.g. INFO) until it receives a correct packet response. The cause is either no response from the control panel (check cable status and COM selection) or receiving NAK characters (erroneous channel, reduce communication speed).

C. Change of communication speed:

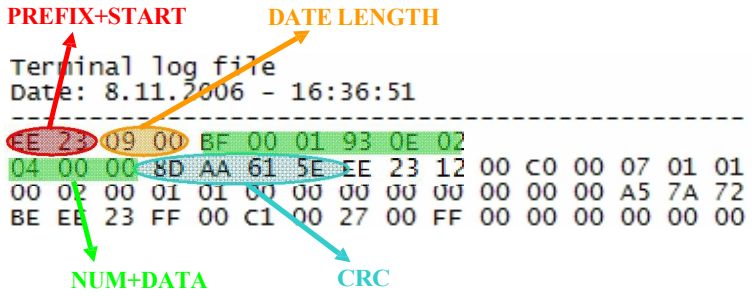
The control panel is able to switch to the speed selected by the application on the PC during reception

1st character of the packet (PREFIX). If this character is garbled, the PBX responds by sending a NAK at 57600 bps. Due to the NAK option, even at a PC receive rate of 9600 bps, this character is received as **0FFh** and the application can retransmit immediately. The

control panel detects the rate during every 1st character of the packet and if it is PREFIX continues to receive at the selected rate and also responds at that rate. After the reply is broadcast it always switches to 57600 bps and waits for the next packet.

4.1.3 Disassembling the packet

Example of captured packets during programming.



Terminal log

4.2 SPI interface

SPI is a serial peripheral interface. It is used for communication between controlling microprocessors and other integrated circuits (EEPROM, A/D converters, displays...).

4.2.1 Distribution of devices on the serial SPI bus

Master

- controls communication by means of a clock signal
- determines which device on the bus it will communicate with using SS-Slave Select(sometimes CS-Chip Select)

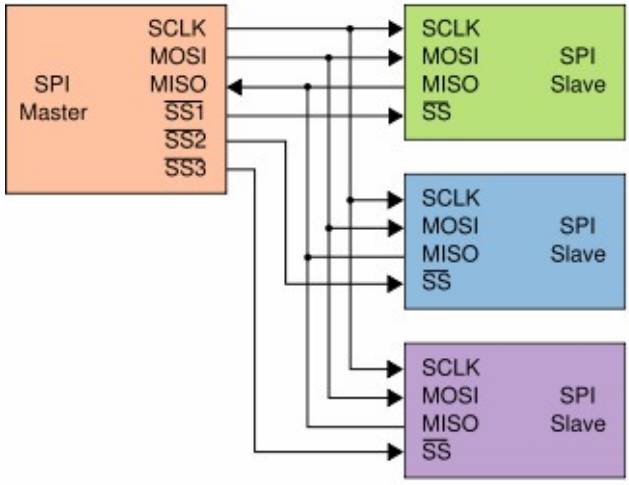
Slave

- transmits according to the clock signal if activated by SS/CS

Communication:

For communication, the Master sets log.0 on the **SS** device it wants to communicate with. Then it starts generating a clock signal to **SCLK** and at that moment both devices send their data, where **MISO** is always Master Output, Slave Input and **MOSI** is Master Input, Slave Output. Once the data has been sent the communication can continue: The *Master continues to supply a clock signal, the SS value does not change* or can be terminated: the *Master stops sending the clock signal and sets SS to log.1.*

The length of the transmitted data is either 8bit (Byte) or 16bit (Word).

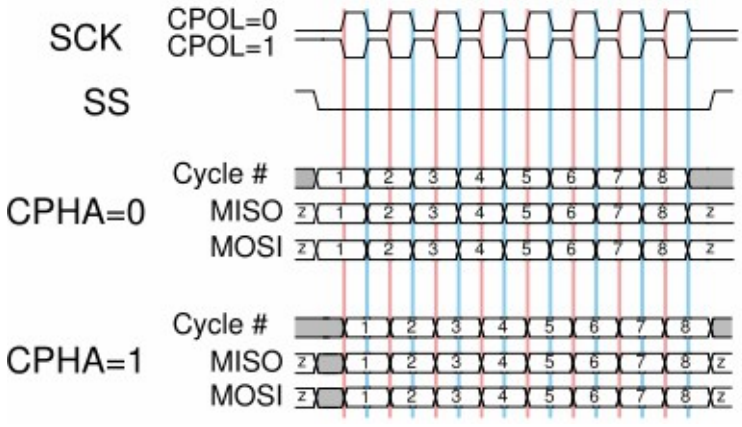


Polarity and phase of the clock signal

The relationship between the clock signal and the data is determined by two configuration bits:CPOL and CPHA.

CPOL= 0; resting level of clock signal log.0

CPOL = 1; the resting level of the hourly sig. is log.1 **CPHA = 0;** the value is read on the rising edge **CPHA = 1; the** value is read on the falling edge



4.3 TCP/IP

(Communication Medea module => User)

4.3.1 TCP/IP protocols

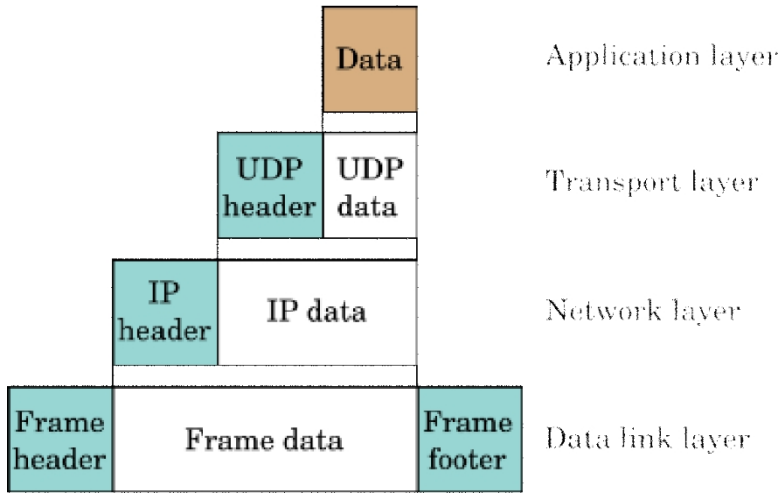
(Brief description focused on the part used in our work)

Comparison to the OSI reference model

(Assignment of logs to layers. Highlighted logs are included in our work)

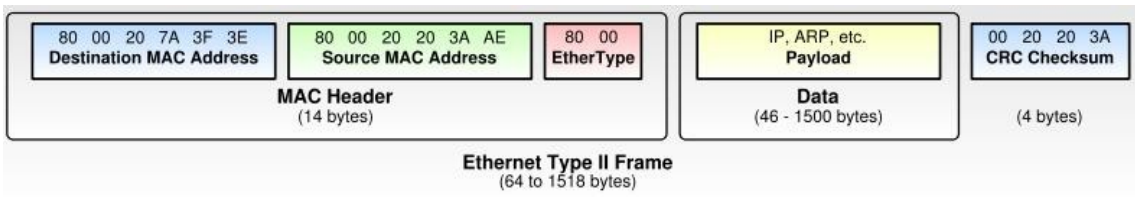
7	**Application**	**HTTP**, SMTP, SNMP, FTP, Telnet, ECHO, SIP, SSH, NFS, RTSP, XMPP, Whois, ENRP
6	**Presentation**	XDR, ASN.1, SMB, AFP, NCP
5	**Session**	ASAP, TLS, SSL, ISO 8327 / CCITT X.225, RPC, NetBIOS, ASP
4	**Transport**	**TCP, UDP,** RTP, SCTP, SPX, ATP, IL
3	**Network**	**IP, ICMP**, **ARP**,IGMP, IPX, OSPF, RIP, IGRP, EIGRP, RARP, X.25
2	**Data Link**	**Ethernet**, Token ring, HDLC, Frame relay, ISDN, ATM, 802.11 WiFi, FDDI, PPP
1	**Physical**	**10BASE-T**, 100BASE-T, 1000BASE-T, SONET/SDH, G.709, T-carrier/E-carrier, various 802.11 physical layers

Example of frame construction



Ethernet Frame (Frame)

Addressing: MAC address; protection code (Frame check seq.) CRC32 (four byte cyclic redundant code); max. length 1518 bytes



The frame data content can be an ARP packet or an IP header.

ARP packet

It is usually used to find out the MAC address of the target computer, assuming we know its IP address. The Operation field usually contains bits marking the packet either as a MAC query (in which case the packet is broadcast, i.e. to everyone) or as a reply with the requested MAC filled in (in which case the packet is unicast, i.e. to the querier only)

+	Bits 0 - 7	8 - 15	16 - 31
0	Hardware stream type (HTYPE)		Protocol type (PTYPE)
32	Hardware length addresses (HLEN)	Protocol length addresses (PLEN)	Operations (OPER)
64	Source Hardware address (usually MAC)		
?	Source protocol address (usually IP)		
?	Target Hardware address (usually MAC)		
?	Source protocol address (usually IP)		

IP header

Addressing: IP address, header is variable length due to the Options field;
protective element: header checksum see: Chapter ENC28J60-Calculating the checksum

+	Bits 0-3	4-7	8-15	16-18	19-31
0	Verse	Length Header	Type of service	Full packet length	
32	Identification			Flags	Fragment Offset
64	Time to Live		Protocol	Header checksum	
96	Source IP address				
128	Destination IP address				
160	Options				
160 or 192+	Data				

The data content after the IP header can be a number of different protocols, I will only mention ICMP, TCP, UDP.

ICMP protocol

ICMP is a protocol for error and control messages (Echo, Destination Unreachable, Redirect, etc.) The "Echo" message type is used by the ping program.

0		31
Message type	Code	Message checksum
Identifier		**Sequence Number**
Date :::		

UDP protocol

UDP provides a non-connective and unreliable message transfer service. Addressing: ports; protection: checksum including Pseudo header.

+	**Bits 0 - 15**	**16 - 31**
0	Source Port	Destination Port
32	Length	Checksum
64	Data	

TCP protocol

TCP provides a connectionless and unreliable message transfer service. Addressing: ports; protection: checksum including Pseudo header.
Connection is established by synchronizing sequence numbers (Three-way handshake)

The sequence number always contains the offset of the first data byte of this segment with respect to the entire sequence of data to be transmitted

The acknowledgement number contains, if the ACK flag is set, the sequence number of the next byte that the receiver is ready to receive.

+	**Bits 0-3**	**4-7**	**8-15**	**16-31**
0	Source Port			Destination Port
32	Sequence number			
64	Acknowledgment number			
96	Data Offset	Reserved	Flags	Window
128	Checksum			Urgent Pointer
160	Options (optional)			
160/192+	Data			

4.3.2 Data in packets

During monitoring, it was found that UDP packets have different data content than TCP packets. TCP packets did not contain the PREFIX, START and CRC characters (see 4.1.2), so the data must be padded before sending. Also missing is the replacement of the EE character with two EE characters. The data must be tested, and when the EE character appears anywhere after the START character, it is duplicated. Data in UDP are complete and have all the requirements.

Note: The XAPI server cannot communicate over UDP.

4.3.3 CRC codes

CRC or Cyclic Redundancy Check is a special hashing function used to detect errors during data transfer or storage. The CRC is calculated before an operation that is expected to have errors. It is sent or stored along with the data. After the data is received, it is independently recomputed from the data. If a different CRC is returned, the transfer is declared an error. In certain cases, the error can be corrected using the CRC. The whole process is tied to a so-called generating polynomial, which is the key to the whole process.

The generating polynomial (GP) is generally an n-bit number, where n is a power of 2 for practical purposes. Thus, in principle, we calculate CRC-16bit, CRC-32bit; CRC-64bit or CRC-128bit, etc. .
The more "multi-bit" the CRC is, the greater the chance of detecting an error. We usually do not invent GPs ourselves, but use already established polynomials with respect to the type of transmission and depth of security.

You can see an example of GP here:

$$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$$

This case is a sample GP for CRC-32bit. It is used according to IEEE 802 for Ethernet and not coincidentally also for our PBX.

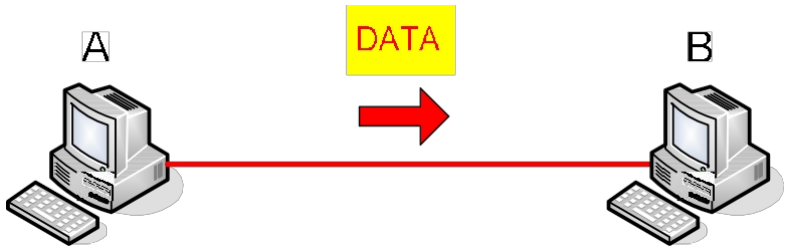
The main operation you will need when calculating the CRC is the XOR:

As a reminder :

```
0 xor 1 = 1
1 xor 0 = 1
0 xor 0 = 0
1 xor 1 = 0
```

I will explain the calculation with a practical example:

We send data from point A to point B :



I choose dates: $x^7 + x^6 + x^3 + x^0$ => **11001001**

I vote GP: $x^3 + x^2 + x^0$ => **1101**

POINT A

So we have the data 11001001, which we secure with polynomial 1101. The first step is "adding" zeros to the end of the data. The number of zeros depends on the highest GP term, in our case 3 zeros (x^3). The next step is division (note that this is not ordinary division, but the XOR mentioned above!). It is also important that after the "division" we are not interested in the result, but in the remainder, so we will not pay attention to the result. And now the pure calculation.

Calculation procedure:

```
11001001000 : 1101 = 10010010
1101
0001100
   1101
   00011
    1101
    1110
    1101
     001100
       1101
       00010
```

11001001000 : 1101 = 10010010, remainder = **010**

After the remainder comes out, we substitute the zeros we added for it.

11001001000 => **11001001010**

And by then, our secure data is ready to be sent.

POINT B

On the receiving side, the data is divided by the same GP and when the remainder comes out zero, the data arrives at point B intact.

11001001010 : 1101 = 10010100
1101
0001100
1101
0001101
1101
0000

If the remainder does not come out zero, we already know something is wrong. We have several options. One is to send a request to repeat the last message, or just to let the sending side know that the data did not arrive correctly. On a bad CRC, the control panel responds with a NAK response for the bad CRC.

5 ENC28J60 integrated circuit

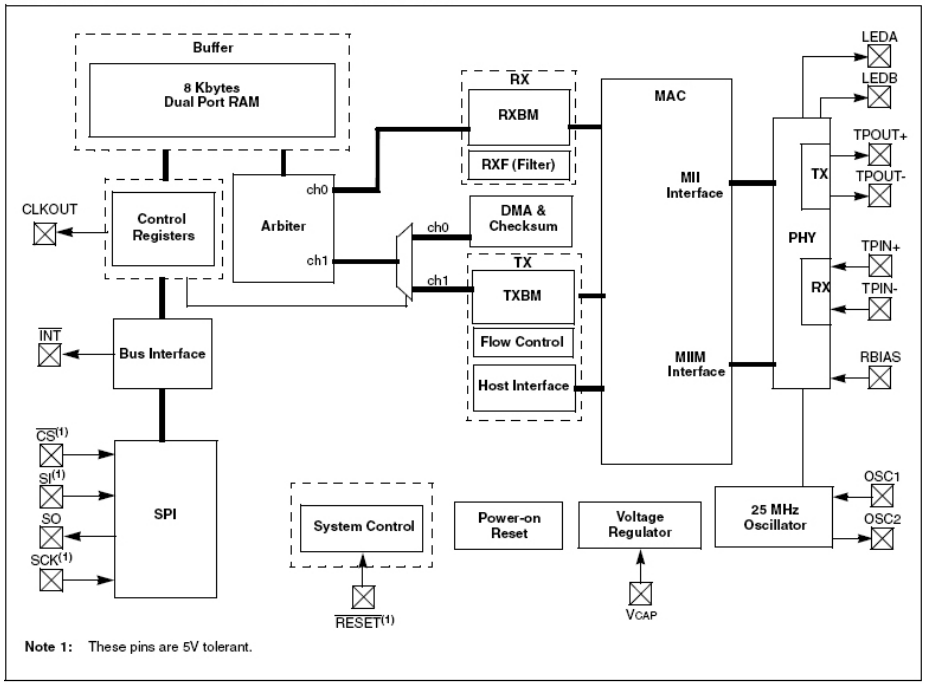
This section is a translation of a selection of parts from the manufacturer's data sheets.

5.1 Basic parameters ENC28J60

- The ENC28J60 (hereafter abbreviated as ENC) is an integrated circuit compliant with IEEE 802.3 and Ethernet industry standards operating at layers 1 and 2 of the OSI model. It is designed for 10BASE-T technology, which is a standard that uses Category 3 UTP and STP (Twisted Pair) cabling and RJ45 connectors designed for a star network topology whose basic network elements are the Hub and Switch. Maximum segment length 100m (328feet). The maximum data rate is 10 Mbps. Max. impedance:100Ω.
- ENC has ...
  - one IEEE 802.3 port with automatic polarity detection and correction
  - SPI interface for communication with the controlling micro-processor.
  - built-in cyclic 8KB memory (Buffer) programmably divided into Transmit and Recieve part.
  - implemented a polynomial and a mechanism to count 16bit checksum for IP protocols.
- Supports both types of media access; Full-Duplex and Semi-Duplex. The access type must be set manually. ENC cannot independently detect which type it is on a given connection. With the Semi-Duplex type enabled, the ENC can be set to automatically re-transmit the frame itself when a collision occurs, until it fails to send or until the number of collisions exceeds a critical limit.
-
- ENC is also capable of adding Padding and CRC32 to the frame itself, which it automatically generates. In addition, it includes a number of programmable receive filters (Unicast, Multicast, Broadcast, CRC mesh, Huge Frame, Pattermatch, Magic Packet and Hash Table Filter (CRC on the destination MAC address of the incoming packet, criteria check)
- Electrical parameters:
  - Power: 3.1 - 3.6 [V]

- 5V TTL inputs, 3.3V logic outputs

FIGURE 1-1: ENC28J60 BLOCK DIAGRAM



5.1.1 ENC28J60 memory:

All ENC memory is implemented as static RAM. It is divided into three types.

Control registers(Control reg:

-registers used to configure and detect ENC status.

-are divided into 4 pages

-they are accessed directly via the SPI interface

-Write/Read Control reg commands.

Ethernet Buffer 8KB

-is a transmit and receive memory, with individual access

-memory allocation is programmable by four control registers

-write read via SPI directly(Write/Read Buffer mem.)

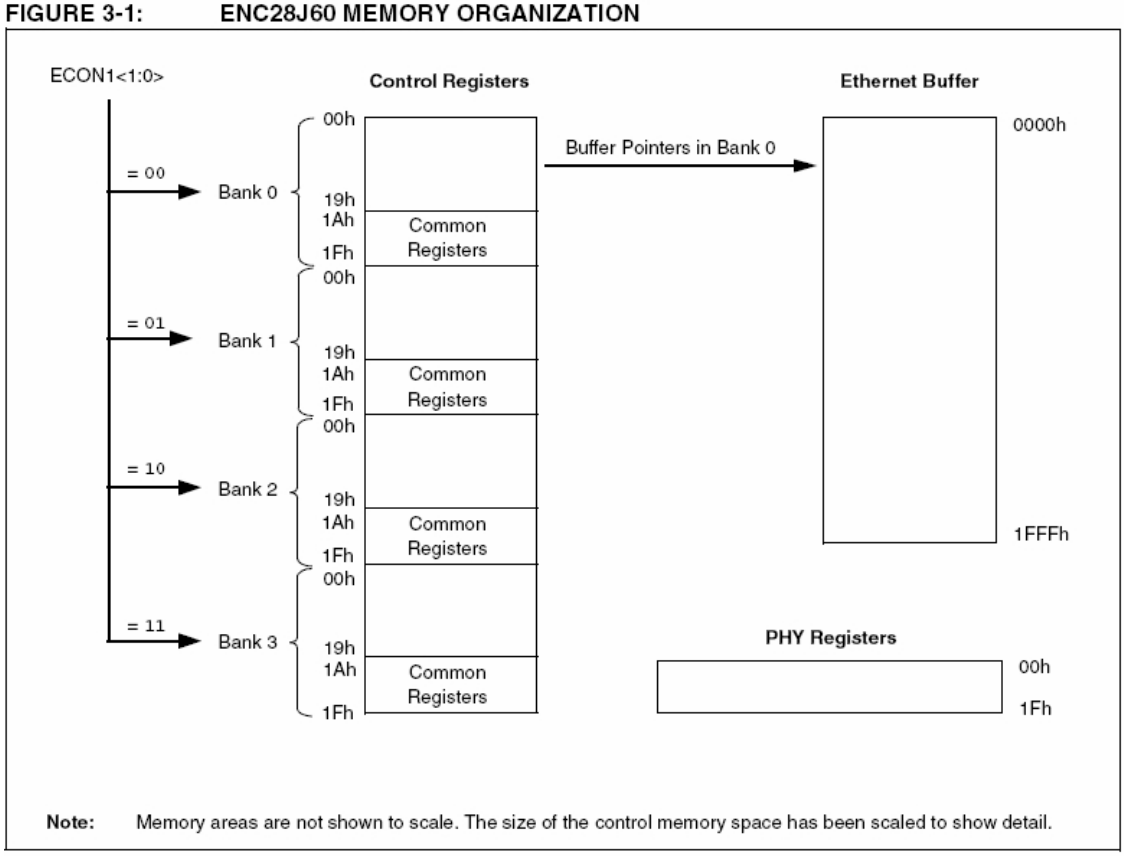
-change automatically incrementing pointers(write/read) via Control reg.

PHY registers

-registers used for configuration, health check, Physical layer health check

-access by writing via Control Registers

FIGURE 3-1: ENC28J60 MEMORY ORGANIZATION



Division of Control Registers into pages

TABLE 3-1: ENC28J60 CONTROL REGISTER MAP

Bank 0		Bank 1		Bank 2		Bank 3	
Address	Name	Address	Name	Address	Name	Address	Name
00h	ERDPTL	00h	EHT0	00h	MACON1	00h	MAADR5
01h	ERDPTH	01h	EHT1	01h	Reserved	01h	MAADR6
02h	EWRPTL	02h	EHT2	02h	MACON3	02h	MAADR3
03h	EWRPTH	03h	EHT3	03h	MACON4	03h	MAADR4
04h	ETXSTL	04h	EHT4	04h	MABBIPG	04h	MAADR1
05h	ETXSTH	05h	EHT5	05h	—	05h	MAADR2
06h	ETXNDL	06h	EHT6	06h	MAIPGL	06h	EBSTSD
07h	ETXNDH	07h	EHT7	07h	MAIPGH	07h	EBSTCON
08h	ERXSTL	08h	EPMM0	08h	MACLCON1	08h	EBSTCSL
09h	ERXSTH	09h	EPMM1	09h	MACLCON2	09h	EBSTCSH
0Ah	ERXNDL	0Ah	EPMM2	0Ah	MAMXFLL	0Ah	MISTAT
0Bh	ERXNDH	0Bh	EPMM3	0Bh	MAMXFLH	0Bh	—
0Ch	ERXRDPTL	0Ch	EPMM4	0Ch	Reserved	0Ch	—
0Dh	ERXRDPTH	0Dh	EPMM5	0Dh	Reserved	0Dh	—
0Eh	ERXWRPTL	0Eh	EPMM6	0Eh	Reserved	0Eh	—
0Fh	ERXWRPTH	0Fh	EPMM7	0Fh	—	0Fh	—
10h	EDMASTL	10h	EPMCSL	10h	Reserved	10h	—
11h	EDMASTH	11h	EPMCSH	11h	Reserved	11h	—
12h	EDMANDL	12h	—	12h	MICMD	12h	EREVID
13h	EDMANDH	13h	—	13h	—	13h	—
14h	EDMADSTL	14h	EPMOL	14h	MIREGADR	14h	—
15h	EDMADSTH	15h	EPMOH	15h	Reserved	15h	ECOCON
16h	EDMACSL	16h	Reserved	16h	MIWRL	16h	Reserved
17h	EDMACSH	17h	Reserved	17h	MIWRH	17h	EFLOCON
18h	—	18h	ERXFCON	18h	MIRDL	18h	EPAUSL
19h	—	19h	EPKTCNT	19h	MIRDH	19h	EPAUSH
1Ah	Reserved	1Ah	Reserved	1Ah	Reserved	1Ah	Reserved
1Bh	EIE	1Bh	EIE	1Bh	EIE	1Bh	EIE
1Ch	EIR	1Ch	EIR	1Ch	EIR	1Ch	EIR
1Dh	ESTAT	1Dh	ESTAT	1Dh	ESTAT	1Dh	ESTAT
1Eh	ECON2	1Eh	ECON2	1Eh	ECON2	1Eh	ECON2
1Fh	ECON1	1Fh	ECON1	1Fh	ECON1	1Fh	ECON1

Summary of all Control Registers, their individual bits and values after reset

TABLE 3-2: ENC28J60 CONTROL REGISTER SUMMARY

Register Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on Reset	Details on Page
EIE	INTIE	PKTIE	DMAIE	LINKIE	TXIE	r	TXERIE	RXERIE	0000 0000	65
EIR	—	PKTIF	DMAIF	LINKIF	TXIF	r	TXERIF	RXERIF	-000 0000	66
ESTAT	INT	BUFER	r	LATECOL	—	RXBUSY	TXABRT	CLKRDY(1)	0000 -000	64
ECON2	AUTOINC	PKTDEC	PWRSV	r	VRPS	—	—	—	1000 0---	16
ECON1	TXRST	RXRST	DMAST	CSUMEN	TXRTS	RXEN	BSEL1	BSEL0	0000 0000	15
ERDPTL	Read Pointer Low Byte ERDPT<7:0>								1111 1010	17
ERDPTH	—	—	—	Read Pointer High Byte (ERDPT<12:8>)					---0 0101	17
EWRPTL	Write Pointer Low Byte (EWRPT<7:0>)								0000 0000	17
EWRPTH	—	—	—	Write Pointer High Byte (EWRPT<12:8>)					---0 0000	17
ETXSTL	TX Start Low Byte (ETXST<7:0>)								0000 0000	17
ETXSTH	—	—	—	TX Start High Byte (ETXST<12:8>)					---0 0000	17
ETXNDL	TX End Low Byte (ETXND<7:0>)								0000 0000	17
ETXNDH	—	—	—	TX End High Byte (ETXND<12:8>)					---0 0000	17
ERXSTL	RX Start Low Byte (ERXST<7:0>)								1111 1010	17
ERXSTH	—	—	—	RX Start High Byte (ERXST<12:8>)					---0 0101	17
ERXNDL	RX End Low Byte (ERXND<7:0>)								1111 1111	17
ERXNDH	—	—	—	RX End High Byte (ERXND<12:8>)					---1 1111	17
ERXRDPTL	RX RD Pointer Low Byte (ERXRDPT<7:0>)								1111 1010	17
ERXRDPTH	—	—	—	RX RD Pointer High Byte (ERXRDPT<12:8>)					---0 0101	17
ERXWRPTL	RX WR Pointer Low Byte (ERXWRPT<7:0>)								0000 0000	17
ERXWRPTH	—	—	—	RX WR Pointer High Byte (ERXWRPT<12:8>)					---0 0000	17
EDMASTL	DMA Start Low Byte (EDMAST<7:0>)								0000 0000	71
EDMASTH	—	—	—	DMA Start High Byte (EDMAST<12:8>)					---0 0000	71
EDMANDL	DMA End Low Byte (EDMAND<7:0>)								0000 0000	71
EDMANDH	—	—	—	DMA End High Byte (EDMAND<12:8>)					---0 0000	71
EDMADSTL	DMA Destination Low Byte (EDMADST<7:0>)								0000 0000	71
EDMADSTH	—	—	—	DMA Destination High Byte (EDMADST<12:8>)					---0 0000	71
EDMACSL	DMA Checksum Low Byte (EDMACS<7:0>)								0000 0000	72
EDMACSH	DMA Checksum High Byte (EDMACS<15:8>)								0000 0000	72
EHT0	Hash Table Byte 0 (EHT<7:0>)								0000 0000	52
EHT1	Hash Table Byte 1 (EHT<15:8>)								0000 0000	52
EHT2	Hash Table Byte 2 (EHT<23:16>)								0000 0000	52
EHT3	Hash Table Byte 3 (EHT<31:24>)								0000 0000	52
EHT4	Hash Table Byte 4 (EHT<39:32>)								0000 0000	52
EHT5	Hash Table Byte 5 (EHT<47:40>)								0000 0000	52
EHT6	Hash Table Byte 6 (EHT<55:48>)								0000 0000	52
EHT7	Hash Table Byte 7 (EHT<63:56>)								0000 0000	52
EPMM0	Pattern Match Mask Byte 0 (EPMM<7:0>)								0000 0000	51
EPMM1	Pattern Match Mask Byte 1 (EPMM<15:8>)								0000 0000	51
EPMM2	Pattern Match Mask Byte 2 (EPMM<23:16>)								0000 0000	51
EPMM3	Pattern Match Mask Byte 3 (EPMM<31:24>)								0000 0000	51
EPMM4	Pattern Match Mask Byte 4 (EPMM<39:32>)								0000 0000	51
EPMM5	Pattern Match Mask Byte 5 (EPMM<47:40>)								0000 0000	51
EPMM6	Pattern Match Mask Byte 6 (EPMM<55:48>)								0000 0000	51
EPMM7	Pattern Match Mask Byte 7 (EPMM<63:56>)								0000 0000	51

Legend: x = unknown, u = unchanged, — = unimplemented, q = value depends on condition, r = reserved, do not modify.

Note 1: CLKRDY resets to '0' on Power-on Reset but is unaffected on all other Resets.

2: EREVID is a read-only register.

3: ECOCON resets to '---- -100' on Power-on Reset and '---- -uuu' on all other Resets.

TABLE 3-2: ENC28J60 CONTROL REGISTER SUMMARY (CONTINUED)

Register Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on Reset	Details on Page
EPMCSL	Pattern Match Checksum Low Byte (EPMCS<7:0>)								0000 0000	51
EPMCSH	Pattern Match Checksum High Byte (EPMCS<15:0>)								0000 0000	51
EPMOL	Pattern Match Offset Low Byte (EPMO<7:0>)								0000 0000	51
EPMOH	—	—	—	Pattern Match Offset High Byte (EPMO<12:8>)					---0 0000	51
ERXFCON	UCEN	ANDOR	CRCEN	PMEN	MPEN	HTEN	MCEN	BCEN	1010 0001	48
EPKTCNT	Ethernet Packet Count								0000 0000	43
MACON1	—	—	—	r	TXPAUS	RXPAUS	PASSALL	MARXEN	---0 0000	34
MACON3	PADCFG2	PADCFG1	PADCFG0	TXCRCEN	PHDREN	HFRMEN	FRMLNEN	FULDPX	0000 0000	35
MACON4	—	DEFER	BPEN	NOBKOFF	—	—	r	r	-000 --00	36
MABBIPG	—	Back-to-Back Inter-Packet Gap (BBIPG<6:0>)							-000 0000	36
MAIPGL	—	Non-Back-to-Back Inter-Packet Gap Low Byte (MAIPGL<6:0>)							-000 0000	34
MAIPGH	—	Non-Back-to-Back Inter-Packet Gap High Byte (MAIPGH<6:0>)							-000 0000	34
MACLCON1	—	—	—	—	Retransmission Maximum (RETMAX<3:0>)				---- 1111	34
MACLCON2	—	—	Collision Window (COLWIN<5:0>)						--11 0111	34
MAMXFLL	Maximum Frame Length Low Byte (MAMXFL<7:0>)								0000 0000	34
MAMXFLH	Maximum Frame Length High Byte (MAMXFL<15:8>)								0000 0110	34
MICMD	—	—	—	—	—	—	MIISCAN	MIIRD	---- --00	21
MIREGADR	—	—	—	MII Register Address (MIREGADR<4:0>)					---0 0000	19
MIWRL	MII Write Data Low Byte (MIWR<7:0>)								0000 0000	19
MIWRH	MII Write Data High Byte (MIWR<15:8>)								0000 0000	19
MIRDL	MII Read Data Low Byte (MIRD<7:0>)								0000 0000	19
MIRDH	MII Read Data High Byte(MIRD<15:8>)								0000 0000	19
MAADR5	MAC Address Byte 5 (MAADR<15:8>)								0000 0000	34
MAADR6	MAC Address Byte 6 (MAADR<7:0>)								0000 0000	34
MAADR3	MAC Address Byte 3 (MAADR<31:24>), OUI Byte 3								0000 0000	34
MAADR4	MAC Address Byte 4 (MAADR<23:16>)								0000 0000	34
MAADR1	MAC Address Byte 1 (MAADR<47:40>), OUI Byte 1								0000 0000	34
MAADR2	MAC Address Byte 2 (MAADR<39:32>), OUI Byte 2								0000 0000	34
EBSTSD	Built-in Self-Test Fill Seed (EBSTSD<7:0>)								0000 0000	76
EBSTCON	PSV2	PSV1	PSV0	PSEL	TMSEL1	TMSEL0	TME	BISTST	0000 0000	75
EBSTCSL	Built-in Self-Test Checksum Low Byte (EBSTCS<7:0>)								0000 0000	76
EBSTCSH	Built-in Self-Test Checksum High Byte (EBSTCS<15:8>)								0000 0000	76
MISTAT	—	—	—	—	r	NVALID	SCAN	BUSY	---- 0000	21
EREVID(2)	—	—	—	Ethernet Revision ID (EREVID<4:0>)					---q qqqq	22
ECOCON(3)	—	—	—	—	—	COCON2	COCON1	COCON0	---- -100	6
EFLOCON	—	—	—	—	—	FULDPXS	FCEN1	FCEN0	---- -000	56
EPAUSL	Pause Timer Value Low Byte (EPAUS<7:0>)								0000 0000	57
EPAUSH	Pause Timer Value High Byte (EPAUS<15:8>)								0001 0000	57

Legend: x = unknown, u = unchanged, — = unimplemented, q = value depends on condition, r = reserved, do not modify.

Note 1: CLKRDY resets to '0' on Power-on Reset but is unaffected on all other Resets.

2: EREVID is a read-only register.

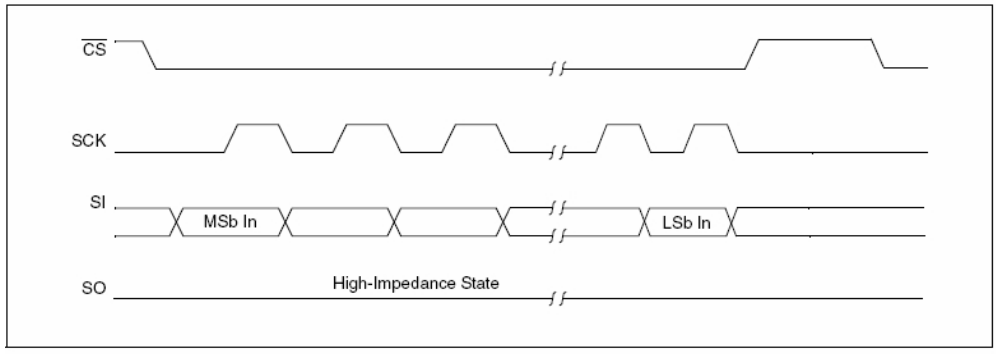
3: ECOCON resets to '---- -100' on Power-on Reset and '---- -uuu' on all other Resets.

5.1.2 Access to ENC28J60 SPI interface:

The ENC supports 8-bit SPI 0.0 slave mode with a clock at logic zero in the idle state

Sending a byte over the SPI is initiated by setting logic 0 on the ENC CS pin (DsPIC PortD8), which puts the ENC side in the command pending state. Now the sending of the command byte/address/data according to the command table can follow. The clock on the SPI is activated and the byte is sent and received at the same time. When the transmit/receive is complete, the CS pin returns to the logic 1-down state.

FIGURE 4-1: SPI INPUT TIMING



SPI commands:

The function of the ENC28J60 is completely dependent on the instructions sent from the control device (DsPIC) via the SPI interface. Instructions are sent in the form of single or multiple byte instructions used to access the Control Memory and Ethernet Buffer. The instruction consists of at least a 3-bit opcode followed by a 5-bit argument containing either a register address or a data constant.

Write and BitSet instructions are followed by one or more bytes of data.

Table of instructions:

TABLE 4-1: SPI INSTRUCTION SET FOR THE ENC28J60

Instruction Name and Mnemonic	Byte 0								Byte 1 and Following							
	Opcode			Argument					Data							
Read Control Register (RCR)	0	0	0	a	a	a	a	a	N/A							
Read Buffer Memory (RBM)	0	0	1	1	1	0	1	0	N/A							
Write Control Register (WCR)	0	1	0	a	a	a	a	a	d	d	d	d	d	d	d	d
Write Buffer Memory (WBM)	0	1	1	1	1	0	1	0	d	d	d	d	d	d	d	d
Bit Field Set (BFS)	1	0	0	a	a	a	a	a	d	d	d	d	d	d	d	d
Bit Field Clear (BFC)	1	0	1	a	a	a	a	a	d	d	d	d	d	d	d	d
System Reset Command (Soft Reset) (SRC)	1	1	1	1	1	1	1	1	N/A							

Legend: a = control register address, d = data payload.

5.1.3 ENC2860 wiring:

Oscillator:

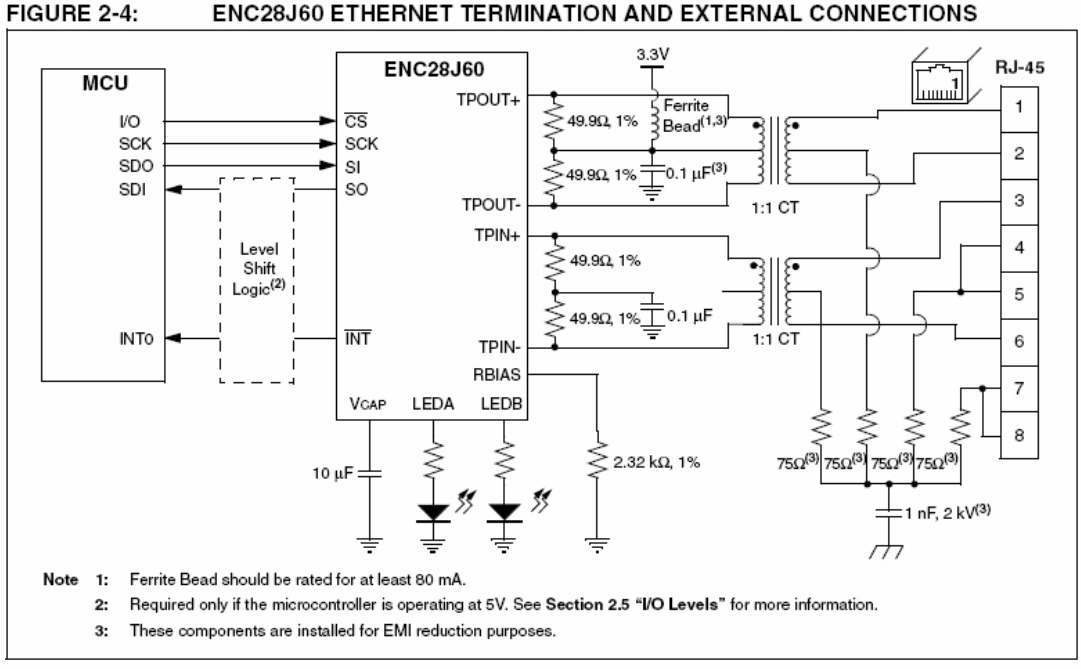
ENC designed at 25 MHz. This can be accomplished by either connecting the crystal to pins OSC1 and OSC2, or by using an external clock source and connecting it to OSC1.

After the Power-On and Power-down restart, wait until the oscillator stabilizes. When this happens the CLKRDY bit in ESTAT (Control Register in ENC) is set to log1.

External connections:

The ENC must be wired according to the datasheets to work properly:

FIGURE 2-4: ENC28J60 ETHERNET TERMINATION AND EXTERNAL CONNECTIONS



Note 1: Ferrite Bead should be rated for at least 80 mA.
2: Required only if the microcontroller is operating at 5V. See **Section 2.5 "I/O Levels"** for more information.
3: These components are installed for EMI reduction purposes.

The physical layer module needs a 2.32KΩ resistor to ground on the RBIAS pin. This resistor affects the output signal aplitude on the TPOUT+/- pins. It is recommended that this be a surface mount type and connected, for interference reasons, with the shortest possible connection to ground.

The ENC is powered by 3.3V, whereas the DsPIC used in this project as a control controller is powered by 5V. The voltage difference needs to be compensated. The ENC inputs(CS, SCK, SI, RESET) are 5V tolerant, but the outputs(SO, INT, CLKOUT) are not at TTL levels, although the DsPIC would probably recognize the levels correctly(3.3 is still within the tolerance) a level lift should be used to be sure, which can be implemented e.g. by using a CMOS logic AND circuit (74HCT08), one input of which is connected to log1. see Figure 2-5.

It is also possible to connect 2 LEDs to the ENC, which can be set to indicate Link(cable connected to another device) and Recieve(incoming packet) status. By connecting LED B, it is possible to set up Full/Half duplex access to the media in hardware. See Figure 2-7

FIGURE 2-5: LEVEL SHIFTING USING AND GATES

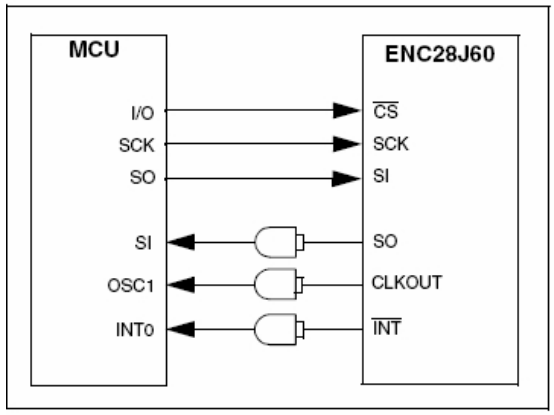
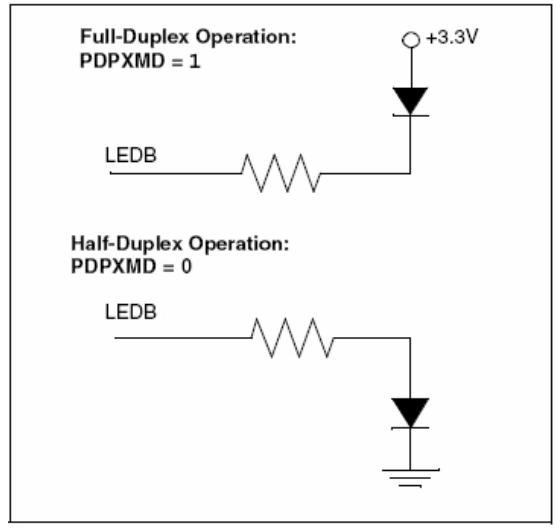


FIGURE 2-7: LEDB POLARITY AND RESET CONFIGURATION OPTIONS



5.1.4 Checksum calculation in ENC28J60:

ENC has a checksum calculation function. Two pointers (divided into the following. and top. byte, EDMAST-pointer to the first byte of the checksum data, EDMAND-pointer to the last byte included in the checksum) define the area in the cyclic Ethernet Buffer for which the ENC should calculate the checksum. Then we activate the start bits of the operation(ECON1.CSUMEN;ECON1.DMAST). When the calculation is finished, EIR.DMAIF is set, which can generate an interrupt. The result is in the EDMACS control registers (H and L upper and lower byte).

The checksum works with the marked contents of the Ethernet Buffer as individual 16-bit numbers (if it is an odd number, the padding 00h is added to complete the 16 bits). These 16-bit numbers are added (the Carry bit for each individual sum is automatically added to the given result) and the complement of the total result to the max. value(0xFFFFh) is the resulting 16-bit checksum.

Example:

values in memory for final sum:{89h, ABh, CDh} sum:

89ABh + CD00h = 156ABh

carry bit shift: 56ABh + 0001h = 56ACh

complement: FFFFh - 56ACh = A953h

checksum is therefore A953h

This method is used for IP, TCP, UDP, ICMP, etc.

Page settings (SetPage subroutine)

The current page number lists the two lowest bits(BSEL0,1) of the ECON1 control register.

REGISTER 3-1: ECON1: ETHERNET CONTROL REGISTER 1

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
TXRST	RXRST	DMAST	CSUMEN	TXRTS	RXEN	BSEL1	BSEL0
bit 7							bit 0

The page change must be done using BitSet(not write2control) instructions, because the page change must not affect other bits in the register!

5.1.5 Write to PHY register (Write2Physical subroutine)

The PHY registers cannot be written to directly by an SPI instruction. This must be done by writing to special Control registers. The following is the writing procedure:

- entry of the PHY register address in the MIREGADR Control Register
- writing the lower 8 bits of data into the Control Reg. MIWRL
- writing the upper 8 bits of data into the Control Reg. MIWRH
- wait until the MISTAT.BUSY bit drops back to 0, or wait exactly 10.24us, or continue, but do not write or read from PHY memory for a given period of time.

5.2 Device initialization

(see 6.0 INITIALIZATION)

Before the device can function properly, it must first be set up. This is done by writing a series of values to some Control Registers. If these registers are divided into upper and lower 8 bits, it is always necessary to write the value to the lower 8 bits first. It is recommended to follow the given procedure.

Splitting the Ethernet Buffer into receiving and transmitting parts

This is done by writing down the starting and ending addresses of the receiving part. The buffer has 8K. The highest possible address is 1FFFh. If we want to split the memory exactly in half, we set the Receive(Recieve) part by saving

0FFEh to ERXST(16bit reg. pointing to the beginning of the received part, must be an even address) 1FFFh to ERXND(16bit reg. pointing to the end of the received part)

All other memory is considered the transmitting part.

ERXRDPT must be set to the same value as ERXST

Receive Filters settings (see Recieve Filters)

Receive filters are set by the ERXFCON register. By a series of enable bits. For most applications it makes sense to consider setting UCEN(Unicast), CRCEN(CRC ok), BCEN(Broadcast), ANDOR(filter pass logic). For normal applications, it is advisable to leave the default value.(UCEN,CRCEN,BCEN=1; rest 0)

REGISTER 8-1: ERXFCON: ETHERNET RECEIVE FILTER CONTROL REGISTER

R/W-1	R/W-0	R/W-1	R/W-0	R/W-0	R/W-0	R/W-0	R/W-1
UCEN	ANDOR	CRCEN	PMEN	MPEN	HTEN	MCEN	BCEN
bit 7							bit 0

- UCEN - passes frames with a MAC address matching the address stored in the MAADR1-6 Control Registers. These are set later.
- CRCEN - passes frames for which it does not detect an error using CRC32, the ANDOR bit does not apply
- BCEN - passes all broadcast frames(MAC= ff:ff:ff : ff:ff:ff). Such frames are used e.g. in ARP protocol, so they are necessary for parsing and response.

- ANDOR (see Figure 8-1 Recieve Filtering using OR logic)
    - Log 1-frame must pass all filters
    - Log 0- frame must pass at least one filter

Whatever the ANDOR value is, the packet must also pass the CRCEN filter if it is enabled.

A packet that does not pass the defined filters will be rejected. The user will not be able to register the rejected frame in any way.

Writing 00h to ERXCON switches the device to promiscuous mode (all incoming packets will be accepted)

Waiting for Oscillator Start-up Timer (see Wainting For OST)

It is necessary to wait for the oscillator to stabilize after Power-on Reset before proceeding with initialization. The stabilization indicator is the ESTAT.CLKRDY bit. Once it is set to 1 we can proceed. (MAC and PHY registers can now be set)

MAC initialization

What follows is a sequence of settings that can only be understood after a deeper understanding of the issue. I prefer to follow the original datasheet - its recommended settings.

In some registers it is necessary to write different values for the Full and Half/Full-Duplex type of multiple media access. The settings shown will be independent of the external wiring of LED B in both cases, which can be set by wiring the connection type in a hardware way. (See External connections; LED polarity)

In all cases, the following entries refer to the registers on page 2.

Full Duplex	**Half Duplex**
0Dh to MACCON1	01h to MACCON1
B3h to MACCON3	B2h to MACCON3
40h to MACCON4	40h to MACCON4
EEh to MAMFLL	EEh to MAMFLL
05h to MAMFLH	05h to MAMFLH
15h to MABBIPG	12h to MABBIPG
12h to MAIPGL	12h to MAIPGL
0100h to PHY.PHCON1	0C to MAIPGH
	0000h to PHY.PHCON1
	0100h to PHY.PHCON2

MAC address settings:

For the Unicast filter to work properly, you must store your own MAC address in the ENC, it is not supplied with the IO. The unique MAC address can be taken from old or broken network cards, for example, and which can then be discarded to avoid problems with collisions of two identical MACs in the network. The MAC address is written to the cont. reg. on page 3 of MAADR1-6, with the first byte(LSB) written to MAADR6.

Interruption when receiving packets:

ENC can generate an interrupt on every incoming packet that passes through the Filters if we set the EIE.PKTIE and EIE.INTIE enable bits or if we want it to generate an interrupt on every packet dropped due to Buffer overflow, we need to reset EIR.RXERIF and set the EIE.RXERIE and EIE.INTIE bits.

Enabling packet reception:

After the RXEN bit is set, the Duplex mode, the pointer indicating the start and end of the receive buffer should not be changed and to prevent receiving unsuitable packets it is recommended to temporarily reset RXEN before changing the ERXFCON and MAC address filters. Once reception is enabled, all packets matching the filters will be written to the receive buffer memory. Those rejected will be discarded and the user will not be able to detect this.

To allow admission

It is necessary to set ECON1, RXEN

5.3 Sending a packet

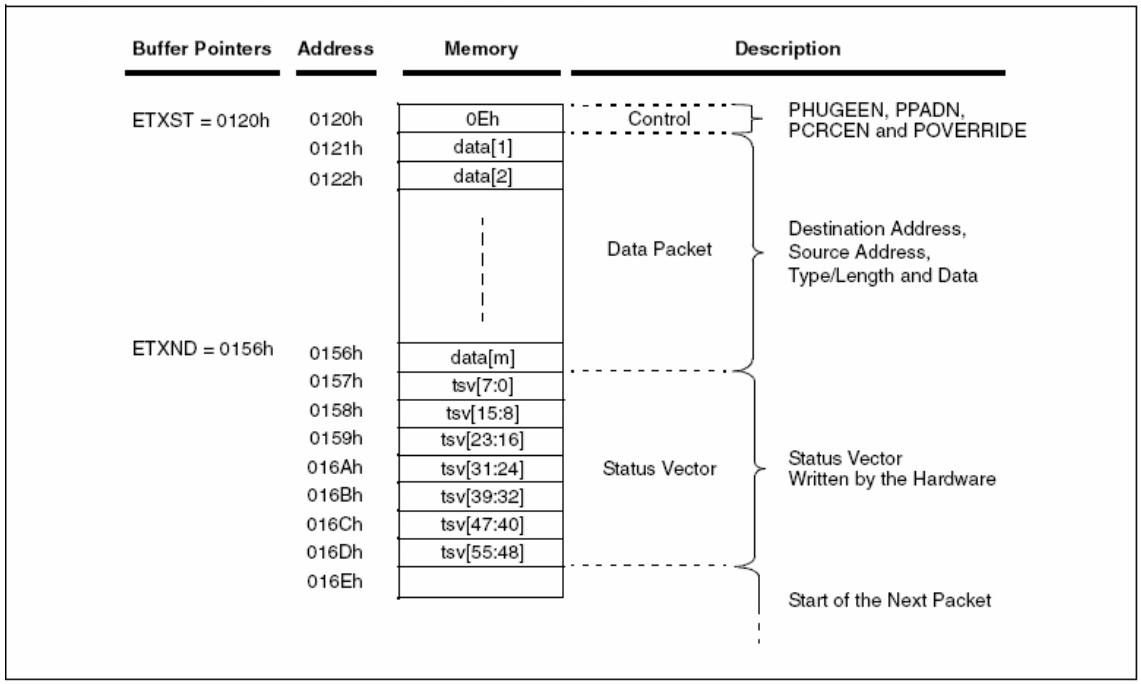
If ENC is initialized correctly it automatically generates Preamble, Start-of-frame, padding(60B or 64B) and CRC32. This setting can also be changed just for a single packet by using the Per-Packet-Control byte(see 7.1 datasheet). The following is the recommended catalog procedure for sending in a packet:

1. Set the ETXST pointer to the even unused (Non-receive) part of the memory, where the packet will start. *If the program does not need to spit out a lot of data at once and cannot wait for packets to be sent, it is sufficient to set this pointer to the same address each time.*
2. Write to the buffer below the value in ETXST, Per-Packet-Control byte (00h for most applications)
3. Write DATA (mac destination address, mac source address, type/length, frame data)
4. Set the ETXND pointer to the last byte of data included in the packet
5. Reset EIR.TXIF bit, set EIE.INTIE to enable interrupts
6. Send packet by setting ECON1.TXRTS

The packet will be sent as soon as possible - as soon as any DMA operation (e.g. Checksum calculation) is finished. Similarly, the DMA operation would wait if a packet is being transmitted.

Once the packet is sent or if its transmission is cancelled TXRTS will be reset and the seven byte *Status Vector* (see chapter) will be written after the transmitted packet. If ESTAT.TXARB is in 1, the packet was not sent correctly. ESTAT.LATECOL determines that the collision occurred after the 64 bytes were transmitted. Half-duplex issues, but this problem should not be too common and in most cases does not need to be treated)

FIGURE 7-2: SAMPLE TRANSMIT PACKET LAYOUT



5.4 Receiving packets

Once receive is enabled in initialization, all packets matching the filters will be written to the receive buffer memory. Rejected ones will be deleted and the user will not be able to detect this in any way.

Once a packet is completely received and written to the buffer, the EPKCNT register increments, EIR.PKTIF (indicating that there are one or more unprocessed packets in memory) is set to 1 and an interrupt will be raised if enabled, and the Hardware Write Pointer (pointing to the address where the next packet will be written; it is software unwritable) automatically increments by 1. EIR.PKTIF resets the hardware after the packet space is freed*, see the chapter Freeing space in the receive memory*.

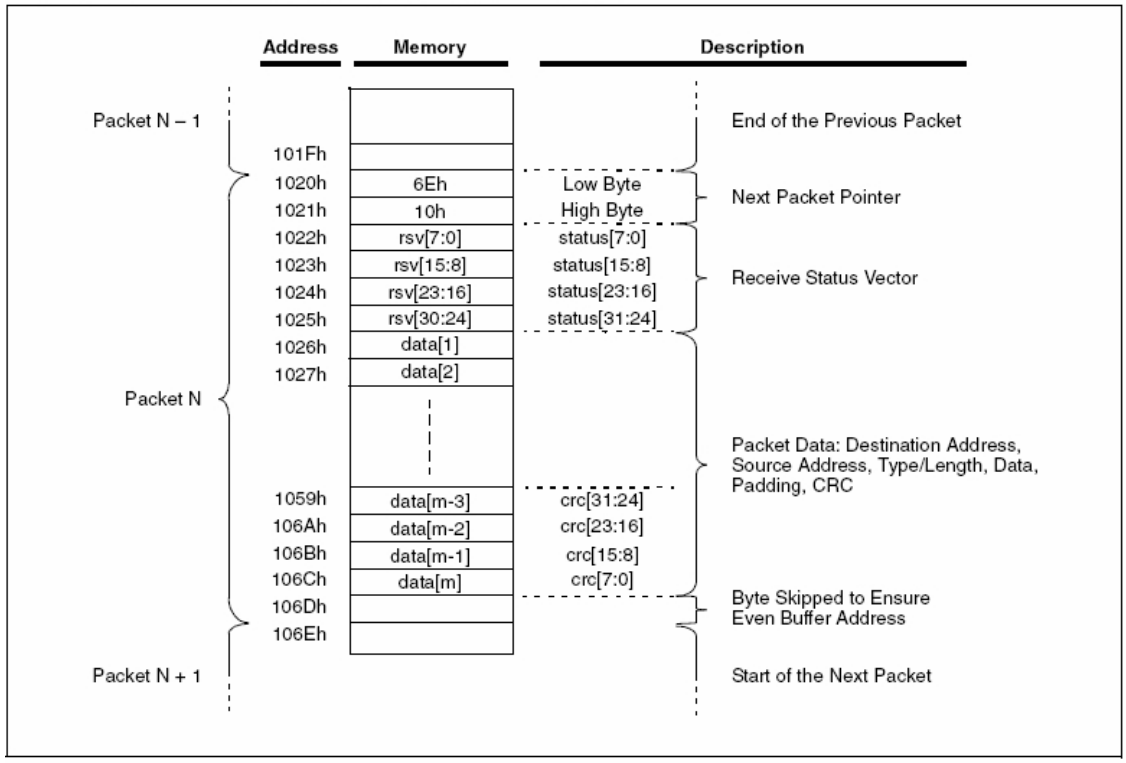
The received packet (see Figure 7-3) written in the buffer has 6 information bytes preassigned.

1. The first two specify the so-called Next Packet Pointer (i.e. the address of the first byte of the next packet) The Next Packet Pointer value is very important for the running of the whole receiving program - I recommend to back it up to the control processor (DsPIC).

2. The following is a 4-byte Status vector (see chapter)
3. After the 6 bits of information, the frame data (destination mac. ad., source mac. ad., type/length, data, padding and CRC32 of the frame) are in memory.

A padding bit can be added at the end so that the first address of the next packet(next packet pointer) is even.

FIGURE 7-3: SAMPLE RECEIVE PACKET LAYOUT



5.4.1 Freeing up space in the receiving memory

Releasing cyclic memory(buffer)

After the user processes a packet (or a part of it) and wants to free the space filled with processed data, he must increase the value of the Recieve Buffer Read Pointer (ERXRDPT). ENC will write cyclically to the Buffer if the value of the Hardware Write Pointer is different from the value of ERXRDPT, so all data below and at the address in the buffer pointed to by the ERXRDPT pointer are protected against being overwritten by new data. When ENC attempts to overwrite an ERXRDPT protected cell, the write is aborted, the value in the cell remains, and the interrupt is invoked if enabled (EIR.RXERIF is set, EIE.RXERIE is enabled for interrupts).

All incoming packets will be dropped until memory is freed.

The change of the pointer will be performed after writing first the lower and then the upper byte.If the user processes all data in a certain packet and wants to free its place, he must increase the value of ERXRDPT so that it points for the processed packet (note that the memory is cyclic, it is best to set the value from the Next Packet Pointer), then it is necessary to set the bit ECON2.PKTDEC. When the user does this the value in EPKTCNT (the register containing the value of unprocessed packets) decrements by 1. If EPKTCNT reaches zero the hardware resets the EIR.PKIF bit (indicating that there is one or more unprocessed packets in memory).

Determining the amount of free space in cyclic memory

By subtracting the values of the ERXRDPT and ERXWRPT(Hardware Write Pointer) registers, the user gets the number of free bytes ready to write new packets.

6 Program in dsPIC30F3013

This section summarizes the most important features of the program used in the Médea project.

6.1 Basic sub-programmes:

In order to simplify, speed up and make the program clearer, we have created subroutines for the most frequently used groups of instructions. Some of them have input parameters and some also return a value. For input and return output values we use wregisters, of which there are a total of 15 (0-14) usable in our DsPIC. For input/output values we mostly use W registers 1-3, the rest are for working calculations.

Subroutines SPISendW1, SPISendW2:

They are the lowest subroutines in the hierarchy designed for direct communication with the SPI interface. They send the lower 8 bits from W1 or W2 and receive the 8 bits sent from the target device (ENC2860).

Subroutines directly using SPISend, setting the CS pin

Write2Control subroutine:

This subroutine is designed to store the value in W2 into the Control Register in ENC at the address stored in W1 on the current page. It sends a complete write control register command. It can be preceded by a prodprog. SetPage to set the page.

Subroutine ReadControl:

It is intended to be read from the Control Register. It is very similar to Write2Control. It has a variant for reading Control Regs of MAC type

BitSet/Clear subroutine

Set the selected value by masking it with the value in W2 of the Control Register at the address in W1.

Write2Buffer/ReadBuffer subroutines

They are designed for reading and writing from the Ethernet Buffer. They send complete write/read buffer mem commands. They send/return values in W1. It can be traversed by a SetWrite/ReadPointer subroutine setting the pointer to the selected value.

SoftReset subroutine

This subroutine invokes a "soft" reset of ENC by sending a command over SPI.

Sub-programmes directly using any of the above sub-programmes

Subroutines SetPage, Write2Physical, SetWrite/ReadPointer, etc...

6.2 Start of the programme

The programme starts in 3 phases:

- DsPIC processor settings (Ports, SPI, Uart, ..)
- Reset ENC28J60 (Hardware reset, Half/Full Duplex switch status detection)
- Initializing ENC28J60 (Reading IP from EEPROM, Initializing, Starting Ethernet interface)

This puts the Médea module in a state of expectation, it is a passive device (it never starts transmitting first). The program in the DsPIC is put into a cycle expecting a change, either pressing one of the buttons or receiving a packet.

6.3 Running the program

After the start, the program is put into the state of expectation (it is a passive device) of either an incoming packet change of state on the Half/Full Duplex jumper, change of state of the Reset IP button.

- When a packet arrives, the program jumps to the subroutine and processes the packet, or sends an appropriate response. (more in chapter Identifying the incoming packet)
- When a state change is detected on the Half/Full Duplex jumper the program resets. Some incoming packets that may be currently being received by the ENC28J60 circuit may be lost, so it is necessary to perform the duplex change only when the device is in an idle state, preferably when the device is turned off completely.

- Pressing the Reset IP button causes the basic IP address setting (192.168.1.5) to be loaded.If some packets are still to be received on the original IP, perform the Reset IP after they have been processed.

6.4 Half/Full Duplex jumper

ENC28J60 does not have autonegotiation function (automatic setting of duplex operation using synchronization pulses) and therefore the duplex operation can be set manually using a jumper and we have two options:

- Half-Duplex - the device is connected to a HUB or a router that does not support Full Duplex
    - jumper is closed
- Full-Duplex - the device is connected to a Switch or a router supporting Full Duplex
    - jumper not connected

Perform duplex change only when the device is at rest.

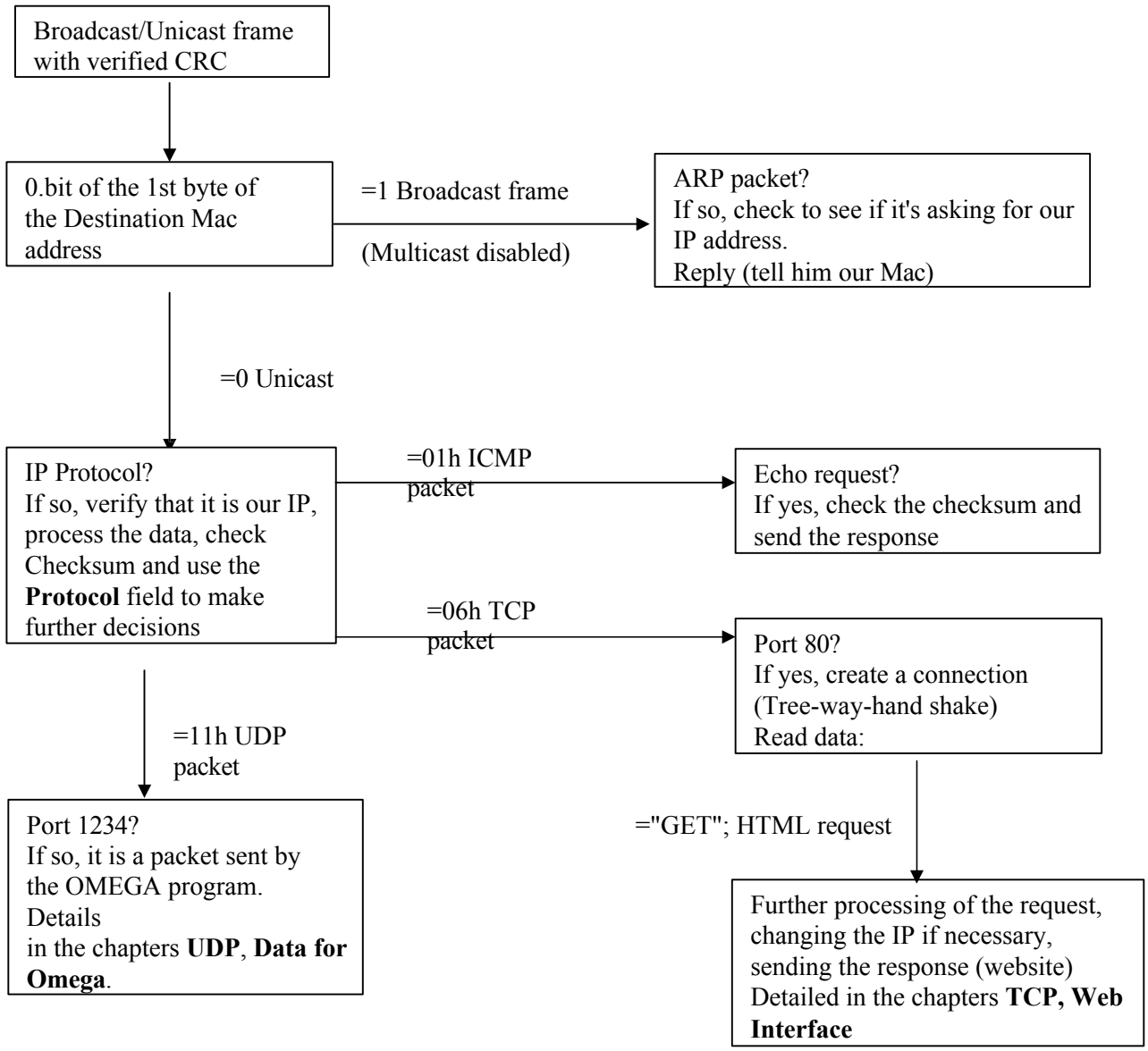
6.5 Reset IP

Normally, the last set IP address is read from the EEPROM every time the program is started and changed using the web interface (see the Web Interface chapter).However, if for some reason the user does not know this address and therefore does not have access to the web page, the user must perform a Reset IP, which sets the address back to the base 192.168.10.111 and then can change it to the desired value.

6.6 Incoming packet identification

(pass through the layers of the OSI reference model)

Once a packet is received the Médea module starts processing it. First, it needs to determine w h a t k i n d of packet it is - it starts the sorting process, the data reading process, which either discards the packet as unwanted/unsupported or identifies it as a request for a specific service and then processes it and sends a response.



6.7 Interruption

We use UART interrupts for both transmit and receive, this will allow for program acceleration.

6.7.1 UART initialization

The initialization of the UART is performed by the _UartInit subroutine. Thus **call _UartInit.**

- U1BRG entry. Speed 57600, assuming eight-phase hinge.
- Writing to U1MODE number 0x8400, i.e. 8 data bits, parity zero, 1 stop bit
- UART activation => setting the UTXEN bit to 1 in the U1STA register.

6.7.2 Interrupt .global reception

Data is stored in memory in the .global receive interrupt

- For UDP: Received and stored in memory
- For TCP: Cut EE 23 header, cut CRC

6.7.3 Interrupting .global transmissions

Transmission of data from the memory to the control panel is done in interruption .global transmission

- For UDP: Forwarding the UDP packet to the PBX
- For TCP: Adding header, doubling EE after START = 23, counting CRC

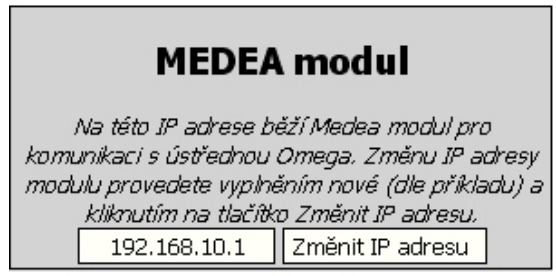
6.7.4 CRC32 counting

For TCP packets it is necessary to calculate the CRC. The subroutine for CRC is called by the command call CRC32. The procedure for using the subroutine is as follows:

- Insert the start address of the data into the w13 register and the length of the data into the data_length cell
- Read data from memory, put it into W0 and call the subroutine call crc32_update
- After the last byte of data the program continues to crc_end and then we get the result in cells crc32_H and crc32_L

Note: Two programs were created, one calculates the CRC according to the method described in 4.3.3, but this method did not work when implemented on data from the control panel. Therefore, a second program was created by rewriting the CRC32 program from the C language. The program was created by Professor Tomáš Kubalík, the leader of our work. Unfortunately, we did not find out how and why the first method does not work.

7 Web interface



autoři: Košař, Kapic

7.1 Change IP address

Changing the IP address is done via the web interface - using any web browser. First, a connection is made between the browser and

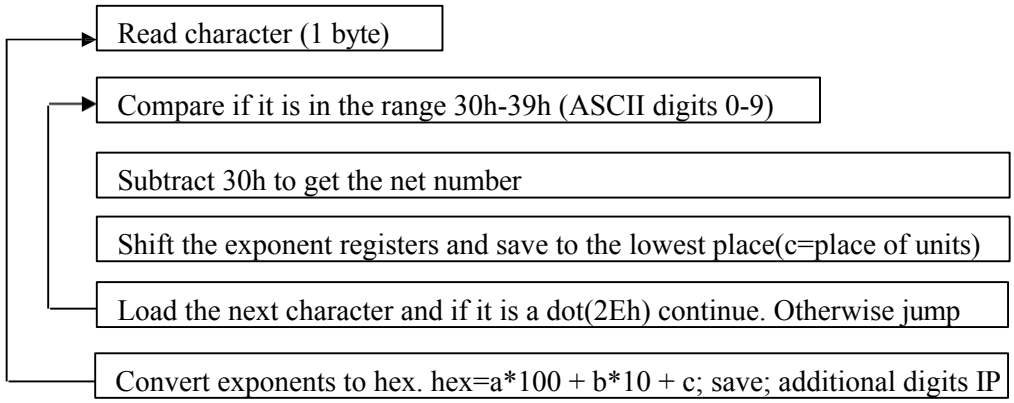
The Médea module creates a TCP/IP connection on port 80 (http protocol). When the Médea module receives a request for a specific action using the parameters of the http GET command, it sends the appropriate html response and possibly changes the IP address in the DsPIC-EEPROM.

Principle:

- TCP segment data contains ASCII text GET / http
  - The medea module sends back a standard web page, using the saved ascii form to fill in the IP address change field
- The TCP segment data contains the ASCII text GET /?IP= followed by the ***correct*** the specified IP in ASCII form
  - the IP image is loaded ascii
  - converts to four hexadecimal numbers
  - both variants of IP expression are stored in EEPROM
  - the response "IP address has been successfully saved" is sent
  - the IP address in RAM is changed
  - the program continues to respond only to the new IP address
- The TCP segment data contains the ASCII text GET /?IP= followed by the ***incorrect*** the specified IP in ASCII form
  - The Medea module sends a standard web page with the warning "Incorrect IP!"

ASCII IP -> HEX IP conversion

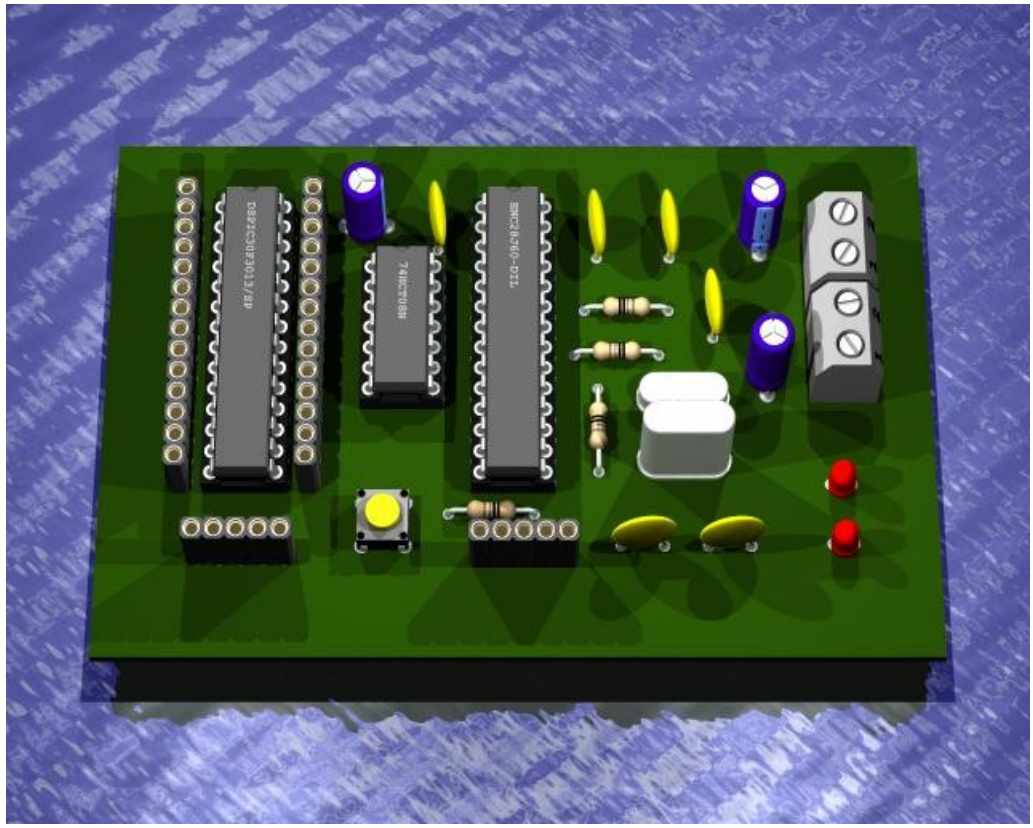
Normally, the IP address is written in decimal form, with each digit separated by a period. Omitting zeros makes the length unfixed. This is the case even when the IP address is expressed in ASCII characters. In this form, it is easy for humans to read and remember. However, in the packets we parse and send, the IP address is expressed in four hexadecimal digits, not separated by anything.



8 Formation of the PCB

In total, two albums were made. The first one was purely for training purposes => Training board. And the second one is ready to be used in the control room.

8.1 Training board

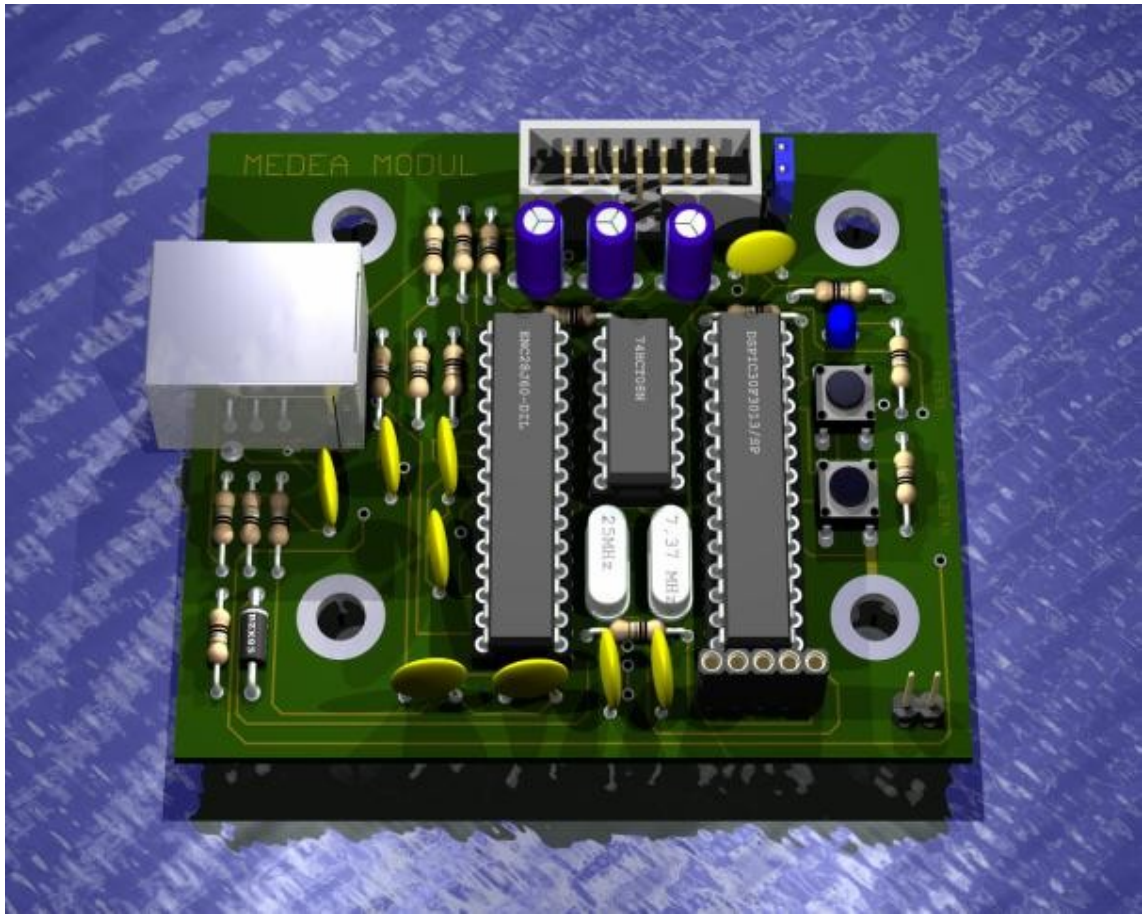


Placement of components on the board in 3D

On this board we verified the SPI communication between PIC and ENC, UART and everything needed to implement our board in TCP/IP networks. There are breakout bars in place for the usual "where could be the error". The breakout bar is also used for programming. Under the PIC (left) are the pins for connecting the Thomas converter, under the ENC are the pins for connecting the transformer and the RJ-45 connector. The power supply pins here are solved by terminal blocks, one for 5V and the other for 3.3V (ENC required), the grounds are connected. The button resets the ENC. 25 MHz (ENC) and 7.3728 MHz (dsPIC) crystals are used. In addition, a 74HCT08 CMOS gate is used to convert levels between the PIC and ENC.

8.2 Medea module version 1.1

The second product we called Medea module. The product is already ready for use in the control panel, it is equipped with mounting holes for fixing. The module is powered by the control panel, from which 5V comes out, the 3.3V power supply is solved by a Zener diode (wiring below). The strip below the PIC is for programming. The board has two buttons, the button located at the top is RESET (resets the PIC - leads to the MCLR) and the button located below it is RESET IP (sets the
default IP address of the module 192.168.10.111). For better orientation during information transfer there are 3 LEDs, two in a column above each other next to the RJ-45 connector and one above the buttons. The module communicates with the control panel via a 14-pin WSL 14G connector (top-most description below). The last important element is the jumper (next to WSL 14G), it sets half-duplex and full-duplex.
Jumper wiring: **HALF = JUMPER CLOSED**, **FULL = WITHOUT JUMPER**



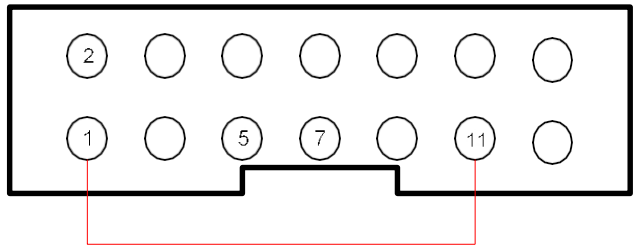
Medea module version 1.1

There was an error in the implementation of the board. **The RX and TX pins of** the 14-pin connector **were swapped**. The error is corrected by a reducer included in the module. See below for a description of the connector.

Immediately after the bug was discovered, another board design was created, version 1.2, where everything is fixed.

8.3 Description of WSL 14G

The module is connected to the control panel by a 14-core cable. The picture below shows the connector on our module from the components view.



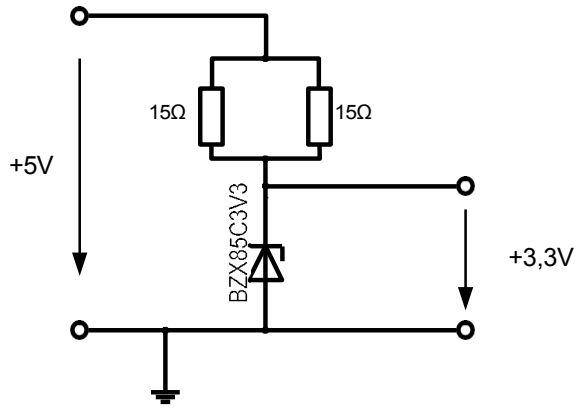
Preview of the WSL 14G mounted on the board from a component perspective

PIN	DESCRIPTION
1	GND
2	+5V
5	RX dsPIC, RX control panel
7	TX dsPIC, TX control panel
11	GND

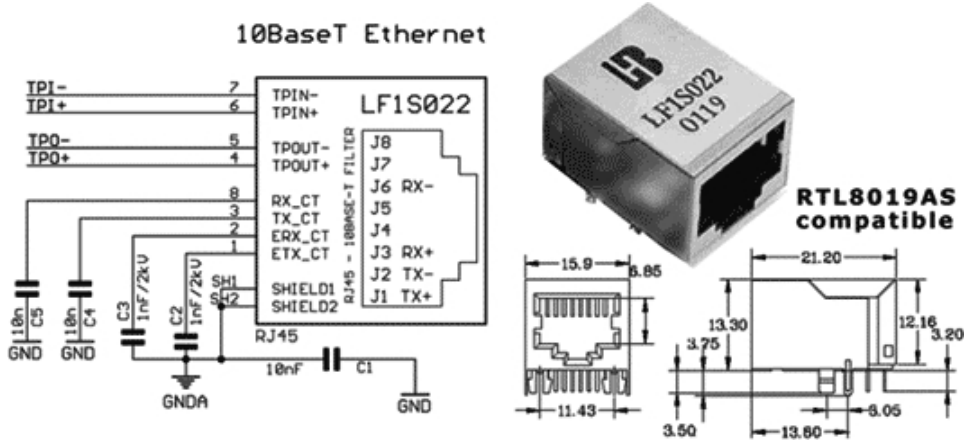
Note: :

1) Pin 1, i.e. GND, is connected to pin 11.
2) In the table you can see that there was no crossing of paths as there should have been (i.e. TX dsPIC, RX PBX). The reduction solves this by swapping wires. I.e. RX to TX.

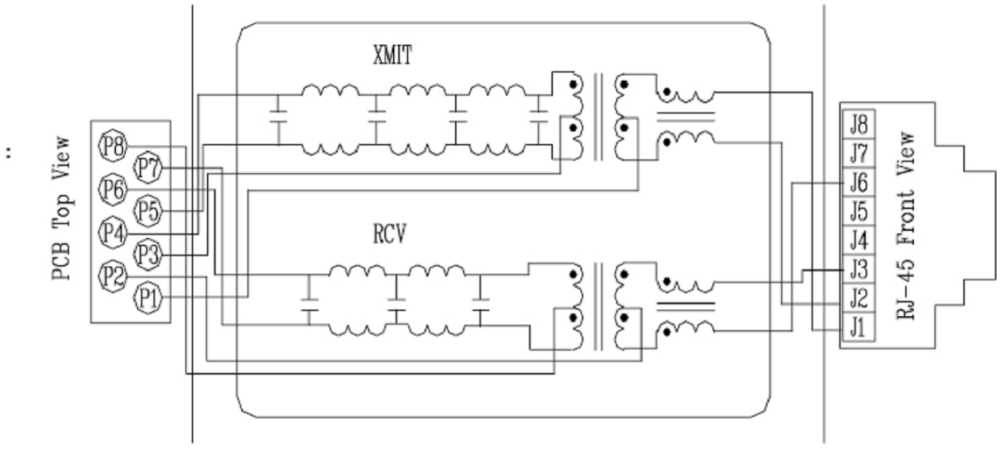
8.4 5V to 3.3V power supply



8.5 LF1S022



When selecting the transformer and connector, we opted for integrity and chose the LF1S022, a decoupling signal transformer for Ethernet 10Base-T in an RJ-45 connector.



Internal wiring diagram

9 Conclusion

The result of our long-term graduation work is a fully functional hardware module for the Ateus Omega control panel. By constructing this product, we have honed a number of skills including: programming in Assembler, circuit board fabrication, monitoring and applying packet communication protocols in practice, and teamwork. We learned how to solve problems related to product implementation. A possible future improvement could be to reduce the power dissipation of the voltage regulator. Finally, we would like to thank Professor Ing. T. Kubalík for his support and help
at a time when the situation seemed unsolvable.

10 References and citations:

Literature:

Šmrha, P. - Rudolf, V.: Internetworking using TCP/IP edition České Budějovice, Kopp 1997, 126 p.

Catalogue sheets-pdf:

Circuits: DSP30F3013, ENC28J60Microchip

LF1S022

Serial communication with Ateus Omega control panel 2N

Internet links:

http://cs.wikipedia.org/wiki/TCP/IP TCP/IP protocols

http://www.cs.vsb.cz/grygarek/LAN/sem/sercomm.html serial transmission

http://cs.wikipedia.org/wiki/Cyklick%C3%BD_redundantn%C3%ADsou%C4%8Det CRC

11 Software used

MPLAB IDE v. 7.42

ASIX UP v. 2.7

EAGLE 4.16 Professional

EAGLE 3D ulp module for EAGLE 4.16

Professional POV-Ray v. 3.6

Visual Basic 6

Wireshark

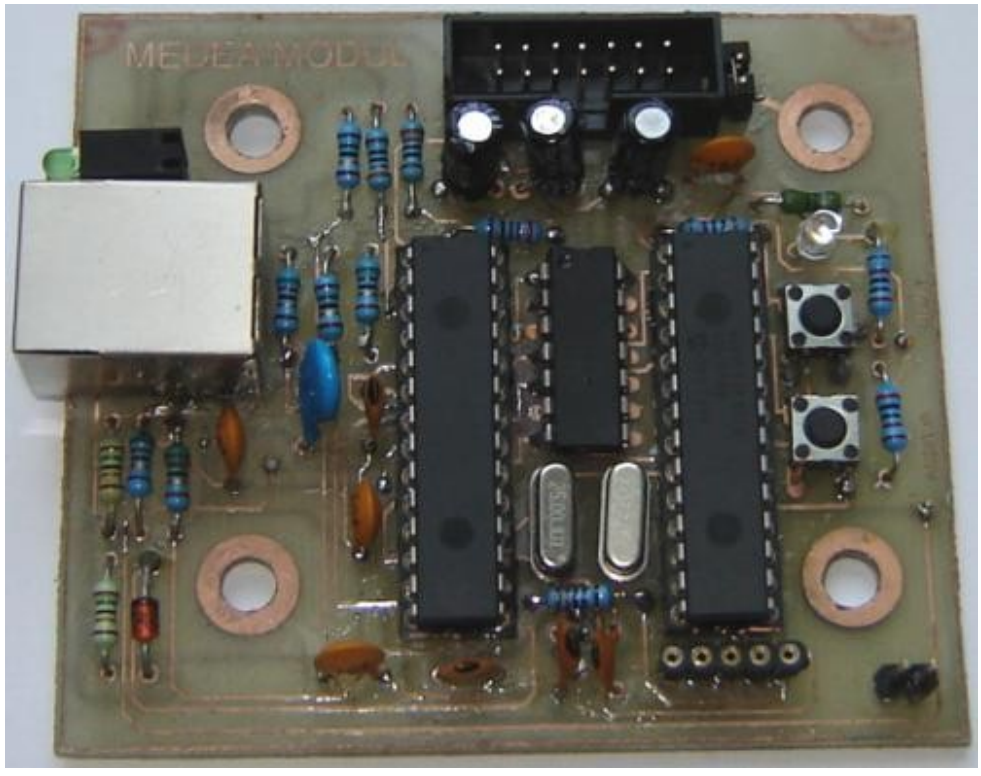
Terminal

IrfanView

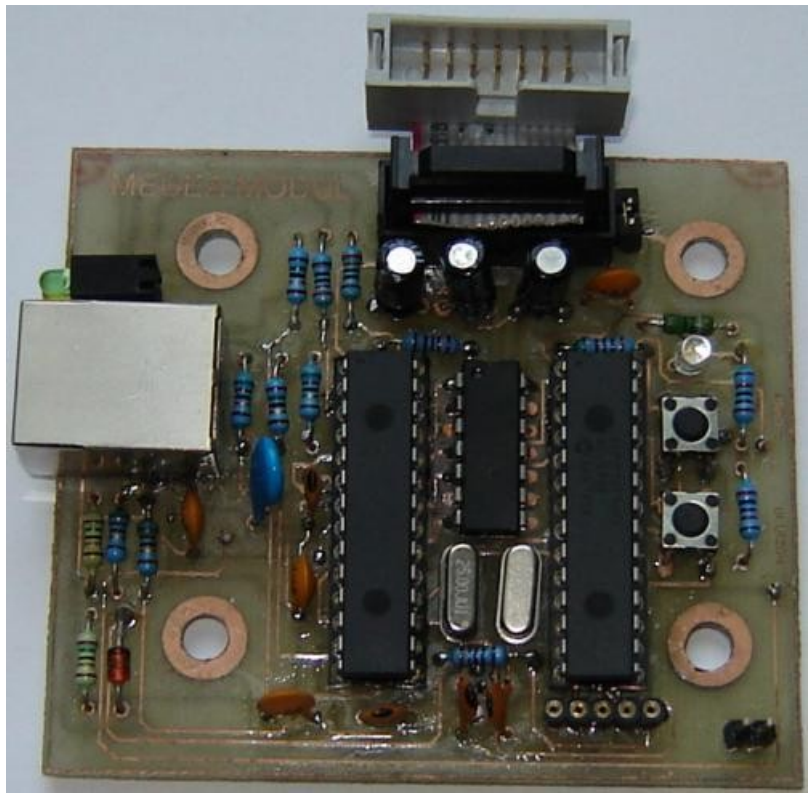
microsoft software package => Word, Visio, PowerPoint

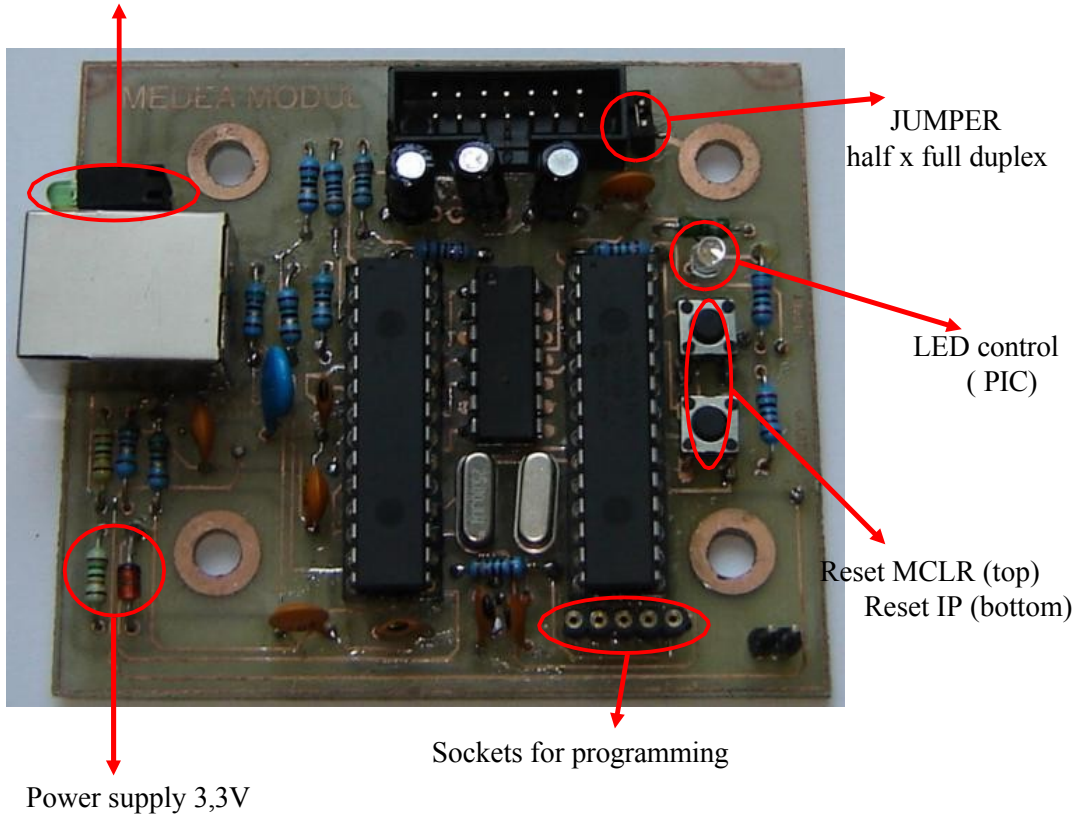
12 Attachments

12.1 Photos of MEDEA MODULE



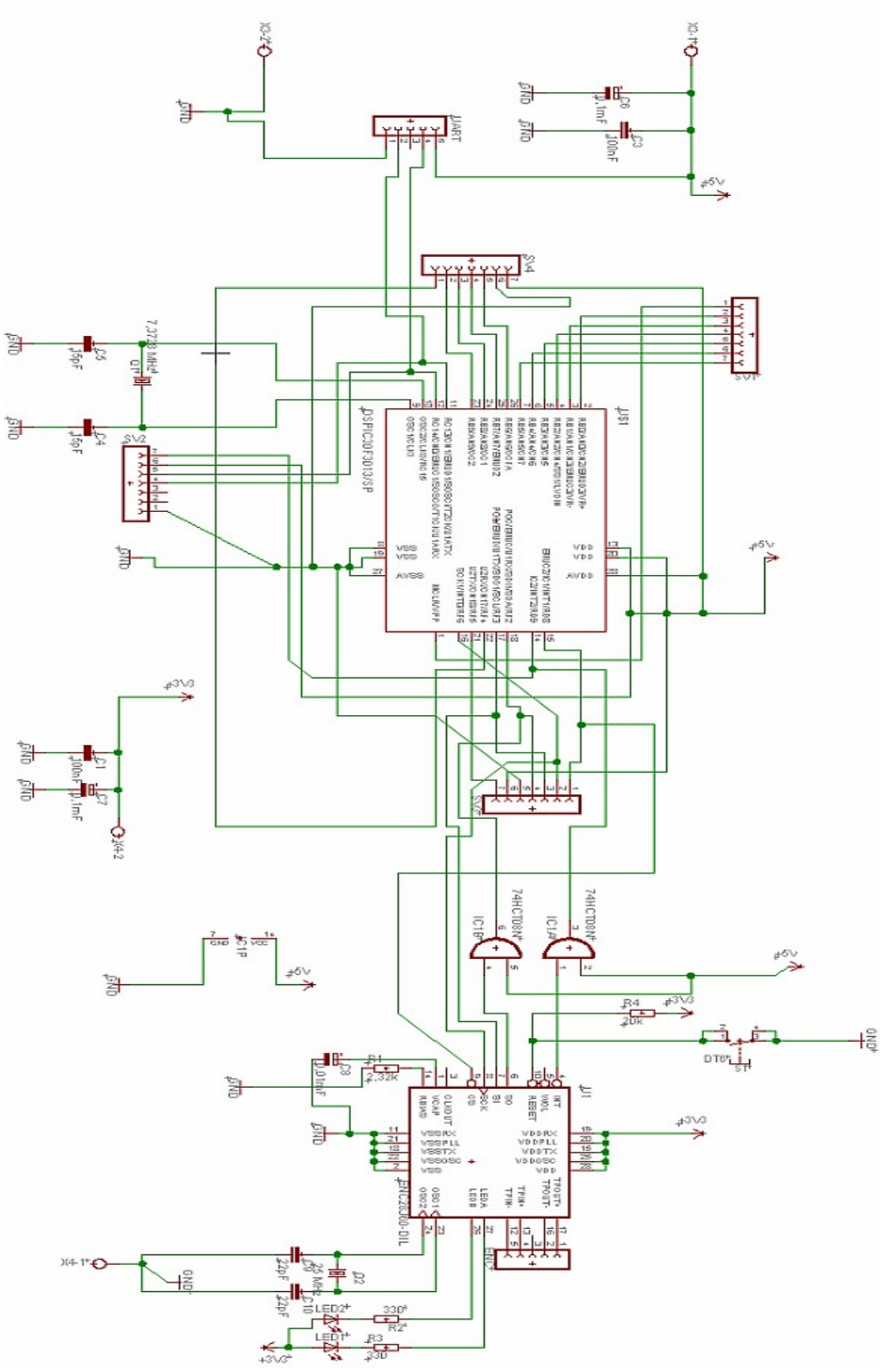
Reduction to MEDEA MODULE



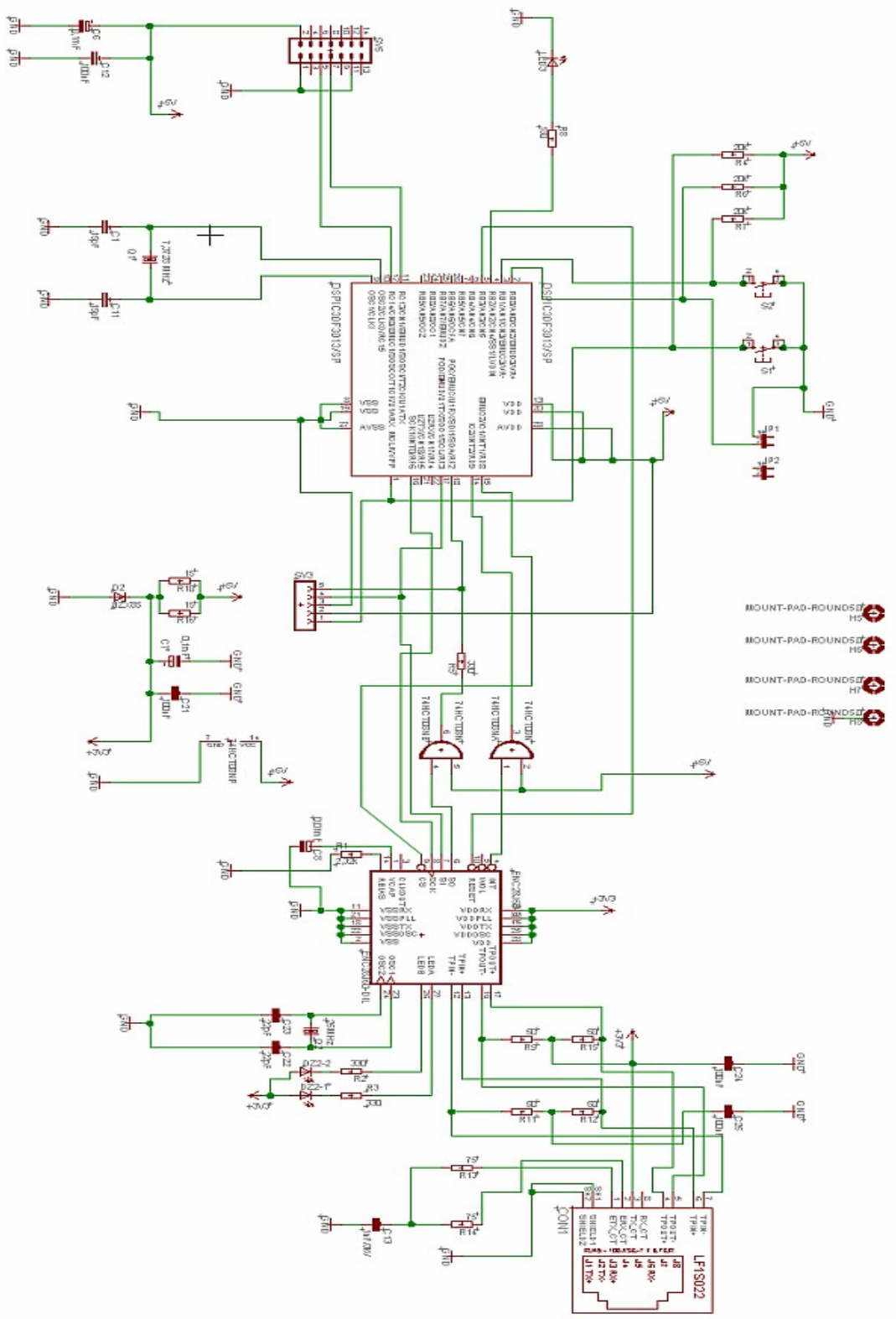

Diode column
JUMPER
half x full duplex
LED control
(PIC)
Reset MCLR (top)
Reset IP (bottom)
Sockets for programming
Power supply 3,3V

12.2 Plate diagrams and designs

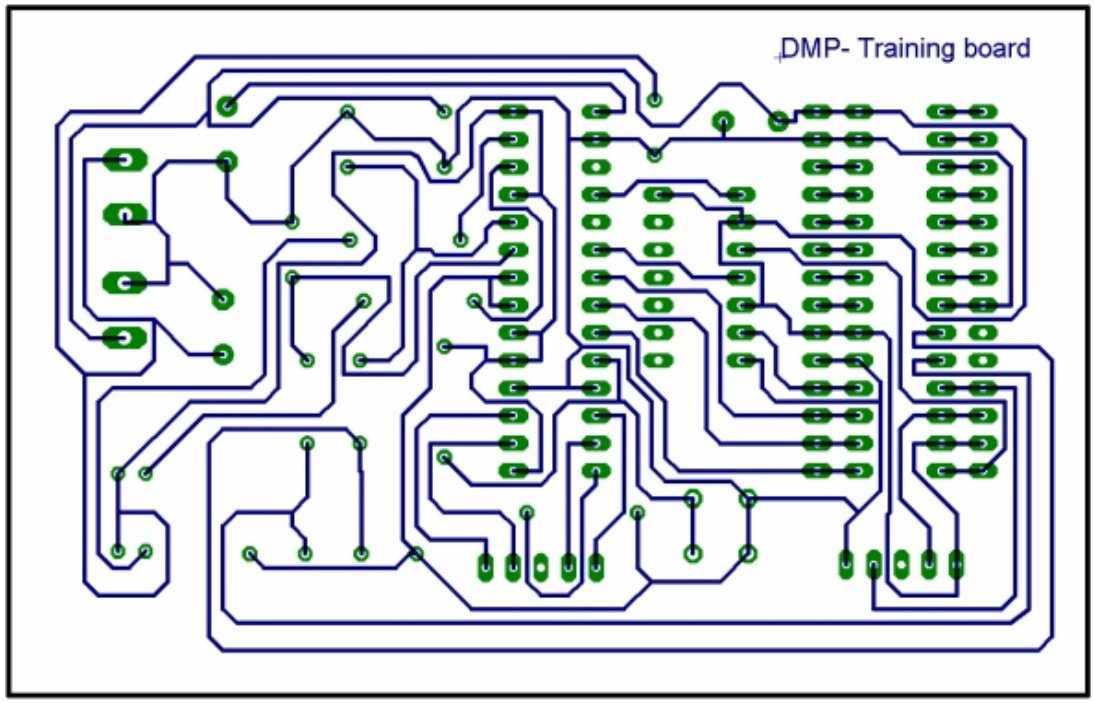
12.2.1 SCHEME - TRAINING BOARD



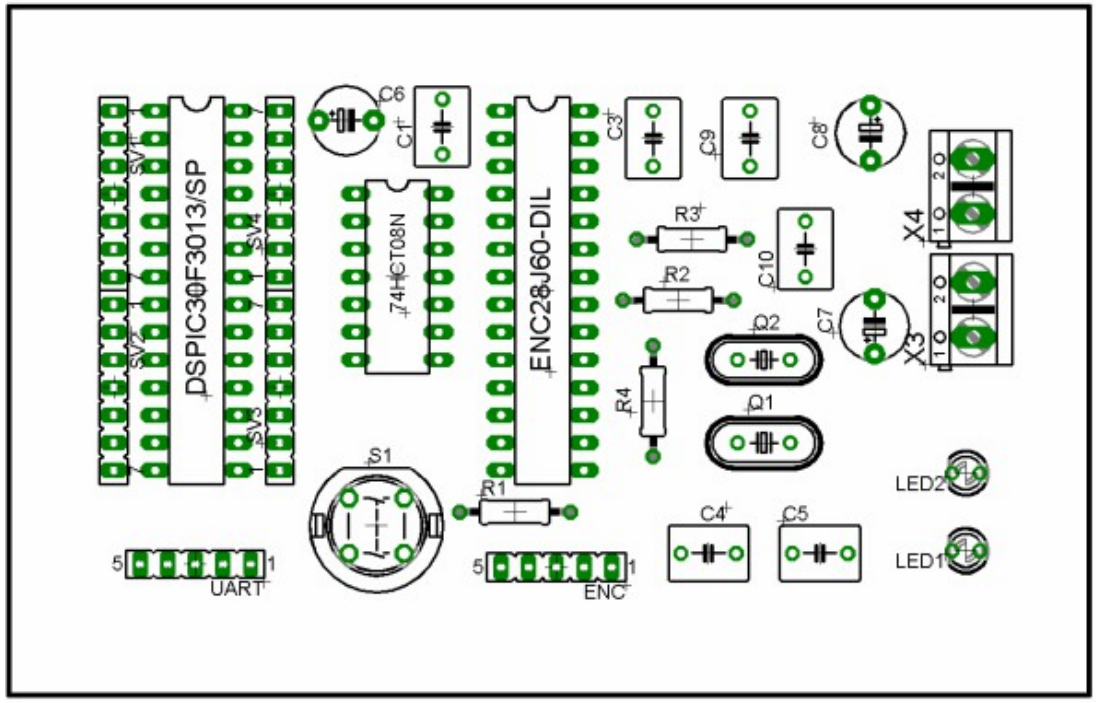
12.2.2 DIAGRAM - MEDEA MODULE



12.2.3 PRINTED CIRCUIT BOARDS - TRAINING BOARD

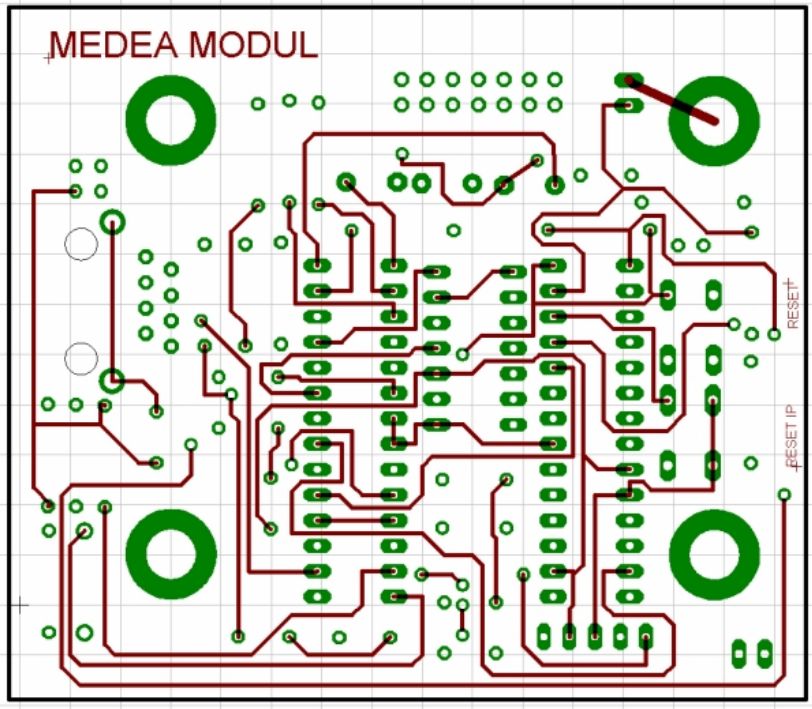


Preview of the PCB on the BOTTOM Training Board layer

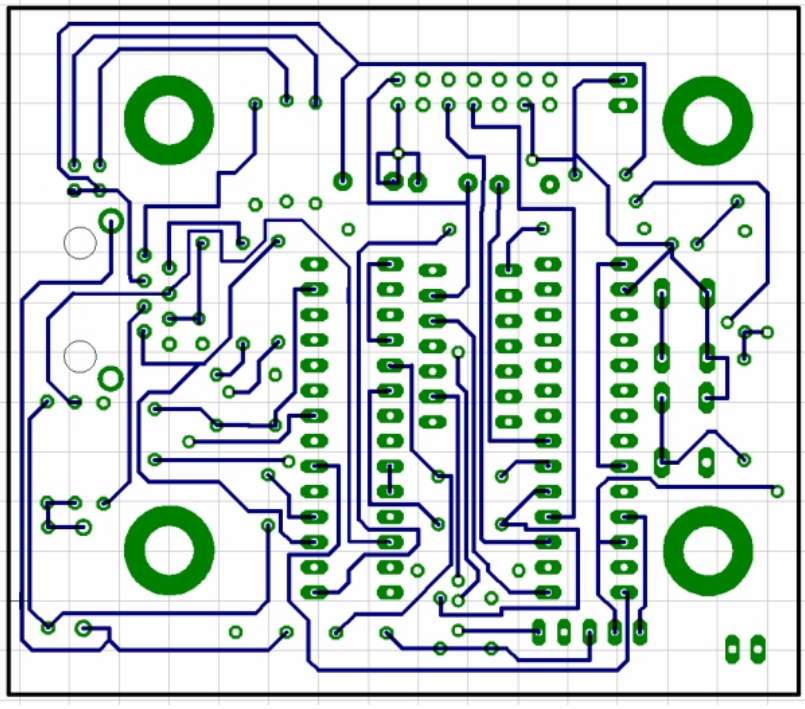


Placement of components on the Training Board

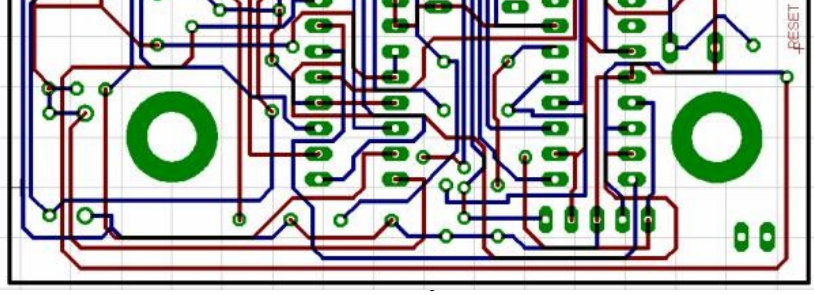
12.2.4 PRINTED CIRCUIT BOARDS - MEDEA MODULE



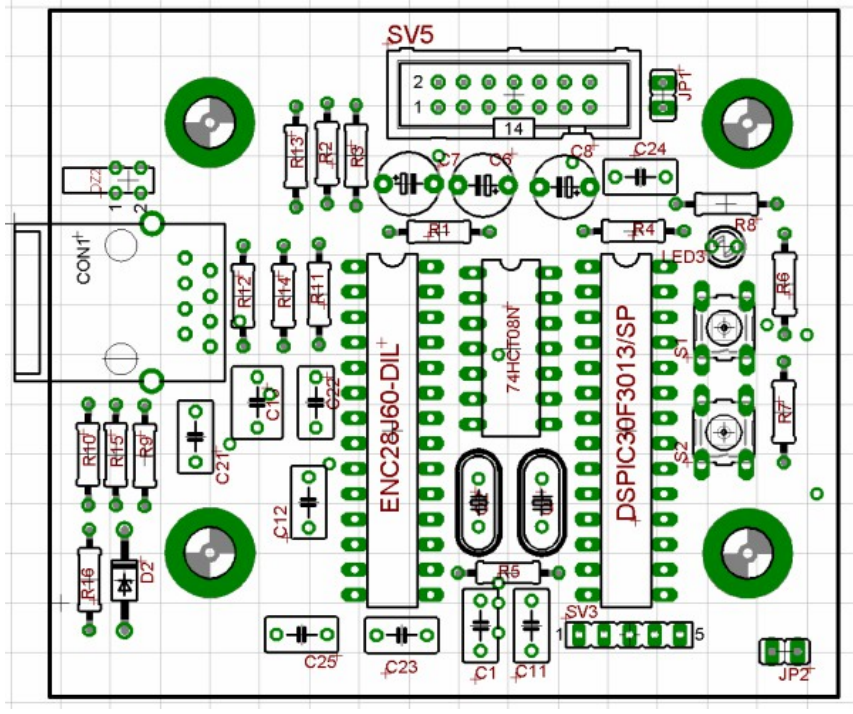
PCB preview of the TOP Medea module layer (from the components view)



Printed circuit board view of the BOTTOM Medea module layer (from the components view)



Preview of the PCB on TOP and BOTTOM layer at the same time (from the view of the semi-particles)



Component layout on the MEDEA MODUL board