

Iosoft Ltd.

PICBASIC PLUS networking

PB_UDP Software Manual

1. Introduction

This manual describes the Iosoft Ltd. PB_UDP source-code package, which provides networking functionality for the Crownhill Associates Proton-Net board, using the PICBASIC PLUS programming language.

Due to the limitations of the BASIC programming language, a full TCP/IP software stack (e.g. Embedded Web server) is not included; instead, a low-level interface employing standard UDP (User Datagram Protocol) communications is provided, that allows for simple communications over a local network or the Internet.

To demonstrate the networking capability, the source-code package includes client & server code for communication with a local PC running the Iosoft 'datagram' utility.

This manual was written for version 1 of the software. You may have been supplied with an updated version, for details see the revision header at the top of the main source file. Although the new version will have improvements and corrections, the underlying principles described in this manual will still apply.

It is impossible for us to provide a complete description of TCP/IP networking within the confines of this manual; newcomers are recommended to read an introductory book on the subject prior to tackling this challenging area, for example 'TCP/IP Lean: Web Servers for Embedded Systems' by Jeremy Bentham (2nd edition ISBN 1-57820-108-X).

For sales and support information on Iosoft products, refer to the Iosoft Ltd. Web site, www.iosoft.co.uk

2. Development environment

Development environment

To use the PB_UDP package, you will need:

1. The source files PD_UDP.BAS, UDP_DATA.INC and UDP_VARS.INC; they may have been supplied within a Zip file, in which case it must be unzipped before putting it in an appropriate directory (e.g. c:\projects\picbasic). There is a second file PB_UDP2.BAS that is only required when demonstrating board-to-board communications (see section 5).
2. A recent version of the Crownhill PICBASIC PLUS compiler. The software was tested with v1.24e, which does include support for 32-bit 'long' variables.
3. A Proton-net target board, with PIC16F877 device.
4. A PICmicro device programmer, that can load the compiled HEX file into a PIC16F877 device.
5. A PC running Windows, with an Ethernet card configured to run TCP/IP.
6. **Either** a crossover Ethernet cable **or** a hub and two straight-through Ethernet cables to link the PC to the target board. An older 10 megabit or a newer 10/100 megabit auto-switching hub may be used.
7. The Iosoft Ltd. 'datagram' network utility for the PC.
8. A straight-through serial cable to connect the target board to an unused serial port on the PC.
9. A terminal-emulator program for the PC (such as Hyperterm supplied with Windows), configured to 9600 baud, no parity, 1 stop bit.

3. Using the software

Getting started

Before looking at the structure of the software, it is worth getting it running in order to check out the development environment.

It is recommended that you set up the target board & PC on their own isolated network, rather than using an existing office network, as it simplifies the initial testing, and avoids the possibility of disrupting office communications.

In order for the PC and target board to communicate, they must not only be physically connected (via a crossover Ethernet cable, or a hub), but they must also have addresses that are in the same **domain**, i.e. must have similar addresses. By default, the PB_UDP package uses an address of 10.1.1.99, which is defined at the top of the Basic file:

```
SYMBOL MYIP1      = 10
SYMBOL MYIP2      = 1
SYMBOL MYIP3      = 1
SYMBOL MYID       = 99
```

It will also try to contact a server (your PC) at the address 10.1.1.3

```
SYMBOL REMIP1     = 10
SYMBOL REMIP2     = 1
SYMBOL REMIP3     = 1
SYMBOL REMIP4     = 3
```

To check whether your PC has a compatible address, look at the TCP/IP properties (details will vary depending on the Windows version).

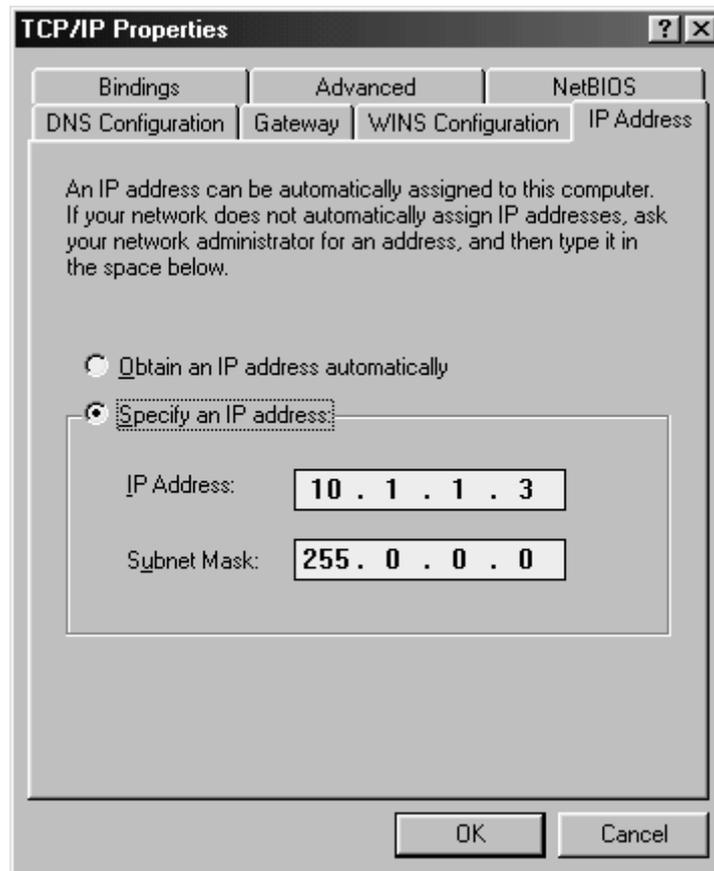


Figure 1: Windows TCP/IP properties

If the PC is set to obtain the IP address automatically, then it won't function in our simplified network, so a fixed IP address should be specified. Be warned that changing the network configuration of a PC is a non-trivial task; it is important to record the existing configuration before any changes (if in doubt, contact the System Administrator for your network before proceeding).

Alternatively, the target board configuration can be changed to reflect the PC settings; for example if the PC is at address 192.168.0.1, then change the Basic code to:

```
SYMBOL MYIP1      = 192
SYMBOL MYIP2      = 168
SYMBOL MYIP3      = 0
SYMBOL MYID       = 99
```

```
SYMBOL REMIP1     = 192
SYMBOL REMIP2     = 168
SYMBOL REMIP3     = 0
SYMBOL REMIP4     = 1
```

The addresses given throughout this manual will also need to be adjusted to match.

It is important that the target board address isn't in use by any other device (or any other target board). The easiest way to guarantee this is to ensure no other devices are physically connected to the network. When connecting to an existing network, extreme care is necessary; an address may appear to be unused (e.g. there is no response to a 'ping') but this may just be because the

device is currently powered down; when it is powered up, it will clash with the target board and cause problems. There is normally one individual (the network administrator) who is responsible for allocating addresses on the network, and he/she should be consulted prior to adding any new devices.

Programming the PICmicro

The PB_UDP source file should be loaded into the compiler and compiled, and the resulting PB_UDP.HEX file programmed into a PIC16F877. There are various types of programming adaptor, so details cannot be given here, but it is important to check the configuration fuse settings before programming the device, namely:

- The watchdog must be disabled.
- Low-voltage programming should be disabled
- The crystal type must be HS (high speed)
- In-circuit debugging must be disabled

The programmed device should be inserted into the target board, and the serial and network cables connected. A terminal emulator running on the PC, set to 9600 baud, no parity, 1 stop bit. On power-up, something similar to the following should be observed:

```
PB_UDP
My IP 10.1.1.99, host 10.1.1.3, gate 10.1.1.100
Tx ARP request
Rx len 60 pcol 806 ARP resp
Tx time request
Rx len 70 pcol 800 IP ICMP Destination unreachable
Tx time request
Rx len 70 pcol 800 IP ICMP Destination unreachable
```

All three network indicators on the target board should be lit, the Tx and Rx LEDs flashing every second, and the system LED toggling on or off every second.

Other possible indications are:

- **No activity: system LED doesn't flash.** The PIC16F877 device programming and/or configuration fuse settings are probably incorrect.
- **The 'link status' LED doesn't light.** The target board Ethernet connection is faulty; the cable may be of the wrong type (crossed vs. uncrossed). If using a hub, check that the corresponding link status LED for the port you are using; communication can only take place if the status LEDs at both ends of the link are illuminated.
- **The serial link only shows 'Tx ARP request' every second.** The target board is trying to contact the PC, but is receiving no response. Check the target board addresses as printed out on start-up, and also the PC address.

If you are in doubt about the PC network configuration, run the utility WINIPCFG, select the network adaptor you are using, and you should see a display similar to the following:

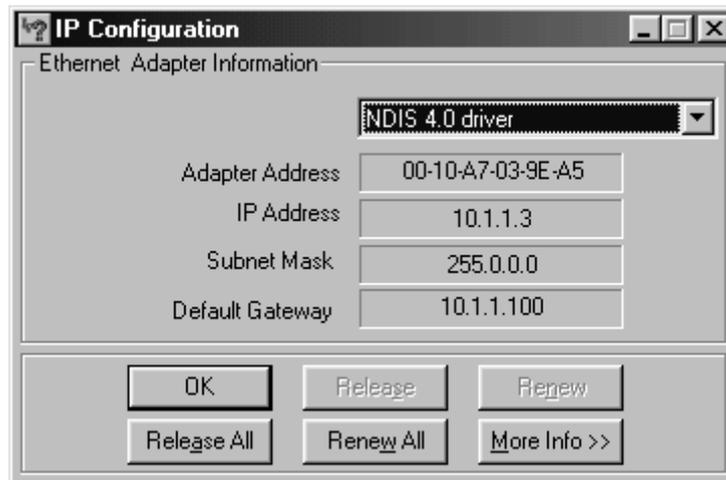


Figure 2: WINIPCFG display

PC time server: the DATAGRAM utility

The target board is contacting the PC every second in an attempt to find out the current time; the 'destination unreachable' message indicates that the PC has received the request, but doesn't know how to respond to it. To give the PC the capability to act as a time server, the Iosoft DATAGRAM utility must be run on the PC. On start-up, the utility presents a small control window, divided into 4 main areas:

- **Local UDP server.** This controls which of 3 services the utility will offer to the network.
- **Destination address.** The utility can initiate network communications, and this area gives the destination address and port number. There is also a timer value which, if non-zero, will cause the communication to be repeated.
- **Transmit UDP data.** The data to be transmitted, in text and hexadecimal notation.
- **Receive UDP data.** Any responses to the outgoing network messages.

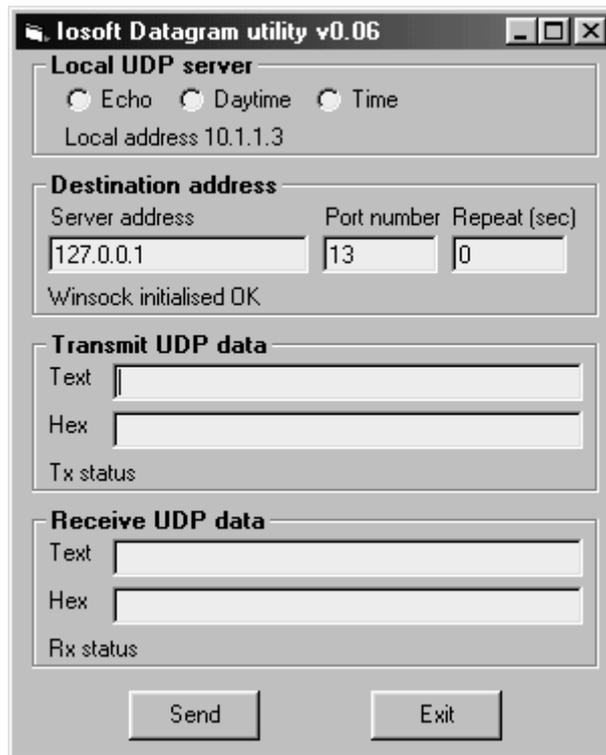


Figure 3: The Iosoft DATAGRAM utility

To enable server operation, it is only necessary to click one of the radio buttons in the top area, we require the 'time' service, so the right-hand button is clicked. The UDP server display briefly changes to 'listening on port 37', then to a count of request that changes every second, e.g. '17 Rx, last from 10.1.1.99 port 1024 len 0'. This confirms that the PC is receiving the time requests, and is returning the time value; the target serial link shows:

```
Rx len 60 pcol 800 IP UDP time
Tx time request
Rx len 60 pcol 800 IP UDP time
Tx time request
```

..and so on, updated every second. The time value received from the PC is displayed on the top right-hand side of the target board LCD, in 24-hour notation, e.g.

```
15:11:04
```

This shows how a simple target board can fetch the current time over the network, instead of having its own real-time clock chip with battery backup, and a user interface to set the time. At a network level, it demonstrates how the client software on the target board can contact the server on the PC, and receive a response.

Target board server: message handler

The target board software also has the ability to process incoming messages, and transmit a reply. To demonstrate this, the DATAGRAM utility is used to send a message to the target board, and display the response.

The datagram utility is set up as follows:

- The server address is set to the target board address 10.1.1.99
- The server port number is set to 4321
- The transmit data is set to the desired text or hex values
- The 'send' button is clicked.

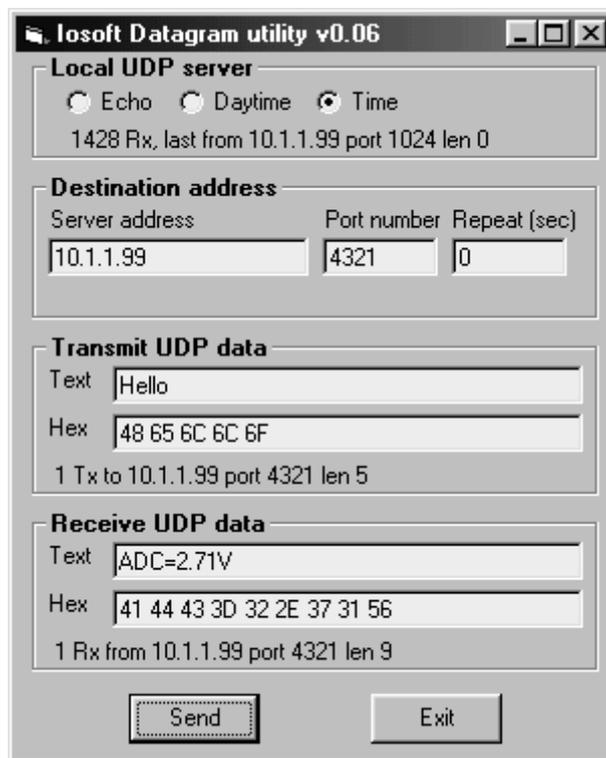


Figure 4: Sending message to target board

The transmitted text ('Hello') appears on the lower line of the target LCD display, and the datagram utility shows a response that is the current voltage measured on the first ADC channel (2.71 volts), which is set by a potentiometer on the board. To see the value change, enter '1' in the 'repeat' box, click 'send', and the voltage value will be redisplayed every second.

Ping diagnostics

The 'ping' utility is often used to diagnose network problems, and the PB_UDP software can respond to pings. Open up a DOS box on the PC, and enter 'ping' plus the target address:

```
C:\WINDOWS>ping 10.1.1.99
Pinging 10.1.1.99 with 32 bytes of data:

Reply from 10.1.1.99: bytes=32 time=35ms TTL=128
Reply from 10.1.1.99: bytes=32 time=34ms TTL=128
Reply from 10.1.1.99: bytes=32 time=34ms TTL=128
Reply from 10.1.1.99: bytes=32 time=34ms TTL=128

Ping statistics for 10.1.1.99:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 34ms, Maximum = 35ms, Average = 34ms
```

Due to buffer size constraints, the maximum ping data size the PB_UDP software can handle is 32 bytes.

Using a gateway

So far, all communications have been restricted to a small desk-top network, but the PB_UDP software has the ability to send & receive messages across larger network, including the Internet.

To do this, it is programmed with the address of a 'gateway', which will forward any outgoing messages to a wider network, and return any responses. The gateway address is given at the top of the source file, by default 10.1.1.100

```
SYMBOL GWIP1      = 10
SYMBOL GWIP2      = 1
SYMBOL GWIP3      = 1
SYMBOL GWIP4      = 100
```

There is also a netmask (default 255.255.255.0), which is used to determine if the desired target address is on the local subnet (no gateway required) or on a remote network (all requests to go though the gateway).

```
SYMBOL MASKIP1    = 255
SYMBOL MASKIP2    = 255
SYMBOL MASKIP3    = 255
```

In theory, all that is necessary is to set the gateway IP address to the correct value for your gateway, the remote IP address to the desired timeserver on the Internet (e.g. the Demon Internet time server 158.152.1.76), and the PB_UDP software will automatically send the time requests out on the Internet, and show the results. In practice, setting up a connection from an office network to the Internet can be quite complicated, as there may be various security systems ('firewalls') in place to stop unwanted messages entering or leaving the site, so it is important to liase with the network administrator before attempting Internet communications.

4. Structure of the software

The PB_UDP software encompasses the following main components:

- Definitions
- Hardware initialisation
- Transmission
- Reception
- Client code
- Server code

Definitions

Extensive use is made of the SYMBOL command to define symbolic constants, for example:

```
SYMBOL LCD_LINE2      = $c0
```

Thereafter, whenever LCD_LINE2 is used the compiler substitutes a value of 0C0h:

```
b = LCD_LINE2: GOSUB LCD_cmd
```

It is convenient to be able to redefine certain byte locations as 16 or 32-bit words, and there is a compiler trick that allows this. For example, the 13th and 14th bytes of the received network frame are a 16-bit protocol value, with the most-significant byte first. The following definition allows these two bytes in the receive buffer to be accessed as if they were one word.

```
DIM rxbuff[RXBUFFLEN] AS BYTE
SYMBOL pcol_          = rxbuff#13
SYMBOL pcol_h         = rxbuff#12
DIM pcol              AS pcol_.WORD
. . .
IF pcol = PCOL_ARP THEN
. . .
```

IP ADDRESS

Symbol definitions are also used for the main IP address definitions

```
SYMBOL MYIP1         = 10
SYMBOL MYIP2         = 1
SYMBOL MYIP3         = 1
'My ID number; last byte of MAC & IP address
SYMBOL MYID          = 99

'4 bytes of remote IP address
SYMBOL REMIP1        = 10
SYMBOL REMIP2        = 1
SYMBOL REMIP3        = 1
SYMBOL REMIP4        = 3
```

```
'4 bytes of gateway IP address
SYMBOL GWIP1      = 10
SYMBOL GWIP2      = 1
SYMBOL GWIP3      = 1
SYMBOL GWIP4      = 100
```

```
'3 bytes of IP address mask
SYMBOL MASKIP1    = 255
SYMBOL MASKIP2    = 255
SYMBOL MASKIP3    = 255
```

If several PB_UDP implementations are on the same network, they must each have a unique value for 'MYID', which is used as the least-significant byte of the address.

MAC ADDRESS

In addition to having a unique IP address, the board must also have a unique hardware (MAC) address. There are various ways this could be achieved, the easiest being to use the 'locally administered' MAC address space provided by the IEEE. If the most-significant byte has a value of 2, then the remaining bytes aren't IEEE-administered, so can be any value we choose (so long as they don't happen to coincide with those of someone else's choosing). The 'MYID' value is used as the least-significant byte of the MAC address; providing it has been set to a unique value for each board on the network, then no two boards will have the same MAC address.

Hardware initialisation

To set up the Ethernet Network Interface Controller (NIC), a large number of hardware registers have to be initialised with specific data values. This has been implemented using DATA statements, with pairs of values representing the register and the byte value, for example

```
SYMBOL RBCR0      = $0a
SYMBOL RBCR1      = $0b
SYMBOL RCR        = $0c
SYMBOL TCR        = $0d
SYMBOL DCR        = $0e
. . .
DATA DCR,         $48          'Data config
DATA RBCR0,       $00          'Clear RBCR
DATA RBCR1,       $00
DATA RCR,         $20          'Rx monitor mode
```

In essence, the software has to fetch each pair of values, and put the second into the register specified by the first:

```
SYMBOL NICADDR    = PORTB
SYMBOL NICDATA    = PORTD
SYMBOL NIC_IOR    = PORTE.0
SYMBOL NIC_IOW    = PORTE.1
SYMBOL NIC_RST    = PORTE.2
. . .
REPEAT
    READ a: READ b
    NICADDR = a: NICDATA = b: GOSUB out_nic
UNTIL a >= T_STOP
```

The out_nic subroutine asserts the write-enable line, so that the data value is written to the required address.

```
'Do a NIC write cycle
out_nic:
    OUTPUT PORTD
    NIC_IOW = 0
    DELAYUS 3
    NIC_IOW = 1
    INPUT PORTD
RETURN
```

TIME DELAY

It takes a finite time for the NIC to respond to some settings, so delays have to be inserted at specific points. This is handled by including a dummy register value (outside the normal 0-1F hex range for NIC registers) to indicate a delay is required, e.g.

```
SYMBOL T_WAIT          = $fd 'Wait x ms
. . .
DATA CMDR,             $21          'Stop, DMA abort (wait 2ms)
DATA T_WAIT,           2            'Wait 2ms
DATA DCR,              $48          'Data config
```

TASK BLOCKS

This idea of dummy register values to indicate special functions has been used extensively, for example there is a dummy value NIC_INIT to mark the start of the initialisation data, and T_STOP to mark the end. This means that the data table can contain several functional blocks, each with a defined beginning and end.

```
DATA NIC_INIT, 0          'NIC initialisation:
DATA RESPORT, $01        'Reset NIC
DATA T_WAIT, 2           'Wait 2ms
DATA CMDR, $21           'Stop, DMA abort (wait 2ms)
. . .
DATA T_STOP, 0           'End of this task

DATA TX_ARPREQ, 0        'Tx broadcast ARP request
DATA ISR, $40           'Clear remote DMA int flag
DATA RSAR0, 0           'Set remote DMA addr
DATA RSAR1, TXSTART
. . .
DATA T_STOP, 0           'End of this task
```

Each block is referred to as a 'task', and is essentially a block of coded instructions that must be executed to initiate the required action, such as setting up the hardware, or transmitting a specific message. To action a task, the following code is used:

```
task = NIC_INIT: GOSUB do_nic_task          'Initialise NIC
. . .
task = TX_ARPREQ: GOSUB do_nic_task        'Send ARP request
```

The subroutine do_nic_task searches the data table for the appropriate code block, and then executes it:

```

do_nic_task:
  IF task > NIC_INIT OR task < TASKBASE THEN RETURN
  RESTORE
  REPEAT
    READ a: READ b
    IF a = T_END THEN RETURN
  UNTIL a = task
  REPEAT
    READ a: READ b
    IF a <= RESPRT THEN NICADDR = a: NICDATA = b: GOSUB out_nic
    IF a = T_WAIT THEN DELAYMS b
    IF a>=T_VOUT1 AND a<=T_VOUT6 THEN GOSUB vout
  UNTIL a >= T_STOP
RETURN

```

Transmission

A transmission task block contains all the commands to set up the NIC, and all the data to be transmitted, for example an ARP request:

```

DATA TX_ARPREQ, 0      'Tx broadcast ARP request
DATA ISR, $40        'Clear remote DMA int flag
DATA RSAR0, 0        'Set remote DMA addr
DATA RSAR1, TXSTART
DATA CMDR, $12      'Start DMA remote write
DATA DPORT, $ff     'Broadcast addr
DATA DPORT, $ff
DATA DPORT, $ff
DATA DPORT, $ff
DATA DPORT, $ff
DATA T_VOUT6, V_LOC_MAC 'Local MAC addr
DATA DPORT, $08     'Ether pcol 0806: ARP
DATA DPORT, $06
DATA DPORT, $00     'ARP hardware type 0001: Ethernet
DATA DPORT, $01
DATA DPORT, $08     'ARP protocol type 0800: IP
DATA DPORT, $00
DATA DPORT, MACLEN  'ARP hardware addr len
DATA DPORT, IPADDRLEN 'ARP pcol addr len
DATA DPORT, $00     'ARP operation 0001: request
DATA DPORT, $01
DATA T_VOUT6, V_LOC_MAC 'Local MAC addr
DATA T_VOUT4, V_LOC_IP  'Local IP addr
DATA DPORT, $ff     'Target MAC addr: broadcast
DATA DPORT, $ff
DATA DPORT, $ff
DATA DPORT, $ff
DATA DPORT, $ff
DATA T_VOUT4, V_REMGATE_IP
DATA T_STOP, 0      'End of this task

```

After the initial register setting, the packet data is put in the NIC buffer (you may care to review the ARP format as described in 'TCP/IP Lean' chapter 3):

```

6-byte destination MAC address: all 1's (broadcast)
6-byte source MAC address: my hardware address
Protocol word: 0806h (ARP)
Hardware type word: 0001h (Ethernet)
ARP protocol word: 0800 (IP)
Hardware address length byte: 6 (Ethernet address length)
IP address length byte: 4

```

```
ARP operation word: 0001 (ARP request)
6-byte local MAC address
4-byte local IP address
6-byte broadcast MAC address
4-byte destination IP address
```

DATA VARIABLES

You will note that yet more dummy register values have been used to represent variables such as the source & destination addresses, so for example the data pair

```
DATA T_VOUT4,    V_LOC_IP
```

Indicates that a 4-byte value should be sent to the NIC data register, consisting of the local IP address. This is handled in the vout subroutine:

```
NICADDR = DPORT
IF b = V_LOC_IP THEN
    NICDATA = MYIP1: GOSUB out_nic
    NICDATA = MYIP2: GOSUB out_nic
    NICDATA = MYIP3: GOSUB out_nic
    NICDATA = MYID: GOSUB out_nic
ENDIF
```

GATEWAY

The data variable V_REMGATE_IP represents the destination IP address, and has an additional feature; if the destination is outside the network, the ARP request will be sent to the gateway, instead of the true destination. The destination address is XORed with the local address and then masked using the netmask value to decide if the gateway IP or remote IP is to be used.

```
a = ((REMIP1^MYIP1)&MASKIP1) | ((REMIP2^MYIP2)&MASKIP2) |
((REMIP3^MYIP3)&MASKIP3)
IF b = V_REMGATE_IP AND a <> 0 THEN
    NICDATA = GWIP1: GOSUB out_nic
    NICDATA = GWIP2: GOSUB out_nic
    NICDATA = GWIP3: GOSUB out_nic
    NICDATA = GWIP4: GOSUB out_nic
    RETURN
ENDIF
IF b = V_REM_IP OR b = V_REMGATE_IP THEN
    i = 0
    REPEAT
        NICDATA = rem_ip[i]: GOSUB out_nic: INC I
    UNTIL I > IPADDRLEN-1:
ENDIF
```

LENGTH CALCULATION

It is necessary to know in advance how much data is being sent to the NIC, in order to set the registers appropriately. A subroutine get_dataalen finds a given task, computes the amount of data it contains, and then sets the NIC length registers accordingly.

```

get_datalen:
  IF task > NIC_INIT OR task < TASKBASE THEN RETURN
  RESTORE
  REPEAT                                'Search for task marker
    READ a: READ b
    IF a = T_END THEN RETURN
  UNTIL a = task
  datalen = 0
  REPEAT                                'Tally data lengths
    READ a: READ b
    IF a = DPORT THEN datalen = datalen + 1
    IF a = T_VOUT2 THEN datalen = datalen + 2
    IF a = T_VOUT4 THEN datalen = datalen + 4
    IF a = T_VOUT6 THEN datalen = datalen + 6
  UNTIL a >= T_STOP
  NICADDR = RBCR0: NICDATA = datalen: GOSUB out_nic
RETURN

```

CLIENT/SERVER REMOTE/RESPONSE ADDRESSES

The PB_UDP code must handle two types of transaction:

CLIENT: initiate a request and wait for a response

SERVER: wait for a request, and send a response

As supplied, the software has two interlinked clients:

ARP CLIENT: requests the MAC address of the time server (or of the gateway, if the server isn't on the local network).

TIME CLIENT: once the MAC address is found, the time is requested at one-second intervals.

There are also three servers:

ARP SERVER: responds to ARP requests for the local IP address.

PING SERVER: responds to pings (ICMP echo requests)

MESSAGE SERVER: accepts message for display on the LCD, returns the current ADC voltage value.

The important point is that the PB_UDP software has no control over the timing of any incoming messages; if the time client has just requested the time value, there is no guarantee that the next message received will be the time response; it could be an ARP or ping request instead. Due to the limited storage space, all messages must be acted upon as soon as they arrive, so the ARP or ping response must be sent even though a time response is expected.

For this reason, two types of destination addresses are used: a **response** address and a **remote** address. The former is used by the server for all its replies, and is just a copy of the incoming source address (i.e. it is used for 'return to sender' replies), while the remote address is a fixed value corresponding to, say, the time server address.

COPYING SOURCE DATA

A frequent requirement is to copy data from an incoming message to the outgoing message. An instance of this is the ARP server, where the original source IP address has to be copied into the outgoing destination IP address field. This is achieved by using the T_VOUT4 4-byte variable definition with the offset of the original source address within the receive buffer:

```
DATA T_VOUT4, 28 'Target IP addr
```

This copies 4 bytes from the receive buffer location 28 to the transmit buffer in the NIC. The same technique can be used with other data sizes, for example copying an incoming 2-byte UDP port number into the outgoing data

```
DATA T_VOUT2, 34 'Destination port (= incoming source port)
```

UDP MESSAGE RESPONSE

The most complex transmission is the UDP message response, as it must conform to UDP & IP formatting, and also include a dynamically-changing voltage value.

```
DATA TX_MSGRSP, 0 'Tx UDP message response
DATA RSAR0, 0 'Set remote DMA addr
DATA RSAR1, TXSTART
DATA CMDR, $12 'Start DMA remote write
DATA T_VOUT6, V_RESP_MAC 'Send to response MAC addr
DATA T_VOUT6, V_LOC_MAC 'Local MAC addr
DATA DPORT, $08 'IP protocol
DATA DPORT, $00
DATA DPORT, $45 'IP v4, header len 20 bytes
DATA DPORT, 0 'Normal service
DATA T_VOUT2, V_IPLLEN 'Length of IP datagram
DATA DPORT, 0 'Identifier
DATA DPORT, 1
DATA DPORT, 0 'No fragmentation
DATA DPORT, 128 'Time to live
DATA DPORT, 17 'Protocol UDP
DATA DPORT, $0 'Dummy checksum
DATA DPORT, $0
DATA T_VOUT4, V_LOC_IP 'Source IP addr
DATA T_VOUT4, 26 'Dest IP addr
DATA DPORT, MSGPORT/256 'Source port
DATA DPORT, MSGPORT//256
DATA T_VOUT2, 34 'Dest port (= incoming source port)
DATA T_VOUT2, V_UDPLEN 'UDP length (header + data)
DATA DPORT, 0 'Disable checksum
DATA DPORT, 0
DATA DPORT, "A" 'UDP data: voltage value
DATA DPORT, "D"
DATA DPORT, "C"
DATA DPORT, "="
DATA T_VOUT4, V_VOLTAGE
DATA DPORT, "V"
DATA T_STOP, 0 'End of this task
```

The points to note are:

Dynamic data. A special variable value V_VOLTAGE has been defined, that returns a 4-character string containing the ADC value in volts, e.g. "1.23".

Data length. The IP and UDP data length has to be computed before executing this task, as follows

```
task = TX_MSGRSP: GOSUB get_dataLEN 'Get response length
iplen = dataLEN - MACHDR_LEN      'Calc IP & UDP len
udplen = iplen - IPHDR_LEN
```

Checksum. The UDP checksum is set to zero to disable it, but the IP checksum can not be disabled. A subroutine is called to compute the checksum just prior to transmission:

```
GOSUB do_nic_task
GOSUB calc_tx_ip_csum: GOSUB transmit
```

Port number. An arbitrary port number has been chosen for the message server:

```
'Port to be used for incoming messages
SYMBOL MSGPORT = 4321
```

Reception

The subroutine 'receive' is used to check the Network Interface Controller (NIC) to see if any frames have been received. The NIC can store up to 6KB of incoming data in its own circular buffer, so it isn't necessary to use interrupts to signal data arrival.

The process by which the data is extracted from the NIC packet buffer is known as 'remote DMA', and is too complex to discuss here; if you need more information, see the National Semiconductor DP8390 family data sheets, since this device is the heart of the NE2000-compatible network controller.

A relatively small buffer is used to store the incoming data, due to the severe RAM constraints of the PIC16.

```
'Size of receive buffer
SYMBOL RXBUFFLEN= 74
DIM rxbuff[RXBUFFLEN] AS BYTE
```

For an incoming UDP packet, the sizes are:

Ethernet header:	14 bytes (2 x 6-byte MAC addresses, protocol word)
IP header:	20 bytes
UDP header:	8 bytes
UDP data:	Up to 32 bytes

A much larger buffer size could be used on a PIC18xxx processor, though a buffer larger than 255 bytes would necessitate some code modifications, as single-byte length values have been used in various places.

If a frame has been received, it is copied into the receive buffer, and the 'rxlen' variable is set to the length; if the frame exceeds the buffer size it is truncated. The frame is then analysed using the check_rx subroutine.

```

GOSUB receive
IF rxlen > 0 THEN
    HRSOUT "Rx len ", DEC rxlen, " pcol ", HEX pcol, " "
    GOSUB check_rx
    HRSOUT $d, $a
ENDIF

```

The check_rx subroutine has to classify the incoming frame, which may be ARP (request or response), IP ICMP (ping request) or IP UDP (message request or time response). If the frame is unrecognised, it is discarded.

During packet reception and decoding, a significant amount of information is printed out to the serial link. Whilst this may be very useful for initial testing, it does significantly increase the response time, so it is recommended that the diagnostic printouts be removed when developing a real-world application.

ARP HANDLER

The ARP message handler must check that the destination IP address matches the local IP address, then branch to handle a request (from a remote host) or a response (to a request we have sent). In the latter case, the remote MAC address is stored for later use, and a flag is set to show the ARP cycle is complete.

```

IF pcol = PCOL_ARP THEN
    idx = ETHLEN + 24 : GOSUB match_myip
    IF matched = 1 THEN
        IF arp_op = $0001 THEN
            'ARP request
            HRSOUT "ARP req"
            txlen = 0
            'Send ARP response
            task = TX_ARPRSP
            GOSUB get_dataalen: GOSUB do_nic_task
            GOSUB transmit
        ENDIF
        IF arp_op = $0002 THEN
            'ARP response
            idx = ETHLEN + 14:
            GOSUB match_remip: GOSUB match_gwip
            IF matched = 1 THEN
                HRSOUT "ARP resp"
                'Get MAC address
                idx = ETHLEN + 8: GOSUB get_remmac
                arped = 1
            ENDIF
        ENDIF
    ENDIF
ENDIF

```

IP HANDLER

An incoming IP message (known as a 'datagram') is checked to ensure the destination IP address matches the local address, then is split into ICMP or UDP handlers.

```

IF pcol = PCOL_IP THEN                                'IP protocol
  idx = 30 : GOSUB match_myip                          'Match my address
  IF matched = 1 THEN
    HRSOUT "IP "
    IF ip_pcol = 1 THEN                                'ICMP protocol
      HRSOUT "ICMP "
      . . .
    ENDIF
    IF ip_pcol = 17 THEN                               'UDP datagram
      HRSOUT "UDP "
      . . .
    ENDIF
  ENDIF
ENDIF
ENDIF

```

ICMP HANDLER

The main ICMP message we'll receive is an Echo Request (ping request), where the incoming data must be echoed back. The Destination Unreachable message is also displayed, since this will be received if we try to access a closed UDP port (i.e. the PC we're contacting isn't running a time server application).

```

IF ip_pcol = 1 THEN                                  'ICMP protocol
  HRSOUT "ICMP "
  IF icmp_type = 8 AND rxiplen < 256 THEN
    iplen = rxiplen                                  'ICMP echo request
    task = TX_ICMPRSP: GOSUB get_dataalen: GOSUB do_nic_task
    icmp_type = 0                                     'Send echo response
    icmp_csum = icmp_csum + 8                         'Adjust checksum
    IF icmp_csum < 8 THEN icmp_csum = icmp_csum + 1
    NICADDR = RBCR0: NICDATA = iplen-20: GOSUB out_nic
    NICADDR = RBCR1: NICDATA = 0 : GOSUB out_nic
    NICADDR = CMDR: NICDATA = $12 : GOSUB out_nic
    NICADDR = DPORT                                  'Copy header & data
    FOR i=0 TO iplen-21
      NICDATA = rxbuff[i+34]: GOSUB out_nic
    NEXT
    GOSUB calc_tx_ip_csum                             'Do IP checksum
    GOSUB transmit                                   'Send the frame
  ENDIF
  IF icmp_type = 3 THEN                               'ICMP dest unreachable
    HRSOUT "Destination unreachable"
  ENDIF
ENDIF
ENDIF
ENDIF

```

UDP HANDLER

The port numbers are used to determine the type of incoming UDP message (confusingly, also known as a 'datagram'). If the source port is a time server, then it is a time response, while if the destination port is my message port, it is a text message.

```

IF rx_srce_port = 37
    HRSOUT "time " 'Response to time request
    . . .
ENDIF
IF rx_dest_port = 4321
    HRSOUT "message " 'Incoming text message
    . . .
ENDIF

```

UDP TIME HANDLER

An incoming time message consists of the a 32-bit value containing the number of seconds since January 1 1900, with the most-significant byte first. We're only interested in the time, which can be extracted using simple modulo arithmetic, and displayed on the LCD. A minor complication is that the compiler treats 32-bit values as **signed** integers, and the current time value is large (C1C964DC hex at the time of writing), which will look like a negative value. To correct this, an integer number of days (24,000) is subtracted to make the seconds count positive without altering the time value.

```

HRSOUT "time " 'Response to time request
rx_udp_dw = rx_udp_dw - $7b98a000 'Subtract days to make +ve
s = rx_udp_dw // 60 'Get hrs, min, sec
rx_udp_dw = rx_udp_dw / 60
m = rx_udp_dw // 60
rx_udp_dw = rx_udp_dw / 60
h = rx_udp_dw // 24
b = LCD_TIMEPOS: GOSUB LCD_cmd 'Display on top LCD line
a = h: GOSUB disp_dec2
b = $3a: GOSUB disp_char
a = m: GOSUB disp_dec2
b = $3a: GOSUB disp_char
a = s: GOSUB disp_dec2

```

UDP MESSAGE HANDLER

An incoming message to UDP port 4321 is displayed on the 2nd line of the LCD

```

idx = UDP_DATA_START 'Display on 2nd LCD line
b = LCD_LINE2: GOSUB LCD_cmd
FOR i=0 to 15 '16 chars on LCD
    b = rxbuff[idx]
    IF idx<rxlen AND b>0 THEN 'Maybe fewer in message
        GOSUB disp_char
        idx = idx + 1
    ELSE
        b = $20: GOSUB disp_char
    ENDIF
NEXT 'Respond with ADC value..

```

A response is sent, giving the current voltage on ADC channel 0. The code is relatively simple, since the outgoing message is pre-formatted using DATA statements, as discussed earlier.

```

centivolts = adcval ** 32000 'Get ADC volts / 100
task = TX_MSGRSP: GOSUB get_dataLen 'Get response length
iplen = dataLen - MACHDR_LEN 'Calc IP & UDP len
udplen = iplen - IPHDR_LEN
GOSUB do_nic_task 'Send response
GOSUB calc_tx_ip_csum: GOSUB transmit

```

5. Communication between target systems

So far, all communication has been between a target board and the PC; to demonstrate board-to-board communications, a second file PB_UDP2.C has been included in the package. It is essentially the same as PB_UDP.C, with minor modifications:

- The IP address has been changed from 10.1.1.99 to 10.1.1.98. This is essential to avoid an IP address clash between the boards
- The remote IP address has been changed from 10.1.1.3 to 10.1.1.99. Instead of requesting data from a PC, the second board will request data from the first board.
- The remote port number has been changed from 37 to 4321. Instead of making time requests, the second board will be sending message requests to the first board, which will return its ADC value.

These changes are sufficient for the second board to poll the first, obtaining its ADC value every second. The UDP message handler also been modified to handle the new incoming messages:

```
IF rx_srce_port = MSGPORT
  HRSOUT "message resp "           'Incoming message response
  idx = UDP_DATA_START           'Display on 2nd LCD line
  b = LCD_LINE2: GOSUB LCD_cmd
  i = 0: REPEAT                  '16 chars on LCD
    b = rxbuff[idx]
    IF idx<rxlen AND b>0 THEN    'Maybe fewer in message
      PRINT b
      INC idx
    ELSE
      PRINT " "
    ENDIF
    INC i
  UNTIL I > 15
ENDIF
```

This is very similar to the other UDP message handler, in that the incoming text (e.g. "ADC=1.23V") is copied to the second line of the display.

Rotating the upper potentiometer on the first board will change the voltage value displayed on the second board, demonstrating that the two boards have been linked via the network.

6. The DATAGRAM utility

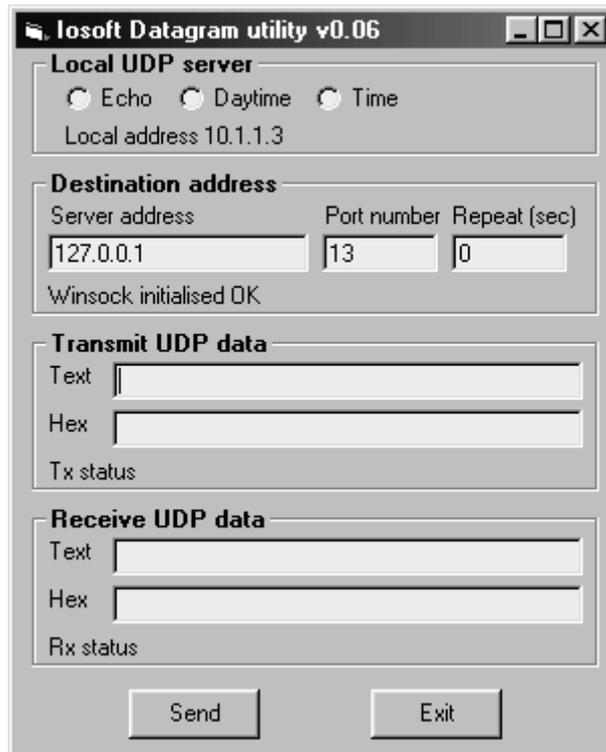


Figure 5: The Iosoft DATAGRAM utility

The Iosoft Datagram utility can perform the following server functions:

- An 'echo' server on UDP port 7. All incoming data is echoed back to the sender.
- An RFC 867 'daytime' server on UDP port 13. The incoming data is discarded, and a date & time string is returned to the sender.
- An RFC 868 'time' server on UDP port 37. The incoming data is discarded, and the number of seconds since 1st Jan 1900 is returned to the sender as an unsigned 32-bit value.

It also has the following UDP client capabilities:

- Sending UDP text or hex strings to a server.
- Receiving UDP text or hex strings in response.

A simple way of testing these capabilities is to employ the 'loopback' IP address 127.0.0.1; for example, enable the daytime server then send a blank (null) message to 127.0.0.1 port 13

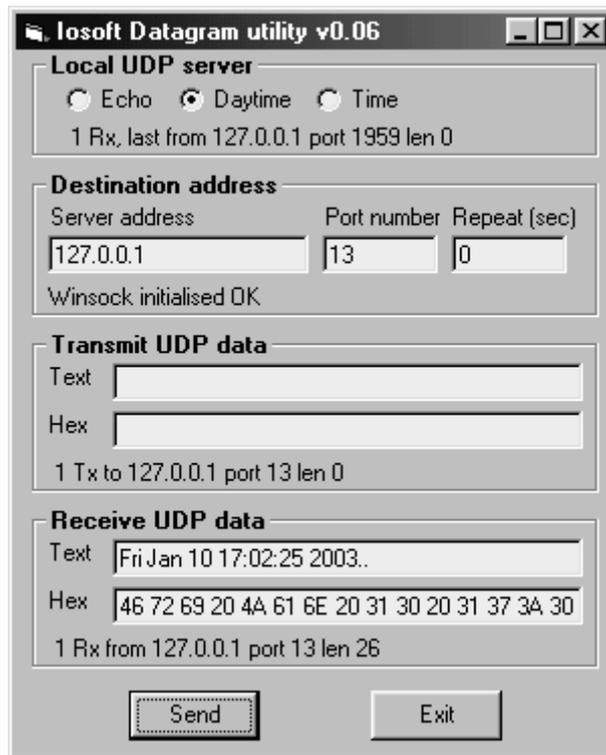


Figure 6: Using the loopback address

Instead of the message going out on the network, it is looped back internally to the datagram utility, which responds as if it had received a normal network request, so returns a date and time string, which is displayed as if it had been received over the network.

Similarly, if you enable the echo server, then send an arbitrary string of characters to 127.0.0.1 port 7, then that string will be echoed back, and displayed as received UDP data.

If a normal (non-loopback) IP address is used, then the UDP message will be sent out on the network, using the normal Windows TCP/IP settings.

JPB 23/1/03

--ends--