UDP/IP OVER ETHERNET FOR 8-BIT MICROCONTROLLERS

Federico de Ceballos
Universidad de Cantabria
federico.ceballos@unican.es

September, 2002

Abstract

With the widespread of the 'net, connecting any system to a private or public Ethernet network is nowadays considered as nearly essential. This has resulted in a movement towards more powerful processors and to those programming languages that provide the necessary libraries. However, connectivity is not something as exclusive or complex as some people would like us to believe. In fact, it is well suited for a lot of systems that can be programmed in Forth.

This paper presents a tiny UDP stack in Forth for the CS8900 Ethernet controller, suitable for immediate use in many 8-bit micros.

Introduction

It is quite common for microcontrollers to be provided with one or more UARTs for serial communication to the outside world. A channel of these characteristics offers limited capabilities (the baud rate can be quite high, however), as well as a single connection (unless other external systems are involved, as it is the case in PPP).

With the help of an external chip plus a few discrete components, it is possible to take a leap forward and bring in Internet access. The new capabilities will be probably limited be the resources available in the microcontroller, yet a simple subset of the protocols currently used can be implemented in a wide range of system with moderate effort.

The coding described in this paper is geared to the Cirrus Logic CS8900A [1], one of the several integrated solutions available.

The paper is divided into two parts. First of all, a short description of the protocols involved is given. Next, the vocabulary developed for a Forth system is presented.

Description of the protocols and messages involved

Ethernet

In this paper we shall be studying Ethernet packets, made of a preamble and a frame with control information and data. The packed travelling through the net are seen by every system connected, therefore some sort of unique hardware addressing is needed. This is achieved by MAC (Media Access Control) addresses.

Field	Size	Description
DA	**6**	**MAC destination address**
SA	**6**	**MAC source address**
Type	**2**	**message type**
Data	**46-1500**	**message dependant information**
FCS	**4**	**frame check sequence**

Table 1

With a normal configuration, we shall only receive packets addressed to us (with our MAC address in *DA*) or broadcast messages (with all bits set in *DA*). If we want to respond to such a message we have to write the data part with the response (taken from the other protocols), move *SA* to *DA*, put our MAC address in *SA*, change *Type* if necessary and send the message to the net (*FCS* is computed by the network processor).

IP

Table 2 shows the fields of an IP packet. As it can be seen, the IP packet (if it isn't long enough) is encapsulated into an Ethernet frame.

Field	Size	Description
DA	*6*	*MAC destination address*
SA	*6*	*MAC source address*
Type	*2*	*message type ($800)*
VerLen	**1**	**version and header length**
TOS	**1**	**type of service**
Length	**2**	**total IP length**
ID	**2**	**identification**
Fragment	**2**	**fragment offset into a big packet**
TTL	**1**	**time to live**
Protocol	**1**	**protocol used in the data portion**
Checksum	**2**	**checksum of the IP part**
IP SA	**4**	**IP source address**
IP DA	**4**	**IP destination address**
Data		**message dependant information**
FCS	*4*	*frame check sequence*

Table 2

If we have received an IP packet and we want to respond to the sender, we have to write the data part with the response (taken from other protocols), write 0 to *Checksum*, move *IP SA* to *IP DA*, put our IP address into *IP SA*, put an adequate value in *TTL*, write the length of the IP part into *Length*, calculate the checksum of the IP part and put the result into *Checksum* and then do the Ethernet frame response as described above.

Of course, several other possibilities are available, all of them found in the corresponding *Request For Comments* [2].

ARP

The Address Resolution Protocol is used in order to obtain the hardware (MAC) address used by a network card with a given IP address. A request is broadcasted so that the owner of the IP address is able to identify itself. This IP address is then used in further correspondence between the two systems.

An ARP message is encapsulated into the Data field of an Ethernet frame, giving the structure shown in table 3.

Field	Size	Description
DA	*6*	*MAC destination address (all bits set)*
SA	*6*	*MAC source address*
Type	*2*	*message type ($806)*
Hardware	**2**	**hardware type is 1 for 10Mb Ethernet**
Protocol	**2**	**protocol is $800**
H Length	**1**	**length of the hardware address (6)**
P Length	**1**	**length of the protocol address (4)**
Operation	**2**	**1 for an ARP request**
Sender HW	**6**	**sender MAC address (= SA)**
Sender IP	**4**	**sender IP address**
Target HW	**6**	**target MAC address (all bits clear)**
Target IP	**4**	**target IP address**
Padding	*18*	*padding in other to get a valid frame*
FCS	*4*	*frame check sequence*

Table 3

We only have to process one kind of ARP message, a broadcast request in which *Target IP* is our assigned IP address. If this condition is fulfilled and the rest of the fields match the indicated values in table 3, we have to do the following steps: move *DA* to *Target HW*, move *Sender IP* to *Target IP*, move our IP address to *Sender IP*, move our MAC address to *Sender HW*, write 2 as *Operation* (ARP response) and then do the Ethernet frame response as described above.

Further details of this protocol can be found in [4].

ICMP

Even if the Internet Control Message Protocol is not strictly needed for UDP purposes, it is nice to be able to find in a standard way whether a system is attached to the network. In order to do this, a *ping* message is sent ("are you there?") and a response is expected ("yes, I am").

The ICMP message is encapsulated into the IP data field with the information shown in table 4.

Field	Size	Description
DA	*6*	*MAC destination address*
SA	*6*	*MAC source address*
Type	*2*	*message type ($800)*
VerLen	*1*	*version and header length*
TOS	*1*	*type of service (0)*
Length	*2*	*total IP length*
ID	*2*	*identification*
Fragment	*2*	*fragment offset into a big packet*
TTL	*1*	*time to live*
Protocol	*1*	*protocol used in the data portion (1)*
Checksum	*2*	*checksum of the IP part*
IP SA	*4*	*IP source address*
IP DA	*4*	*IP destination address*
ICMP Type	**1**	**type of message (8)**
Code	**1**	**a code asigned to the message, probably 0**
ICMP Check	**2**	**checksum of the ICMP part**
Data	**varies**	**some values, probably fixed**
FCS	*4*	*frame check sequence*

Table 4

When we receive an IP message with *Protocol* set to 1 and *ICMP Type* set to 8, we know that an echo is being requested. All we have to do is: write 0 into *ICMP Type*, write 0 into *ICMP Check*, calculate the checksum of the ICMP part and put the result into *ICMP Check* and then do the IP response as described above.

Further details of this protocol can be found in [5].

UDP

Finally, an UDP message is also encapsulated into an IP message with the information shown in table 5.

If we decide to answer an UDP message directed to us from some port in some remote system, we have to swap the contents of *SP* and *DP*, copy the data and set *Length* to the appropriate value, write 0 into *UDP Check*, calculate the checksum of the UDP part and put the result into *UDP Check* and then do the IP response as described above.

Further details of this protocol can be found in [3].

Field	Size	Description
DA	*6*	*MAC destination address*
SA	*6*	*MAC source address*
Type	*2*	*message type ($800)*
VerLen	*1*	*version and header length*
TOS	*1*	*type of service*
Length	*2*	*total IP length*
ID	*2*	*identification*
Fragment	*2*	*fragment offset into a big packet*
TTL	*1*	*time to live*
Protocol	*1*	*protocol used in the data portion*
Checksum	*2*	*checksum of the IP part*
IP SA	*4*	*IP source address*
IP DA	*4*	*IP destination address*
SP	**2**	**source port**
DP	**2**	**destination port**
Length	**2**	**total UDP length**
UDP Check	**2**	**checksum of the UDP part**
Data	**varies**	**message dependant information**
FCS	*4*	*frame check sequence*

Table 5

Accessing the 8900 from Forth

This part describes some sets of words used to access the CS8900 chip. The source code (AVR assembler for the low-level words and Forth for the rest) is available from the author.

Low level words

These words allow the user to physically access the chip using seven output signals (/IOR, /IOW, /CHIP-SELECT and a four bit data bus) and eight bi-directional signals that compose the data bus.

```
-IOR              clear /IOR signal
-IOW              clear /IOW signal
-SEL              clear /CHIP-SELECT signal
+IOR              set /IOR signal
-IOW              set /IOW signal
-SEL              set /CHIP-SELECT signal
ADDR ( a )        set the addr bus
DATA! ( c )       write an 8bit value to the data bus
DATA@ ( c)        read an 8bit value from the data bus
DIN               set the data bus as input
DOUT              set the data bus as output
(INIT)            set the direction of the ports at start-up
```

Writing the IO registers

The low-level words presented before are only needed to allow the user to write to or read from the different registers that compose the IO space.

```
IO! ( c a )       write a value to an 8bit register
IO@ ( a -- c )    read a value from an 8bit register
```

Accessing the chip

With this baggage, we can define words to access the individual 16-bit registers, including those available from the PacketPage memory.

```
RXTX! ( x )       write to the transmit-receive data register
RXTX@ ( -- x )    read from the transmit-receive data register
TX-CMD! ( x )     write to the transmit command register
TX-LEN! ( x )     write to the transmit length register
PTR! ( x )        set the packet data pointer
PACKET! ( x )     write to the packet data register
PACKET@ ( -- x )  read from the packet data register

PP! ( x a )       set the contents of a packet data register
PP@ ( a -- x )    fetch the contents of a packet data register
```

Using the UDP vocabulary

Finally, this last set of words allows high level access to the UDP protocol. Please note that ARP and ICMP processing is hidden into UDP@.

```
/8900 ( -- flag )
      initialise the chip, return true if OK

UDP ( -- addr )
      return the address of the data part in the message buffer

UDP@ ( -- size true | false )
      check whether a new message is available. If so, copy the
      message into the buffer and return its size and true.
      Otherwise, return false

UDP! ( size )
      reply to the previous received message

UDP-CLIENT-IP ( -- dx )
      return the client IP address

UDP-CLIENT-PORT ( -- x )
      return the client UDP port number

UDP-SERVER ( -- x )
      return the destination UDP port number

UDP-SERVER! ( x )
      change the destination UDP port number
```

The meaning of each word should be clear from the description. UDP-SERVER! is used when responding to a message using a different port from the one it was addressed to.

Bibliography

[1] Cirrus Logic. *CS8900A Ethernet Controller Product Data Sheet*. 2001.

[2] RFC 760. *Internet Protocol*. 1980.

[3] RFC 768. *User Datagram Protocol*. 1980.

[4] RFC 826. *Ethernet Address Resolution Protocol*. 1982.

[5] RFC 792. *Internel Control Message Protocol*. 1981.