

Q68 Ethernet Device Driver

User Manual

This is an Ethernet driver for the Q68 computer. It implements an IP device driver for your Local Area Network (LAN). It can be used to communicate with other Q68 computers, or QL emulators that have the IP device driver built in, and support UDP and TCP connections. Such as QPC2.

The driver sends and receives standard IP data packets, So it can also talk to other devices and systems. This implementation is not a complete IP stack, so you may find some things are not fully supported.

This driver is intended for the Q68 computer only. It will **not** work on any other QL compatible computer.

This version supports UDP data packets and a pseudo TCP protocol. The TCP does not support all the usual TCP features. **It does not support errors, or lost packets.** So all the data packets must arrive, and in the correct order. This should be OK over a LAN, but you may have problems if you try to use the TCP over the internet.

The goal of this TCP implementation, is to support the IP Network driver. So the Q68 can operate a QL like network over a LAN.

The driver will also handle ARP packets to handle MAC address requests, and Ping packets. Anything else sent to the driver is liable to be thrown away.

Many of the system calls from the QDOS IP device driver by Richard Zidlicky have been implemented in this driver. Although some features of the system calls may be missing. See the section on system calls.

QPAC2 'channels' will display information on open Ethernet channels (courtesy of routines by Wolfgang Lenerz)

Acknowledgements

Thanks to Peter Graf, Martyn Hill, and Wolfgang Lenerz for their help and advice in development and testing of the driver.

Introduction

This device driver creates an IP stack with limited functionality. So don't expect full compatibility with a real IP network.

Note that the unlike TCP where the receipt of data packets is acknowledged back to the sender. The UDP protocol is very much a 'fire and forget' system, with no guarantee of the data packets getting through to the destination. The CP2200 Ethernet controller IC used in the Q68 can only store up to 8 data packets, or 4K bytes of data (whichever comes first) before it starts to ignore received data packets.

In the background, the driver tries to keep the receive buffer in the CP2200 empty. However if data packets arrive too quickly, it may not be able to keep up. In which case data packets may get lost. If you wish to avoid this, you will need to implement your own acknowledgement system.

Getting Started

LRESPR the Q68NET_BIN file. This should produce a message in #0, of

```
Q68 Ethernet Driver    Vx.xx
Initializing CP2200
```

If you are running SMSQ/E, the initialization messages should be in the system language. English, French, German, and Italian are currently supported.

If you also receive a message of `Self Initialization timed out or Auto-negotiation failed`. Then there was a problem during initialization of the CP2200 Ethernet controller IC, You can try to re-initialize the CP2200 with the **ETH_INIT** command.

You may also use the **ETH_ERRNO** function for a clue to what went wrong

You now need to assign an IP Address to the Q68 with the **ETH_SETIP** command

e.g. **ETH_SETIP "172.16.0.20"**

You may want to set a subnet mask with the **ETH_SUBNET** command, The default is 255.255.255.0

If you have a router, you can use the **ETH_GATEWAY** command to set it.

Example usage

In this example I am using a Q68 and a Windows computer(XP) running QPC2 V4.05. Alter the values given to suit your requirements.

Q68

LRESPR the driver, and when initialization is complete.

ETH_SETIP "172.16.0.20" This is the IP Address for the Q68.

OPEN_NEW#4,"udp_172.16.0.20:5800" Sets the Q68 to open it's network port 5800.

QPC2 (Assuming that the Windows computer is on the same network "172.16.0.xx")

OPEN_IN#4,"udp_172.16.0.20:5800" Sets QPC2 to talk to the Q68's port 5800.

On the Q68, Enter **REPeat loop:INPUT#4,a\$:PRINT LEN(a\$)!!!a\$**

Do this before the next the next step. The reason being that, it sets the Q68 waiting for input. If you tried to **PRINT#4** from the Q68 now. The Q68 will not know where to send to, As it does not know the IP address of the other computer.

On QPC2, Enter **PRINT#4,"Hello There"** This should appear on the Q68.

You may need to do this twice. For some reason QPC2 may not send the first string, It sends the Carriage Return on the end, but not the actual string.

Enter **PRINT#4,FILL\$("X",2000)** Which should also appear on the Q68.

This is sent as a fragmented packet, as it is longer than the 1472 bytes that will fit in one data packet.

Break into the loop on the Q68, and reverse the process.

In QPC2, Enter **REPeat loop:INPUT#4,a\$:PRINT LEN(a\$)!!!a\$**

and in the Q68 enter and run the program

```
100 a$="hello"
110 FOR n = 1 TO LEN(a$)
120 PRINT#4,a$(n);
130 END FOR n
140 PRINT#4
```

and then try, **a\$=FILL\$("Z",2000) : GO TO 110**

Why not **PRINT#4,"hello"**? Well **INPUT#** in the QPC2, UDP IP device driver does not seem to like more than one character at a time being sent in a data packet.

Opening Channels

I have tried as far as possible to follow the IP Device Driver as used in some QL emulators for the opening of channels

There are 3 devices available SCK_, UDP_, and TCP_.

OPEN just creates a socket of the requested type/protocol. An IP address & port are not required.

OPEN_IN creates a socket of the requested type/protocol. It sets the peer address for UDP sockets, or opens a connection to a server for TCP. The IP address and Port must be specified.

OPEN_NEW creates a socket of the requested type/protocol. It sets the host address for UDP sockets. The IP Address used in the **OPEN_NEW** command, should be the IP Address of the computer it's used on, or "0.0.0.0"

Note- It's a bit counter intuitive, but OPEN_IN creates an output channel, and OPEN_NEW creates an input channel. However once the connection has been established, the channels are then bi-directional. This is how the OPEN commands are defined in the QDOS IP device driver.

syntax: *channel_number* := *numeric_expression*
 socket_type := **SCK_** | **UDP_** | **TCP_**
 IP_address := IP Address in IPv4 numbers-and-dots notation
 port := Integer between 0 and 65535
 IP_specifier := *socket_type_IP_address:port*

OPEN#*channel_number,socket_type*
OPEN_IN#*channel_number,IP_specifier*
OPEN_NEW#*channel_number,IP_specifier*

FOPEN([#*channel_number*,]*socket_type*)
FOP_IN([#*channel_number*,]*IP_specifier*)
FOP_NEW([#*channel_number*,]*IP_specifier*)

example: i. **OPEN#4,SCK_**
 ii. **OPEN_IN#5,"TCP_192.168.0.10:5800"**
 iii. **OPEN_NEW#ch,"UDP_192.168.0.5:5800"**

ETH_INIT

FETH_INIT

ETH_INIT will attempt to (Re)Initialize the CP2200 Ethernet controller.

FETH_INIT is a function version of **ETH_INIT** and will return 0, or an error code

Returns a Not Found error if the Ethernet driver cannot be found.

syntax: **ETH_INIT**
 FETH_INIT

example: i. **ETH_INIT**
 ii. **PRINT ETH_INIT**

ETH_MAC\$

The function **ETH_MAC\$** will return the MAC address of the CP2200 Ethernet controller as a dash separated string.

syntax: **ETH_MAC\$**

example: **PRINT ETH_MAC\$**

ETH_SETIP

ETH_SETIP will set the IP Address that the Q68 will use.

Setting, or resetting the IP address will cause a gratuitous ARP request to be broadcast over the network, to inform any other computers of it's change.

syntax: *IPaddress := string_expression*

 ETH_SETIP IPaddress

example: i. **ETH_SETIP "192.168.0.10"**
 ii. **ETH_SETIP address\$**

ETH_GETIP\$

The function **ETH_GETIP\$** will get the IP Address that was set with the **ETH_SETIP** command.

syntax: **ETH_GETIP\$**

example: i. **PRINT ETH_GETIP\$**
 ii. **a\$ = ETH_GETIP\$**

ETH_SUBNET

ETH_SUBNET will set the subnet mask that the Q68 will use.

syntax: *mask := string_expression*

 ETH_SUBNET mask

example: i. **ETH_SUBNET "255.255.255.0"**
 ii. **ETH_SUBNET a\$**

ETH_SUBNET\$

The function **ETH_SUBNET\$** will get the subnet mask that was set with the **ETH_SUBNET** command.

syntax: **ETH_SUBNET\$**

example: i. **PRINT ETH_SUBNET\$**
 ii. **a\$ = ETH_SUBNET\$**

ETH_GATEWAY

ETH_GATEWAY will set the gateway (router) IP address that the Q68 will use.

syntax: *IPaddress := string_expression*

ETH_GATEWAY *IPaddress*

example: i. **ETH_GATEWAY "192.168.0.1"**
 ii. **ETH_GATEWAY a\$**

ETH_GATEWAY\$

The function **ETH_GATEWAY\$** will get the gateway (router) IP address that was set with the **ETH_GATEWAY** command.

syntax: **ETH_GATEWAY\$**

example: i. **PRINT ETH_GATEWAY\$**
 ii. **a\$ = ETH_GATEWAY\$**

ETH_NETNAME

ETH_NETNAME will set the Q68's network name. The network name can be up to 26 characters long.

syntax: *netname := string_expression*

ETH_NETNAME *netname*

example: i. **ETH_NETNAME "Ethernet Q68"**
 ii. **ETH_NETNAME a\$**

ETH_NETNAME\$

The function **ETH_NETNAME\$** will return the network name given by the **ETH_NETNAME** command.

syntax: **ETH_NETNAME\$**

example: i. **PRINT ETH_NETNAME\$**
 ii. **a\$ = ETH_NETNAME\$**

ETH_PING

ETH_PING will send up to 4 Ping requests over the network, to the specified IP Address. The results will be sent to either the specified, or the default channel (#1).

syntax: *channel_number := numeric_expression*
 IPaddress := string_expression

ETH_PING [#*channel_number*,] *IPaddress*

example: i. **ETH_PING "192.168.0.1"**
 ii. **ETH_PING#4, a\$**

ETH_ERRNO

The function **ETH_ERRNO** will return the last driver error that occurred. This is a specific error number for the Ethernet driver, and is not the same as the QDOS error that may have been displayed.

If you encounter a QDOS error message while using the driver. It may not be a very helpful error message, as there are only a limited number of QDOS error messages. Using **ETH_ERRNO** may supply you with a more helpful error report. See the list of extended error messages at the rear of this document.

ETH_ERRNO will return 0, for no current error.

After using **ETH_ERRNO**, the last error will be cleared.

syntax: **ETH_ERRNO**

example: i. **PRINT ETH_ERRNO**
 ii. **a = ETH_ERRNO**

ARP_ADD

ARP_ADD will add an IP Address and MAC Address into the ARP table.

syntax: *IPaddress := string_expression*
 MACaddress := string_expression

ARP_ADD *IPaddress,MACaddress*

example: i. **ARP_ADD "192.168.0.20","01-02-03-04-05-06"**
 ii. **ARP_ADD address\$,mac\$**

ARP_REMOVE

ARP_REMOVE will remove an ARP table entry. If no parameter is supplied, then all the ARP table entries will be removed.

syntax: *IPaddress := string_expression*

ARP_REMOVE *IPaddress*

example: i. **ARP_REMOVE "192.168.0.20"**
 ii. **ARP_REMOVE**

ARP_LIST

ARP_LIST will list all the ARP table entries to the supplied channel, or will default to channel #1. The list will be in the format, IP Address, MAC Address.

syntax: *channel := numeric_expression*

ARP_LIST [#channel]

example: i. **ARP_LIST**
ii. **ARP_LIST #2**
iii **ARP_LIST #chan**

QPAC2 Channels

If you have an Ethernet channel open. When you look at QPAC2's Channels. Information will be displayed about the open channel.

The channel information will be displayed in the format, Socket type, Local port number, Peer IP address, Peer port number

e.g. **UDP_53760:172.16.0.9:5800**

A UDP connection, of my port 53760, to port 5800 on IP address 172.16.0.9

Supported System Trap Calls

See the UQLX documentation for details of these system traps.

Trap #2

D0	Name	Notes

\$01	IO_OPEN	D3=0-2
\$01	IP_ACCEPT	D3=LISTENing channel ID
\$02	IO_CLOSE	

Trap #3

D0	Name	Notes	

\$00	IO_PEND		
\$01	IO_FBYTE		
\$02	IO_FLINE		
\$03	IO_FSTRG		
\$05	IO_SBYTE		
\$07	IO_SSTRG	D2 is word sized, So should limit data size to 32K	
\$48	FS_LOAD		
\$49	FS_SAVE		
\$50	IP_LISTEN		
\$51	IP_SEND	data size limited to 64K	See notes
\$52	IP_SENDTO	data size limited to 64K	See notes
\$53	IP_RECV		See notes
\$54	IP_RECVFM		See notes

\$58	IP_BIND
\$59	IP_CONNECT
\$5B	IP_GETHOSTNAME
\$5C	IP_GETSOCKNAME
\$5D	IP_GETPEERNAME
\$64	IP_GETSERVBYNAME
\$65	IP_GETSERVBYPORT
\$6E	IP_GETPROTOBYNAME
\$6F	IP_GETPROTOBYNUMBER
\$72	IP_INET_ATON
\$73	IP_INET_ADDR
\$74	IP_INET_NETWORK
\$75	IP_INET_NTOA
\$76	IP_INET_MAKEADDR
\$77	IP_INET_LNAOF
\$78	IP_INET_NETOF
\$7C	IP_ERRNO

Notes:

IP_SEND and IP_SENDTO	D1 flag is not supported
IP_RECV and IP_RECVFM	D1 flag only supports MSG_PEEK

Ethernet driver error messages

If the Ethernet driver encounters a problem, it may store an error message that is more useful than any QDOS error message that may also be returned.

These error messages may be examined with the **ETH_ERRNO** function.

There is only ever one error number that is stored, and it is the last error encountered.

After using the **ETH_ERRNO** function, the error code is reset to zero.

Receive errors

- 1 Insufficient memory to read packet into
- 2 Read packet validation failed, checksum mismatch
- 3 Received packet from unexpected MAC address
- 4 Unable to process TCP control bits
- 5 Unexpected TCP segment numbers
- 6 Unexpected TCP sequence number

- 8 Last packet was not transmitted successfully

Send errors

- 10 Timeout waiting for transmit buffer to empty
- 11 Protocol not found when creating a packet
- 12 Failure sending ARP request
- 13 Failure sending Ping reply
- 14 Attempt to send too many packets with received ACK's (TCP)
- 15 Failure sending a SYN,ACK

Open errors

- 20 Invalid parameters for requested OPEN type
- 21 No managed ports available
- 22 Unable to acquire MAC address for specified IP address
- 23 Requested port already in use
- 24 No response from requested TCP server for a SYN request

I/O errors

- 30 Error creating checksum for transmission
- 31 Attempt to send more than 64K bytes
- 32 Bad sockaddr length
- 33 Fragmented packet incomplete before life ran out
- 34 I/O timeout while waiting for a MAC address
- 35 IP_LISTEN queue out of range
- 36 Not a TCP channel
- 37 Out of memory creating a dummy channel definition block
- 38 The queue for a listening channel is full

Close errors

- 40 Timed out waiting for a TCP close acknowledgement
- 41 While in FIN-WAIT-1, Timed out waiting for an ACK
- 42 While in FIN-WAIT-2, Timed out waiting for a FIN (last ACK)
- 43 While in LAST-ACK, Timed out waiting for final ACK
- 44 Unexpected FIN received

IP Trap errors

- 50 Error reading a database entry

CP2200 Initialization errors

- 100 Timed out while waiting for controller to reset
- 110 Timed out while waiting for auto-neg in Physical layer

Copyright and Disclaimer

This driver should not cause any problems, damage, or loss of data. However by using this device driver, you do so at your own risk, and I do not accept responsibility for any damage, or loss of data.

The driver may contain portions of, or be based on routines from the SMSQ/E source code. (Although it may be hard to find them)

Licence for SMSQ/E

Copyright (c) 1989-2012, by

Tony Tebby
Marcel Kilgus
Bruno Coativy
Fabrizio Diversi
Phoebus Dokos
Thierry Godefroy
Jérôme Grimberty
George Gwilt
John Hall
Mark Swift
Per Witte
Wolfgang Lenerz

collectively called the "COPYRIGHT HOLDERS".

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL COPYRIGHT HOLDERS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.