From Metcalfe and Boggs

Subject: Alto Ethernet Interface

Inter-Office Memorandum

To: Ethernet Distribution

Date: June 22, 1974

From: Metcalfe and Boggs

Location: Coyote Hill

Subject: Alto Ethernet Interface

Organization: Parc CSL and SSL

XEROX

This reference memo defines the Alto Ethernet Interface as seen from within the Alto's emulator. While we expect that we ourselves will write most of the software which deals with this interface, you may find the specifications useful for some special BCPL programming of your own. This memo will be revised and extended with experience; we welcome suggestions and comments.

You have received this memo because your name is on the newly established Ethernet distribution list maintained by Adrienne Payne. Please remove your name from the list if memos like this one are of no interest to you.

Like other Alto device interfaces, the Ethernet uses several reserved locations in Alto memory to receive its commands and to report its status in cooperation with an emulator program.

These locations are (from "ETHERNET\ETHERSYMS.BC"):

manifest [ EPLoc=#600] //Post location
manifest [ EBLoc=#601] //Interrupt bit location
manifest [ EELoc=#602] //EOT count location, Posted
manifest [ ELLoc=#603] //Load location (mask)
manifest [ EILoc=#604] //Input count location
manifest [ EIPLoc=#605] //Input pointer location
manifest [ EOLoc=#606] //Output count location
manifest [ EOPLoc=#607] //Output pointer location
manifest [ ESLoc=#610] //Serial Number location
manifest [ ESpare=#612] //Spare

GEPLoc receives status information (see below) when an Ethernet command (see below) completes. GEPLoc may hold a bit to be set into NWW (new wake-ups waiting) when an Ethernet command completes so as to interrupt the Alto's emulator (see memos on the Alto interrupt system). GELoc gets the number of remaining buffer words at command completion; it is used for computing the length of an input packet -- PacketLength=(GELoc)-(GELoc).

If GESLoc is zero, we say that your Alto is promiscuous; the Ethernet microcode will accept packets regardless of their destination address. A packet addressed to zero, please note, is a broadcast packet and will be accepted regardless of GESLoc.

June 22, 1974

Alto Ethernet Interface
When the Ethernet finishes processing one of its SIO commands, it posts the results in @EPLoc. The left-hand byte (bits 00 to 07) of the post code carries the microcode's explanation of what happened as a result of the command. The right-hand byte (bits 08 to 15) carries the hardware status at the time of the post. Both bytes of @EPLoc must be interpreted to tell exactly what happened. Here are the codes to be found in the left-hand byte and their mnemonics:

manifest [PostInDone=0] //Input flow terminated, maybe AOK
manifest [PostOutDone=1] //Output flow terminated, maybe AOK
manifest [PostIBOverflow=2] //Incoming packet overflowed buffer
manifest [PostOutLoadOverflow=3] //16 collisions, load overflow
manifest [PostwordCountZero=4] //User (you) provided zero-length buffer
manifest [PostAborted=5] //Flow aborted for some reason (reset?)
manifest [PostNoverHappon=6] //Serious hardware/firmware bug

The hardware status bits found in the right-hand byte of @EPLoc (after a post) are low; they are normally one. If zero, they signal the following conditions:

| Bit 15 | Incomplete transmission, discard input |
| Bit 14 | Output command issued causing post |
| Bit 13 | Input command issued causing post |
| Bits 13&14 | Reset command issued |
| Bit 12 | CRC register not zero, discard input |
| Bit 11 | Collision (not used by software) |
| Bit 10 | Input data late, discard input |
| Bits 8&9 | Should always be 1; if zero, failure of hardware |

Routine checking of post codes can be speeded with the use of patterns. These patterns are formed from the appropriate left-hand microcode byte and normal hardware status.

manifest [EtherOutputOK=#777] //Output was successful
manifest [EtherInputOK=#377] //Input was successful

Here is how to reset the Ethernet interface:

and EtherReset()=valof
[ @EPLoc=0 //Clear posting location
  StartI0(EtherResetCommand) //Request hardware/firmware reset
  results (@EPLoc ne 0) //Return boolean ]

The EtherReset routine should always return the boolean true; if not, the Ethernet firmware/hardware package is malfunctioning or not installed.

Timing, and time-outs in particular, are important in Etherneting. Here are two useful (and trivial) timing routines which support the following Ethernet examples.

static [Timer] //Holds number of "tics" to time-out
and SetTimer (Microseconds) be
[ Timer=(Microseconds/70)+1 //Convert from Microseconds to tics ]
and TimedOut()=valof
[ Timer=Timer-1 //Count down Timer to wait for time-out
  resultis (Timer le 0) ]

Here is a simple (and not always adequate) way to get an Ethernet packet:

and GetEther (Buffer,Time)=valof
[ GEICLoc=BufferI0 //Length
  @EPLoc=Buffer+1 //Pointer to packet's first word
  SetTimer(Time) //Establish how long wait
  GetPost: //Come here to look for new post code
    GEICLoc=0 //To look for non-zero
    StartI0(EtherInputCommand) //Input?
    while ((@EPLoc eq 0)&(not TimedOut())) loop
      if TimedOut() then results false
      if (@EPLoc eq EtherInputOK) then results true
      goto GetPost //keep waiting ]

-3-

-4-
Here is a simple (and not always adequate) way to send an Ethernet packet:

```
and PutEther(Buffer,Time)=valof
   [GEOCLoc=Buffer!0 //Only send what I tell you
   @EOPLoc=Buffer+1
   @EICLoc=0 //No input please
   @EIPLoc=0
   SetTimer(Time) //time-out
   GetPost: //Come here to look for new post
   @EPLoc=Buffer+1
   @EICLoc=0 //No Input please
   @EPLoc=0 //To look for non-zero
   StartO(EtherOutputCommand) //Go!
   while ((GEOCLoc & (not TimedOut()))) loop
      If TimedOut() then result: false
      If ((GEOCLoc & EtherOutputOK()) then result: true
      goto GetPost //Keep trying
   ]
```

If an input buffer is specified during an output command, the Ethernet interface will look for incoming packets during its retransmission waits (if any); an input post code will result if a packet comes in under an output; the output will not be done.

Here is a way to wait for an Ethernet post should one of the above (or any) operation failed to post in time:

```
and WaitEther(Time)=valof
   [SetTimer(Time)
   while ((GEOCLoc & (not TimedOut()))) loop
      results & EPLoc
   ]
```

The EtherAround routine accepts a time-out count for a prespecified output (and pre-zorded post location) and returns the output status obtained after quickly starting an input.

```
and EtherAround(Count)={table[
   NEG+SA0+DA1; //NEG 0,1 Make count negative
   LOAD+ACO+PREL+20; //LOAD 0,1, Output Command
   STARTIO; //SIO Start Ethernet Output
   LOAD+AC0+IND+PREL+16; //LOAD 0,GEPLoc Get Post location
   MOV+SA0+DA0+SZR; //MOV 0,0,SR Chk for post
   JUMP+PREL+(4 &#377); //JMP -4. Keep waiting
   DORTN; //JMP 1,3 Timed-out
   LOAD+AC1+PREL+13; //EtherInputOK
   SUB+SA0+DA0+SZR; //Input under output?
   DORTN; //Input under output
   MOV+SA0+DA1; //MOV 0,1 Move post code to safe place
   SUB+SA0+DA0; //SUB 0,0 Make a zero
   STORE+AC0+IND+PREL+8; //STORE 0,GEPLoc Zero post loc
   LOAD+AC0+PREL+5; //LOAD 0,C2 Input Command
   STARTIO; //SIO Start Ether Input
   MOV+SA1+DA0; //MOV 1,0 Return post code
   DORTN; //JMP 1,3 Receiver started
   EPLoc; //Address of Ethernet Post location
   EtherInputCommand; //EtherInputCommand
   EtherOutputCommand; //EtherOutputCommand
   EtherInputOK; //posted when input comes in under output
   ]) (Count)
```

DISTRIBUTION / ETHERNET

BACHRACH, Bob
BATES, Roger
BELEN, Peter
BEGGS, David
CLARK, Larry
DEUTSCH, Peter
DUVALL, Bill
ELKIND, Jerry
ENGLISH, Bill
FAIRBAIN, Doug
FIALA, Ed
FLEGAL, Bob
GECKIE, Chuck
GERHARDT, Suzan
LAMPORT, Butler
LIDDLE, David
McCREIGHT, Ed
McDANIEL, Gene

MELVIN, John
METCALFE, Bob
PARISH, Vicki
RIDER, Ron
SROCH, John
SIRONIT, Charles
SPROULL, Bob
STURGES, Howard
STONEHART, Dan
TAFT, Ed
TAYLOR, Bob
THACKER, Chuck
TREICHEL, Jeanie
WILMER, Mike
ZELINSKY, Mara