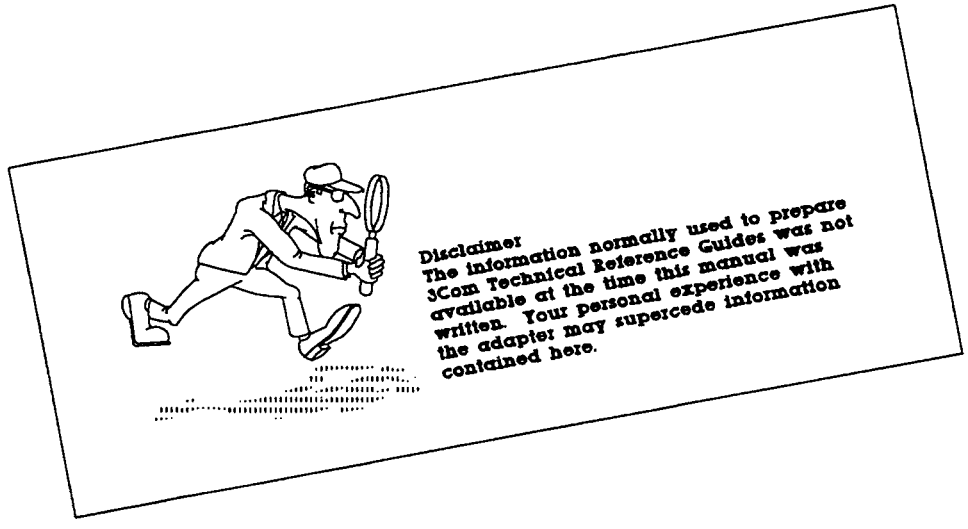
EtherLink (3C501) Adapter
Technical Reference



5400 Bayfront Plaza, Santa Clara, CA 95052

Manual Part No. 6405-00
Originally published November 26, 1988. Printed in the U.S.A.

Contents

List of Figures

Introduction

The EtherLink adapter is a low-cost Ethernet controller/transceiver for IBM Personal Computers which conforms to the Ethernet Specification, Version 1.0, 30 September 1980, as published by DEC, Intel and Xerox. With one minor exception, it implements levels one and two of the Open Systems Interconnect Model of the International Standards Organization:

Level One Functions, Physical Layer:

- Coax/station electrical isolation.
- Bit transmission/reception.
- CarrEtherLink adapter sense.
- Transmit collision detection.
- Encoding/decoding.
- Preamble generation/removal

Level Two Functions, Data Link Layer:

- Frame check sequence generation/checking.
- Carrier deference.
- Transmit collision enforcement.
- Collision fragment (runt) filtering.
- Bad packet filtering.
- Address recognition.

The controller on the EtherLink adapter incorporates a VLSI Ethernet Data Link Controller, the See 8001 or EDLC, and a single, 2Kbyte packet buffer designed to operate with the 8237A DMA controller found on the IBM System Board. It also has provisions for external loopback and can use one of the interrupt channels of the 8259A interrupt controller on the system board. In addition to a conventional Ethernet controller, the EtherLink adapter contains an Ethernet transceiver providing complete Level 1 and Level 2 functionality on one printed circuit card.

Two versions of the EtherLink adapter controller exist. The two versions may be differentiated by their respective assembly numbers. These numbers are printed on the cards. One version has ASSY 0345- printed along the connector edge of the card. The other version has ASSY 34-0780- printed along the edge opposite the connector. These versions have minor differences which are noted where they apply.

Architecture

The EtherLink adapter has a single 2Kbyte packet buffer (large enough for the longest Ethernet packet) shared between transmit and receive. Once a packet is transferred to the buffer for transmission and a transmit initiated, the software must intervene only in case of a collision. To receive a packet, software selects one of several address recognition modes; when a packet of interest is detected, the controller places it in the packet buffer. When enabled for multicast recognition, address screening of multicast packets must be performed by the system software.

The software interface to the EtherLink adapter is a block of sixteen registers in the I/O space of the 8088. These registers are used to write commands, read status information, and access packet data. In contrast with IBM's practice, the EtherLink adapter's base address is jumper settable (32 values on ASSY 0345 and 64 values on ASSY 34-0780). The EtherLink adapter has a 4Kbyte ROM for program storage. The base address of the ROM in memory space is also jumper settable (32 values on ASSY 0345 and 256 on ASSY 34-0780).

Access to the packet buffer switches between the system bus and the network under software control. Network access can be receive, transmit with automatic rollover to receive, or loopback.

One of the I/O registers provides a one byte window on the packet buffer. Another register, GP, the general purpose buffer pointer, holds the address of the byte visible through the packet buffer window. Reading and writing the window automatically increments GP permitting sequential access to the packet buffer from the system bus. Writing GP then reading or writing the window gives the effect of random access.

The packet buffer can be loaded and unloaded using the 8237A DMA controller on the system board to repetitively read or write the window. The EtherLink adapter can request DMA service, detect the end of the transfer and interrupt when the DMA is done.

All interrupts go through the 8259A interrupt controller on the system board. Each type of EtherLink adapter interrupt is enabled independently of other types; however, the EtherLink adapter has only one interrupt line to the system. This arrangement requires software to scan the EtherLink adapter status registers to determine the cause of an interrupt. Consistent with IBM PC practice, both DMA service request and interrupt request line drivers can be disabled.

System Interface

The EtherLink adapter has two 11-bit buffer address pointers (registers), the General Purpose pointer, GP, and the Receive Pointer, RP. RP is used as the buffer pointer while the controller receives packets from the network. Software can read and reset RP. GP is used by the controller during transmit and by software to address the packet buffer when the buffer is available to the system bus. Software can read and write GP. GP automatically increments when the packet buffer is read, either by the system or while transmitting to the network. Loopback operation uses both GP and RP.

Transmit packets are end-aligned within the packet buffer. Software uses GP when filling the buffer. Software must reload GP with the address of the first byte of the packet before initiating a transmission.

Receive packets are front-aligned in the buffer. After the EtherLink adapter receives a packet, RP contains the packet length in bytes. If the packet length exceeds the legal maximum, the first 2048 bytes will be saved in the buffer. After the 2048th byte, RP locks up preventing any buffer overwrite; reading a packet length of 2048 (800 hex) from RP indicates a packet of at least 2048 bytes.

During loopback, the controller reads the end-aligned transmit packet from the buffer and writes the received packet front-aligned in the buffer. A maximum length packet can be looped back. The received packet may overwrite part of the transmitted packet. Loopback requires the EtherLink adapter to be connected to an Ethernet by the onboard or external transceiver, or fitted with special BNC or DA-15 loopback plugs by the user.

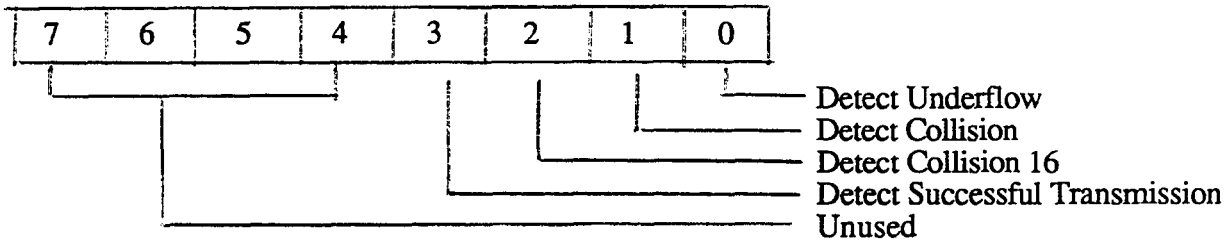
The EtherLink adapter's Ethernet station address is stored in a PROM, whose contents are accessible through another one byte window register similar to the window register used to access the packet buffer. available in locations zero thru five with station address byte zero at address zero. Bits 0-2 of GP are used to address the PROM. However, unlike access to the packet buffer, reading the station address does NOT auto-increment GP; software must explicitly increment it.

The EtherLink adapter provides two sets of registers for the Ethernet station address. One set is read only (the PROM); the other set is write only. Software must program the station address by setting the write only register set. The station address in the PROM serves only to provide a "hint" about what the station address should be.

EtherLink Adapter Controller Register Map

	Read	Write
0		Station Addr 0
1		Station Addr 1
2		Station Addr 2
3		Station Addr 3
4		Station Addr 4
5		Station Addr 5
6	Receive Status	Receive Command
7	Transmit Status	Transmit Command
8	GP Buffer Pointer [LSB]	GP Buffer Pointer [LSB]
9	GP Buffer Pointer [MSB]	GP Buffer Pointer [MSB]
A	RCV Buffer Pointer [LSB]	RCV Buffer Pointer Clear
B	RCV Buffer Pointer [MSB]	
C	Ethernet Address Prom Window	
D		
E	Auxiliary Status (CSR)	Auxiliary Command (CSR)
F	Buffer Window	Buffer Window

Transmit Command Register



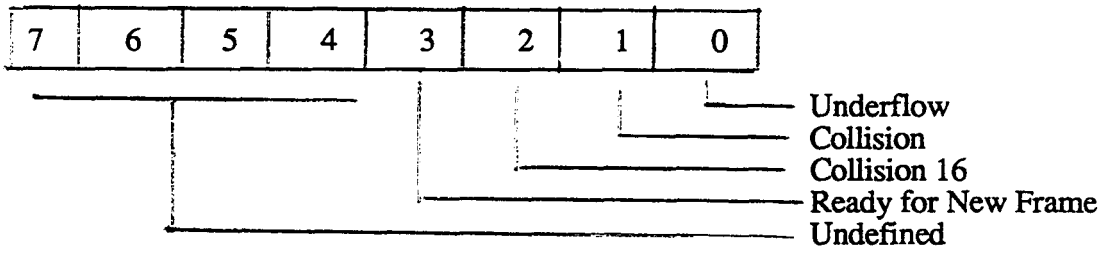
A packet transmission can terminate for any of four different reasons: successful transmission, collision, sixteenth successive collision without successful transmission, or underflow. After each collision, data remains in the packet buffer undisturbed, but software must reset GP and explicitly restart the transmission. Once restarted, the adapter delays the appropriate amount of time before actually retransmitting the packet. After the sixteenth consecutive collision further attempts to retransmit should be abandoned on the assumption that the network is overloaded or has failed.

Underflow occurs only when transmitting packets with bad FCS, for diagnostic purposes, and should not be seen during routine operation of the controller.

Software may choose whether to ignore or detect any of the four conditions listed above. A one in the corresponding bit position detects the condition; a zero ignores the condition. Detecting a condition is not sufficient to generate an interrupt. Software must also have set Request Interrupt and DMA Enable (RIDE) and/or Interrupt Request Enable (IRE, on ASSY 34-0780 only) for the EtherLink adapter to generate interrupts.

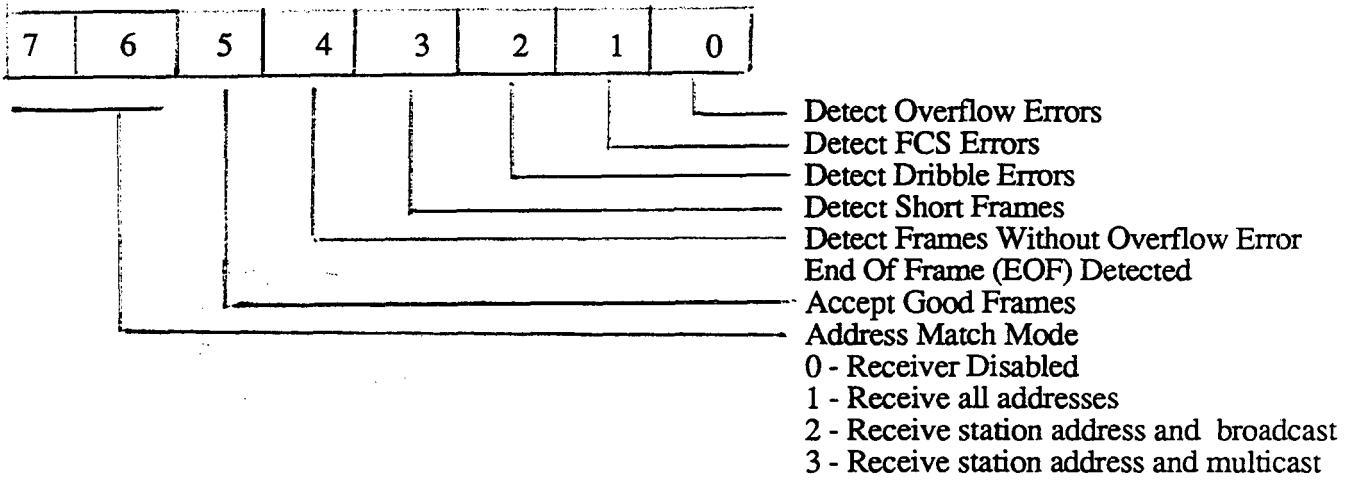
The details concerning underflow and interrupts are contained in the description of the Auxiliary Command Register below.

Transmit Status Register



The controller loads the Transmit Status Register only after each transmission or attempted transmission. If interrupts are enabled for transmit, reading the status register clears the interrupt.

Receive Command Register



Software can program the EtherLink adapter to detect only certain classes of packets (packets of interest); all unwanted packets are discarded automatically. Discarded packets require no software intervention. After a packet is discarded RP is reset and the controller is enabled to receive the next packet of interest. The EtherLink adapter will generate interrupts on packets of interest only if RIDE and/or IRE (ASSY 34-0780 only) is set.

The Address Match Mode controls which packets to accept by examining their destination addresses. If the match mode is zero, the EtherLink adapter will not detect any packet; mode one accepts packets regardless of the contents of their destination addresses. Modes two and three compare the destination of each packet with the station address registers stored in adapter registers zero through five.

Other bits in the Receive Command Register allow software to further define packets of interest. The EtherLink adapter only accepts (loads the buffer with) well formed packets (legal size, no FCS error and no overflow). Setting Accept Good Frames defines good frames (legal size, no overflow and no FCS error) as frames of interest. Dribble errors are allowed on good frames as long as there are no other errors. Setting Detect Dribble Errors causes the EtherLink adapter to detect dribble errors, the packet will be loaded into the buffer if it has no other errors.

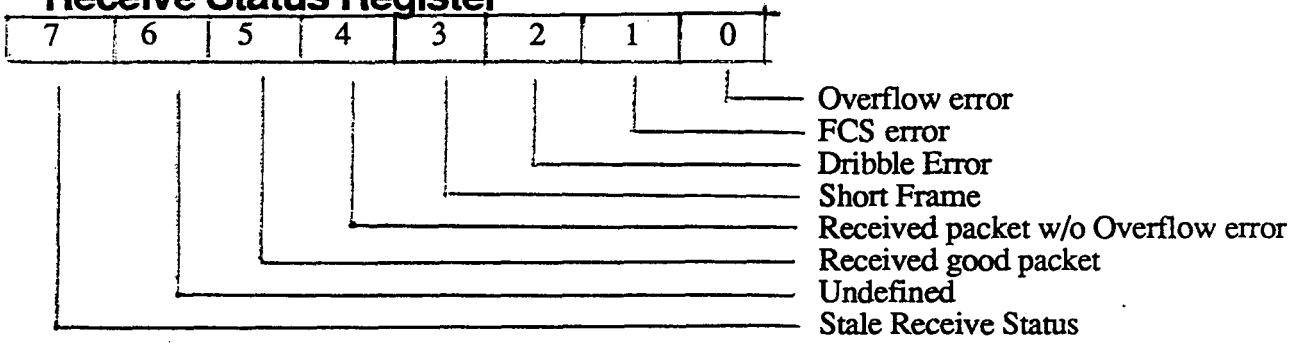
All other bits are useful only to detect packets with errors. The controller will discard packets with errors, however software can detect them in order to keep counts for diagnostic purposes. Short frames are packets whose length is less than 60 bytes, excluding preamble and FCS; these are probably collision fragments. FCS error means the four byte FCS computed on receipt did not match the FCS in the packet. (Note that an Ethernet "alignment error" is equivalent to a packet with dribble and FCS errors). Overflow errors happen when the controller detects a packet of interest while the packet buffer is not available. The buffer may contain a previously received packet, or it may belong to the system bus.

Table 1. Discard/EOF Condition Truth Table

Frame Status	Interesting Packet	Discard	EOF
Good Frame	No	No	Yes
	Yes	No	Yes
Overflow	No	Yes	No
	Yes	Yes	No
Short Frame (Runt)	No	Yes	Yes
	Yes	Yes	Yes
FCS Error	No	Yes	Yes
	Yes	Yes	Yes
Dribble Error	No	No	Yes
	Yes	No	Yes
Address Mismatch	N/A	Yes	No

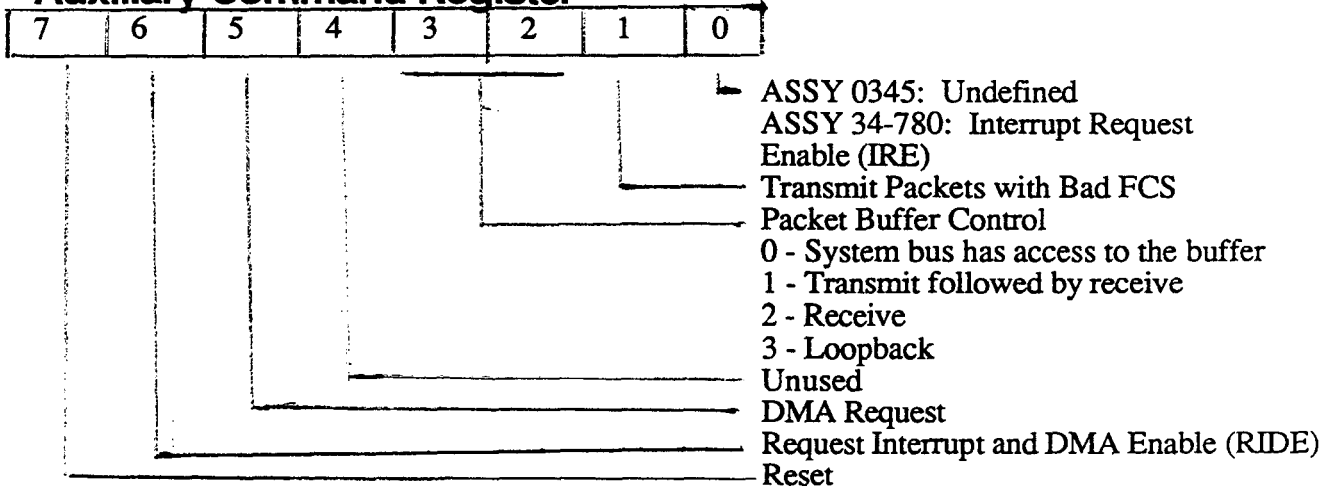
(not accurate)

Receive Status Register



Software defines packets of interest by setting the Receive Command Register. The controller changes the status register after every packet detected on the network whether or not it was interesting. If the controller detects an interesting packet, the Stale Receive Status goes to zero; once the Stale Receive Status is zero the controller discards all packets until software reads the status register; this guarantees that software reads the status associated with the detected packet. Reading the status register sets the Stale Receive Status back to one; the EtherLink adapter can then detect the next interesting packet on the network. However, if the controller is not in receive mode all interesting packets will cause overflows. The Stale Receive Status bit will not go to one in this case unless overflow errors are set as packets of interest. Reading the Receive Status Register clears any receive interrupts.

Auxiliary Command Register



Writing the Reset bit to one resets all control and status registers in the EtherLink adapter. Software must explicitly set this bit to zero after setting it to one. Leaving Reset on has the effect of perpetually resetting the controller.

The Request Interrupt and DMA Enable (RIDE) bit permits the EtherLink adapter to drive both the Interrupt Request (IRQ) and DMA Service Request (DRQ) signals on the system bus. Jumpers on the EtherLink adapter card select the DMA channel (either 1 or 3 on ASSY 0345, or 1, 2 or 3 on ASSY 34-0780) and interrupt channel (either 3 or 5 on ASSY 0345, or any one of 2 thru 7 on ASSY 34-0780). When RIDE is zero, the EtherLink adapter cannot generate DMA transfers and can only generate interrupts if IRE is one (IRE is available only on ASSY 34-0780). Bits in the transmit and receive command registers can be set to detect certain conditions; however, no interrupts can result until RIDE or IRE is a one. RIDE and IRE may be set to one simultaneously. When RIDE, and IRE on ASSY 34-0780, are zero, the interrupt and DMA channels selected are tri-stated per IBM convention.

Software must manipulate RIDE, IRE and interrupts with care. When RIDE is zero the state of the associated IRQ and DRQ lines on the system bus can be undefined. Leaving these lines in an undefined state when their associated DMA and interrupt channels are active can result in strange and unpredictable behavior. Software must insure that the associated IRQ and DRQ lines are not used by other peripheral devices before setting RIDE or IRE to one. Neither setting of RIDE or IRE is safe under all circumstances!

Setting DMA Request to one starts a DMA transfer. The EtherLink adapter interrupts at the completion of the transfer. Setting DMA Request to zero, disables DMA Service request, clears DMA Done, and clears the interrupt.

Bits 2 and 3 of the auxiliary command register control access to the packet buffer. If both bits are zero, the buffer "belongs" to the system bus; software is free to read and write the buffer without interference from the network. If either of the bits are not zero the packet buffer belongs to the network.

If bit 3 is one, the controller will accept one packet from the network. The setting of the receive command register causes the EtherLink adapter controller to receive and/or detect only "interesting packets" all other packets will be discarded. Received packets are loaded into the packet buffer starting at address '000H'. After a packet has been loaded into the packet buffer the size, in bytes, excluding preamble and FCS, remains in RP.

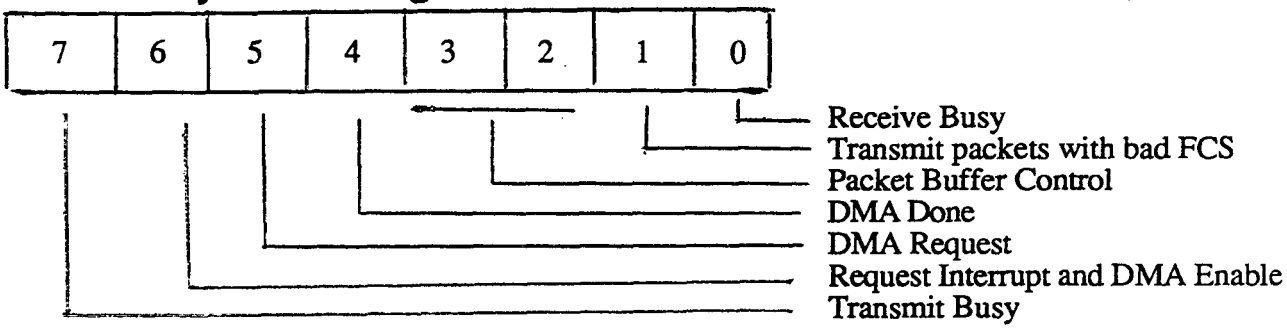
If bit 2 goes to one, the controller transmits the packet buffer contents. The transmission starts at the address left in GP by the software and ends when GP reaches '7FFH'. GP counts up.

If both bits 2 and 3 are one, the controller transmits the packet buffer and simultaneously receives the packet from the network writing the packet back into the beginning packet buffer.

Setting bit 1 of the auxiliary command register to one causes the EtherLink adapter to transmit packets with bad FCS. This bit is useful for testing the receive FCS circuitry.

The low order bit (bit 0) of the Auxiliary Command Register is undefined on EtherLink adapters with assembly number 0345. Software running on EtherLink adapters with assembly number 48-0747 may use this bit, Interrupt Request Enable (IRE), to enable interrupts on packets of interest only. IRE permits the adapter to drive the Interrupt Request signal, IRQ. Jumpers on the adapter card select the interrupt channel (either 3 or 5 on ASSY 0345, or any one of 2 thru 7 on ASSY 34-0780). If IRE is off the EtherLink adapter can drive IRQ only if RIDE is on. Software which does not use DMA should use IRE in place of RIDE. However the cautions described in relation to the RIDE bit and interrupts must be observed.

Auxiliary Status Register



Software starts a DMA transfer by programming the proper channel of the 8237A DMA controller on the system board and setting RIDE, and DMA Request to one on the EtherLink adapter. When the DMA transfer ends, DMA Done goes to one; software clears DMA Done by setting DMA Request to zero.

Receive Busy goes to one whenever the controller is armed to receive a packet; this happens automatically after transmitting a packet, or whenever software sets the Packet Buffer Control to receive or loopback. Receive Busy goes to zero after the controller accepts a packet. Software must wait 800 nanoseconds after receive busy goes to zero before reading the receive status register.

Transmit Busy is meaningful only when the packet buffer control is set for loopback or transmit; while the packet buffer is switched to the bus or is in receive mode Transmit Busy will be set. Transmit Busy remains at one when software starts a transmit by setting the Packet Buffer Control to one. Transmit Busy goes to zero upon a collision or a successful transmission. Software can distinguish between these two cases by examining the transmit status register. Transmit Busy is set to one when software switches the packet buffer control to the system bus (by setting bits 2 and 3 of the AUX COMMAND REG to zero).

EtherLink Adapter Programming

To Transmit a Packet

1. **Set the Packet Buffer Control to zero; this gives the system bus access to the buffer.**

2. **Load the packet into the buffer so that the last byte of the packet coincides with the last byte of the packet buffer. When GP equals 7FF hex it points to the last byte of the packet buffer.**

3. **Load GP so that it points to the first byte of the packet in the buffer.**

4. **Start the transmission by setting the Packet Buffer Control to one.**

5. **Read the transmit status register to determine whether there was a collision or a successful transmission. The transmission terminates when Transmit Busy goes to zero.**

In case of collision, set the Packet Buffer Control to zero, reload GP, and set the Packet Buffer Control to one; this retransmits the packet. Again wait for Transmit Busy to go to zero; then read the transmit status register to determine why the transmission terminated. Receiving packets requires both one time initialization of the controller and manipulation of the EtherLink adapter for each packet that arrives. The one time initialization includes reading the station address PROM, loading the station address registers, and setting the receive command register. In the programming example the routines "getaddr" and "setaddr", read the station address PROM and write the station address registers respectively.

To Receive a Packet

1. **Clear RP and set the Packet Buffer Control to two; this initializes the receive pointer (RP) and allows the controller to load the packet buffer from the network.**

2. **800 nanoseconds after Receive Busy goes to zero, the receive status register has the status of the packet just received. The size of the packet, in bytes, is in RP. Software must set the Packet Buffer Control to zero before reading the packet from the buffer.**

The following code, written in C, initializes the controller, transmits a single packet of 1000 bytes, and then receives well formed broadcast and packets addressed only to the station. The main program is found at the end of the example. The routines "inb", "inw", "outb", and "outw" read and write words and bytes on the IBM PC's I/O bus; the routine "inbs" reads a byte sign extended into a word. All of the examples are polled I/O; no use is made of the interrupt circuitry.

Transmit and Receive Code Example

In the following example, the EtherLink adapter is designated by its development code, IE4.

```
/* the various EtherLink adapter command registers */
#define IE4(num) (0x300+0x10*num)
#define EDLC_ADDR(num) (num)/* EDLC station address, 6 bytes */
#define EDLC_RCV(num) ((num)+0x6)/* EDLC receive command and status */
#define EDLC_XMT(num) ((num)+0x7)/* EDLC XMIT command and status */
#define IE4_GP(num) ((num)+0x8)/* transmit, station address PROM bp */
#define IE4_RP(num) ((num)+0xa)/* receive buffer pointer */
#define IE4_SAPROM(num) ((num)+0xc)/* station address prom window*/
#define IE4_CSR(num) ((num)+0xe)/* IE4 command and status */
#define IE4_BFR(num) ((num)+0xf)/* 1 byte window on packet buffer */

/* bits in EDLC_RCV, interrupt enable on write, status when read */
#define EDLC_NONE0x00/* match mode in bits 5-6, write only */
#define EDLC_ALL0x40/* promiscuous receive, write only */
#define EDLC_BROAD0x80/* station address plus broadcast */
#define EDLC_MULTI0xc0/* station address plus multicast */
#define EDLC_STALE0x80/* receive CSR status previously read */
#define EDLC_GOOD0x20/* well formed packetes only */
#define EDLC_ANY0x10/* any packet, even those with errors */
#define EDLC_SHORT0x08/* short frame */
#define EDLC_DRIBBLE0x04/* dribble error */
#define EDLC_FCS0x02/* CRC error */
#define EDLC_OVER0x01/* data overflow */

#define EDLC_RERROR(EDLC_SHORT|EDLC_DRIBBLE|EDLC_FCS|EDLC_OVER)
#define EDLC_RMASK(EDLC_GOOD|EDLC_ANY|EDLC_RERROR)

/* bits in EDLC_XMT, interrupt enable on write, status when read */
#define EDLC_IDLE0x08/* transmit idle */
#define EDLC_160x04/* packet experienced 16 collisions */
#define EDLC_JAM0x02/* packet experienced a collision */
#define EDLC_UNDER0x01/* data underflow */

/* bits in IE4_CSR */
#define IE4_RESET0x80/* reset the controller */
#define IE4_RIDE0x40/* request interrupt/DMA enable */
#defineIE4_DMA0x20/* DMA request */
#define IE4_EDMA0x10/* DMA done */
#define IE4_LOOP0x0c/* 2 bit field in bits 2 and 3, loopback */
#define IE4_RCVEDLC0x08/* gives buffer to receive */
#defineIE4_XMTEDLC0x04/* gives buffer to transmit */
#define IE4_SYSBFR0x00/* gives buffer to processor */
#define IE4_CRC0x02/* causes CRC error on transmit */
#define IE4_RCVBSY0x01/* receive in progress */
```

```
/* miscellaneous sizes */
#define BFRSIZ0x800/* number of bytes in a buffer */
#define RUNT   60/* smallest legal size packet, no fcs */
#define GIANT 1514/* largest legal size packet, no fcs */

error(s) char *s; {printf("%s ", s);}

/* call DOS to test for keystroke, if its ^C get back to DOS */
sense_key() {char c;
      if (dos(0xb)) {if ((c = dos(8)&0177)==3) exit(0); return(c);}
      else return(0);}

/* low level transmit routines */
ie4reset(base) short base; {
      outb(0xa, 5);/* turn off DMA channel 1 */
      outb(0x21, 0xa0);/* mask interrupt level 5 */
      outb(IE4_CSR(base), IE4_RESET);/* reset them */
      outb(IE4_CSR(base), 0);
      if (inb(IE4_CSR(base)) != 0x80 ) error("Can't reset IE4_CSR");
      if ((inb(EDLC_XMT(base))&0x0f) != 0)
            error("Can't clear EDLC_XMT");
      if ((inb(EDLC_RCV(base))&0x9f) != EDLC_STALE)
            error("Can't reset EDLC_RCV");}

xmt_start(base, size) short base; int size; {
      char c= inb(EDLC_XMT(base));
      outb(IE4_CSR(base), IE4_RIDE); /* make sure out of transmit mode */
      outw(IE4_GP(base), BFRSIZ-size);/* before zapping counter */
      outb(EDLC_XMT(base), EDLC_16|EDLC_JAM|EDLC_UNDER|EDLC_IDLE);
      outb(IE4_CSR(base), IE4_RIDE|IE4_XMTEDLC);}
      /* returns 0 for successful transmit
            1 timed out
            2 collision
            3 data underflow
            4 idle not set after transmit
            5 16 collisions */
```

```
xmt_wait(base, size, stall) short base; int stall, size; {
      int i; char c;
      if ( (inb(IE4_CSR(base))&IE4_XMTEDLC) == 0)
            error("buffer not switched to transmit, xmt_wait");
      i = stall;
      do {
            if ( inbs(IE4_CSR(base)) < 0 ) continue;
            c = inb(EDLC_XMT(base));
            if (c&EDLC_UNDER) return(3);/* underflow */
            if (c&EDLC_16)    return(5);/* 16 successive collisions? */
            if (c&EDLC_JAM)   return(2);/* collision? */
            if (inw(IE4_GP(base))==0x800) {
            if (!(inb(EDLC_XMT(base)) & EDLC_IDLE)) return(4);
            return(0);}}
      while(i-->=0);
      return(1);}

retransmit(base, mode, size) short base, mode, size; {
      int i = 0, org = BFRSIZ-size, k;
      if ( (inb(IE4_CSR(base))&IE4_XMTEDLC) == 0)
      error("buffer not switched to transmit, retransmit");
      while ( inbs(IE4_CSR(base)) < 0 )
            if (++i > 1000) {error("retransmit timed out"); break;}
      outb(IE4_CSR(base), IE4_RIDE); /* make IE4 idle */
      outw(IE4_GP(base), org);
      if ( (inb(EDLC_XMT(base))&(EDLC_JAM|EDLC_16)) == 0 ) return;
      outb(IE4_CSR(base), IE4_RIDE|mode);}

/* returns 0 on failure, 1 on success */
xmt_done(base, size, stall) short base, size, stall; {
      retry: switch(xmt_wait(base, size, stall)) {
            case 1: error("Transmit timed out"); ie4reset(base);
            case 0: return(1);
            case 2:
                  error("Jam");
                  retransmit(base, IE4_XMTEDLC, size);
                  sense_key();
                  goto retry;
            case 3:error("underflow on transmit"); ie4reset(base); break;
            case 4:error("idle not set after xmt"); ie4reset(base); break;
            case 5:ie4reset(base);
                  error("excessive collisions");
                  break;
            default: error("xmt_done: bad argument");}
      return(0);}

/* low level receive routines */
```

```
rcv_start(base, mode) short base; char mode; {
      outb(EDLC_RCV(base), EDLC_NONE);
      outb(IE4_CSR(base), IE4_RIDE);
      outw(IE4_RP(base), 0);
      inb(EDLC_RCV(base));/* he'll discard until we read the status */
      outb(IE4_CSR(base), IE4_RIDE|IE4_RCVEDLC);
      outb(EDLC_RCV(base), mode|EDLC_GOOD);}

rcv_wait(base, stall) short base; int stall; {
      char status; int i = stall;
      do {
            if (inb(IE4_CSR(base))&IE4_RCVBSY) continue;
            status = inb(EDLC_RCV(base))&(EDLC_STALE|EDLC_RMASK);
            if ( (status&(EDLC_ANY|EDLC_RERROR)) != 0 ) return(status);}
      while(i-->=0);
      return(0);}/* timed out */

rcv_chk(status) char status; {
      if (status&EDLC_FCS) error("FCS error");
      if (status&EDLC_DRIBBLE) error("dribble error");
      if (status&EDLC_OVER) error("overflow on receive");
      if (status&EDLC_SHORT) error("size");}

rcv_done(base, stall) short base; int stall; {
      char status;
      if ((status = rcv_wait(base, stall)) == 0) return(0);
      if (status < 0) {error("not fresh status"); return(-1);}
      outb(IE4_CSR(base), IE4_RIDE|IE4_SYSBFR);/* give buffer to processor*/
      outb(EDLC_RCV(base), EDLC_NONE);/* shut down the EDLC */
      rcv_chk(status);
      return(status&0xff);}/* guaranteed to be non-zero at this point */

getaddr(base, cp) short base; char *cp; {int i;
      for(i=0; i<6; i++) {
            outw(IE4_GP(base), i);
            *cp++= inp(IE4_SAPROM(base));}}

setaddr(base, cp) short base; char *cp; {int i;
      for(i=0; i<=5; i++) outb(EDLC_ADDR(base)+i, cp[i]);}

/* fill packet with constant pattern */
fill_pkt(base, size, pat) short base, size, pat; {
      int i; char pathi = pat>>8;
      /* Watch out!  This routine knows that a short is two bytes. */
      outb(IE4_CSR(base), IE4_RIDE|IE4_SYSBFR);
      size= (size+1) & ~1;/* align packet on word boundary */
      outw(IE4_GP(base), BFRSIZ-size);
      for(i=size>>1; i>0; i--) {
            outb(IE4_BFR(base), pat); outb(IE4_BFR(base), pathi);}}
```

```
xmt_pkt(base, size, stall) short base; int size, stall; {
      xmt_start(base, size); xmt_done(base, size, stall);}

rcv_pkt(base, rcv_mode) {
      int status, stallcon = 0x400;
      rcv_start(base, rcv_mode);
      while((status = rcv_done(base, stallcon))==0) sense_key();
      return(status);}

main() {
      char myaddr[6]; int ie4 = IE4(0), size, i;

      /* one time only initialization */
      ie4reset(ie4);         /* leaves buffer switched to system bus */
      getaddr(ie4, myaddr);       /* read station address from PROM */
      setaddr(ie4, myaddr);       /* set the station address */
      printf("3Com IE4 Programming Example Version 1.0\\r\\n");
      printf("My station address is ");
      for (i=0; i<6; i++) printf("%02x ", myaddr[i]&0xff);
      outb(EDLC_RCV(ie4), EDLC_ALL|EDLC_GOOD);

fill_pkt(ie4, 1000, 0x5555); /* fill packet with constant pattern*/
      xmt_pkt(ie4, 1000, 1000);/* transmit packet of 1000 bytes */
      /* receive those packets */
      printf("\\r\\nStart receive loop\\r\\n");
      while (1)
            if (rcv_pkt(ie4, EDLC_ALL|EDLC_GOOD) > 0) {
                  size = inw(IE4_RP(ie4));/* that's the size in bytes */
                  printf("%d ", size);}
            else ie4reset(ie4);}
```

Setting the Jumpers

The factory jumper settings on the EtherLink adapter work with software supplied by 3Com. Only extraordinary circumstances or use with non-3Com software require alteration of the factory settings.

Jumpers are used on the EtherLink adapter to select the I/O configurations of the controller. Jumpers on the adapter select the following:

- Interrupt channel used by the adapter.
- DMA channel used by the adapter. Two jumpers are required to select a DMA channel, DMA Request (DRQ) and DMA Acknowledge (DACK). These jumpers must select the same channel.
- The base I/O address of the adapter.

- The base memory address and enable for the available 4K by 8 onboard PROM.
- Ethernet connector (which one of the two connectors on the backplate is attached to the Ethernet). The EtherLink adapter may be attached to the network through either the BNC connector (silver cylindrical connector) using the onboard transceiver or through the DIX connector (the 15 pin D-connector) using a standard Ethernet transceiver.

The two versions of the EtherLink adapter have different degrees of selectability. Table 2 and Table 3 below describe the functions and factory settings for the jumpers on ASSY 0345 and ASSY 34-0780 respectively. The DIX/BNC jumper on ASSY 0345 is a shorting block which fits over 2 of 3 pins. The DIX/BNC selection on ASSY 34-0780 is done using an eight position shorting plug which fits in 1 of 2 sixteen pin IC sockets.

NOTE: BE CAREFUL when changing the jumpers. If the jumpers are improperly installed, it is possible to short together +5V and GND.

Table 2. Jumper Function and Factory Setting on ASSY 0345

Function	Bus Signal	Legend	Factory Setting
DIX/BNC Select		SW1	BNC
DMA Channel Select	DRQ 1,3	JP1	DRQ 1
	DACK 1,3	JP2	DACK 1
Interrupt	IRQ 3,5	JP3	IRQ 3
I/O Base Address	AD 09		1, not selectable
	AD 08	JP4	1
	AD 07	JP5	0
	AD 06	JP6	0
	AD 05	JP7	0
	AD 04	JP8	0
PROM Base Address	AD 19		1, not selectable
	AD 18	JP9	1
	AD 17	JP10	1
	AD 16		0, not selectable
	AD 15		1, not selectable
	AD 14	JP11	1
	AD 13	JP12	0
	AD 12	JP13	0
PROM Enable		JP14	Disabled

Table 3. Jumper Function and Factory Settings on ASSY 34-0780

NOTE: BE CAREFUL when changing the jumpers. If the jumpers are improperly installed, the adapter will malfunction.

Function	Bus Signal	Legend	Factory Setting
DIX/BNC Select		BNC or DIX	BNC
DMA CHANNEL Select	DRQ 1,2,3 DACK 1,2,3	REQ 1,2,3 ACK 1,2,3	DMA 1 DMA 1
Interrupt	IRQ 2-7	2-7	INT 3
I/O Base Address	AD 09 AD 08 AD 07 AD 06 AD 05 AD 04	9 8 7 6 5 4	1 1 0 0 0 0
PROM Base Address	AD 19 AD 18 AD 17 AD 16 AD 15 AD 14 AD 13 AD 12	19 18 17 16 15 14 13 12	 1 1 0 1 1 0 0
PROM Enable		Enable/Disable	Disable

Ethernet Interface

The EtherLink adapter provides two options for connection to an Ethernet, selectable by a jumper. The first is the standard, DA-15 DIX outlet, which uses fully compatible signalling. This outlet attaches to a standard transceiver cable, which in turn is connected to any Ethernet transceiver for use with standard "thick" Ethernet cable.

The other Ethernet interface uses the onboard transceiver and is designed to be used with, "thin", low-cost (50 ohm) RG-58A/U coax as the Ethernet media. The integral transceiver is attached to the cable via a single BNC connector on the box, to be mated with a BNC "T" pre-installed on the RG-58 coax. The station can be coupled and uncoupled without affecting network operation. The integral transceiver provides complete electrical isolation. The RG-58 Ethernet is electrically compatible with standard Ethernet coax. In fact, the RG-58 Ethernet can be attached to standard Ethernet by simply coupling them with an N-series/BNC adapter, and EtherLink adapters can communicate with any other station on the RG-58 or yellow coax. One drawback of the RG-58 Ethernet is that the distance limitation is more severe: approximately 300 meters of an RG-58-only segment.

Known Bugs Found on Early Production Boards

Receiving a runt packet can lock up the controller while in receive mode regardless of the setting of Detect Short Frames, bit three of the Receive Command Register. Reading the Receive Status Register after reception of a runt permits reception of further packets. This is not a problem for software using polled I/O; such software can read the Receive Status Register in the same loop that checks the Auxiliary Status register for Receive Busy. Interrupt driven software must set Detect Short Frames to insure that runts generate interrupts; interrupt level software can then read the status register in order to receive subsequent packets.

It is possible to get one false interrupt for each write to the Receive or Transmit Command Register. Software can distinguish false interrupts from true ones by examining Receive Busy and Transmit Busy, bits zero and seven respectively of the Auxiliary Status Register. Well written software would routinely check for these conditions before taking action to disturb the state of the controller. However, software running at main program level cannot prevent false interrupts this way because an interrupt would occur before main program level could read the status register. Interrupt level would then be responsible for clearing the interrupt by reading the status registers.