RC 11189 (#50146) 5/9/85
Computer Science 9 pages

Research Report

PC Protocol Card: An Interface to the ALAN Network Switch (User Guide)

Yannick Thefaine
Brent Hailpern
Lee Hoevel
Sandy Liles
Kenneth J. Perry

IBM T. J. Watson Research Center
Yorktown Heights, NY

IBM Research Division
San Jose · Yorktown · Zurich

RC 11189 (#50146) 5/9/85
Computer Science 9 pages

PC Protocol Card: An Interface to the ALAN Network Switch (User Guide)

Yannick Thefaine
Brent Hailpern
Lee Hoevel
Sandy Liles
Kenneth J. Perry

IBM T. J. Watson Research Center
Yorktown Heights, NY

Introduction:

The PC Protocol Card is an SDLC communication adapter to interface the IBM Personal Computer with the Alan local area network. It consists of a microprocessor (Intel 8086), a four-channel DMA (Intel 8237) and a serial protocol controller (Intel 8274). Four-Kbytes of ROM support the microprocessor and thirty-two Kbytes of memory are allocated for program storage and communication buffers. The on-board processor manages communication for the host processor resulting in little host overhead during PC to PC communication. PC control commands are issued through a bi-directional port, while data exchange is accomplished with the DMA. A multiprotocol serial controller (MPSC) circuit converts parallel data to serial data. The serial link supports an RS-422 driver/receiver. Twisted-pair cabling connects PCs to the ALAN local area network switch. See "Alan: A (Circuit-Switched) Local Area Network", IBM RC 10960.

Specifications:

Processor.....Intel 8086/5
Read Only Memory.....4 Kbytes
Random Access Memory.....32 Kbytes
Direct Memory Access.....Intel 8237/5
Clock cycle.....210 nsec
Multiprotocol Serial Controller...Intel 8274
Serial Link Data Rate.....895 kbaud
Serial Link Interface.....RS-422 Twisted-pair
Power requirements.....2.75 Amps at 5 volts

Processor (Intel 8086):

The processor (8086) executes a control program stored in ROM (Read Only Memory). Two-Kbytes of ROM are allocated for even addresses and two-Kbytes for odd addresses. The address space is from FF000H to FFFFFH. The Protocol Card Rom code begins at (F000:F000).

When a hardware reset occurs, a jump is made to absolute location (F000:FFF0).[1]

The Protocol Card is initialized only on booting the system. The handshaking sequence between Protocol card and PC is as follows:

- send FFH to card in order to boot card
- wait for 7CH or less than 08H from card
- send 7EH to card to continue
- wait for zero from card
- send zero to card to clear status register
- if 7CH from card, mark good card setup, else mark bad

The ROM code checks the registers, flags, and DMA channels. If OK, memory refresh is activated and memory is checked leaving the memory cleared to zero. The Protocol card then waits for the PC to request the initialization status. When the status is requested, the 8-bit port is enabled by writing 01H to I/O port 60H (see below), the status is sent, and the card jumps to the ALAN link support.

Direct Memory Access (Intel 8237):

The Direct Memory Access integrated circuit (Intel 8237) contains four independent channels with a full 64k address and word count capability. Channel 1 and 2 are interfaced with the receiver and transmitter buffers of a Multiprotocol Serial Controller (8274). Channel 0 is used for storage refresh, while channel 3 handshakes with the PC DMA channel 1 for data block transfer. Prior to a data block tranfer, the PC and the Protocol Card DMA's must be properly initialized. Direction of data flow is predetermined by setting a register (see below). The Protocol Card must initiate the start of DMA data transfer.

DMA PORT ADDRESSES:

- 20H , 22H - count and address of channel 0
- 24H , 26H - count and address of channel 1
- 28H , 2AH - count and address of channel 2
- 2CH , 2EH - count and address of channel 3
- 36H - mode register
- 38H - clear internal flip/flop
- 3EH - general mask register
- 34H - single mask register

The DMA is programmed as referenced in the Intel manual.[2]

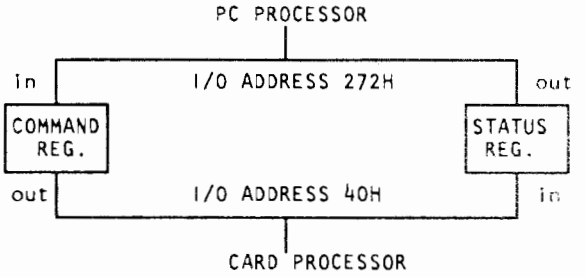
---

[1] Intel Microsystem Components Handbook Vol.1 p.3-343, 1985

[2] Intel Microsystem Components Handbook Vol.1 p.2-88, 1985

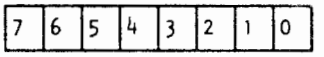
PC to Protocol Card interface:

The PC accesses the Protocol Card through a pair of bi-directional registers. I/O port address 272H is dedicated for the PC to Protocol Card interface and port address 40H is assigned to the Protocol Card-PC interface.



The PC's processor (8088) supports an eight-bit bus whereas the Protocol Card processor (8086) supports a sixteen-bit bus. The PC is interfaced with the lower part of the 8086 bus. This bus can only access bytes located at even address boundaries. Therefore, the Protocol Card's DMA steps by two bytes for each data transfer to the PC.

The Protocol Card has a one-byte Block Transfer register at address 80H. Setting the register's least significant bit initiates a data transfer from the Protocol Card to the PC. Setting the two least significant bits initiates data transfer from the PC to the card memory.

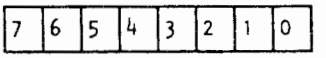


BLOCK TRANSFER REGISTER
ADDRESS 80H

- bit0 =1 transfer from Protocol Card to PC
- bit1 =1 transfer from PC to Protocol Card
- bit2-7 unused

The Block Transfer register must be set by the 8086 processor after DMA setup.

An Interface Control Register at I/O address 60H is accessed by the 8086 processor. The control signals from the Protocol Card to the PC are controlled by this register. The least significant bit, if on, activates the control lines; if the least significant bit is off, the control lines are in a high impedance state. When the 8086 processor reads this register, the Protocol Card personalized address is returned in the high nibble of the register and the state of the interface is indicated by bit zero.



PC INTERFACE CONTROL REGISTER
ADDRESS 60H

- bit0 =0 all signals in high impedance

- bit0 = 1 all signals are operational
- bit1 = 1 used to interrupt the PC (IRQ2)
- bit2-7 unused for a write
- bit1-3 unused for a read
- bit7-4 address of Protocol Card for a read; bit4 is the LSB.

The card is personalized by setting a four-bit binary switch. As described above, the PC interface control command enables or disables the interface to the PC. When the interface is disabled, other features can use the same DMA channel and I/O port address. When the least significant digit of the command code matches the value set in the address switches on the Protocol Card, the card is enabled. If it does not match, the Protocol Card is disabled.

PC - Card common port protocol

The PC makes requests of the card through the common port using a one-byte protocol. The format of the request byte is:

- 01 - send message to remote PC
- 02 - receive message from remote PC
- 03 - PC request for file buffer
- 04 - state request
- 05 - request for card's port ID
- 06 - request for card to force interrupt to PC
- 07 - ask card to send awake to remote port
- 1D - poke byte into RAM
- 1E - peek at RAM (n bytes)
- 1f - download to card through common port (not implemented)
- 2n - card status request
- 4n - reset request (n : 0 - card, not 0 - switch)
- 6n - open to port n
- 8n - close request
- An - send header for file transfer
- Cn - send buffer for file transfer

META-PROTOCOL PC REQUESTS:

- FC - jump to test code in RAM
- FD - set break point
- FE - resume in ROM code
- FF - reboot Protocol Card

The handshake protocol between the PC and Protocol Card excludes the use of 00H and 7FH as valid data. Therefore, to send DWORD data items, bits 0-6 are sent first with bit 8 = 1 to handle 0. Bits 8-13 are mapped into bits 0-6 of another data byte with bit 8 = 1 again. Thus, integer values sent through the common port are limited to 0..16383.

Serial Protocol Controller (Intel 8274)

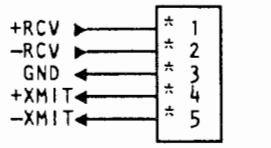
The Multiprotocol Serial Controller consists of two independent full-duplex channels (A and B). Each channel can be individually initialized. The channels are nearly identical, except that

vector interrupt is supported only by channel B. The MPSC is programmed as referenced in the Intel manual.[3] Port address C6H is used to read the channel A status registers (RR0,RR1) or to write the Command/Parameter (WR0...WR7). All registers are accessed via the command port (WR0). An internal pointer register selects which of the command or status registers will be read or written during a command/status access. Port address C4H is used to read or write the channel A data buffer. To start transmission, the first byte of a frame must be written to this register. If more bytes have to be transmitted, the DMA takes the bus and fetches successive bytes from memory. This process continues until the byte DMA count is exhausted.

On a 'character available' in the MPSC status register (RR0), the DMA stores the incoming bytes in the memory receive buffer. The MPSC can be programmed to handle protocols such as MONOSYNC, BYSYNC, SDLC or HDLC. We use SDLC. A frame consists of an opening flag followed by data bytes and the end-of-message closing flag. The CRC field is sent but not processed by the receiving processor. A data synchronized clock is generated on the Protocol Card to sample data bits as they are received. The clock frequency is 14.31818 Mhz/16. Tranmission is done at the same baud rate.

RS-422 interface:

The serial interface consists of a pair of RS-422 devices. The link is twisted-pair wires. The RS-422 connector is a nine pin female D-shell connector. Only pins one through five are used.



RS-422 Connector

General Program Sequence:

1. After boot, run through the card checkout code to find any failing components and setup memory refresh.
2. Wait for a status request from the PC and respond. If checkout is good, initialize data variables, and get own port ID from the cluster switch.If initializing errors occured, the card will send to the PC:
   - 01 - flag for register error
   - 02 - DMA channel error
   - 03 - refresh error
   - 04 - memory error
3. Setup DMA to read one byte from MPSC. This is always active if no other DMA transfer is active - equivalent to a read enable.
4. Poll until one of the following occurs:
   - PC request via the 8-bit port
   - DMA completion
   - Timer = zero (timeout)
   - Process action and return to poll

DATA VARIABLES AREA:

[3] Intel Microsystem Components Handbook Vol.1 Application Note AP-134, 1985

Data variables are located in RAM addressed from 0000:0000. The variables cannot have initially defined values since the RAM test clears memory.

Software specifications:

1. PROTOCOL CARD ROM SOFTWARE: The Protocol Card software provides an interface between the PC application program (via a device driver) and a remote PC or (Protocol Card) via the ALAN switch). The card communicates to external interfaces through:
   - 8274 Intel MPSC (SDLC) - to the cluster switch
   - 8237 Intel DMA - to PC device driver
   - 8-bit I/O port - to PC device driver

   The protocol used between the card and the switch is explained below. The card software is laid out in the following format:
   - Data variables start at 0000:0000, the beginning of the RAM area.
   - Code area begins at F000:F000, the start of the standard ROM code for the 8086 processor.
   - Boot area begins at F000:FFF0, the address for the 8086 reset sequence.

   The data area is consecutive bytes of storage for local variables, but communication via the 8237 DMA controller uses only the even bytes of memory. Therefore, data transfers require twice the normal storage for buffer space.
2. DEVICE DRIVER INTERFACE SOFTWARE:

   So that PC application programs and users may use the network, a device driver(DD) is provided to send requests to the Protocol Card(CC). Access to the DD is via function calls to DDWRITE. DDWRITE is an entry point in the module DDSUB.OBJ.

   FUNCTION DDWRITE(MSGPTR:WORD): WORD; EXTERN;

   RESPONSE: value returned by function is request dependent.

   REQUEST: DD function to be invoked

   MSGPTR@ FORMAT:

REQUEST (word)	TARGET (word)	OFFSET (word)	SEGMENT (word)

   a. 1 - Open to another PC (requires target)
      - Response: 0 = ok
      - 254 = already open
      - 255 = not opened

   b. 2 - Close path

   c. 3 - Send message (offset+segment --) data buffer)
      - Response: 0 = ok
      - 255 = path not opened

   d. 4 - Get message (offset+segment --) data buffer)
      - Target: maximum number of characters to be received
      - 0 = no maximum

- Response: 0 = ok
- 255 = no buffer available

e. 5 - Status request (offset+segment --› data buffer)
- Response in data buffer
- integer: 0
- byte1: x'80' = port closed
- x'4n' = open to port n
- byte2: x'nn' = number of messages queued

f. 6 - Reboot CC

g. 7 - Reboot switch port

h. 8 - Send file (requires target and buffer address)
- Target: buffer size mod 64
- Response: 0 = buffer sent
- 255 = not opened/sent

i. 9 - Receive file (requires buffer address)
- Response: 0 = buffer received
- 255 = buffer not available

j. 10 - State request
- Response: 0n = switch state
- 255 = switch not responding

k. 11 - Who am I ?
- Response: n = my ID (0-31)
- 255 = card not yet initialized

l. 12 - Set/reset interrupt vector (IRQ2)
- 1 = set
- 0 = reset
- bufoff,bufseg = address of interrupt routine

m. 13 - Initiate card invoked interrupt

n. 14 - Send bytes (requires target and buffer address)
- Target: buffer size (0-7)
- Response: 0 = buffer sent
- 255 = not opened/sent

o. 15 - Return branch entry point to DD (offset+segment --›data buffer). This is used by interrupt handlers that cannot call the device driver through the conventional DOS interrupt.

- Response in data buffer:
- +0 = 0
- +2 = offset of branch entry
- +4 = segment of branch entry

p. 29 - Poke byte (requires message buffer)
- message length = 6
- message data = nnaaaa (nn & aa = 30-3f for hex nibbles)

- nn = byte to be poked

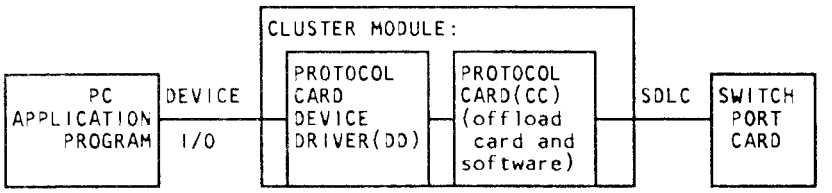
q. 30 - Peek (requires message buffer)

- message length = 6
- message data = nnaaaa (nn & aa = 30-3f for hex nibbles)
- nn = number of bytes to read
- aaaa = starting address to be read

TARGET: port number (0-31) to communicate with, if applicable
OFFSET: relative address of user buffer to store data reply,if applicable
SEGMENT: segment address corresponding to the relative address
MSGBUF: message buffer contains 2 byte length (exclusive) and data

Protocol For Cluster Module (Protocol Card / Device Driver):

1. INPUTS:
   - local - requests from PC application programs
   - remote - SDLC messages from (or transmitted via) the Alan switch port card
2. OUTPUTS:
   - local - responses to PC application programs
   - remote - SDLC messages to (or transmitted via) the Alan switch port card



3. "SDLC" MESSAGE HEADERS (one-byte protocol):
   - 01H - close connection
   - 02H - reset switch port
   - 03H - request to switch port for my ID
   - 04H - request to switch for state
   - 05H - request to switch for remote ID
   - 06H - awake request to remote
   - 0001nnnn len = (nnnn+1) * 64
   - 001xxxxx open request to port xxxxx
   - 010xxxxx ack to port xxxxx
   - 011xxxxx response to remote open with my ID
   - 100xxxxx not used
   - 101xxxxx variable length message
   - 110xxxxx variable length message (xxxxx-1 * 32)
   - 1110ABCC switch response to state request where:
     a. A - current state
     b. B - previous state
     c. CC - why changed
   - F0H - 1111xxxx other (not assigned)

Acknowledgments:

Many thanks to Dale Junod, William Hoyt, Michael Cassera, and the valuable team of the Research Center Central Services whose support permitted us to build this hardware. We also thank Julio Escobar (MIT graduate summer student) for the help he gave us in debugging the hardware during development.