PCnet Family Software Design Considerations

AMD

Application Note

INTRODUCTION

The purpose of this application note is to help you understand the software interfaces of the PCnet family of devices so you'll be able to design an efficient, reliable device driver. Key programming sequences are presented and tips on the operational characteristics are given where necessary.

ACCESS OPERATIONS

I/O Resources

Controller I/O resources coexist with other system resources in the I/O address space of your system. Controller resources are located at specific offsets from the base I/O address you assigned to the device, and are organized as indicated in Table 1.

Table 1. 16-Bit I/O Resources

Offset (Hex)	Bytes	Register
00	16	Address PROM
10	2	Register Data Port (RDP)
12	2	Register Address Port (RAP)
14	2	Reset
16	2	Configuration Data Port (IDP/ BDP)
18	2	Vendor Specific Word
1A	2	Reserved
1C	2	Reserved
1E	2	Reserved

For the default case of Word I/O accesses, the controller responds to 16-bit accesses at offsets 00h through 16h. Offset 18h, the Vendor Specific Word, is not implemented by the controller. Address offsets 1Ah through 1Eh are reserved for future AMD use and should not be implemented if upward compatibility to future AMD devices is desired.

The Register Address Port is shared by the RDP and the IDP/BDP to save I/O locations. The Vendor Specific Word (VSW) is not implemented; this particular I/O address is reserved for customer use and will not be used by future AMD products. If more than one Vendor Specific Word is needed, it is suggested that RAP also be shared with the VSW.

For the optional Double Word I/O access mode (see Table 2), which is available only in the PCnet-32 and PCnet-PCI controllers, the controller responds to 32-bit accesses at offsets 0x00 through 0x1C. Except for CSR88 all registers contain only two bytes of valid data in bits 15-0. The upper two bytes are reserved for future use. Reserved bits must be written as zeros, and when read, are undefined.

Table 2. 32-Bit I/O Resources

Offset (Hex)	Bytes	Register
00	4	Address PROM
04	4	Address PROM
08	4	Address PROM
0C	4	Address PROM
10	4	Register Data Port (RDP)
14	4	Register Address Port (RAP)
18	4	Reset
1C	4	Bus Configuration Data Port (BDP)

The Register Address Port is shared by the RDP and the BDP to save registers.

Bus Slave Operation

Your software automatically causes bus slave operation whenever you attempt writing or reading the I/O resources of a PCnet device. A PCnet device enters slave mode operation during I/O accesses when the $\overline{IOR}$ or $\overline{IOW}$ pins (PCnet-ISA/PCnet-ISA$^+$ only) are asserted. This mode is used for accessing the Register Address Pointer (RAP), the Register Data Port (RDP), and other I/O resources.

Publication# **19669** Rev: **A** Amendment/**0**
Issue Date: **March 1995**

I/O Register Accesses

To access a Control Status Register, you must complete a two step operation. First, you perform an I/O write to the RAP using the appropriate CSR number as your data. Then, you perform an I/O read or write access to the RDP. Note that the contents of the RAP is latched and is not changed until rewritten; therefore, the register select cycle is not required in order to repeatedly access the same CSR.

These software accesses should be coded as 16-bit accesses, even if the PCnet device is hardware configured for 8-bit I/O bus cycles. It is acceptable (and transparent) for the motherboard to turn a 16-bit software access into two separate 8-bit hardware bus cycles. The motherboard accesses the low byte before the high byte and the PCnet device has circuitry to specifically support this type of access.

To access an ISA Configuration Register (ISACSR) on PCnet-ISA and PCnet-ISA$^+$ devices, the RAP must be written first with the register number, followed by the read or write access to the IDP. These software accesses should be coded as 16-bit accesses.

Performing a single I/O read from the reset register causes an internal device reset. The I/O access may be 8 or 16 bits wide, and I/O write accesses are ignored.

Address PROM Access

By performing I/O reads to offsets 0x00 to 0x0F from the I/O Base Address you can read the contents of the address PROM (e.g., the 48-bit Ethernet address or vendor specific information). For PCnet-ISA or PCnet-ISA$^+$ systems, reading these locations causes accesses to an external PROM. For PCnet-32 or PCnet-PCI reading these locations causes accesses to internal registers that are loaded from an external serial EEPROM after a hardware reset.

PCnet-ISA and PCnet-ISA$^+$ support 8 and 16-bit I/O accesses while PCnet-32 and PCnet-PCI support 8, 16, or 32-bit accesses.

Boot PROM Access (PCnet-ISA/ PCnet-ISA$^+$ Only)

The boot PROM is a memory resource located at the address selected by the IOAM1-0 pins in bus master mode or the $\overline{\text{BPAM}}$ input in shared memory mode. It may be software accessed as an 8-bit or 16-bit resource but the latter is recommended for best performance.

INITIALIZATION BLOCK

Each PCnet Ethernet controller is custom configured by way of a programmable Initialization Block which is read from system memory.

The Initialization Block must be allocated as a contiguous chunk of memory and may reside at almost any address in system memory. There are two constraints on the Initialization Block address. The full address must fit in the address fields provided in CSR1 and CSR2 and the address must be aligned on a 16-bit word boundary (32-bit boundary for 32-bit mode).

Table 3. 16-Bit Initialization Block

15		0	
MODE			IADR+00
PADR[15:0]			IADR+02
PADR[31:16]			IADR+04
PADR[47:32]			IADR+06
LADRF[15:0]			IADR+08
LADRF[31:16]			IADR+10
LADRF[47:32]			IADR+12
LADRF[63:48]			IADR+14
RDRA[15:0]			IADR+16
RLEN	Reserved	RDRA[23:16]	IADR+18
TDRA[15:0]			IADR+20
TLEN	Reserved	TDRA[23:16]	IADR+22

During your system design process, you must make certain decisions about how the PCnet controller will be configured in order to meet the requirements of your implementation.

You then design your device driver to allocate and setup the Initialization Block using operational parameters which meet your requirements. Your driver is also responsible for starting the initialization procedure by writing the appropriate control bits in Control Status Register 0.

By default, the Initialization Block for the PCnet family of controllers is twelve 16-bit words and may aligned on any word boundary.

For PCnet-32 and PCnet-PCI versions of the controller, which are capable of operating in 32-bit mode, the Initialization Block may be configured as seven double words aligned on a double word boundary (see Table 4). When designing your device driver, remember that the 32-bit version of the Initialization Block employs afield layout which is somewhat different from the16-bit layout.

Table 4. 32-Bit Initialization Block

31				0	
TLEN	RES	RLEN	RES	MODE	IADR+00
PADR[31:0]					IADR+04
Reserved		PADR(47:32)			IADR+08
LADRF[31:0]					IADR+12
LADRF[63:32]					IADR+16
RDRA[31:0]					IADR+20
TDRA[31:0]					IADR+24

Your software must load the 24 or 32-bit physical address of the initialization block into CSR1 and CSR2. Remember that writing to any CSR is a two-step process: First, write the CSR number to the RAP, then write the data to the RDP. Also, note that the address stored in CSR1 and CSR2 is a physical address, not a 80x86 segment:offset pair.

Use of the initialization block is optional. Your software can write directly to the registers that the initialization procedure normally loads. Also, your software can modify any register after it has been loaded by the initialization procedure. If you don't use the initialization block, you must be careful not to set the INIT bit in CSR0. This would cause the controller to load whatever garbage the contents of CSR1 and CSR2 happen to point to.

If the initialization block is not used, then the software must initialize the following registers directly:

CSR 8-11	Logical Address Filter
CSR 12-14	Physical Address
CSR 15	Mode
CSR 24-25	Base Address of RCV Ring
CSR 30-31	Base Address of XMT Ring
CSR 47	Polling Interval (Not necessary for PCnet-ISA and PCnet-ISA$^+$)
CSR 76	RCV Ring Length
CSR 78	XMT Ring Length

RLEN and TLEN Fields

The receive length (RLEN) and transmit length (TLEN) fields of the Initialization Block are used to program the number of entries in the receive and transmit descriptor rings. See Table 5.

For programming these fields, you must first choose one of the eight or ten possible ring sizes, determine the correct encoded value, then write this value to the RLEN or TLEN field.

Table 5. RLEN/TLEN Encoding

RLEN/TLEN	Number of Descriptors
0000	1
0001	2
0010	4
0011	8
0100	16
0101	32
0110	64
0111	128
1000	256*
1001	512*
11XX	512*
1X1X	512*

**32-bit mode only*

You perform this process independently for each descriptor ring because your descriptor rings are not required to be the same size.

If your implementation requires descriptor rings of other sizes, from 1 to 65535 entries per descriptor ring is possible, you may override the standard values programmed in the RLEN and TLEN fields. Information on configuring larger or odd size descriptor rings is provided later in this section.

RDRA and TDRA Fields

The Receive Descriptor Ring Address (RDRA) and Transmit Descriptor Ring Address (TDRA) fields are programmed with the physical address of the receive and transmit descriptor rings which you have allocated in system memory. Note that these are 32-bit physical

addresses, not the logical segment:offset or selector:offset pointers used by 80X86 processors.

LADRF Field

The Logical Address Filter (LADRF) is a programmable 64-bit mask that is used to accept incoming packets based on Logical (Multicast) Addresses. If the first bit of an incoming destination address is a logical zero, the address is a physical address and is compared against the physical address that was loaded through the Initialization Block. If the first bit of an incoming address is a logical one, the address is deemed a logical address and is passed through the logical address filter.

The incoming logical address is sent through the CRC generator circuit producing a 32-bit result. The high order 6 bits of the result are used to select one of the 64 bit positions in the Logical Address Filter. If the selected filter bit is a one, the address is accepted and the packet is copied to a receive buffer.

The broadcast address, which is all ones, does not go through the Logical Address Filter and is normally enabled, for reception regardless of the contents of LADRF. The reception of broadcast packets can be disabled by writing to the MODE register (CSR 15). If the Logical Address Filter is loaded with all zeroes, all incoming logical addresses except broadcast will be rejected.

Note that this is not a perfect filter. It just reduces the number of multicast frames the controller accepts. Depending on how the upper layer uses multicast frames, your driver probably should maintain a list of acceptable multicast addresses and throw away any multicast frame whose address does not appear in the list.

PADR Field

The physical address (PADR) field is programmed with a 48-bit value representing the unique node address assigned by the IEEE and used for internal address comparison. PADR[0] is the first address bit transmitted on the wire, and must be zero. The six-hex-byte nomenclature used by the IEEE maps to the PCnet family address register as follows: the first byte is PADR[7:0] with PADR[0] being the least significant bit of the byte. The second IEEE byte maps to PADR[15:8], the third, fourth, fifth and sixth IEEE bytes map to PADR[23:16], PADR[31:24], PADR[39:32], PADR[47:40].

MODE Field

The Mode Register (CSR15) allows alteration of the chip's operating parameters. The Mode field of the Initialization Block is copied directly into CSR15. Normal operation is the result of configuring the Mode field with all bits zero.

DESCRIPTOR RING ACCESS MECHANISM

Transmit and Receive Descriptor Rings

The basic organization of buffer management is a circular queue of tasks in memory called a descriptor ring with transmit and receive operations described by separate descriptor rings.

Each descriptor ring must be allocated as a contiguous area of memory and typically consists of from 1 to 128 multi-word entries called descriptors. Given this organization, from 1 to 128 tasks may be queued on a descriptor ring awaiting execution by the PCnet device. Information on configuring larger descriptor rings is provided later in this section.

Table 6. 16-Bit Transmit Descriptor

Address	15	14	13	12	11	10	9	8	7-0
CXDA+00h	TBADR[15:0]								
CXDA+02h	OWN	ERR NO_ FCS	ADD_/	MORE	ONE	DEF	STP	ENP	TBADR[23:16]
CXDA+04h	1	1	1	1	BCNT				
CXDA+06h	BUFF	UFLO DEF	EX	LCOL	LCAR	RTRY	TDR		

**CXDA = Current Transmit Descriptor Address*

Table 7. 16-Bit Receive Descriptor

Address	15	14	13	12	11	10	9	8	7-0
CRDA+00h	RBADR[15:0]								
CRDA+02h	OWN	ERR	FRAM	OFLO	CRC	BUFF	STP	ENP	RBADR[23:16]
CRDA+04h	1	1	1	1	BCNT				
CRDA+06h	0	0	0	0	MCNT				

**CRDA = Current Receive Descriptor Address*

For PCnet-ISA and PCnet-ISA$^+$ controllers, transmit and receive descriptors are 8 bytes long. The PCnet-32 and PCnet-PCI controllers use either 8- or 16-byte descriptors, depending on the value of the SWSIZE field in CSR58. The larger descriptors contain all of the information in the smaller ones plus some extra information. The differences are as follows:

- The 16-byte descriptors support 32-bit addresses whereas the 8-byte ones support 24-bit addresses.
- The larger receive descriptors contain two extra fields: a receive collision count and a runt packet count that indicate the number of collisions detected and the number of runt packets that were addressed to this station since the last packet was received.
- The 16-byte transmit descriptor has an extra field for the transmit retry count, which is the number of times the controller had to back off while attempting to transmit this packet.
- The 8-byte descriptors must be aligned on 8-byte boundaries, while the 16-byte descriptors must be aligned on 16-byte boundaries.
- The locations of fields within the descriptors are different for the two descriptor types.

Transmit descriptors contain a buffer address pointer, status, and the number of bytes to transmit from the corresponding buffer. Receive descriptors contain a buffer address pointer, status, the size of the receive buffer in bytes, and the size of the received message, in bytes.

The only limitations on a descriptor ring address are that the address must fit in the address fields (RDRA and TDRA) in the Initialization Block, and address bits 1:0 for 8-byte descriptors or bits 2:0 for 16-bytedescriptors must be zero in order to meet the alignment requirement.

The BCNT fields of the transmit and receive descriptors are 12-bit negative numbers representing the twos complement of the buffer size in bytes. The MCNT field of the receive descriptor is a 12-bit positive number representing the length of the received message in bytes.

For either transmit or receive operations, multiple data buffers may be chained together to accommodate a packet which is longer than the current buffer size. On transmit, when you know a packet occupies more than one buffer, you use the Start of Packet and End of Packet bits to inform the controller of the situation. On receive, when the controller fills the first buffer and uses additional buffers, it informs you of the situation by using the Start of Packet and End of Packet bits.

To ensure proper queuing and de-queuing of message buffers, each buffer descriptor is owned by one entity at a time: the PCnet device or the host device driver. The OWN bit in each descriptor indicates whether the descriptor entry is owned by the host (OWN = 0) or by the controller (OWN = 1). On transmit, you set this bit after filling the data buffer, and the controller clears this bit after transmitting the contents of the buffer. On receive, the controller clears this bit after filling the buffer, and you set this bit after emptying the buffer.

***Caution**: Once you set the OWN bit you relinquish ownership of the descriptor and must not make further changes to any field in the corresponding descriptor.*

The location of the descriptor rings, and their lengths, are presented to the controller in the Initialization Block which is accessed during the initialization procedure performed by the PCnet device.

Table 8. 32-Bit Transmit Descriptor

Address	31	30	29	28	27	26	25	24	23-16	15-12	11-0
CXDA+00h	TBADR[31:0]										
CXDA+04h	OWN	ERR NO_ FCS	ADD_/	MORE	ONE	DEF	STP	ENP	RES	1111	BCNT
CXDA+08h	BUFF	UFLO DEF	EX	LCOL	LCAR	RTRY		TDR		RES	TRC*
CXDA+0Ch	RESERVED										

CXDA = Current Transmit Descriptor Address

**TRC is four bits wide (bits 3-0); bits 11-4 are reserved.*

Table 9. 32-Bit Receive Descriptor

Address	31	30	29	28	27	26	25	24	23-16	15-12	11-0
CRDA+00h	RBADR[31:0]										
CRDA+04h	OWN	ERR	FRAM	OFLO	CRC	BUFF	STP	ENP	RES	1111	BCNT
CRDA+08h	RCC								RPC	0000	MCNT
CRDA+0Ch	RESERVED										

CRDA = Current Receive Descriptor Address

When initializing the controller with the Initialization Block, the descriptor ring size fields, RLEN and TLEN, are used to specify the number of descriptors per ring. The size and encoding characteristics of these fields limit you to 128 descriptors per ring in 16-bit mode (512 descriptors per ring in 32-bit mode) with the number of descriptors limited to a power of two.

However, you are able to manually specify a descriptor ring with up to 65535 entries by directly writing the receive and transmit ring length registers (CSR76, CSR78 respectively). In addition to being able to specify a larger number of descriptors, you may specify any number of descriptors, not just those which are a power of two. If you choose to use this method of initialization, remember to first configure your Initialization Block and invoke the controller initialization sequence, then perform the direct register accesses to override the values read from the Initialization Block.

***Caution**: If you choose to override the RLEN and TLEN initialization values by writing directly to CSR76 and CSR78, you must write your descriptor ring entry counts as 16-bit negative numbers representing the twos complement of the number of entries in the ring.*

Transmit and Receive Buffers

Transmit and receive buffers may be located at any address in system memory, need not be allocated as contiguous buffers, and have no alignment requirements.

Buffers may be dynamically allocated randomly throughout system memory. Since you specify the starting address for each individual buffer, you are not required to allocate buffers as contiguous blocks.

Buffers may be allocated on any boundary in system memory. Since the controller is able to access buffer data on a byte boundary, you are not required to ensure buffer alignment.

PCnet controllers have only one FCS generator that is used by the transmitter to append the FCS field to the frame, and by the receiver to do multicast filtering and to verify the FCS.

In allocating space for buffers, remember that the receiver stores the FCS field in the receive buffer, whereas the software does not write FCS data to the transmit buffer. Therefore a receive packet takes up 4 more bytes of buffer space than a transmit packet of the same size.

DIAGNOSTICS

Internal Loopback

Normal operation of a PCnet family Ethernet controller is as a half-duplex device. However, in order to allow your device driver to perform an on-line operational test of the controller, a pseudo full duplex mode is provided. The loopback facilities of the MAC Engine, and dual internal FIFOs, allow full operation to be verified without disturbance to the network.

In loopback mode, simultaneous transmission and reception are enabled with the following constraints:

1. *Packet length may be as large as the length field allows and as short as 8 bytes*
2. *FCS may be generated and appended to the output serial bit stream or may be checked on the input serial bit stream, but not both at the same time*
3. *Packets should be addressed to the node itself*
4. *Multicast frames are not accepted when FCS logic is connected to transmit*

During loopback operation, the FCS logic can be allocated to the receiver by setting the Disable Transmit FCS bit (CSR15: DXMTFCS[3]).

If DXMTFCS = 0, the MAC Engine will calculate and append FCS to the transmitted message. The receive message passed to the host will therefore contain an additional four bytes of FCS. In this loopback configuration, the receive circuitry cannot detect FCS errors if they occur.

If DXMTFCS = 1, the last four bytes of the transmit message must contain the (software generated) FCS computed for the transmit data preceding the FCS bytes. The MAC Engine will transmit the data and verify the FCS at the receiver.

When performing an internal loopback, no frame will be transmitted to the network. However, when the controller is configured for internal loopback, the receiver will not be able to detect network traffic.

External Loopback

While configured for external loopback operation, the controller transmits frames onto the network and receives network traffic. For external loopback tests using the T-MAU, an external loopback connector is required since 10BASE-T hubs do not normally feed a station's transmitter output back to that same station's receiver.

INITIALIZATION PROCEDURE

Following a major system event such as the initial power-on sequence or a system wide reset, your device driver software must be designed to perform several fundamental steps which comprise the initialization procedure. The initialization procedure ensures that the PCnet controller is configured to meet your system needs.

There are two ways to initialize the controller. You can use the automatic initialization described below, or you can write directly to the same registers loaded by the automatic initialization procedure.

To use the automatic initialization your device driver must:

1. *Allocate and setup the Initialization Block in system memory*
2. *Disable the PCnet controller by setting the STOP control bit in CSR0*
3. *Load the address of the Initialization Block into device registers CSR1 and CSR2*
4. *Enable the PCnet device initialization procedure by setting the INIT control bit in CSR0*

The Initialization Block contains operating parameters necessary for device operation. Once the Initialization Block is setup in system memory, you must set the STOP bit (CSR0: bit 2) and the device must be programmed with the address of the Initialization Block in CSR2 and CSR1. Address bits 15 through 0 are placed into CSR1, and the rest of the address bits are placed in CSR2. Note that this is the physical address.

The automatic PCnet device initialization includes reading the Initialization Block from system memory to obtain the operating parameters. The Initialization Block is read when the INIT bit (CSR0: bit 0) is set. The INIT bit should be set before or concurrent with the STRT bit (CSR0: bit 1) to insure correct operation. On completion of the read operation and after internal registers have been updated, the IDON bit (CSR0: bit 8) is set and an interrupt generated if the INEA (CSR0: bit 6) bit is set. Remember to install your interrupt service routine, and enable the appropriate interrupt at the interrupt controller, before invoking the device initialization process, if interrupts are used. (Alternative method: Make sure INEA is 0 during initialization, so that the interrupt service routine doesn't have to handle IDON.)

Since initialization is not performed very often, you can reduce the overhead in your interrupt service routine if you poll the Initialization Done (IDON) bit in CSR0.

As the Initialization Block is read during the initialization procedure, the various information and control fields are loaded into specific Control Status Registers. When the initialization procedure is complete, you may change or override configuration settings by directly accessing the Control Status Registers.

It is possible to duplicate the effect of initialization via the Initialization Block by writing the appropriate bits in the appropriate registers. Either method may be used at the discretion of the programmer. However, If the registers are written directly, the INIT bit (CSR0: bit 0) must not be set or the Initialization Block will be read in, overwriting previously written information. Direct initialization must set up the Polling Interval (CSR47).

REINITIALIZATION PROCEDURE

The transmitter and receiver section can be turned on via the Initialization Block (MODE field: DTX[1] and DRX[0]). The state of the transmitter and receiver are

monitored through CSR0 (RXON[5] and TXON[4]). The device should be reinitialized if the transmitter and/or the receiver were not turned on during the original initialization, and it was subsequently required to activate them or if either section shut off due to the detection of an error condition such as transmit underflow or transmit buffer error.

An alternative approach is to write CSR15 directly. The STOP bit (CSR0: bit 2) must be set before writing CSR15. Subsequently setting the STRT bit (CSR0:bit 1) without reinitialization does not make PCnet family devices behave the way the LANCE does. The PCnet family reloads the transmit and receive descriptor pointers with their respective base addresses, something the LANCE does not do.

RESET (H_RESET, S_RESET, STOP)

There are three ways to reset a PCnet controller: a hard reset by asserting the RESET pin (H_RESET), a read access to the RESET address (i.e., offset 0x14 for 16-bit I/O, offset 0x18 for 32-bit I/O), and a soft reset by setting the STOP bit (CSR0: bit 2). Reset causes the device to cease operation and clear its internal logic.

For PCnet-ISA a read access to the RESET address has the same effect as asserting the RESET pin. For PCnet-ISA$^+$, PCnet-32, and PCnet-PCI, reading the RESET address does not affect the information loaded into controller registers from the configuration EEPROM.

Reading the RESET register sets T-MAU to Link Fail state. Software must wait for Link Pass before it can transmit. (Receive works in Link Fail.) Link status can be read from one of the LED registers (BCR4 by default). Setting the STOP bit does not affect the T-MAU.

NORMAL OPERATION

Transmit Sequence Overview

This discussion on the transmit sequence is limited to those actions required by your device driver to transfer control of one or more packets to the PCnet family Ethernet controller for transmission on the network.

The PCnet controller subsystem is assumed to be initialized and operational. This means the controller has been programmed to operate with features and modes that support your requirements. This also means the transmit descriptor ring has been allocated and initialized and the controller has been programmed with the information necessary to manage the transmit descriptor ring and transmit buffers.

Your device driver physically allocates and initializes the descriptor ring, and owns descriptor ring memory, maybe even transmit buffer memory. However, logical ownership is passed back-and-forth between your driver and the controller in the process of queuing packets, transmitting packets, and releasing descriptors and buffers.

Transmit buffers are chunks of memory that are allocated and filled by other entities in the system, such as a file transfer application. When the application decides it's time to transmit the contents of these buffers, a transfer request is generated and passed down through the protocol stack. Eventually, your driver receives a transmit request with the address and length of the associated buffers and performs packet queuing and descriptor updating. The controller will find that it owns the descriptor(s) you've prepared, get the address of the transmit buffer(s), and complete the transmission.

Packet Queuing

Packet queuing takes place when you associate a specific transmit buffer with the next available transmit descriptor. In a system that uses segment:offset style addresses, your device driver is required to translate the segment:offset style address to a 24-bit or 32-bit physical address before updating the transmit descriptor. In 16-bit mode, the least significant 16 bits of the physical buffer address are loaded into the TMD0 element of the descriptor, and the most significant eight bits of the physical address are loaded into bits 7-0 of the TDM1 element of the descriptor. In 32-bit mode, the 32-bit physical buffer address is loaded into TMD0. The next step is to load the buffer byte count, which is the 12-bit two's complement of the buffer length, into the BCNT field of the descriptor. You program the byte count field with the size of the transmit packet, in bytes, not the raw size of the buffer.

Descriptor Update

The final step of the transmit sequence for your device driver is the descriptor update. Certain control bits, such as STP, ENP, ADD_FCS, and especially the OWN bit, must be updated in the TMD1 element of the descriptor. Your specific implementation dictates how and when the first three bits are used but the OWN bit is universal. This is the key bit that transfers ownership of the descriptor(s) to the controller, causing the controller to begin processing transmit descriptors and associated transmit buffers.

If your implementation results in the use of multiple descriptors per transmit packet, or if you support multiple packets per transmission opportunity, your device driver must fully prepare all transmit buffers and related descriptors before setting the OWN bits in reverse order in the descriptor ring.

Receive Sequence Overview

This discussion of the receive sequence identifies actions required by your device driver to receive one or more packets from the PCnet family Ethernet controller.

Your device driver must keep track of incoming receive packets, perform packet removal, and update descriptors. These tasks are typically handled by a software loop that processes receive packets that span multiple

receive buffers and processes multiple packets queued in the receive buffers.

Your driver physically allocates and initializes the descriptor ring, and owns descriptor ring memory, maybe even receive buffer memory. However, logical ownership is passed back-and-forth between the controller and your driver in the process of receiving packets, filling receive buffers, and releasing descriptors and buffers.

The PCnet controller subsystem is assumed to be initialized and operational. This means the controller has been programmed to operate with features and modes that support your requirements. This also means that the receive descriptor ring has been allocated and initialized and the controller has been programmed with information necessary to manage the receive descriptor ring and receive buffers. In addition, your device driver must have already transferred descriptor/buffer ownership to the controller by setting the ownership bit (RMD1: OWN[15]=1 for 16-bit descriptor or RMD1: OWN[31]=1 for 32-bit descriptor) in each descriptor.

Packet Queuing

When you initially enable the receiver, the controller begins polling the first entry of the receive descriptor ring to determine descriptor/buffer ownership. Receive descriptor polling repeats on a fixed 1.6 millisecond cycle. (The polling interval can be programmed via CSR47 for PCnet-32 and PCnet-PCI controllers.) The polling process allows the controller to identify an empty receive buffer and become prepared to capture the next incoming packet.

If the controller has access to an empty receive buffer when a packet arrives, it begins transferring data from the internal receive FIFO to the receive buffer using burst mode. The number of data transfer cycles contained within a single bus cycle is, in general, dependent on the programming of the DMAPLUS option (CSR4: bit 14), the DMA Burst Register (CSR80: DMABR[7:0]), and the Bus Timer Register (CSR82).

The controller performs a look ahead operation between periods of data transfer. Look ahead operations allow the controller to perform data chaining, that is, to continue receiving a packet that exceeds the size of the current receive buffer by transferring data to the next empty buffer. The look ahead operation consists of reading the next descriptor to determine if the controller has ownership of the next descriptor and to determine the address and size of the next empty receive buffer.

When the packet being received requires data chaining and the controller does not own another empty buffer, an error condition exists causing the controller to abort the receive operation and update the current descriptor status. This includes clearing the ownership bit (OWN) and setting the receive buffer error (BUFF) bit. Note that the message byte count field (MCNT) is not valid when a buffer error occurs.

When data chaining is required and an empty receive buffer is available, the controller releases the full buffer to your device driver by clearing the ownership bit in the corresponding descriptor. The controller continues receiving the packet by transferring data to the next empty buffer.

It is important to note that an interrupt is not generated for each receive descriptor update when a receive packet spans multiple descriptors and buffers. The interrupt is generated only when receive interrupt status is set (CSR0: RINT[10] = 1) following the update of the final descriptor for the current packet (where the ENP bit is set).

The controller repeats this cycle of capturing data, transferring data, performing the look ahead operation, and data chaining, until packets stop arriving or the controller has no more empty receive buffers.

Packet Removal

In addition to the fundamental requirement that you detect packet arrival, you must assess each received packet by evaluating available status. Reporting error conditions and forwarding acceptable packets to your client layer are basic responsibilities. To ensure a higher level of performance, you must provide a very efficient receive buffer emptying function to maintain a supply of empty buffers for the controller.

Received packet forwarding requires that you identify a specific receive buffer containing packet information, and that you identify an empty system buffer to hold the contents of the receive buffer.

To detect packet arrival, use controller status polling or install and use an interrupt service routine. When using controller status polling, you'll have to monitor status bits INTR, RINT, and MISS in CSR0 (bits 7, 10, and 12 respectively). When using a hardware interrupt to activate your driver, just check for RINT and MISS.

To assess the condition of the received packet, evaluate controller status (CSR0: ERR, MISS, MERR, RINT, INTR, RXON) and descriptor status (RMD1: OWN, ERR, FRAM, OFLO, CRC, BUFF, STP, and ENP). Flushing error packets and forwarding acceptable packets are the assumed default guidelines.

Software-based receive descriptor and system buffer pointers, initialized and maintained by your device driver, provide the identity of full receive buffers and empty system buffers.

Functionally, the receive descriptor ring is a circular queue. You must keep track of the bounds of the queue and update a receive descriptor pointer as you process descriptors in the order of packet arrival. At initialization,

your pointer contains the address of the first descriptor in the receive descriptor ring.

Upon detection of packet arrival, your receive descriptor pointer identifies the descriptor to check for ownership. If you own the descriptor and packet status is error free, you'll have access to the address of the receive buffer in RMD0[15:0] and RMD1 (HADR[7:0]) and the message byte count in RMD3 (MCNT[11:0]).

***Caution**: For those cases when an empty system buffer is not available and you are unable to empty the receive buffers, your device driver must comply with error policies established outside the scope of this document. While waiting for an empty system buffer, you have to choose between losing the oldest packets in the receive buffers in order to capture the most recent packets or losing the most recent packets to preserve the oldest packets received.*

With the address and length of the receive buffer known, the next step is to identify an empty system buffer and move the contents of the receive buffer to the system buffer. Your system buffer pointer identifies the empty system buffer used as the destination in the block move. The system buffer pointer is incremented to the next empty buffer at the conclusion of the move.

Remember that the buffer addresses stored in the descriptors are physical addresses. If you are using an 80X86 type processor, you may have to convert these physical addresses back to the segment:offset or selector: offset form before moving data from one buffer to another.

Descriptor Update

You must update the corresponding descriptor, after forwarding (or flushing) a packet, to transfer ownership to the controller for receiving subsequent packets.

When returning to the descriptor to set the ownership bit, be sure to check the Start Of Packet and End Of Packet status bits (RMD1: STP[9] and ENP[8]) for the possibility of a packet that spans multiple buffers.

Updating your driver's receive descriptor pointer to the next descriptor in the ring is the final step in the receive function.

If you are implementing a software policy that supports emptying two or more packets per receive indication, you use the updated descriptor pointer to evaluate the next descriptor before exiting the receive function. When you support this concept and another packet is waiting, your driver proceeds as when receiving a packet arrival notification.

Interrupt Handling

The PCnet controller signals important events to the host processor with hardware interrupts. The primary purpose of an Interrupt Service Routine (ISR) is to perform a special unit of work whenever the supported controller asserts an interrupt to signal the occurrence of an important event.

The use of hardware interrupts, in place of software polling, allows your host CPU to process other tasks until the controller requires attention and keeps your device driver from hogging the system bus with continuous status reads. In addition, the controller receives attention much sooner because it preempts lower priority tasks.

Unless special or unusual conditions exist, your device driver should handle PCnet controller interrupts with an ISR.

Controller interrupts directed to the host processor are generated upon:

1. *Completion of the initialization procedure*
2. *The reception of a packet*
3. *The transmission of a packet*
4. *The end of transmission of a packet*
5. *Detection of the transmission of a frame longer than 1518 bytes*
6. *Detection of a transmitter jabber error*
7. *Detection of a missed packet*
8. *Detection of a memory error*
9. *Detection of an internal counter overflow (Missed Packet, Receive Collision)*

Your ISR can determine the actual cause of the interrupt request by examining the BABL, MISS, MERR, RINT, TINT, and IDON bits in CSR0 and the MPCO, RCVCCO, TXSTRT, and JAB bits in CSR4.

Polling Tip: In a system where software polling substitutes for a hardware interrupt, your device driver should monitor the Interrupt status bit (INTR[7]) in Control Status Register 0.

The controller hardware interrupt is a logical OR of several interrupt causing events (e.g., BABL, MISS, MERR, MPCO, RCVCCO, RINT, TINT, IDON, JAB or TXSTRT). After asserting the interrupt request pin, and before the interrupt request is acknowledged, the controller does not generate another transition on the interrupt pin if another event occurs.

For each interrupt request, it's possible that your ISR will find multiple interrupt generating conditions. For example, when a packet arrives and an interrupt occurs, a single packet is waiting for you. However, if ISR latency is high, additional packets could be pending when you evaluate controller status. The same possibility exists for a transmit interrupt or other event interrupts also.

Therefore, it is wise to scan both the Receive and Transmit Descriptor Rings for multiple changes of descriptor ownership whenever an interrupt occurs.

The controller hardware interrupt remains asserted until one of the following conditions is met:

- A one is written to each bit in CSR0 and CSR4 that indicates an interrupt condition
- The controller RESET pin is asserted
- The STOP bit is set (CSR0: bit 2)

The Interrupt Service Routine can cause the interrupt signal to go inactive by reading CSR0 and CSR4 then writing back the values just read.

The driver can minimize software reentry problems by clearing the IENA bit at the start of the ISR then setting it just before the end of the ISR. This way the ISR can run with interrupts enabled.

Interrupt Latency

Interrupt latency due to hardware (e.g., signal assertion time, Programmable Interrupt Controller response time, CPU response time), is somewhat uncontrollable and beyond the influence of your device driver. However, interrupt latency due to excessive software execution in your ISR is your direct responsibility. The design of your device driver must account for interrupt latency and keep it to a minimum.

Minimizing interrupt latency can improve performance in three major areas: your driver is capable of higher overall throughput, your driver is less likely to lose track of the order of significant events, and your driver is less likely to cause artificial overrun conditions and lose incoming packets.

In a network environment with alternating periods of high and low activity, you can improve throughput and minimize packet loss by managing a software-based status FIFO from your interrupt service routine. If you allocate enough space to record controller status on a per descriptor basis, you should be able to record status in real time, from inside the ISR, then process status and descriptors outside the ISR during the time interrupts are inactive.

Status Processing

Status processing comprises three basic functions: capturing controller status, detecting significant or relevant status bits, performing an applicable unit of work.

As a general rule, your device driver will spend most of its time efficiently and correctly transmitting and receiving packets without causing errors. Therefore, when interrupts occur and controller status is captured and analyzed, your device driver should first scan for receive packet status, then scan for transmit packet status, and last, scan for error status.

Receive packet status usually gets top priority in order to minimize packet loss which results in additional network traffic and unnecessary packet delays. Delaying transmit packet status checking is acceptable, up to a point, since packets are only delayed and are not required to be requeued and retransmitted. Checking for errors last is acceptable because they don't occur frequently.

Normal status processing should include capturing and scanning the CSR0 and CSR4 registers, and the appropriate receive and transmit descriptors.

Error Status Processing

Controller status is available to your device driver in CSR0 and CSR4. The order in which your device driver processes these status bits is determined by the level of importance to your implementation.

Status bit ERR (CSR0: bit 15) is set by the ORing of BABL, MERR, CERR, and MISS. ERR is also set when the JAB bit (CSR4: bit 1) is set. ERR remains set as long as any of the error flags are true. ERR is read only: write operations are ignored.

Status bits BABL and MERR (CSR0: bits 14 and 11, respectively) should be recognized as indicating fatal error conditions. BABL and MERR are cleared by writing a logical one to the respective bit position.

The MISS status bit (CSR0: bit 12) is considered to be non-fatal. MISS is cleared by writing a logical one to the bit position.

- When the controller detects a collision error and sets the CERR bit, the ERR bit (CSR0: bit 15) is also set. In addition, beware that this error condition does not cause an interrupt to be asserted by the controller. CERR usually indicates that an external transceiver is not connected to the AUI port or that the 10BASE-T cable is not connected.
- When the controller has lost an incoming receive frame because a receive descriptor was not available and sets the MISS bit, the ERR bit (CSR0: bit 15) is also set. However, be aware that this bit is the only immediate indication of this error condition since there is no receive descriptor to be updated. This bit is included for compatibility with LANCE software. The PCnet family devices also maintain a count of missed frames in CSR112. You may want to mask MISS interrupts by setting the MISSM bit in CSR3 and ignore the MISS bit in CSR0.

Status bit JAB (CSR4: bit 1) is considered a fatal error indicator. JAB is cleared by writing a logical one to the bit position.

LED CONTROL

PCnet family controllers support driving up to four externally mounted LEDs. For PCnet-ISA and PCnet-ISA$^+$ one LED control pin indicates only 10BASE-T Link Status while the other LED control pins are programmable. For PCnet-32 and PCnet-PCI all LED control pins are programmable.

For each programmable LED control pin, you can enable multiple status indications. A programmed control pin then indicates the logical OR of the corresponding enabled status conditions. You can enable monitoring of the following status conditions: Collision Activity, Jabbering, Receive Activity, Receive Polarity, Link Status on Twisted Pair Interface, and Transmit Activity.

For the PCnet-ISA and PCnet-ISA$^+$ controllers, you use I/O write accesses to ISACSR registers 5, 6, and 7 to perform LED control pin programming.

For the PCnet-32 and PCnet-PCI controllers, you use I/O write accesses to BCR registers 4, 5, 6 and 7 to perform LED control pin programming. (BCR6 is not available on PCnet-PCI.)

FAMILY DEVICE DRIVER CONSIDERATIONS

Common Features and Functions

The devices which make up the PCnet family of Ethernet controllers represent evolutionary improvements of the basic design introduced with the LANCE controller. Many features, functions, and internal registers are identical. In most cases, new features and functions are controlled and monitored with additional bits in existing registers and with additional registers.

When planning the design of your device driver, you could design a specialized driver which handles a single controller from the family or you could design a driver which could dynamically adapt to the different family members. Table 10 shows common features and functions is provided to help you design your device driver.

Table 10. Common Features and Functions

Feature/Function	PCnet-ISA	PCnet-ISA+	PCnet-32	PCnet-PCI
ISO 8802-3 (ANSI/IEEE 802.3)	standard	standard	standard	standard
Am7990 (LANCE) Software	run as is	run as is	minor change	minor change
Am79C900 (ILACC) Software	N/A	N/A	minor change	minor change
PCnet-ISA Software	run as is	run as is	run as is[1]	run as is[1]
NE2100 (LANCE) Software	run as is	run as is	run as is	run as is
NE1500T (PCnet-ISA)Software	run as is	run as is	run as is	run as is
ISA Bus	standard	standard	N/A	N/A
VESA VL-Bus	N/A	N/A	standard	N/A
PCI Bus	N/A	N/A	N/A	standard
Am386DX (32-bit) Mode	N/A	N/A	standard	standard
Am486 (32-bit) Mode	N/A	N/A	option	option
16-bit Initialization Block	only	only	default	default
16-bit Descriptors	only	only	default	default
32-bit Initialization Block	N/A	N/A	option	option
32-bit Descriptors	N/A	N/A	option	option
Auto EEPROM Initialization	N/A	standard	standard	standard
I/O Resource Registers	5[2]	5[2]	5[3]	5[3]
16-bit I/O Accesses/Range	Y/ [4]	Y/ [4]	Y/ [5]	Y/ [5]
32-bit I/O Accesses/Range	N/A	N/A	Y/ [6]	Y/ [6]
CSRs (0-126)	standard	superset	superset	superset
Bus Configuration Registers	0-7 (ISA)	0-7 (ISA)	0-21[7]	0-21[7]
Software Reloc. Resources	N/A	Plug ’n Play	Plug ’n Play	PCI Config
				Space
Programmable Interrupts	1	1	1 of 4	1
DMA (DRQ/DACK)	4[8]	4[8]	1[9]	1[9]
LED Outputs/Programmable	4/3	4/3	4	3
EADI	option	option	option	N/A

Notes:

1. *Major code changes required to take advantage of full 32-bit operation.*
2. *PROM, RDP, RAP, Reset, IDP*
3. *PROM, RDP, RAP, Reset, BDP*
4. *24 bytes beginning with base address.*
5. *32 bytes beginning with base address. Byte offsets 0-23 are active, bytes offsets 24-31 are reserved for future use.*
6. *32 bytes beginning with base address. Byte offsets 0-31 are active. RDP, RAP, and BDP contain two bytes of valid data.*
7. *Bus Configuration Registers 0-21 are a superset of the PCnet-ISA set, minor code changes may be required.*
8. *ISA DMA channels used only for bus arbitration, thus, PCnet family works with either an 8 or 16-bit DMA channel.*
9. *PCnet-32 and PCnet-PCI use request/grant interfaces and a central bus arbiter.*

Unique Features and Functions

PCnet-ISA Controller

Single Chip System Solution with ISA Bus Interface

The PCnet-ISA controller is a single chip Ethernet system solution that interfaces directly with the PC-AT Industry Standard Architecture (ISA) bus. The highly integrated 120-pin (EIAJ PQFP) VLSI device reduces the adapter part count and cost, and is applicable for applications demanding higher system throughput.

PCnet-ISA contains an ISA bus interface unit, DMA Buffer Management Unit, ANSI/IEEE 802.3 Media Access Control engine, ISO 8802-3 (ANSI/IEEE 802.3) defined Attachment Unit Interface (AUI), IEEE 802.3 (Type 10BASE-T) Twisted Pair Transceiver Media Attachment Unit (T-MAU), support for external remote boot PROMs, support for external address PROMs, and individual 136-byte transmit and 128-bytereceive FIFOs.

The individual 136-byte transmit and 128-byte receive FIFOs reduce system overhead by providing sufficient latency during packet transmission and reception thus minimizing intervention during normal (i.e., avoidable) network error recovery.

An integrated Manchester Encoder/Decoder provides the Physical Layer Signaling functions required for a fully compliant IEEE 802.3 station and eliminates the need for an external Serial Interface Adapter (SIA).

Upgrading LANCE Code to PCnet-ISA Code— NE2100 Compatibility

PCnet-ISA is register compatible with the Am7990 (LANCE) Ethernet controller and its DMA Buffer Management Unit supports the LANCE descriptor software model. The controller is software compatible with Novell NE2100 and NE1500T Ethernet device drivers and the AMD Am1500T and PCnet-ISA device drivers.

Network software written for the LANCE based NE2100 board should run on a PCnet-ISA based board without modification.

The following list of differences between the PCnet-ISA controller and an NE2100 board could affect your existing software.

- Internal loopback with or without Manchester Encoder/Decoder (MENDEC)
- Byte Swap, ALE Control, and Byte Control bits in CSR3 are not used
- CSR3 and CSR4 are accessible without setting the STOP bit
- Setting the STOP bit is not equivalent to hardware reset
- Reset does not clear the Register Address Port (RAP)
- Memory error (MERR) does not turn off the transmitter and receiver
- Memory error (MERR) has a slightly different meaning on PCnet-ISA
- CERR has a slightly different meaning in10BASE-T mode
- An I/O write after an I/O read to finish a software reset is not required
- All internal registers can be accessed by software
- There are no restrictions on the minimum size of transmit or receive buffers
- Interrupt bits can be masked independently
- Frame Check Sequence (FCS) can be omitted for individual frames
- Pad bytes for short packets can be added and stripped automatically
- Missed Frame and Receive Collision Counters are provided
- Optional mode to accept runt packets enabled by software
- Chip ID register allows positive ID of the controller
- Programmable Memory Read and Memory Write signals
- I/O base address assignment and detection
- Optional power down mode for T-MAU circuitry is enabled by software
- Idle state of the AUI drivers is controlled by software
- Port selection (i.e., AUI or T-MAU) by jumper, by software, or automatically
- The 10BASE-T link test function can be disabled by software
- Automatic (T-MAU) polarity correction feature can be disabled by software
- In T-MAU mode two receiver threshold voltages are selected by software
- The meaning of three LED driver outputs is programmable by software
- New External Address Detection Interface is enabled by software
- Optional initialization by software writing directly to control registers
- Initialize (INIT) and Start (STRT) bits can be set with a single I/O write
- I/O resources accessed indirectly with Register Address Port (RAP)
- Descriptor rings can contain from 1 to 65535descriptors each
- Interrupt at beginning of packet transmission is software controlled

- Accept/Reject broadcast packets by software control
- Accept/Reject frames matching physical address by software control
- Accessing certain active registers requires stopping the controller
- Transmit and Receive FIFO Watermarks are software controlled
- Maximum DMA burst length is software controlled
- Transmit descriptor polling can be disabled by software
- The transmission two-part deferral algorithm can be disabled by software
- An alternate backoff algorithm can be selected by software

PCnet-ISA Changes

The following functions are available in the LANCE but behave differently in the PCnet-ISA controller without affecting existing software.

Loopback

The loopback facilities of the MAC engine allow full operation to be verified without disturbance to the network. Loopback operation is affected by the state of the loopback control bits (LOOP, MENDECL, and INTL) in CSR15. This affects whether the internal MENDEC is considered part of the internal or external loopback path.

CSR3 Bits Now Reserved

In CSR3 of the LANCE the BSWP, ACON, and BCON bits allowed you to configure the bus interface. With PCnet-ISA a hardware reset makes the controller compatible with the NE2100 card so these are reserved bits in PCnet-ISA and must be written as zero.

With PCnet-ISA, CSR3 and CSR4 are always accessible, not just when the STOP bit is set.

CSR3 Bits Now Used

In the LANCE, there were control register bits marked as reserved. In PCnet-ISA, some of the reserved bits are used to control new options. Existing NE2100 software that writes zeros to these reserved positions will run properly on the PCnet-ISA without modifications.

CSR3 of PCnet-ISA contains the following new control bits: Babble Mask (BABLM), Missed Frame Mask (MISSM), Memory Error Mask (MERRM), Receive Interrupt Mask (RINTM), Transmit Interrupt Mask (TINTM), Initialization Done Mask (IDONM), Disable Transmit Two Part Deferral (DXMT2PD), and Enable Modified Back-off Algorithm (EMBA).

MODE Register (CSR15) Bits Now Used

The following bits have been added to the mode register: Disable Receive Broadcast (DRCVBC), Disable Receive Physical Address (DRCVPA), Disable Link Status (DLNKTST), Disable Automatic Polarity Correction (DAPC), MENDEC Loopback Mode (MENDECL), Low Receive Threshold (LRT)/Transmit Mode Select (TSEL), and Port Select (PORTSEL).

TMD1 Bit Now Used

This new bit adds dynamic control of FCS generation on a frame-by-frame basis.

Stopping PCnet-ISA

In the LANCE, setting the STOP bit is the equivalent of asserting $\overline{\text{RESET}}$. The controller ceases operation, clears internal logic, forces all three-state buffers to the high impedance state, and enters an idle state with the STOP bit in CSR0 set.

With the PCnet-ISA controller, setting the STOP bit is not the equivalent of asserting $\overline{\text{RESET}}$. Setting the STOP bit clears the appropriate status bits, but it does not change any control bits that affect the configuration of the device.

The STOP bit resets and disables the transmit and receive state machines. It resets the transmit and receive FIFOs and resets the Next Receive Descriptor Address and Next Transmit Descriptor Address Registers. All data in the transmit and receive FIFOs are lost when the STOP bit is set. The Next Receive Descriptor Address and Next Transmit Descriptor Address Registers point to the first descriptors in the transmit and receive descriptor rings.

If the STOP bit is set in the middle of a transmit or receive operation, the PCnet-ISA will complete the current DMA burst, but it will not issue a TINT or RINT interrupt before it turns off the receiver and transmitter. The PCnet-ISA does not violate the OWN bit convention when the STOP bit is set. If an OWN is set, the corresponding buffer has not been completely filled by the receiver or transmitted by the transmitter.

When your software stops the controller with the STOP bit, several steps are required before resuming operation with the STRT bit. Your software should process all packets in the receive queue, reinitialize the transmit and receive descriptors, then queue up any outgoing packets that were not sent before the STOP bit was set. Your software is not required to reinitialize any configuration registers in the PCnet-ISA.

Register Address Port (RAP)

The contents of the Register Address Port are not changed by setting the STOP bit or by hardware reset so this register powers up in an unknown state. LANCE software that expects the Register Access Port to be cleared by a reset must be modified to work with PCnet-ISA.

Memory Errors

The Memory Error (MERR) bit does not turn off the transmitter or the receiver in PCnet-ISA. This is unlikely to be a problem with software written for the LANCE.

MERR occurs in the PCnet-ISA chip when $\overline{\text{DACK}}$ has not been asserted 250 ms after DRQ occurs. In the LANCE MERR occurs when $\overline{\text{READY}}$ has not been asserted within 25.6 µs after the address has been asserted on the DAL lines. This prevents the LANCE from locking up the system if it tries to access non-existent memory. However, accessing non-existent memory locations over the ISA bus does not cause the system to lock up, because the IOCHRDY line, which is used to insert wait states, is normally high and must be forced low by the addressed memory to extend the cycle.

Collision Error (CERR)

CERR in the LANCE indicates that the Signal Quality Error test (SQE) failed. This means that the external transceiver did not activate the collision inputs to the AUI within 20 network bit times after the chip stopped transmitting. CERR has the same meaning in PCnet family controllers when the AUI port is selected. However, when the T-MAU is selected, CERR indicates that the T-MAU was in the link fail state when the chip was transmitting.

Software Controlled Hardware Reset

An I/O read access of the Reset Register at I/O offset 0x14 creates an internal reset pulse. The controller responds to an internal reset pulse differently from when the RST pin is asserted or the when STOP bit is set. A reset invoked by reading the Reset Register causes certain bits in the Command Status Registers to be automatically cleared. However, the internal reset has no effect on bits in the Bus Configuration Registers.

The NE2100 LANCE based family of Ethernet cards require an I/O write access to the Reset Register following a read access to the Reset Register. The PCnet-ISA controller does not have the same requirement, however, if your device driver performs the extra write access there are no negative side effects.

PCnet-ISA Enhancements

This section describes programmable functions of the PCnet-ISA that are not available in the LANCE. The software controls these new functions by writing to a collection of registers that are not present in the LANCE and by writing to certain bits that are not used in LANCE registers.

Accessing Internal Registers

PCnet-ISA supports 108 internal registers compared to six in the LANCE. Most of these registers are used only for debugging and production testing and are never touched by normal device driver software.

The PCnet-ISA registers are divided into two groups: control and status registers (CSRs) and ISA Configuration Registers (ISACSRs). Accessing either group is a two step procedure. First, your software must load the Register Address Port with the number of the desired register. Then your software can read from or write to the Register Data Port to access a CSR or the ISACSR Data Port (IDP) to access an ISACSR.

Buffer Sizes

PCnet-ISA places no restrictions on the minimum size of transmit or receive buffers. This means that a packet can be made up of one or more small buffers containing protocol headers plus one or more large buffers containing data. These buffers are linked together via descriptors in the transmit descriptor ring.

Masking Interrupt Bits

PCnet-ISA gives you the ability to mask individual interrupt sources. By setting the appropriate mask bits, your software can allow some conditions to cause an interrupt while disabling other classes from causing an interrupt.

Dynamic FCS Generation

Frame Check Sequence (FCS) generation can be enabled or disabled on a packet-by-packet basis. This feature is useful for bridge applications in which a packet should be passed from one network to another while preserving the original FCS.

The Disable Transmit FCS (DXMTFCS) bit disables FCS generation in general, while the ADD_FCS bit overrides DXMTFCS for an individual packet. To use this feature, set the DXMTFCS bit in the MODE register, then for each packet that should be sent with FCS generation, set the ADD_FCS bit in the TMD1 field of the packet descriptor. When the packet occupies more than one buffer, you set the ADD_FCS bit only in the first descriptor.

Automatic Pad Generation and Stripping

To simplify and speed up your driver software and to help save system bus bandwidth, PCnet-ISA can be programmed to handle packets with less than 64 data bytes. Transmit frames can be automatically padded to extend them to 64 data bytes (excluding preamble). During reception of an 802.3 frame the pad field can be stripped automatically.

Setting the Auto Pad Transmit (APAD_XMT) bit in CSR4 enables automatic pad generation on transmit, and setting the Auto Strip Receive (ASTRP_RCV) bit in the same register enables automatic pad stripping on receive.

PCnet-ISA uses the length field of the IEEE 802.3 header to determine the number of bytes to strip. It uses the total number of bits transmitted to determine the number of bytes to pad. A major difference between

the Ethernet and IEEE 802.3 specifications is that Ethernet uses the two bytes following the source address field for a type field rather than for a length field. Since all Ethernet type codes are numerically greater than 1500, automatic padding and stripping can be used with both Ethernet and IEEE 802.3 protocols.

Missed Frame Count

PCnet-ISA provides an internal 16-bit counter (Missed Frame Count: CSR112) to record the number of frames addressed to this node that were not saved because the receiver was disabled or because the receive FIFO overflowed. The Missed Frame Counter, which is accessible by software at any time, is cleared by setting the STOP bit or by asserting a hardware reset. When the Missed Frame Counter rolls over from 65535 to 0, PCnet-ISA sets the Missed Packet Counter Overflow (MPCO) bit in CSR4 and issues an interrupt signal if this interrupt is enabled (IENA = 1 in CSR0 and MPCOM = 0 in CSR4).

This counter eliminates the need for the Missed Frame (MISS) bit in CSR0, but this bit is included for compatibility with the LANCE. The MISS bit causes an interrupt every time a packet is missed, while the MPCO bit causes an interrupt only after 65536 packets have been missed.

Your software can extend the effective length of this counter by maintaining the high order bits of an extended counter in system memory. The high order bits are incremented by the interrupt service routine when an MPCO interrupt occurs.

Since the Missed Frame Counter continues to count with the receiver disabled, your software can use this feature to create a simple network traffic monitor. When you set the Promiscuous Mode and Disable Receiver bits in the MODE register, you don't need to worry about providing receive descriptors or processing received frames. The Missed Frame Counter will keep an accurate count of the total number of packets transmitted over the network and your software can periodically fetch and record the count.

Receive Collision Count

PCnet-ISA provides an internal 16-bit counter (Receive Collision Count: CSR114) to record the number of regular and late collisions. The Receive Collision Counter, which is accessible by software at any time, is cleared by setting the STOP bit or by asserting a hardware reset.

In a non 10BASE-T network, when the AUI interface is active, all receive collisions are detected and counted.

This counter does not increment when the active interface port is T-MAU because there are no receive collisions in a 10BASE-T network.

Accepting Runt Packets

PCnet-ISA can be programmed to accept properly addressed runt packets through the use of a test mode. Enabling this feature is a three step operation. First, set the Enable Test Mode Operation (ENTST) bit of CSR4 to enable the Buffer Management Scratch Test Register (BMSTR) in CSR124. Second, set the Runt Packet Accept (RPA) bit in CSR124. Third, reset the ENTST bit of CSR4 to disable further accesses to the Buffer Management Scratch Test Register. General networking software should never set bits other than the RPA bit in the BMSTR. The other control bits are reserved for factory use during IC testing.

Chip ID Register

PCnet-ISA provides a 32-bit Chip ID Register in CSR88 and CSR89. Four well-defined fields in the Chip ID Register allow your device driver to make a positive identification of the controller. The four fields are as follows: bit 0 is always a one, bits 11-1 contain the Manufacturer ID, bits 27-12 contain the Part Number, and bits 31-28 contain the Version. The Manufacturer ID (per JEDEC Publication 106-A) for Advanced Micro Devices is 00000000001 (binary), the Part Number for PCnet-ISA is 0x0003, and the Version is a 4-bit pattern that is silicon revision dependent.

Programmable Memory Read and Memory Write Signals

The widths of the memory read and memory write signals asserted by PCnet-ISA during a DMA transfer are programmable via the ISA Bus Configuration Registers. The value in the Master Mode Read Active Register (ISACSR0) controls the width of the active portion of the read signal (MEMR), while the value in the Master Mode Write Active Register (ISACSR1) controls the width of the active portion of the write signal (MEMW). The values stored in ISACSR0 and ISACSR1 define the number of 50 ns periods that the respective command signal is active.

I/O Base Address Assignment and Detection

The I/O resources of PCnet-ISA can be assigned to four different base addresses. I/O base addresses 0x300, 0x320, 0x340, and 0x360, are selected with external jumpers controlling the IOAM1-0 input pins.

Although your software cannot read the values of these address select pins, you can attempt to read the Chip ID Register for each of the four possible addresses. You can also attempt to read the address PROM, if available, which is located at offset 0x00 from the I/O base address.

Optional Power Down Modes for T-MAU Circuitry.

PCnet-ISA supports two power down modes, Sleep and Auto-Wake, for reduced power consumption by the T-MAU circuitry in critical battery powered applications.

Sleep mode is enabled when the $\overline{SLEEP}$ pin is asserted and the AWAKE bit (ISACSR2: bit 2) is reset. Upon entering Sleep mode, PCnet-ISA goes into a permanent deep sleep. Assertion of the $\overline{RESET}$ pin is the only method of reviving PCnet-32. Sleep mode is the default power down mode.

Auto-Wake mode is enabled when the $\overline{SLEEP}$ pin is asserted and the AWAKE bit is set. Upon entering Auto-Wake mode, the T-MAU receive circuitry remains enabled even while the $\overline{SLEEP}$ pin is asserted. The $\overline{LED0}$ output pin also continues to function, indicating a good 10BASE-T link if there are link beat pulses or valid frames present. The $\overline{LED0}$ pin can drive external hardware that deasserts the $\overline{SLEEP}$ pin. This configuration effectively wakes the system when there is any activity on the 10BASE-T link. Upon awakening the controller, your device driver must allow 0.5 seconds for the internal analog circuits to stabilize.

TSEL

The state of the AUI drivers during idle is programmable through the Transmit Mode Selection (TSEL) bit in the MODE register. TSEL must be set to zero (the default) for IEEE 802.3 or Ethernet 2 transformer coupled networks. It should be cleared for direct coupled Ethernet1 networks.

Network Interface Port Selection

PCnet-ISA can be connected to an IEEE 802.3 network via one of two network interface ports. The Attachment Unit Interface (AUI) provides an IEEE 802.3 compliant differential interface to a remote MAU or an on-board transceiver. The 10BASE-T interface provides a twisted-pair Ethernet port.

Port selection is made by software or by hardware, or it can be made automatically by the link pulse detection logic. The External MAU Select (XMAUSEL) bit in the MAU Configuration Register (ISACSR2) allows your software to specify the mechanism that determines port selection. If XMAUSEL is set, port selection is determined by the state of the MAU Select (MAUSEL) pin, otherwise, port selection is determined by the Port Select (PORTSEL) bits in the MODE register. The only valid choices for the PORTSEL bits are 00 to select the AUI interface or 01 to select the 10BASE-T interface.

If the Auto Select (ASEL) bit in ISACSR2 is set, the state of the XMAUSEL bit is ignored, and the selection is made automatically. If the T-MAU detects valid link pulses on its Receive Data (RXD) inputs, the T-MAU is selected. Otherwise, the AUI is selected.

T-MAU Control-Jabber

If the T-MAU detects a jabber condition, it will set the Jabber Error (JAB) bit in CSR4, thus causing an interrupt. Setting the Jabber Error Mask (JABM) bit in CSR4 will mask out this interrupt.

T-MAU Control-Link Test

Setting the Disable Link Status (DLNKTST) bit in CSR15 will prevent the PCnet-ISA from monitoring link pulses. This feature is used for twisted pair networks that do not use link pulses.

T-MAU Control-Polarity

Setting the Disable Automatic Polarity Correction (DAPC) bit in CSR15 disables the automatic polarity correction. The actual polarity as detected by the T-MAU receiver regardless of the state of the DAPC bit can be read in the following rather roundabout way: One of the three LED drivers in ISACSR5, ISACSR6, ISACSR7, can be programmed to indicate the actual polarity detected by the T-MAU. Your software can test the state of this LED driver by reading the LED output (LEDOUT) bit of the corresponding LED register.

T-MAU Control-Receive Threshold

The receiver threshold voltage can be lowered so that the PCnet-ISA can be used with twisted pair cables that are longer than the maximum specified in the 10BASE-T standard. To enable this feature set the Low Receive Threshold (LRT) bit in CSR15.

Programmable LEDs

PCnet-ISA provides four LED driver output pins. One of these ($\overline{LED0}$) displays the T-MAU link status. $\overline{LED0}$ is not programmable. The other three LED drivers can be programmed to display the status of one or more of the following internal signals from the T-MAU: collision, jabber, link, receive status, receive polarity, and transmit status.

If one LED driver is programmed to display more than one signal, the output will be the logical OR of the selected signals. For example, if the receive (RCVE) bit is set in the LED1 (ISACSR5) register, the LED1 pin will be active whenever the MAU is receiving a packet. If the RCVE and transmit (XMTE) bits in the LED1 register are both set, the $\overline{LED1}$ pin is active whenever the MAU is receiving or transmitting.

Since network events are extremely short, each of the three programmable LED driver circuits can be connected to pulse stretcher circuits that will cause the LEDs to be turned on long enough for humans to be able to notice them. Each pulse stretcher is enabled by setting the pulse stretcher enable (PSE) bit in the appropriate LED register (ISACSR5-7).

The state of an LED driver signal before the pulse stretcher can be read at any time by examining the LEDOUT bit of the appropriate LED register. This technique can be used to test the polarity of the twisted pair receiver inputs (normal or reversed).

External Address Detection Interface (EADI)

A software enabled External Address Detection Interface (EADI) allows external hardware address filtering

in parallel with frame reception and address comparison in the MAC Station Address Detection (SAD) block. When the EADI is enabled (ISACSR2: EADISEL, bit 3=1), four PCnet-ISA pins lose their default function and are remapped as EADI control pins. LED control pins $\overline{\text{LED1}}$, $\overline{\text{LED2}}$, and $\overline{\text{LED3}}$ become EADI output control pins SF/BD, SRD, and SRDCLK. The MAU select pin MAUSEL becomes the EADI input control pin $\overline{\text{EAR}}$.

When external address detection hardware is used, it may be convenient to set the disable receive broadcast (DRCVBC) bit in the MODE register, thereby disabling the automatic reception of broadcast messages. The internal physical address detection logic can be turned off by setting the Disable Receive Physical Address (DRCVPA) bit in the MODE register. Also, the internal logical address filtering can be disabled by filling the Logical Address Filter (LADRF) in CSR8-11 with zeros.

Automatic or Direct Initialization

PCnet-ISA is initialized similar to the LANCE by setting up an Initialization Block in memory, then setting the Initialization (INIT) bit in CSR0. A minor difference is that with PCnet-ISA, the INIT and STRT bits can be set at the same time, whereas with the LANCE these bits must be set in two separate I/O operations.

As an alternative to automatic initialization, PCnet-ISA can be initialized directly by writing to the appropriate Control and Status Registers. If you use this method the INIT bit in CSR0 should not be set.

I/O Resource Access

To access a Control Status Register, you must complete a two step operation. First, you perform an I/O write to the Register Address Port (RAP) using the appropriate CSR number as your data. Then, you perform an I/O read or write access to the RDP. Note that the contents of the RAP is latched and is not changed until rewritten; therefore, the register select cycle is not required inorder to repeatedly access the same CSR.

These software accesses should be coded as 16-bit accesses, even if the PCnet device is hardware configured for 8-bit I/O bus cycles. It is acceptable (and transparent) for the motherboard to turn a 16-bit software access into two separate 8-bit hardware bus cycles. The motherboard accesses the low byte before the high byte and the PCnet device has circuitry to specifically support this type of access.

To access an ISA Configuration Register (ISACSR), the RAP must be written first with the ISACSR number, followed by the read or write access to the IDP. These software accesses should be coded as 16-bit accesses.

Descriptor Ring Length Initialization

The LANCE restricts the sizes of descriptor rings to integer powers of 2. PCnet-ISA enforces the same limitation when you use auto initialization but provides you with the ability to override the limit and set the number of transmit or receive descriptors to any number from 1to 65535.

After performing initialization, by writing directly to the Control and Status Registers or by invoking auto initialization, you load the two's complement of the desired value directly into CSR76 for the receive descriptor ring or CSR78 for the transmit descriptor ring.

Start of Transmit Interrupt

PCnet-ISA sets the Transmit Start (TXSTRT) bit in CSR4, and optionally asserts an interrupt, when it begins transmission of a frame. The Transmit Start interrupt occurs if the Transmit Start Mask (TXSTRTM) bit in CSR4 is cleared. TXSTRTM is set by default on a controller reset.

Disable Receive Broadcast

The reception of broadcast packets is enabled by default with a controller reset and is disabled by setting the Disable Receive Broadcast (DRCVBC) bit in CSR15. This feature is used for protocols that do not support broadcast addressing, except as a function of multicast.

Disable Physical Address Detection

The reception of frames matching the physical address of the node is disabled by setting the Disable Receive Physical Address (DRCVPA) bit in CSR15. Frames matching the address of the node are not recognized although the frame may be accepted by the EADI mechanism.

Accessing Active Registers

From time to time your software may find it necessary to alter the operational parameters contained in certain active Control and Status Registers. You must stop the controller by setting the STOP bit in CSR0 before accessing the following control registers.

Table 11. Accessible Active Registers

Register	Register Name
CSR1	Initialization Address Register (IADR[15:0])
CSR2	Initialization Address Register IADR[23:16])
CSR8	Logical Address Filter (LADRF[15:0])
CSR9	Logical Address Filter (LADRF[31:16])
CSR10	Logical Address Filter (LADRF[47:32])
CSR11	Logical Address Filter (LADRF[63:48])
CSR12	Physical Address Register (PADR[15:0])
CSR13	Physical Address Register (PADR[31:16])
CSR14	Physical Address Register (PADR[47:32])
CSR15	Mode Register
CSR76	Receive Ring Length Register
CSR78	Transmit Ring Length Register
CSR80	Burst and FIFO Threshold Control Register
CSR82	Bus Activity Timer

While it is not necessary to do a LANCE type automatic initialization after the STOP bit has been set, it is necessary to clean up the transmit and receive queues as described earlier in this chapter (Stopping PCnet-ISA).

Tuning DMA Operations

PCnet-ISA lets you tune transmit and receive DMA operations by the controller to the response and throughput capacity of your system bus. You can specify different values for the Transmit FIFO Start Point (XMTSP), the Transmit FIFO Watermark (XMTFW), and the Receive FIFO Watermark (RCVFW).

The Transmit FIFO Watermark is the number of FIFO write cycles attempted on a transmit DMA operation from system memory to the FIFO. The transmit FIFO must have enough empty space to accommodate this number of write cycles for the DMA to begin. You can select 8, 16, or 32 write cycles when programming the XMTFW field in CSR80. The default is 64 write cycles.

The Transmit FIFO Start Point is the number of bytes that must be loaded into the FIFO before the controller begins transmitting on the network. When the entire frame is in the FIFO, transmission begins regardless of the value in XMTSP. You can select 4, 16, 64, or 112 bytes written when programming the XMTSP field in CSR80. The default is 64 bytes written.

The Receive FIFO Watermark is the number of bytes that must be present in the FIFO before the controller begins a receive DMA operation from the FIFO to system memory. At least 64 bytes have to be received before the controller attempts to perform the receive DMA. This eliminates the need for your software to react to receive frames that are runts or suffer from a collision during the slot time (512 bit times). If you have enabled the Runt Packet Accept feature, the receive DMA begins when the receive byte count matches the RCVFW threshold or on detection of a complete valid frame (regardless of length). You can select 16, 32, or 64 bytes received when programming the RCVFW field in CSR80. The default is 64 bytes received.

DMA Burst Length Control

There are two ways for you to limit the maximum time the PCnet-ISA controller retains control of the system bus during a DMA transfer burst. You can specify a range of values for the DMA Burst Register (DMABR) and for the Bus Activity Timer (DMABAT).

The DMA Burst Register contains the maximum allowable number of transfers during a single DMA mastership cycle. You can specify a transfer count in the range of 1 to 256 by writing values 0x00 through 0xFF. The default value loaded on reset is 0x10. The DMA Burst Register is not used to limit the number of descriptor transfers.

You can disable the DMA Burst Register by setting the DMAPLUS bit in CSR4. When disabled, the DMA transfer continues until the transmit FIFO is full or the receive FIFO is empty.

The Bus Activity Timer contains the maximum allowable time a single bus mastership cycle is asserted by the controller. This is a count down timer, with a resolution of 100 nanoseconds per tick, programmed with a 16-bit unsigned number corresponding to the number of ticks desired. A value of zero limits the controller to one bus cycle per bus mastership period. To allow a maximum bus activity period of 51 microseconds you program the timer with a value of 510 (0x01FE).

You enable this timer by setting the Bus Timer Register Enable (TIMER) bit in CSR4.

Both counters can be enabled at the same time. In this case PCnet-ISA releases the bus when either counter is exhausted.

Disabling Descriptor Polling

By default, the controller automatically performs periodic polling of the next available descriptor to determine if ownership has changed in favor of the controller. If you want to eliminate these automatic accesses and recover

the bus bandwidth for other uses, you can disable automatic polling by setting the Disable Transmit Polling (DPOLL) bit in CSR4.

When automatic polling is disabled and the controller is quiescent, with respect to transmitting packets, your software must notify the controller when new packets are ready for transmission. With your new packets queued and the OWN bits in the corresponding transmit descriptors set, you notify the controller by setting the Transmit Demand (TDMD) bit in CSR0. The controller processes all queued packets before returning to the transmit idle mode.

Disabling Transmit Two Part Deferral

The IEEE/ANSI 802.3 Standard (ISO/IEC 8802-3 1990) requires that the CSMA/CD MAC monitor the medium for traffic by watching for carrier activity. When carrier is detected, the media is considered busy and the MAC should defer to the existing message. The ISO 8802-3 (IEEE/ANSI 802.3) Standard also allows an optional two part deferral after a receive message.

The PCnet-ISA MAC engine implements the optional two part deferral algorithm and it's enabled by default on reset. Your software can disable this feature by setting the Disable Two Part Transmit Deferral (DXMT2PD) bit in CSR3.

Alternate Backoff Algorithm

The ISO 8802-3 (IEEE/ANSI 802.3) Standard requires use of a "truncated binary exponential backoff" algorithm that provides a controlled pseudo random mechanism to enforce the collision backoff interval before re-transmission is attempted.

The PCnet-ISA MAC engine implements an optional alternative algorithm that suspends the counting of the slot time/IPG during the time that receive carrier sense is detected. It effectively accelerates the increase in the backoff time in busy networks, and allows nodes not involved in the collision to access the channel while the colliding nodes await a reduction in channel activity. Your software can enable this option by setting the Enable Modified Backoff Algorithm (EMBA) bit in CSR4.

Programmable Registers

The PCnet-ISA device has more than 70 programmable registers with most of these for debugging and production testing purposes. You should limit access to the registers shown in Table 12 and Table13.

Table 12. PCnet-ISA Accessible ISA Bus Configuration Registers

ISA Bus Configuration Registers				
ISACSR0	MSRDA, Master Mode Read Active (Default: 5) Programs Width of Master Mode Memory Read (MEMR) Signal			
ISACSR1	MSWRA, Master Mode Write Active (Default: 5) Programs Width of Master Mode Memory Write (MEMW) Signal			
ISACSR2	MC, MAU Configuration (Default: 1)			
	Reserved[15:4]			
	0000			
	0008	EADISEL		
	0004	*AWAKE*		
	0002	*ASEL*		
	0001	*XMAUSEL*		
ISACSR3	Reserved (Default: N/A)			
ISACSR4	Reserved (Default: N/A)			
ISACSR5	LED1 Control and Status Register (Default: 0084)			
	8000	*LEDOUT*	0008	*RXPOLE*
	0080	*PSE*	0004	*RCVE*
	0010	*XMTE*	0002	*JABE*
			0001	*COLE*
ISACSR6	LED2 Control and Status Register (Default: 0008)			
	8000	*LEDOUT*	0008	*RXPOLE*
	0080	*PSE*	0004	*RCVE*
	0010	*XMTE*	0002	*JABE*
			0001	*COLE*
ISACSR7	LED3 Control and Status Register (Default: 0090)			
	8000	*LEDOUT*	0008	*RXPOLE*
	0080	*PSE*	0004	*RCVE*
	0010	*XMTE*	0002	*JABE*
			0001	*COLE*

Table 13. PCnet-ISA Accessible Control and Status Registers

	ISA Bus Configuration Registers							
Register	**Contents**							
CSR0	PCnet Controller Status (Default: 0004)							
	8000	*ERR*	0800	*MERR*	0080	*INTR*	0008	*TDMD*
	4000	*BABL*	0400	*RINT*	0040	*IENA*	0004	*STOP*
	2000	*CERR*	0200	*TINT*	0020	*RXON*	0002	*STRT*
	1000	*MISS*	0100	*IDON*	0010	*TXON*	0001	*INIT*
CSR1	Initialization Block Address, IADR[15:0] (Default: N/A)							
CSR2	Initialization Block Address, IADR[23:16] (Default: N/A)							
CSR3	Interrupt Masks and Deferral Control (Default: 0000)							
	8000	*0*	0800	*MERRM*	0080	*0*	0008	*EMBA*
	4000	*BABLM*	0400	*RINTM*	0040	*0*	0004	*0*
	2000	*0*	0200	*TINTM*	0020	*0*	0002	*0*
	1000	*MISSM*	0100	*IDONM*	0010	*DXMT2PD*	0001	*0*
CSR4	Test and Features Control (Default: 0115)							
	8000	*ENTST*	0800	*APAD_XMT*	0080	*0*	0008	*TXSTRT*
	4000	*DMAPLUS*	0400	*ASTRP_RCV*	0040	*0*	0004	*TXSTRTM*
	2000	*TIMER*	0200	*MPCO*	0020	*RCVCCO*	0002	*JAB*
	1000	*DPOLL*	0100	*MPCOM*	0010	*RCVCCOM*	0001	*JABM*
CSR8	Logical Address Filter (LADRF[15:0]) Default: N/A							
CSR9	Logical Address Filter (LADRF[31:16]) Default: N/A							
CSR10	Logical Address Filter (LADRF[47:32]) Default: N/A							
CSR11	Logical Address Filter (LADRF[63:48]) Default: N/A							
CSR12	Physical Address (PADR[15:0]) Default: N/A							
CSR13	Physical Address (PADR[31:16]) Default: N/A							
CSR14	Physical Address (PADR[47:32]) Default: N/A							
CSR15	Mode Register (Default: 0000)							
	8000	*PROM*	0800	*DAPC*	0080	*PORTSEL*	0008	*DXMTFCS*
	4000	*DRCVBC*	0400	*MENDECL*	0040	*INTL*	0004	*LOOP*
	2000	*DRCVPA*	0200	*LRT/TSEL*	0020	*DRTY*	0002	*DTX*
	1000	*DLNKTST*	0100	*PORTSEL*	0010	*FCOLL*	0001	*DRX*
CSR76	Receive Ring Length (Default: N/A)							
CSR78	Transmit Ring Length (Default: N/A)							
CSR80	Burst and FIFO Threshold Control (Default: 2810) RCVFW[13:12], Receive FIFO Watermark 0000 *Request DMA when 16 bytes are present* 1000 *Request DMA when 32 bytes are present* 2000 *Request DMA when 64 bytes are present* 3000 *Reserved* XMTSP[11:10], Transmit Start Point 0000 *Start transmission when 4 bytes are present* 0400 *Start transmission when 16 bytes are present* 0800 *Start transmission when 64 bytes are present* 0C00 *Start transmission when 112 bytes are present* XMTFW[9:8], Transmit FIFO Watermark 0000 *Request DMA when 8 cycles will fit in FIFO* 0100 *Request DMA when 16 cycles will fit in FIFO* 0200 *Request DMA when 32 cycles will fit in FIFO* 0300 *Reserved* DMABR[7:0], DMA Burst Register							

Table 13. PCnet-ISA Accessible Control and Status Registers (Continued)

	ISA Bus Configuration Registers
Register	**Contents**
CSR82	Bus Activity Timer (Default: N/A)
CSR88	Chip ID Register (Default: N/A) Part Number continued[15:12], 4-bit pattern: 0011 Manufacturer ID[11:1], 11-bit pattern: 0000 0000 001 Reserved[0], 1-bit pattern: 1
CSR89	Chip ID Register (Default: N/A) Version[15:12], 4-bit pattern, dependent on revision of silicon Part Number[11:0], 12-bit pattern: 0000 0000 0000
CSR112	Missed Frame Count (Default: N/A)
CSR114	Receive Collision Count (Default: N/A)
CSR124	Buffer Management Scratch Test (Default: N/A) RPA[3], Runt Packet Accept 0008 *Force CORE receive logic to accept runt packets*

PCnet-ISA+ Controller

Upgraded Single Chip System Solution with ISA Bus Interface

The PCnet-ISA+ controller is a single chip Ethernet system solution that interfaces directly with the PC-AT Industry Standard Architecture (ISA) bus. The highly integrated 132-pin (JEDEC PQFP) VLSI device reduces the adapter part count and cost, and is applicable for applications demanding higher system throughput.

New Support For Microsoft Plug 'n Play ISA Specification

PCnet-ISA+ is compatible with the Plug 'n Play ISA Specification, jointly written by Intel and Microsoft Corporation. The Plug 'n Play specification describes a hardware and software mechanism that enables resolution of conflicts between Plug 'n Play ISA cards. Requirements for system-wide resources (e.g., memory addresses, I/O addresses, DMA channels, and Interrupt Request lines) are evaluated on a card-by-card basis and are allocated among the cards to eliminate conflict.

The overall tasks of Plug 'n Play configuration software are:

- Isolate an installed ISA card
- Read the card's resident resource information
- Identify the type of card and configure its resources
- Locate and load the applicable device driver

Cards that are compatible with the Plug 'n Play ISA Specification are electrically and functionally compatible with standard ISA cards and inter-operate in an ISA system without conflict. However, a system that contains one or more standard ISA cards may not be fully auto-configurable.

Configuration information, maintained in a standard read-only format, identifies the card and describes the system resources required by the card (e.g., memory address space, I/O address space, DMA channel supported, interrupt request supported).

New Support for Auto-Configuration EEPROM

PCnet-ISA+ automatically reads the contents of a serial EEPROM that conforms with the National Semiconductor Microwire interface. Following a controller reset, the controller configuration information, including the 48-bit IEEE address and the Plug 'n Play configuration information is read from the EEPROM and loaded into the appropriate PCnet-ISA+ control registers.

If the adapter is set up as a boot device (per the Plug 'n Play Specification), it becomes active at reset and operates as a normal AT card. When the adapter is first installed, you run a configuration program to set the configuration information (e.g., I/O address, memory map, etc.).

If the adapter is not set up as a boot device, it remains inactive at reset and becomes initialized by the Plug 'n Play software.

The EEPROM contains the following information:

- All ISACSRs
- 48-bit IEEE Ethernet address
- EISA configuration bits
- I/O port address bits
- IRQ enable bits
- Boot ROM address enable bits
- Serial data stream for programming external logic
- Plug 'n Play ISA configuration information

When the EEPROM does not exist, or in the case where there's an EEPROM checksum error, PCnet-ISA+ waits in the Plug 'n Play WAIT_FOR_KEY state. Your device driver can gain access to the PCnet-ISA+ I/O resources by writing a special AMD initialization string to the Plug 'n Play Address Port at 0x279. The AMD initialization string is 32 hex bytes as follows:

6B,	35,	9A,	CD,	E6,	F3,	79,	BC,
5E,	AF,	57,	2B,	15,	8A,	C5,	E2,
F1,	F8,	7C,	3E,	9F,	4F,	27,	13,
09,	84,	42,	A1,	D0,	68,	34,	1A.

Additional Interrupt Request Lines and DMA Channels

PCnet-ISA+ supports the connection of up to eight programmable interrupt lines (3, 4, 5, 9, 10, 11, 12, and 15) to the ISA system bus. IRQ15 is a dual purpose pin that acts as IRQ15 when the IEEE address is stored in an EEPROM and as Address Prom Chip Select (APCS) when the card manufacturer chooses to store the IEEE address in a parallel PROM. Unless changed by your software after initialization, IRQ3 is the default interrupt and each interrupt is a pulse rather than a level to support an edge triggered Programmable Interrupt Controller (PIC).

In Master Mode operation, PCnet-ISA+ supports the connection of up to four DMA Channels (3, 5, 6, and 7) to the ISA system bus. These four DMA channels are seen on the ISA bus interface as four pairs of DMA Request (DRQ) and DMA Acknowledge (DACK) lines.

Supports Sixteen I/O Base Address Boundaries

Following power-on or reset, I/O address 0x300 is the default Base Address for PCnet-ISA+. Your device driver can write to a Plug 'n Play Register and select from sixteen different Base Address boundaries ranging from 0x200 through 0x3E0.

Supports Eight Boot PROM Locations

Following power-on or reset, memory address 0xC0000 is the default Boot PROM address for PCnet-ISA+. Your device driver can write to a Plug 'n Play Register and select from eight different address boundaries ranging from 0xC0000 through 0xDC000.

Your device driver can write to a Plug 'n Play Register and select one of four PROM sizes (8K, 16K, 32K, or 64K) or you can specify no Boot PROM selected.

Revised External Address Detection Interface (EADI)

A software enabled External Address Detection Interface (EADI) allows external hardware address filtering in parallel with frame reception and address comparison in the MAC Station Address Detection (SAD) block. When the EADI is enabled (ISACSR2: EADISEL, bit 3=1), four PCnet-ISA pins lose their default function and are remapped as EADI control pins. LED control pins $\overline{\text{LED1}}$, $\overline{\text{LED2}}$, and $\overline{\text{LED3}}$ become EADI output control pins SF/BD, SRD, and SRDCLK. The Disable Transceiver pin DXCVR becomes the EADI input control pin $\overline{\text{EAR}}$.

Revised Support for External (LED) Status Indicators

For the PCnet-ISA+ controller, you use I/O write accesses to ISACSR registers 5, 6, and 7 to perform LED control pin programming. ISACSR5 controls the status indicated on the $\overline{\text{LED1}}$ control pin and defaults to Twisted Pair MAU Receive Polarity Status. ISACSR6 controls the status indicated on the $\overline{\text{LED2}}$ control pin and defaults to Twisted Pair MAU Receive Status. ISACSR7 controls the status indicated on the $\overline{\text{LED3}}$ control pin and defaults to Twisted Pair MAU Transmit Status.

Each control register can be programmed to display one or more of the following status indications: Collision Activity, Jabbering, Receive Activity, Receive Polarity on Twisted Pair Interface, Transmit Activity, and Receive Address Match.

Your device driver can now reverse the polarity of the LED2 output pin by setting ISACSR6, bit 14.

Revised Reset Register (offset 0x14) Accesses

An I/O read access of the Reset Register at I/O offset 0x14 creates an internal reset pulse. The controller responds to an internal reset pulse differently from when the RST pin is asserted or the when STOP bit is set. A reset invoked by reading the Reset Register causes certain bits in the Command Status Registers to be automatically cleared. However, the internal reset has no effect on bits in the Bus Configuration Registers.

The NE2100 LANCE based family of Ethernet cards require an I/O write access to the Reset Register following a read access to the Reset Register. The PCnet-ISA+ controller does not have the same requirement, however, if your device driver performs the extra write access there are no negative side effects.

PCnet-32 Controller

Single Chip System Solution with Multiple Bus Interface

The PCnet-32 controller is a single chip Ethernet system solution that interfaces directly with the Am486DX, Am386DX, or VESA VL-Bus local-bus. The highly integrated 160-pin VLSI device reduces the adapter part count and cost, and is applicable for applications demanding higher system throughput.

PCnet-32 contains a local-bus interface unit, DMA Buffer Management Unit, ANSI/IEEE 802.3 Media Access Control engine, ISO 8802-3 (ANSI/IEEE 802.3) defined Attachment Unit Interface (AUI), IEEE 802.3 (Type 10BASE-T) Twisted Pair Transceiver Media Attachment Unit (T-MAU),

support for auto-configuration EEPROM, and individual 136-byte transmit and 128-byte receive FIFOs.

PCnet-32 has the built-in capability of automatically selecting either the AUI port or the T-MAU.

The individual 136-byte transmit and 128-byte receive FIFOs reduce system overhead by providing sufficient latency during packet transmission and reception thus minimizing intervention during normal (i.e., avoidable) network error recovery.

An integrated Manchester Encoder/Decoder provides the Physical Layer Signaling functions required for a fully compliant IEEE 802.3 station and eliminates the need for an external Serial Interface Adapter (SIA).

The PCnet-32 Ethernet controller, in response to configuration pin settings at reset, operates using one of three interfaces: Am486 local bus, Am386 local bus, or the VESA VL-Bus interface.

Revised Programmable Interrupt Support

Supports either 2 or 4 programmable interrupt lines.

Optional 32-bit I/O Mode

For the default case of Word I/O accesses, the PCnet-32 responds to 16-bit accesses at offsets 0x00 through 0x16.

For the optional Double Word I/O access mode, the controller responds to 32-bit accesses at offsets 0x00 through 0x1C. The RDP, RAP, and BDP registers, at offsets 0x10, 0x14, and 0x1C respectively, contain only two bytes of valid data in bits 15-0. The upper two bytes are reserved for future use. Reserved bits must be written as zeros, and when read, are undefined.

The selection of Word I/O mode (WIO) or Double Word I/O mode (DWIO) is accomplished by two methods.

- A hardware reset function
- Automatic mode switching due to double word I/O write to offset 0x10

The PCnet-32 mode setting defaults to word I/O (i.e., DWIO=0) after a hardware reset.

Your device driver can switch to Double Word I/O mode by performing a 32-bit I/O write to the RDP at offset 0x10. Note that the RDP offset remains the same for both I/O modes even though I/O resource mapping changes when the I/O mode changes.

Accesses to non-double word address boundaries and accesses of less than four bytes are not allowed while in DWIO mode. An I/O write access may cause unexpected reprogramming of PCnet-32 control registers. A read access will yield undefined values.

Once the controller enters DWIO mode, only a hardware reset can restore WIO mode.

LANCE, ILACC, and PCnet-ISA Compatibility

PCnet-32 is register compatible with the Am7990 (LANCE), PCnet-ISA, and optionally with the Am79C900 (ILACC) Ethernet controllers. The DMA Buffer Management Unit supports the LANCE and PCnet-ISA descriptor software models and, optionally, the ILACC software structures. The controller is software compatible with Novell NE2100 and NE1500T Ethernet device drivers and the AMD Am1500T and PCnet-ISA device drivers.

Your device driver can force compatibility with ILACC style 32-bit data structures such as the Initialization Block and the Transmit and Receive Descriptor Rings. Note, that the Initialization Block follows a format that is different from the 16-bit version. Both versions of the relevant data structures are illustrated in the Initialization Block and Descriptor Ring Access Mechanism sections near the beginning of this chapter.

ILACC compatibility is enabled by performing a single I/O write to BCR20. You must set the I/O Style Register (bits 7-0) to 0x01.

Revised Reset Register (offset 0x14) Accesses

An I/O read access of the Reset Register at I/O offset 0x14 creates an internal reset pulse. The controller responds to an internal reset pulse differently from when the RST pin is asserted or the when STOP bit is set. A reset invoked by writing the Reset Register causes certain bits in the Command Status Registers to be automatically cleared. However, the internal reset has no effect on bits in the Bus Configuration Registers.

The NE2100 LANCE based family of Ethernet cards require an I/O write access to the Reset Register following a read access to the Reset Register. The PCnet-32 controller does not have the same requirement, however, if your device driver performs the extra write access there are no negative side effects.

New Software Relocatable Mode

After power-up, in the absence of an EEPROM with specific configuration information, the PCnet-32 controller enters Software Relocatable Mode. While in this mode, PCnet-32 snoops on I/O accesses on the system bus but does not directly respond to I/O accesses. When the controller detects a special sequence of 12 consecutive I/O accesses, it presumes that system control software is attempting to relocate the controller to another 32-byte block in the I/O address space.

Following twelve consecutive I/O byte-write accesses to address 0x378, the data written is evaluated for the presence of a special enabling string. If the first four bytes of data are 0x41, 0x4D, 0x44, and 0x01, the subsequent eight bytes of data become the new I/O Base Address, bus interface configuration information, and interrupt configuration information.

Data from accesses 5 and 6 go to BCR16 (I/O Base Address Lower), data from accesses 7 and 8 go to BCR17 (I/O Base Address Upper), data from accesses 9 and 10 go to BCR2 (Miscellaneous Configuration), and the final two bytes from accesses 11 and 12 go to BCR21(Interrupt Control).

When internal control register updating is complete, the controller leaves Software Relocatable Mode and begins responding to all I/O accesses directed to the 32 bytes of I/O address space that begins at the I/O Base Address location.

Revised Internal Loopback Mode

Loopback mode allows the PCnet-32 controller to operate in a pseudo full duplex mode for test purposes. The loopback facilities of the MAC engine allow verification of full operation without disturbance of the network. When in loopback mode, the FCS generator must be allocated to the receiver in order to test the multicast address detection feature of the MAC. All other features, such as automatic transmit padding and receive pad stripping, operate identically in loopback as in normal operation.

In loopback mode, the PCnet-32 controller now allows you to use frames with as few as 8 bytes of data by default. The controller automatically enables Runt Packet Accept during loopback regardless of the setting of RPA (CSR124, bit 3).

Revised Power Down Modes for T-MAU Circuitry

PCnet-32 supports two power down modes, Coma and Snooze, for reduced power consumption in critical battery powered applications.

Coma mode is enabled when the $\overline{\text{SLEEP}}$ pin is asserted and the AWAKE bit (ISACSR2: bit 2) is reset. Upon entering Coma mode, PCnet-32 goes into a permanent deep sleep with the T-MAU circuitry powered down.

Snooze mode is enabled when the $\overline{\text{SLEEP}}$ pin is asserted and the AWAKE bit is set. Upon entering Snooze mode, the T-MAU receive circuitry remains enabled even while the $\overline{\text{SLEEP}}$ pin is asserted. The $\overline{\text{LNKST}}$ output pin also continues to function, indicating a good 10BASE-T link if there are link beat pulses or valid frames present. The $\overline{\text{LNKST}}$ pin can drive external hardware that deasserts to the $\overline{\text{SLEEP}}$ pin of the controller. This configuration effectively wakes the system when there is any activity on the 10BASE-T link. Upon awakening the controller, your device driver must allow 0.5 seconds for the internal analog circuits to stabilize.

Revised External Address Detection Interface (EADI)

A software enabled External Address Detection Interface (EADI) allows external hardware address filtering in parallel with frame reception and address comparison in the MAC Station Address Detection (SAD) block. When the EADI is enabled (BCR2: EADISEL, bit 3=1 and BCR21: REJECTDIS, bit 7=0), four PCnet-ISA pins lose their default function and are remapped as EADI control pins. LED control pins LED1, LED2, LEDPRE3, become EADI output control pins SF/BD, SRDCLK, and SRD. The Interrupt Request 2 pin INTR2 becomes the EADI input control pin EAR.

Revised External (LED) Status Indicators

For the PCnet-32 controller, you use I/O write accesses to BCR registers 4, 5, 6, and 7 to perform LED control pin programming. BCR4 controls the status indicated by the $\overline{\text{LNKST}}$ pin and defaults to Link Status. BCR5 controls the status indicated on the LED1 control pin and defaults to Twisted Pair MAU Receive Polarity Status. BCR6 controls the status indicated on the LED2 control pin and defaults to Twisted Pair MAU Receive Status. BCR7 controls the status indicated on the LED3 control pin and defaults to Twisted Pair MAU Transmit Status. These four control registers are fully programmable by your device driver.

PCnet-PCI Controller

Single Chip System Solution with PCI Bus Interface

The PCnet-PCI controller is a single chip Ethernet system solution that interfaces directly with the Peripheral Component Interconnect (PCI) local-bus. The highly integrated 132-pin VLSI device reduces the adapter part count and cost, and is applicable for applications demanding higher system throughput.

The PCnet-PCI controller contains a local-bus interface unit, DMA Buffer Management Unit, ANSI/IEEE 802.3 Media Access Control engine, ISO 8802-3(ANSI/IEEE 802.3) defined Attachment Unit Interface (AUI), IEEE 802.3 (Type 10BASE-T) Twisted Pair Transceiver Media Attachment Unit (T-MAU), support for auto-configuration EEPROM, and individual 136-byte transmit and 128-byte receive FIFOs.

The PCnet-PCI controller has the built-in capability of automatically selecting either the AUI port or the T-MAU.

The individual 136-byte transmit and 128-byte receive FIFOs reduce system overhead by providing sufficient latency during packet transmission and reception thus minimizing intervention during normal (i.e., avoidable) network error recovery.

An integrated Manchester Encoder/Decoder provides the Physical Layer Signaling functions required for a fully compliant IEEE 802.3 station and eliminates the need for an external Serial Interface Adapter (SIA).

New Driver Support with Interrupt Line Register

The Interrupt Line Register (PCI Configuration Space: offset 0x3C) contains the 8-bit Interrupt Request (IRQ) number assigned to the controller by the system's Power On-Self Test (POST). The POST writes the IRQ number and your device driver reads the value when

it's time to install the Interrupt Service Routine. Writing to this register has no effect on the operation of the controller. This register is intended to be used for passing information from the operating system to the driver.

Optional 32-bit I/O Mode

For the default case of Word I/O accesses, the PCnet-PCI responds to 16-bit accesses at offsets 0x00 through 0x16.

For the optional Double Word I/O access mode, the controller responds to 32-bit accesses at offsets 0x00 through 0x1C. Except for CSR88, all CSRs and BCRs contain only two bytes of valid data in bits 15-0. The upper two bytes are reserved for future use. Reserved bits must be written as zeros, and when read, are undefined.

The selection of Word I/O mode (WIO) or Double Word I/O mode (DWIO) is accomplished by two methods.

- A hardware reset function
- Automatic mode switching to DWIO mode due to double word I/O write to offset 0x10

The PCnet-PCI mode setting defaults to word I/O (i.e., DWIO=0) after a hardware reset.

Your device driver can invoke Double Word I/O mode by performing a 32-bit I/O write to the RDP at offset 0x10. Note that the RDP offset remains the same for both I/O modes even though I/O resource mapping changes when the I/O mode changes.

Accesses to non-double word address boundaries and accesses of less than four bytes are not allowed while in DWIO mode. An I/O write access may cause unexpected reprogramming of PCnet-PCI control registers. A read access will yield undefined values.

Once the controller enters DWIO mode, only a hardware reset can restore WIO mode.

LANCE, ILACC, and PCnet-ISA Compatibility

The PCnet-PCI controller is register compatible with the Am7990 (LANCE) and optionally with the Am79C900 (ILACC) Ethernet controllers. The DMA Buffer Management Unit supports the LANCE and PCnet-ISA descriptor software models and, optionally, the ILACC software structures. The controller is software compatible with Novell NE2100 and NE1500T Ethernet device drivers and the AMD Am1500T and PCnet-ISA device drivers.

Your device driver can force compatibility with ILACC style 32-bit data structures such as the Initialization Block and the Transmit and Receive Descriptor Rings. Note, that the 32-bit Initialization Block follows a format that is different from the 16-bit version. Both versions of the relevant data structures are illustrated in the Initialization Block and Descriptor Ring Access Mechanism sections near the beginning of this document.

ILACC compatibility is enabled by performing a single I/O write to BCR20. You must set the I/O Style Register (bits 7-0) to 0x01.

Revised Reset Register (offset 0x14) Accesses

An I/O read access of the Reset Register at I/O offset 0x14 creates an internal reset pulse. The controller responds to an internal reset pulse differently from when the RESET pin is asserted or the when STOP bit is set. A reset invoked by reading the Reset Register causes certain bits in the Command Status Registers to be automatically cleared. However, the internal reset has no effect on bits in the Bus Configuration Registers.

The NE2100 LANCE based family of Ethernet cards require an I/O write access to the Reset Register following a read access to the Reset Register. The PCnet-PCI controller does not have the same requirement, however, if your device driver performs the extra write access there are no negative side effects.

Revised Internal Loopback Mode

Loopback mode allows the PCnet-PCI controller to operate in full duplex mode for test purposes. The loopback facilities of the MAC engine allow verification of full operation without disturbance of the network. When in loopback mode, the FCS generator must be allocated to the receiver in order to test the multicast address detection feature of the MAC. All other features, such as automatic transmit padding and receive pad stripping, operate identically in loopback as in normal operation.

In loopback mode, the PCnet-PCI controller now allows you to use frames with as few as 8 bytes of data by default. The controller automatically enables Runt Packet Accept during loopback regardless of the setting of RPA (CSR124, bit 3).

Revised BCR16, BCR17, BCR18, BCR19, BCR20 (others?) Support

BCR16 and BCR17, the I/O Base Address registers on the PCnet-32 controller, are not used by PCnet-PCI. Writes to these locations have no effect on the operation of the device.

BCR18 and BCR19 have reserved bits that were defined in BCR18 and BCR19 of the PCnet-32 controller.

Revised External (LED) Status Indicators

For the PCnet-PCI controller, you use I/O write accesses to BCR registers 4, 5, and 7 to perform LED control pin programming. BCR4 defaults to Link Status. BCR5 controls the status indicated on the $\overline{\text{LED1}}$ control pin and defaults to Receive. BCR7 controls the status indicated on the $\overline{\text{LED3}}$ control pin and defaults to Twisted Pair MAU Transmit Status. These three control registers are fully programmable by your device driver.

Revised Auto-Configuration Mode

The PCnet-PCI controller supports the 64-byte header portion of the PCI Configuration Space as specified by the PCI Local Bus Specification. The configuration registers necessary to identify the PCnet-PCI controller and supported functions are implemented.

The configuration registers are accessible only by PCI configuration cycles and may be accessed following power-on while the EEPROM is being read.

In the absence of an EEPROM with specific configuration information, registers that are normally loaded from the EEPROM are set to their default values.

Revised Power Down Modes for T-MAU Circuitry

The PCnet-PCI controller supports two power down modes, Coma and Snooze, for reduced power consumption in critical battery powered applications.

Coma mode is enabled when the $\overline{\text{SLEEP}}$ pin is asserted and the AWAKE bit (BCR2:bit2) is reset. Upon entering Coma mode, PCnet-PCI goes into a low power state with the T-MAU circuitry powered down. Coma mode is the default power down mode.

Snooze mode is enabled when the $\overline{\text{SLEEP}}$ pin is asserted and the AWAKE bit is set. Upon entering Snooze mode, the T-MAU receive circuitry remains enabled even while the $\overline{\text{SLEEP}}$ pin is asserted. The $\overline{\text{LNKST}}$ output pin also continues to function, indicating a good 10BASE-T link if there are link beat pulses or valid frames present. The $\overline{\text{LNKST}}$ pin can drive external hardware that deasserts the $\overline{\text{SLEEP}}$ pin of the controller. This configuration effectively wakes the system when there is any activity on the 10BASE-T link. Upon awakening the controller, your device driver must allow 0.5 seconds for the internal analog circuits to stabilize.

Revised External Address Detection Interface (EADI)

The PCnet-PCI controller does not support an External Address Detection Interface (EADI).

Dynamic Device Identification

Each device in the PCnet family provides a unique chip identification number located in the Chip ID Register in CSR88 and CSR89. Your software can read these registers and determine the exact device type.

The 16-bit identification numbers for the PCnet family are as follows.

Table 14. Chip ID Numbers

Controller	Chip Identification Number CSR 88 CSR89
PCnet-ISA	xxxx 0000 0000 0000 0011 0000 0000 0011
PCnet-ISA+	xxxx 0010 0010 0110 0000 0000 0000 0011
PCnet-32	xxxx 0010 0100 0011 0000 0000 0000 0011
PCnet-PCI	xxxx 0010 0100 0010 0000 0000 0000 0011

GUIDE TO WRITING PCnet FAMILY DRIVERS

A network adapter driver is the interface between the network operating system and the adapter hardware. The details of this interface are intimately connected to the operating system, so that the idea of writing a generic driver is not really feasible. Different systems assign more or less of the network protocol generation duties to the adapter driver. Usually though, for transmission the upper layer software generates the whole frame except for preamble and FCS and passes to the adapter driver the size and address of the buffer or buffers that contain the data to be sent. On receipt of a frame the adapter driver usually passes to the upper layer software the entire frame except for preamble and FCS without interpreting any header information. On the other hand some adapter drivers use the Ethernet type field of the received frame to determine which protocol stack to pass the frame to.

Most adapter drivers include four types of functions:

1. *The initialization routines set up data structures and initialize the hardware.*
2. *The status and control routines perform functions such as managing the multicast tables, resetting the hardware, and reporting statistics.*
3. *The transmit routine passes frames to the hardware and starts transmissions.*
4. *The interrupt service routine (ISR) handles incoming packets, services end of transmission interrupts, collects statistics, and handles hardware errors.*

Pseudocode for a very simple PCnet controller driver is shown at the end of this application note.

Initialization

The requirements of initialization routines vary greatly from one operating system to another. Part of the initialization task is to set up links to the upper layer software. Specifically the upper layer needs to be able to call the driver's status and control routine and its transmit routine. In the other direction the driver must be able to call the upper layer's receive routine after a packet has arrived.

The initialization tasks that are found in all PCnet family device drivers are setting up the initialization block and the descriptor rings, allocating memory for transmit and receive buffers, initializing the hardware, and

installing an interrupt vector. Most drivers also set up counters to keep track of the number of packets and bytes sent and received and the number of various types of errors encountered.

The driver also needs to set up pointers into the transmit and receive descriptor rings. Usually you need one pointer into the receive ring and two into the transmit ring. The pointer into the receive ring shows which descriptor will point to the next received frame. The pointers into the transmit ring point to the head and tail of a queue of frames that are in the process of being sent. The tail of queue pointer is used by the transmit routine to find the next available descriptor, while the head of queue pointer is used by the transmit interrupt service routine to find the status of frames that have just been sent.

The hardware setup requirements of course depend on the type of computer and the operating system. For ISA bus computers, the driver must first find out what I/O base address is assigned to the PCnet controller so that it can access the controller's registers. It must also set up the host DMA controller and programmable interrupt controller.

To set up the host DMA controller in an ISA bus system, the driver must find out what DMA channel is assigned to the controller and set that channel into the cascade mode. In the cascade mode the DMA controller performs bus arbitration (determines which potential bus master should gain control of the bus), but does not provide addresses and read/write signals. The driver must also clear the mask bit for the selected DMA channel.

To set up the programmable interrupt controller (PIC) in an ISA bus system, the driver must find out what interrupt channel is assigned to the controller and clear the PIC's mask bit corresponding to that channel.

The driver initialization routine must also set up the appropriate interrupt vector so that an interrupt from the PCnet controller will cause the interrupt service routine to be executed.

Status And Control Routines

These routines are highly dependent on the conventions of the network operating system. They can include functions to retrieve statistics, reset the hardware, change the number and sizes of buffers, etc.

Two common control functions deserve mention. These are the functions that add addresses to and delete addresses from the multicast table. When the driver adds an address to the multicast table, it should also set the corresponding bit in the Logical Address Filter (LADRF) so that the PCnet controller will accept frames directed to this multicast address. The driver determines which bit to set by applying to the multicast address the same Cyclic Redundancy Check (CRC) algorithm that the PCnet controller uses to verify the FCS field of the frame. It then uses the six highest order bits of the resulting CRC code as a pointer to the bit in the LADRF. AC program to perform this calculation is shown at the end of this section.

Once the bit is selected, the driver sets the appropriate bit in the LADRF field of the initialization block and reinitializes the PCnet controller. Since the STOP bit in CSR0 must be set before the device can be initialized, the hardware descriptor pointers are set to point back to the base addresses of the descriptor rings. The driver must also set its three descriptor pointers back to the start of the descriptor rings, set the OWN bits of all transmit descriptors to 0, and set the OWN bits of all receive descriptors to 1.

When the driver deletes an address from the table, it has to do more than just clear the corresponding bit in the LADRF. This is true because more than one multicast address can map to one bit in the LADRF. The most straight forward way to adjust the LADRF after an address has been removed from the table is to clear the LADRF then recalculate the filter bits for all addresses in the table. Since changing the multicast table is a very rare event, the speed of the routines that add and delete addresses is not critical.

The Transmit Routine

When the upper layer has a packet to send, it calls the driver's transmit routine and passes to it information about the packet. This information includes the size and location of the buffer or buffers that contain the packet. Depending on the design of the driver, the driver either copies the packet data to one or more of its own buffers, or it writes to the next available transmit descriptor the physical address of the buffer containing the packet. In either case the driver starts the transmission process by setting the OWN bit of the appropriate descriptor(s). The driver can expedite the transmission by setting the Transmit Demand TDMD bit in CSR0 to avoid waiting for the next transmit poll, which happens about every1.6 ms.

The Interrupt Service Routine

The most complicated part of a driver is the interrupt service routine (ISR). The ISR must handle any hardware errors, the reception of packets, and the end of transmission interrupts. The ISR also increments any appropriate statistics counters.

When an interrupt occurs, the ISR reads CSR0 and optionally CSR4 to determine the cause of the interrupt. CSR0 and CSR4 are designed so that the driver can clear the interrupt condition by writing back the values just read. This clears the old interrupt flags without disturbing any flags that may have been set since CSR0 was read. It is a good idea to leave the Interrupt Enable (IENA) bit in CSR0 cleared until the end of the ISR. This will prevent a second network event from causing a

nested call to the ISR even though CPU interrupts may be enabled. At the end of the ISR, you can disable CPU interrupts, set the IENA bit, then execute the return from interrupt command. Setting the IENA bit in CSR0 does not affect any other bits in CSR0. Therefore, if a second network event has occurred while the ISR was executing, the ISR will be called a second time without losing any information.

If the interrupt was caused by the end of a transmission, the TINT bit in CSR0 will be set. The ISR must check for errors in the descriptor that the start of transmit queue pointer points to. Depending on the requirements of the operating system, the ISR may also call a routine in the upper layer software to do some post processing such as releasing buffers. Finally, the ISR must move the head of transmit queue pointer to the next descriptor.

If the interrupt was caused by an incoming frame, the ISR examines the current receive descriptor to look for errors and to get the size of the frame. If there are no errors, and multicast addressing is allowed, the ISR must next get the destination address from the buffer that the descriptor points to. If the least significant bit of the first byte of the address is ONE, the address is multicast, and the driver must search the multicast table (a software data structure) to see if this address is in the list of acceptable addresses. Note that logical address filter mechanism is not a perfect filter. The driver must still maintain a multicast address list in software.

If there are no errors and the frame passes the filter tests, the driver then calls the upper layer's receive routine and passes to it the size and location of the received frame. Since the message count in the receive descriptor includes the FCS field, the driver may have to subtract 4 from the message count, depending on whether or not the upper layer receive routine expects the size to include the FCS. Also note that the buffer address in the descriptor is a physical address, which may have to be converted to a different format such as a 80x86segment:offset pointer.

On returning from the upper layer receive routine, the driver sets the OWN bit in the descriptor to return the descriptor and its associated buffer to the PCnet controller, and it increments its own receive descriptor pointer.

Since another network event may have occurred while the first interrupt was being processed, the driver should loop back and test the OWN bits of the next receive and transmit descriptors and continue until all outstanding transmit and receive frames have been processed.

PCnet Driver Pseudocode

The following pseudocode describes the principal routines for a simple driver for a PCnet family Ethernet controller.

The initialization routine:

```
initialize()
{  reset_PCnet_controller;
   setup_DMA_controller;
   setup_interrupt_controller;
   setup_initialization_block;
   setup_receive_descriptor_ring;
   setup_transmit_descriptor_ring;
   start_tx_queue = first_tx_descriptor;
   end_tx_queue = first_tx_descriptor;
   current_rx_queue = first_rx_descriptor;
   install_interrupt_vector;
   write_CSR0 (INIT);

   do
   {read_CSR0();
   }  until IDON == 1;

   write_CSR0 (STRT | IENA);
}
```

The transmit routine:

```
transmit()
{
   Examine OWN bit of tx descriptor at tx_queue_tail;
   if (OWN == 1) return “Out of buffers”;

   Get buffer address from descriptor at end_tx_queue;
   Copy packet to tx buffer;
   Set up BCNT field of descriptor;  /* Write the negative of the packet size. */
   Set OWN, STP, and ENP bits of descriptor;
   tx_queue_tail = tx_queue_tail + 1;
   if (tx_queue_tail > last_tx_descriptor) tx_queue_tail = first_tx_descriptor;
   Set TDMD bit in CSR0;
   return “OK”;
}
```

The interrupt service routine:

```
isr()
{  status = read_CSR0();
   write_CSR0 (status & !IENA);  /* Turn off interrupts from PCnet device. */
   enable system interrupts;
   do
   {repeat = service_xmt();
   repeat = repeat OR service_rcv();
   } until repeat == 0;

   if (status & ERR) inc_error_counters();
   disable interrupts;
   write_CSR0 (IENA);  /* Turn on interrupts from PCnet device. */
   return_from_interrupt;
}
```

```
service_xmt()
{  Examine OWN bit of tx descriptor at tx_queue_head;
   if (OWN == 0)  return 0;

   Examine ERR bit of tx descriptor at tx_queue_head;
   if (ERR == 1)  update_tx_error_counters();
   else update_frame_&_byte_count_statistics();

   increment tx_queue_head;
   if (tx_queue_head > last_tx_descriptor)  tx_queue_head = first_tx_descriptor;

   return 1;
}

service_rcv()
{  Examine OWN bit of rx descriptor at current_rx_desc;
   if (OWN == 1)  return 0;

   Examine ERR bit of rx descriptor at current_rx_desc;
   if (ERR == 1)
   {update_rx_error_counters();
   goto release_descriptor;
   }

   Examine MCNT field of descriptor at current_rx_desc;
   if (MCNT is not a legal size)
   {update_rx_error_counters();
   goto release_descriptor;
   }

   Examine DA field of buffer that rx descriptor points to;
   if (not multicast address OR address is in multicast table)
   {update_frame_&_byte_count_statistics();
   call upper layer receive routine;
   }

release_descriptor:
   Set OWN bit in descriptor at current_rx_desc;
   increment current_rx_desc;
   if (current_rx_desc > last_rx_desc)  current_rx_desc = first_rx-desc;
   return 1;
}
```

The following C Program generates LADRF bits for multicast addresses:

```
/************************************************************
*   hash.c      Rev 0.1   7/25/91
*
*   Generate a logical address filter value from a list of
*   Ethernet multicast addresses.
*
*  Input:
*      User is prompted to enter an Ethernet address in
*      Ethernet hex format:  First octet entered is the first
*           octet to appear on the line. LSB of most
*           significant octet is the first bit on the line.
*           Octets are separated by blanks.
*      After results are printed, user is prompted for
*      another address.
*
*      (Note that the first octet transmitted is stored in
*      the LANCE as the least significant byte of the Physical
*      Address Register.)
*  Output:
*      After each address is entered, the program prints the
*      hash code for the last address and the cumulative
*      address filter function. The filter function is
*      printed as 8 hex bytes, least significant byte first.
************************************************************/
#include
void updateCRC (int bit);
    int adr[6],     /* Ethernet address */
        ladrf[8],   /* Logical address filter */
        CRC[33],    /* CRC register, 1 word/bit + extra control bit */
        poly[] =    /* CRC polynomial. poly[n] = coefficient of
                       the x**n term of the CRC generator polynomial. */
           {1,1,1,0, 1,1,0,1,
            1,0,1,1, 1,0,0,0,
            1,0,0,0, 0,0,1,1,
            0,0,1,0, 0,0,0,0};
void main()
{
        int k,i, byte;  /* temporary array indices */
```

```
       int hashcode;   /* the object of this program */
       char buf[80];   /* holds input characters */
   for (i=0;i
   printf (“Enter Ethernet addresses as 6 octets separated by blanks.\n”);
   printf (“Each octet is one or two hex characters. The first octet \n”);
   printf (“entered is the first octet to be transmitted. The LSB of \n”);
   printf (“the first octet is the first bit transmitted. After each \n”);
   printf (“address is entered, the Logical Address Filter contents \n”);
   printf (“are displayed, least significant byte first, with the \n”);
   printf (“appropriate bits set for all addresses entered so far.\n”);
   printf (“       To exit press the
key.\n\n“);
 while (1)
  {
  loop:
      printf (“\nEnter address:  ”);
      /* If 1st character = CR, quit, otherwise read address. */
      gets (buf);
      if ( buf[0] == ’\0’) break;
      if (sscanf (buf, “%x %x %x %x %x %x”,
            &adr[0], &adr[1], &adr[2],&adr[3],&adr[4],&adr[5])
          != 6)
          {  printf
              (“Address must contain 6 octets separated by blanks.\n”);
             goto loop;
           }
      if ((adr[0] & 1) == 0)
           {  printf (“First octet of multicast address ”);
              printf (“must be an odd number.\n”);
              goto loop;
           }
      /* Initialize CRC */
      for (i=0; i
      /* Process each bit of the address in the order of transmission.*/
      for (byte=0; byte
         for (i=0; i
             updateCRC ((adr[byte] >> i) & 1);
      /* The hash code is the 6 least significant bits of the CRC
         in reverse order:  CRC[0] = hash[5], CRC[1] = hash[4], etc.
```

```
       */
       hashcode = 0;
       for (i=0; i
       /* Bits 3-5 of hashcode point to byte in address filter.
          Bits 0-2 point to bit within that byte.  */
       byte = hashcode >> 3;
       ladrf[byte] |= (1 < (hashcode & 7));
       printf (“hashcode = %d (decimal)  ladrf[0:63] = ”, hashcode);
       for (i=0; i
          printf (“%02X ”, ladrf[i]);
       printf (“ (LSB first)\n”);
    }
}
void updateCRC (int bit)
{
    int j;
    /* shift CRC and control bit (CRC[32]) */
    for (j=32; j>0; j--) CRC[j] = CRC[j-1];
    CRC[0] = 0;
    /* If bit XOR (control bit) = 1, set CRC = CRC XOR polynomial. */
    if (bit ^ CRC[32])
       for (j=0; j
}
```

Trademarks

AMD, the AMD logo, and combinations thereof are trademarks of Advanced Micro Devices, Inc.

Am186, Am386, Am486, Am29000, *b*IMR, eIMR, eIMR+, GigaPHY, HIMIB, ILACC, IMR, IMR+, IMR2, ISA-HUB, MACE, Magic Packet, PCnet, PCnet-*FAST*, PCnet-*FAST*+, PCnet-Mobile, QFEX, QFEXr, QuASI, QuEST, QuIET, TAXIchip, TPEX, and TPEX Plus are trademarks of Advanced Micro Devices, Inc.

Microsoft is a registered trademark of Microsoft Corporation.

Product names used in this publication are for identification purposes only and may be trademarks of their respective companies.