Architectural Choices for Network Performance

National Semiconductor
Application Note 873
Larry Wakeman
John Von Voros
February 1993

TABLE OF CONTENTS

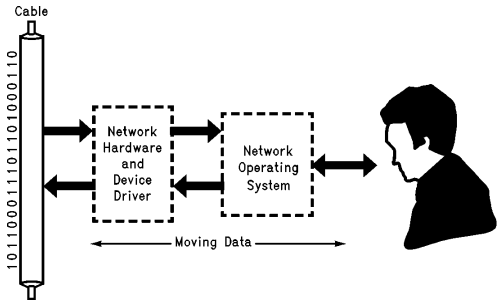
INTRODUCTION
NETWORK PERFORMANCE
Defining Network Performance
Where's the Bottleneck?
Measuring Performance
ARCHITECTURAL OVERVIEW
Hardware Interfaces
I/O Mapped Slave
Shared RAM Slave
Simple Bus Master
Buffered Bus Master
Intelligent Bus Master
Buffer Management Architectures
Transmit Requirements
Receive Requirements
Operating System Requirements
Hardware Packet Buffering Schemes
Simple DMA Buffering
Buffer Ring DMA
Linked List Packet Handling
SYSTEM APPLICATIONS
PC®/PC-AT® Client Adapter
PC (ISA Bus) Server Adapter
PC Motherboard Applications
PS/2® and EISA Server Adapter
PC System Bus (CPU Bus)
PC System I/O Bus Design
CONCLUSION

INTRODUCTION

Recently the popularity of networking has grown, and as a result virtually any type of computer system and many peripherals are incorporating facilities to connect to a network. With the integration of the network function onto just a few IC's, the design of the interface circuitry to the network's physical interface is becoming simpler. However, the design of the interface can be implemented many different ways, with varying tradeoffs.

The basic design tradeoffs for interfacing a system to Ethernet are fairly simple:

1. **Performance.** Generally this is measured in terms of the amount of data transmitted and received in a given time period. The more data the better. As shown in *Figure 1*, the purpose of a Network interface is simply to: Move Data. The interface should be as fast and efficient as possible.



TL/F/11784–1

FIGURE 1. The Success of a Network Interface is it's Ability to Quickly, and Safely Move Data to/from the User

2. **Low Cost.** Obviously the user would like to pay as little as possible for the network connection.
3. **Compatibility.** Both software and hardware compatibility to established industry standards is crucial. In the PC market, this means the ability to work with standard software (i.e., Novell's NetWare® and Microsoft's Windows for Workgroups®) and hardware. In the non-PC market, interoperability with other network components is the key to successful integration into an existing network.

Unfortunately, these simple goals can lead to choosing dramatically different designs for the Ethernet interface subsystem. The diversity of computer architectures (both hardware and software) requires a unique balance of all of these criteria.

This paper will concentrate on the application of Ethernet in PC type computer systems (i.e., Intel 286, 386, 486 CPUs). Both hardware and software issues will be addressed as they pertain to performance, cost, and compatibility. The considerations presented here are applicable to other computer systems as well.

NETWORK PERFORMANCE

Obviously one of the major tradeoff's in developing a network interface solution is performance versus cost.

But: What is network performance?

Defining Network Performance

Network performance is a measure of the ability of a particular network configuration to move data from one computer to another. Typically, this data movement occurs from a server to a workstation or client. Unfortunately, there is no standard method of measuring and benchmarking performance, due to the multitude of network and node configurations. We shall dissect the components of performance in an attempt to describe the roles that hardware and software play in a typical network.

RRD-B30M75/Printed in U. S. A.

When a user makes a request for information or data not resident on his own computer (like opening a memo in his word processor when the memo is located on a remote system), the network's pieces must all respond to this request. The user's perception of performance is determined by how long he must wait for the information to appear on his screen.

When an inquiry is made on the network for some information, a complex set of transactions occurs. The user's computer operating system tells the network protocol to send a message to the server asking for the information. The protocol software then instructs the driver to send the request to the hardware interface, which in turn sends data over the cable to the server. After the packet has been received, an acknowledgement is sent to the server indicating that a valid transfer was accomplished.

The reverse procedure occurs on the server end. When the hardware receives the request, the driver is instructed to pass it on to the network protocol. The computer operating system takes the request from the protocol and issues a response (the actual data) which is sent back through the network in a similar fashion.

Each of these software and hardware components manipulates the requests and responses to ensure proper delivery of the data to/from each destination. This process is shown in *Figure 2*. Each step requires time for the software and hardware to execute its piece of the job. The sum total of the time it takes for each operation to occur is the performance of the transaction.

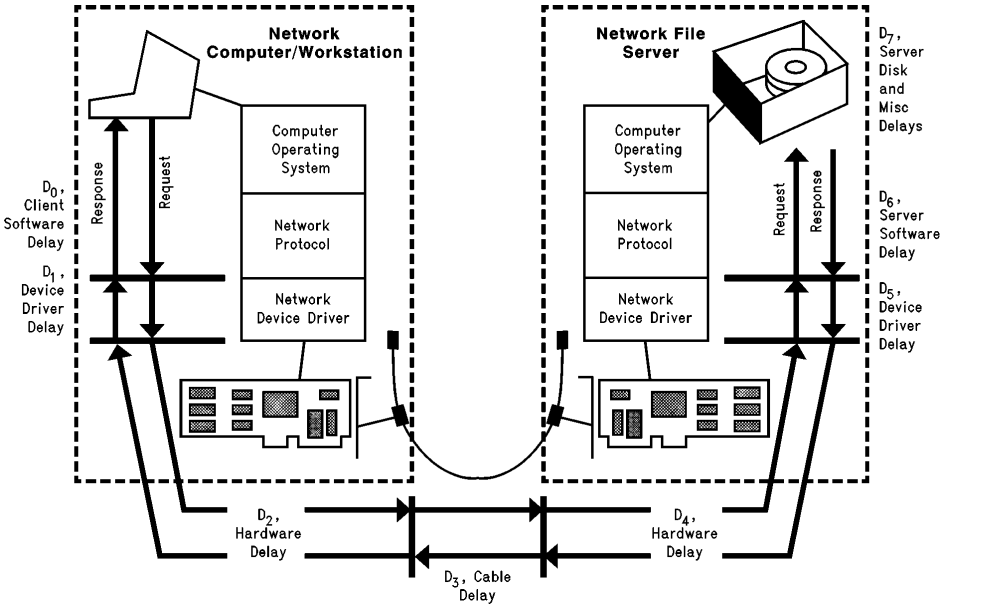
WHERE'S THE BOTTLENECK?

There are many variables in the performance equation, and the network interface card is only one factor. Much like Ethernet cards, the overall performance of a system can be divided into hardware and software issues. On the hardware side, the speed of a network is determined by the performance of the server and all of the attached workstations. The network operating system is a major contributor to the overall latency of a network.

The response time of a server or workstation is affected by CPU speed, bus bandwidth, network loading, and the speed and topology of the transmission media (such as coax, twisted-pair, optical fiber, etc.). The server should provide sufficient disk cache memory and a fast hard disk subsystem to minimize delays. Typically, throughput on a loaded network segment can be reduced to under 20% of maximum by random disk I/O requests. Improving individual workstation throughput has very little impact on the overall network since it only affects a small percentage of the total load. The expense of outfitting a high performance server can be amortized over the cost of the entire network since all users will benefit.

The network operating system (NOS) can also have a dramatic affect on Ethernet throughput. The two main functions of the NOS are to move large blocks of data around in RAM and manage the disk I/O subsystem. Excessive copying of data or poor file management will result in poor LAN performance.

Since each item in the request/response path contributes to overall performance, it is desirable to minimize delays through each section. From the Ethernet hardware developer's perspective, the efficiency of the NOS and the cable throughput are fixed. The only areas available for improvement of the network hardware performance are optimization of the driver and the bus interface design. It is important to recognize that the hardware and driver are a small (but important) part of the total performance equation. In most



TL/F/11784–2

FIGURE 2. Representation of System Delays that Impact Performance

cases doubling or tripling performance of the network hardware will have a much smaller impact on total performance. (Whereas doubling or tripling the CPU performance could have a much greater affect.)

As can be seen in *Figure 2*, the total speed at which requests/responses can be handled is dictated by the serial path shown. For the analytically minded:

$$\text{Performance} = \frac{K_P}{\Sigma(\text{Request Delays})}$$

Basically, network performance is the reciprocal of the sum of the request delays for the measured transactions. K_P is a multiplier constant to convert to kbytes/sec.

A single request/acknowledge delay is the sum of the individual delays as shown in *Figure 2*.

$$\text{Request Delay} = (D0 + D1)\,(2\,K_{CPU1}) + (D5 + D6)\,(2\,K_{CPU2}) + 2(D2 + D3 + D4) + D7$$

D0, D1, D3, D5, D6 are all software delays and thus are multiplied by the performance of the system processors, while D2, D3, D4 are network hardware delays that are not affected by software performance (ideally). D7 is the delay due to system hardware other than the network interface, such as a disk and controller.

Measuring Performance

Generally, one would like to measure these delays and calculate the throughput of a particular network configuration. Most benchmarks can only evaluate the summation of these delays. This is done by measuring the actual data throughput, usually in kbytes/sec or in seconds for a particular task.

Now, of course every good marketer has his favorite benchmark, but the validity of a benchmark can be very deceiving. Most Ethernet vendors cite the Novell **PERFORMx** benchmark as a performance measure, this is valid but does not accurately model the request/response of a real network. Some testing labs have developed scripts that simulate user transactions in an attempt to create a more representative view of network speed, and these tend to be more valid. However, due to the ease of testing and general availability of the **PERFORM** program most comparisons utilize this program's measurements.

The perform benchmark tends to measure the throughput of the network interface, the protocol software, and the system CPU. The data being transferred is cached in the server's memory, so disk drive speed is not a factor. This means that the largest delay for a system is not taken into consideration.

Another important aspect of performance is the amount of time the server CPU is idle. The more CPU bandwidth that is available the greater the potential for the server to do other processing or to handle more information. This becomes important when the server is being used as a database engine or if multiple Ethernet cards are used concurrently. A server that is oversaturated will drop packets, and thus decrease performance because each of these packets will have to be retransmitted. Generally, CPU utilization indicates the potential for better performance rather than actual performance. In the NetWare world, the CPU utilization measure is made using the **MONITOR** program which is run on the server. A meaningful benchmark should contain both the throughput and the CPU utilization figures. For example, a system with 100 kbyte/sec throughput with a 10% utilization (90% idle) should be better than a 100 kbytes/sec with a 90% utilization. When making comparisons between architectures, it is important to use the same equipment for each test since the benchmarks are also measuring the server's and workstations' performance. The network should have at least six workstations to ensure heavy Ethernet traffic.

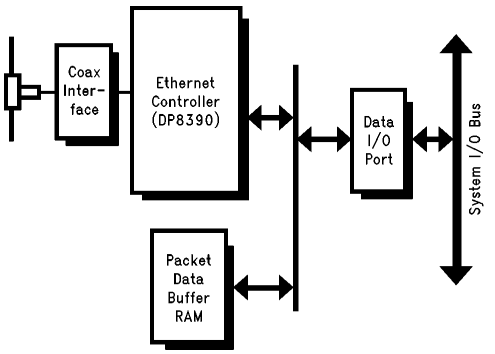
ARCHITECTURAL OVERVIEW

The architecture of an Ethernet card encompasses a hardware interface to the system as well as a software interface, which is really the packet buffer management that the hardware implements with the device driver software. This section will cover the hardware aspects first since this is very much the systems designer's decision. Secondly a description of various software interfaces is described, although in most cases this is actually defined by the integrated controller chosen for a particular design.

Hardware Interfaces

Before discussing actual applications, it is useful to categorize Ethernet interfaces. Once done, this can be applied to various applications to determine the best fit for each application. A summary can be found in Table 1. The interfaces from the Ethernet subsystem to the host's system bus can be divided into roughly 5 categories as follows:

1. **I/O Mapped Slave:** In this design, the adapter interfaces to the system via a limited number of I/O ports, usually 16 bytes–64 bytes. The interface has dedicated RAM to buffer network data, usually 8k to 64 kbytes. A simple block diagram is shown in *Figure 3*. National's DP8390 core controllers have on-chip logic to ease implementation of this interface.



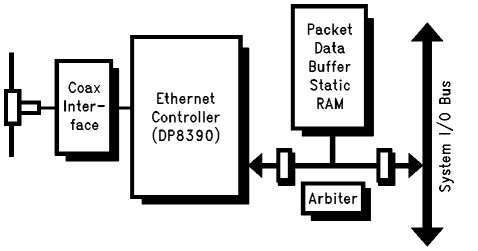
TL/F/11784–3

FIGURE 3. Diagram Showing Major Blocks of the I/O Mapped Architecture

Advantages: A very simple interface, tends to be low cost. Does not occupy large address space (important for PC's with many peripherals competing for common address allocations). Places little performance requirements on the system bus, and so is ideal for systems that have poor bus bandwidth or long bus latencies.

Disadvantages: May be slightly lower in performance. Requires dedicated buffer RAM which can add to cost of the interface.

2. **Shared RAM Slave:** This architecture utilizes a RAM that is dual ported (usually a static RAM with an arbiter rather than an integrated dual port RAM) to enable the network interface and the main system to communicate through a common "window" of memory, as shown in *Figure 4*. The DP8390 has request/acknowledge logic to simplify implementation of this interface.



TL/F/11784–4

FIGURE 4. Basic Block Diagram of the Shared Buffer RAM Design

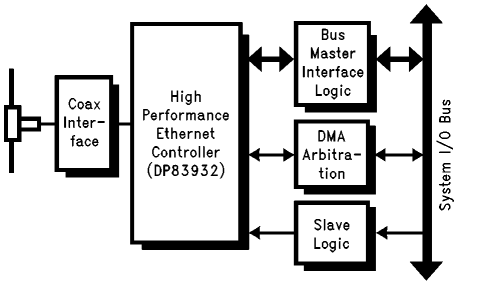
Advantages: A fairly simple interface that is slightly higher in performance than the I/O Slave since the data is directly accessible to the system via the buffer RAM.

Disadvantages: Tends to be more complex than the I/O interface, and thus more expensive. The additional cost is due to the logic required to dual port the buffer RAM which includes a couple of extra PALs, 4–6 extra octal buffers, and some added logic for software control. (However, as integrated or ASIC solutions become available this extra logic can be absorbed inexpensively making the solution cost equivalent to an I/O mapped design.) This architecture places slightly greater performance requirements on the system bus than the I/O Slave since the system is contending for the buffer RAM with the network interface.

The Shared RAM architecture is not the best choice when the host system has a limited addressing range and/or its memory cycles aren't significantly faster than its I/O cycles. In a PC-AT compatible application, the Shared RAM design typically has a 30% faster bus transfer speed, however when the effects of driver and Network Operating System (NOS) overhead are considered, the advantage of the Shared RAM design is reduced to 10% or less. In Micro Channel or EISA systems, the difference in I/O vs memory transfer rates is less, so the performance disparities would be reduced (assuming the network hardware does not present any other constraints).

Note: These relative numbers do not include disk access overhead, so the performance difference seen by a user will typically be lower.

3. **Simple Bus Master:** For the simple bus master interface, the network peripheral directly requests the system bus, and when granted, takes control of the bus and directly places packet data into system memory *(Figure 5)*. The performance of this design is heavily dependent on the sophistication and speed of the DMA channel logic, and the bus itself. In the past, most Ethernet controllers did not have sufficient bus speed or buffer management to support this type of architecture, hence it's relatively new emergence has coincided with the introduction of high performance controllers such as the SONIC™ DP83932.



TL/F/11784–5

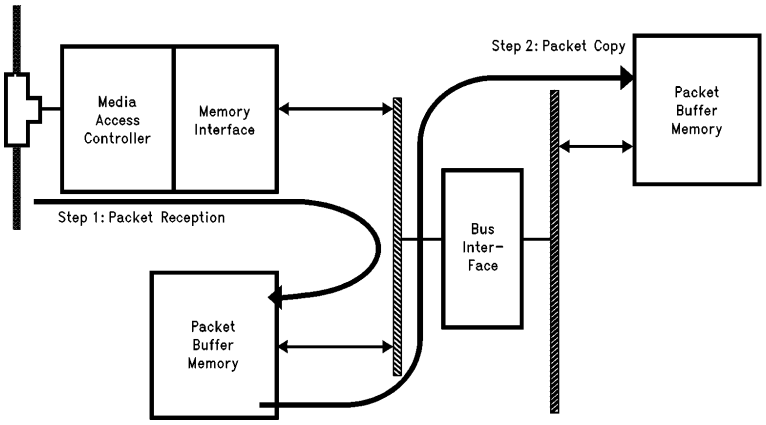
FIGURE 5. Simple Bus Master Block Diagram

In order to understand memory buffer structures, it is useful to compare the packet movement of a Shared Memory or I/O Interface (local memory) to that of the Bus Master design. In a local memory design *(Figure 6a)*, the data is first buffered, then copied to the system memory. The Bus Master, on the other hand *(Figure 6b)*, places the data directly into system memory. This latter approach is significantly faster because local memory designs require an extra read and write cycle to move the contents of the local buffer RAM into system memory. In theory, the Bus Master can eliminate additional data copies. Performance is reduced if the Bus Master cannot buffer directly to the NOS and a data copy has to be executed. The fact that local memory requires data buffering in two steps is not as significant to performance as the method of moving the data into system memory (i.e., DMA, I/O channel, etc.). Bus latency is one of the prime considerations when deciding to use a bus master approach. Newer generation controllers rely on a FIFO to buffer data until the bus becomes free for transfers. Depending on the computer, this delay can be longer than the maximum time allowed by the FIFO. When a FIFO overrun (or underrun) occurs, the packet must be retransmitted. In theory, the maximum bus latency tolerated by a controller can be calculated by the equation:

$$\text{Latency}_{max} = \frac{\text{Depth of FIFO (Bytes)}}{10\ \text{Mbits/S} \times 0.125\ \text{Byte/Bit}}$$

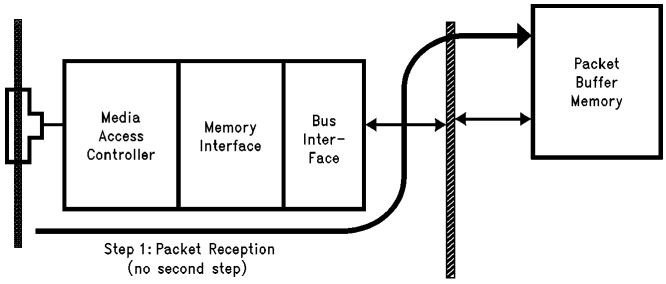
There are two classes of bus masters which for this document we will call a MAC (Media Access Control) Bus Master, and a Bus Master (for lack of better names). The distinction is that a MAC Bus Master becomes owner of the bus, but utilizes some form of system or external DMA controller to actually move the data. In other words the MAC Bus Master cannot generate addressing for the received or transmitted data. The Bus Master, on the other hand, utilizes a DMA controller that is built into the MAC. This DMA controller is capable of controlling data movement in a reasonably sophisticated way, and can place data into or take data from any desired location.

Advantages. Properly designed with enough intelligence in the packet buffering (i.e., not typically a MAC Bus Master), this implementation provides a very high performance data transfer throughput. If the DMA master is sophisticated enough, data can be placed directly into system memory, thus eliminating extraneous data copying by the driver software as is required by I/O mapped and Shared RAM designs.



TL/F/11784–6

(a) Shared Memory or I/O Architecture



TL/F/11784–7

(b) Bus Master Architecture

FIGURE 6. Comparison of Data Movement

Disadvantages. In order to achieve high performance, the DMA machine must be sufficiently sophisticated (not a MAC Bus Master). In many low performance bus systems, direct bus ownership is not supported. In other buses, such as ISA, the bus is not sophisticated enough to arbitrate between potential bus owners without tying up the CPU unnecessarily. In some high performance bus interfaces, the latency from bus request to bus grant can be very long and require on-card buffering of the data to avoid dropping packets.

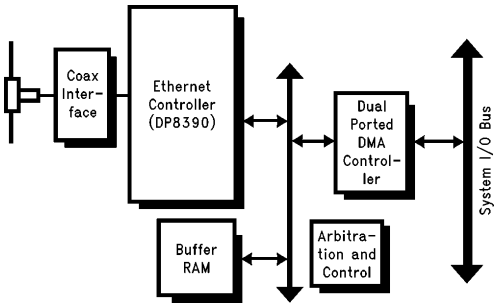
When comparing software driver performance, the efficiency of the driver plays a bigger role in Bus Master designs. This is because the Bus Master's hardware transfer is very efficient and the overhead of the driver is a bigger percentage of the data throughput. Typically in a PC (ISA bus) the bus master data transfer rate is 2–2.5 times that of an I/O based design. When software overheads are included, however, the Bus Master design typically achieves a performance increase over the I/O design. (Driver inefficiencies in reality can reduce this by about 5%.)

Note: These relative numbers do not include disk access overhead, so the performance difference seen by a user will typically be lower.

4. **Buffered Bus Master.** In this design the network packet data is DMA'd by a network controller through the on-card bus into a buffer RAM *(Figure 7)*. The packet data is then tranferred to the system by additional logic that DMA's the data across the system bus into main memory. The performance of this architecture is comparable to that of the standard bus master with a marginally higher use of CPU bandwidth.

   Advantages. This architecture provides high performance close to the simple bus master design even in situations where there are extremely long bus latencies (i.e., EISA).

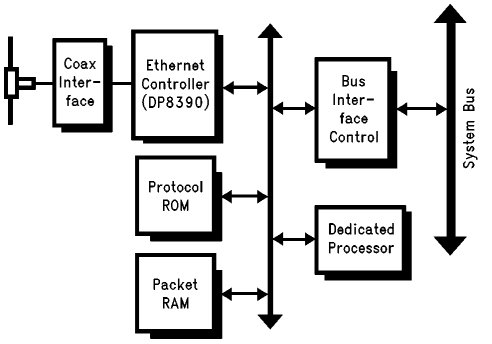
   Disadvantages. When compared to the simple bus master interface, the cost of implementation is higher since this design requires additional buffer RAM and a complex system bus DMA channel in addition to the network interface's DMA channel. If the DMA interface is not sophisticated, the performance will be lower and the software driver will have to do additional processing.



TL/F/11784–8

FIGURE 7. Buffered Bus Master Block Diagram

5. **Intelligent Bus Master.** This design has a general purpose processor dedicated to the network interface for processing packet data *(Figure 8)*. For low-end cards, the processor does not do protocol processing but only performs packet data manipulation and controls access between the system and the network interface. On high end designs, the dedicated network CPU does protocol processing which off-loads this task from the main system. The transfer of data to the system may be via an I/O, shared RAM, or bus master interface.



TL/F/11784–9

FIGURE 8. Intelligent Bus Master Block Diagram

Advantages. When using a processor with sufficient performance, this solution offers the highest performance of any of the solutions. This design also allows for the highest bus latencies, since on board RAM can store many packets. This architecture can off-load the server's CPU from processing the low level protocol tasks and can thus achieve very low server utilization relative to other technologies.

Disadvantages. Very costly in terms of hardware and component count. In order to achieve significant packet throughput advantages, the dedicated processor must be able to process packets at least as fast as the host CPU. In many cases, the low end cards are less efficient than simple bus master cards in terms of packet throughput.

In most practical examples, medium performance 16-bit processors are chosen and this choice tends to offer lower packet throughput than any of the other architectures. Typical CPU loading is roughly half that of a non-intelligent bus master.

BUFFER MANAGEMENT ARCHITECTURES

Just as the performance of a particular hardware implementation depends on how fast data is transferred to the system, the packet buffer management scheme helps determine how fast data can be transferred from the hardware to the Network Operating System. A great hardware design can be foiled by a poor software interface.

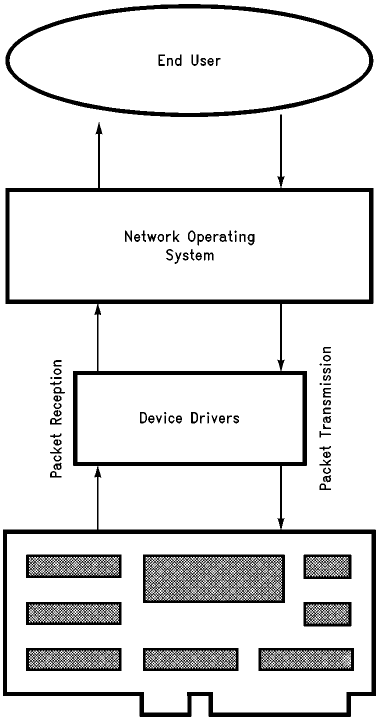
High performance software designs will reduce software complexity and directly provide the data to the NOS in a way that the NOS expects. There are cost/performance tradeoffs in this interface as well, and so there are various buffering methods.

Before discussing the types of buffering that hardware may choose to implement, it is first useful to look at what operating systems expect for packet data. The goal is to minimize the device driver overhead, so a look at what information and in what form the NOS expects it is important.

Figure 9 shows a representation of how the user sends and receives data to/from the NOS. For sending packets, the NOS breaks up the data into smaller pieces so that they can fit within an Ethernet frame. The driver then takes this data and presents it to the hardware. On reception, the hardware gives data to the driver which in turn passes this data to the NOS. The NOS translates the data (if necessary) to a form acceptable to the user's application.

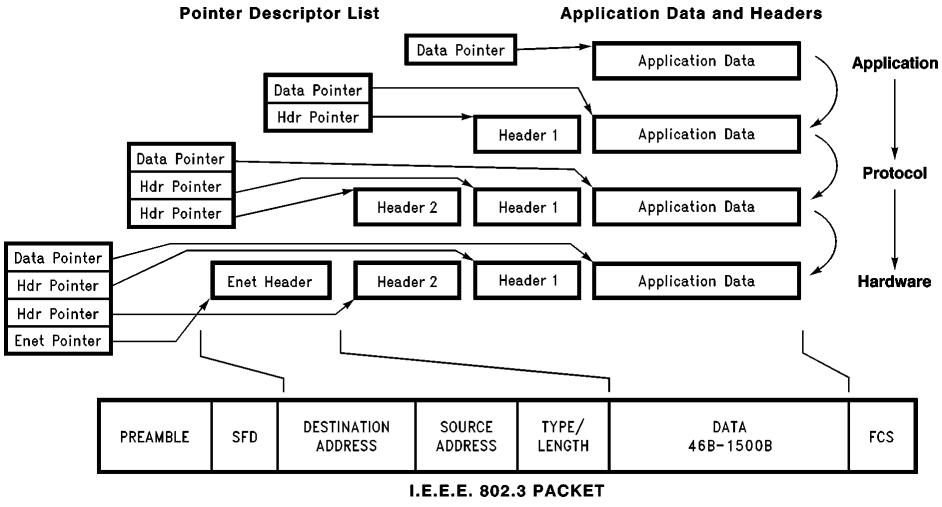
Transmit Requirements

The simpler part of the packet buffering scheme is the transmission of a packet. In this instance, the packet starts from the user's application, and the NOS prefixes network data, usually referred to as headers, to the application data as shown in *Figure 10*. These headers can contain protocol, routing, or application specific information. The most efficient method of ''prefixing'' is to create a pointer list which describes the data pieces' locations rather than copying all the data into a contiguous area. The driver receives this list from the NOS and then sends the information to the hardware. The network interface controller should be able to use this list with as little driver software manipulation as possible. The hardware and its associated driver must also be able to queue multiple requests since the network cable can only send one packet at a time.



TL/F/11784–10

FIGURE 9. User-Cable Data Movement through NOS to Driver to Hardware



TL/F/11784–11

FIGURE 10. Transmit Packet Creation

Receive Requirements

One might assume that the receive packet handling should be very similar to the transmit, but there are a number of differences. First, unlike a transmit packet, the NOS, driver, and hardware have no idea when a packet may arrive, what kind of packet (i.e., IPX, TCP/IP, DECnet, etc.) it is going to receive, and how may packets may be received in a given time. While a packet to be transmitted is statically resident in memory until it is operated on, a packet being received must have sufficient memory allocated to it prior to reception. The amount of memory set aside must be equal to the maximum packet size since there is no way to predetermine a packet's length.

These unknowns require a different type of buffer management. Since a packet could be received at any moment, the driver or the hardware must allocate memory before the packet arrives. The system should provide enough memory to handle several packets at a time in case the packets are received faster then they can be processed. This memory allocation is accomplished in hardware by most architectures. A dedicated packet buffer RAM on the network card allows sufficient space to receive multiple packets (this capability is the reason that dedicated hardware RAM is used). All but the simple bus master architecture have a dedicated packet buffer.

In the simple bus master, the driver must allocate a dynamic pool of system memory, and so the structure of the receive portion of the driver depends more on the memory allocation scheme of the NOS than the network side. Thus for simple bus masters to be effective, they should implement a memory allocation/management scheme similar to that of the NOS to simplify data manipulation.

Another consideration is that ultimately the packet will be given to the host's operating system, so the hardware/driver should present the data in a compatible format. The NOS will fragment the packet to remove all of the headers and give the data to the receiving application in the form it accepts.

As can be seen in *Figure 11 a–c*, there are several possible schemes for dealing with received packets. In cases where only one packet type is being received, the hardware/software may be able to fragment the packet into it's multiple headers as it is received and to place each header into a separate area for manipulation (called protocol fragmentation buffering). In some cases, this scheme is efficient since data copying can be eliminated (in theory). The down side for the NOS is that each layer must keep track of several pointers. When multiple protocols and packet types are involved, this type of scattering is difficult because the hardware will have to receive enough of the packet to determine its type, and then either the hardware or software must find appropriate memory to place the fragments. This effort is required because different packet types have different header lengths and contents.

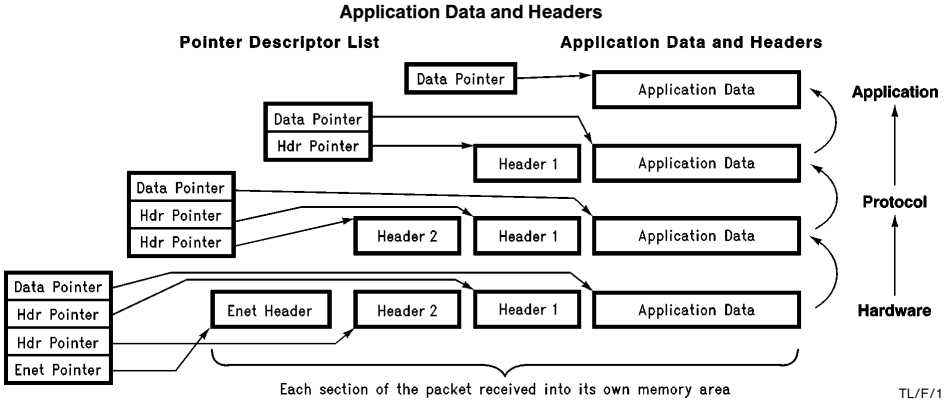
Another possible reception method is to fragment the packet into small buffers, usually 256 or 512 bytes. This will result in a large Ethernet packet being chopped into 3 pieces *(Figure 11b)*. There are several advantages to this technique:

1. Operating system memory management uses these block sizes, so memory allocation is simplified by not having to allocate large contiguous memory blocks.
2. Most Ethernet packets are relatively small, usually <256 bytes. This type of memory scheme is more memory efficient than if a packet were contiguously buffered to a single area. This is because each memory area must be able to accommodate a full Ethernet packet 1518 bytes, and if a small 100 byte packet is received then the rest of the packet buffer will contain unused memory. (However these days operating systems with huge multi-megabytes of memory are common, and a few extra kbytes of packet buffers is typically not a big problem.)
3. In some schemes, the beginning of a new packet may be buffered directly at the end of the previous packet which causes additional fragmentation and hence more pointers (but is more memory efficient).

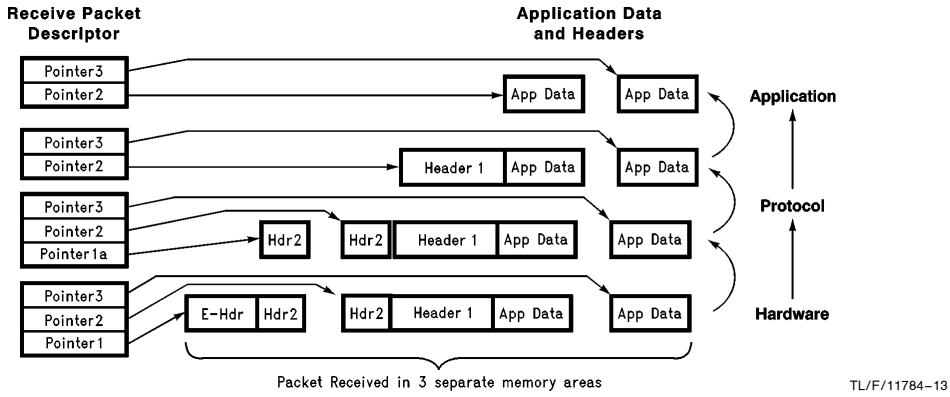
The problem with this second scheme is performance. When a packet is scattered as shown in *Figure 11b*, multiple pointers are required to keep track of the packet, this means that there is more software overhead associated with maintaining the pointers, and it is possible that some fields within the packet may be split between two buffers. Fortunately, this split is not likely to occur in the packet headers if >64 byte buffers are used, but the application data will be split, and may need to be copied to a contiguous buffer by the NOS prior to handing it off to the receiving application.

The problems with the two associated schemes can be overcome by having the hardware buffer the packet into a single contiguous memory area *(Figure 11c)*. This allows the protocol software to have only one pointer to describe a packet. When a header is processed, the old pointer is thrown out, and the next header's pointer is easily created. The packet data arrives at the application in contiguous form. Multiple pointers to packet headers can easily be created if the software requires it. In multiple packet type applications, it is easier/cheaper for the software (as opposed to hardware) to determine the type of protocol/packet type for the received packet and manipulate the data based on this determination.

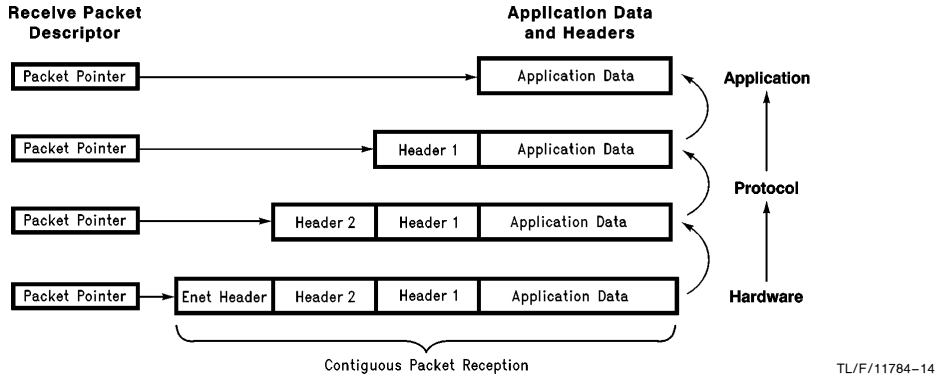
POINTER DESCRIPTOR LIST



(a) Receive Packet Scattered by Section



(b) Received Packet Scattered Based on Small Allocated Memory Blocks



(c) Received Packet Buffered to a Contiguous Memory Area

FIGURE 11

Operating System Requirements

As has been described, the optimal buffering scheme for packets depends, in part, on the way the NOS interacts with the device driver and the hardware. At the device driver level, the NOS defines a programming interface for the exchange of packets between the hardware/driver and the NOS. The type of interface helps to determine the desired buffering scheme.

Note: Hardware architecture also affects this choice.

For non-embedded applications which use standard operating systems, the most prevalent transmit schemes provide the driver with a list of locations for the various portions of the packet. The driver/hardware then assembles the packet to prepare for transmission. Several schemes are used for receiving packets, but contiguous packet reception is the most popular. Packet scattering based on small memory block allocation is also quite common.

Most operating systems require a certain byte alignment on received packets to conform to their memory management schemes. For example, 32 bits OS's usually require double word alignment, while PC DOS (an 8-bit OS), most often demands byte alignment. In general, the transmit alignment is usually byte oriented because the header fragments generated may be an odd number of bytes.

Hardware Packet Buffering Schemes

Keeping in mind the general characteristics outlined above, several hardware packet buffering techniques can be compared. The NOS requirements do not change based on the hardware architecture or the buffering scheme chosen, so when the hardware does not provide optimal algorithms, the device driver software is required to complete the job.

When the hardware packet buffering scheme minimizes bottlenecks (particularly software overhead), the theoretical performance of the driver/hardware set will increase. This section compares major buffering schemes and how they map into the NOS operations, hopefully revealing an indication of the better architectures.

The several schemes can be categorized as follows:

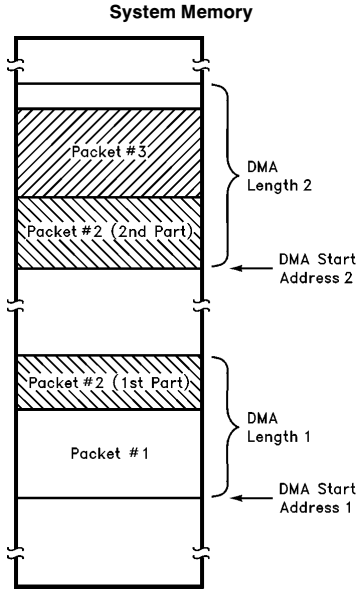
1. Simple DMA: Utilizing a simple start address and length, the system DMA performs all memory transfers.
2. Buffer Ring: A block of memory is setup as a recirculating ring where data at the top of the memory block automatically wraps to the bottom, and pointers track the used/open memory space.
3. Linked Lists: A number of descriptor structures that describe blocks of memory. Each block can contain either a part of a packet, a single packet, or multiple packets.
4. Protocol Translation: This scheme must be implemented on an intelligent card since the on board CPU performs driver tasks as well as protocol processing. The designer can customize the hardware using any combination of the three previous buffering schemes if the native CPU is not needed for protocol translation. This buffering scheme, due to it's unique programmability, will not be discussed separately.

SIMPLE DMA BUFFERING

Typically this buffering architecture is used when the Ethernet Hardware is a simple MAC (Media Access Controller) Bus Master card that uses the system's DMA to provide a low cost solution. It is possible that the System DMA can be used in conjunction with a hardware scheme that includes a dedicated buffer RAM (like Shared RAM).

Reception tends to be a problem for the simple MAC bus master, so this is discussed first. Incoming packets are buffered in a small FIFO and a request is made for the system DMA controller to transfer the data. Simple bus master cards that do not use local memory must have access to the system bus before the FIFO fills, or the packet will be dropped. In addition, the host CPU must be able to allocate new blocks of memory as they are needed, which can be significant in terms of software overhead. For ISA based PC's, the DMA transfer rate is between 1.0–2.0 Mbytes/second which is not sufficient to keep up with the Ethernet data rate.

The transmit operation requires the DMA controller to transfer data from the host's memory to the controller's FIFOs *(Figure 12)*. If the FIFOs are not large enough to buffer the maximum packet size, care must be taken to avoid a FIFO underrun because partial packets will be transmitted on the network. Once again, the host's CPU is responsible for all memory management.



TL/F/11784–15

FIGURE 12. System Memory Packet Reception

Buffer Ring DMA

For this architecture, the controller's memory manager utlizes a ring structure comprised of a series of contiguous fixed length buffers in a local RAM *(Figure 13)*. The ring is typically divided into a transmit and receive section by the driver software. During reception, incoming data is buffered to the receive portion of the ring and then transferred to the system by the local DMA channel. The memory manager is responsible for three basic functions during reception: linking receive buffers for long packets, recovery of buffers when a packet is rejected, and recirculation of memory blocks that have been read by the host.

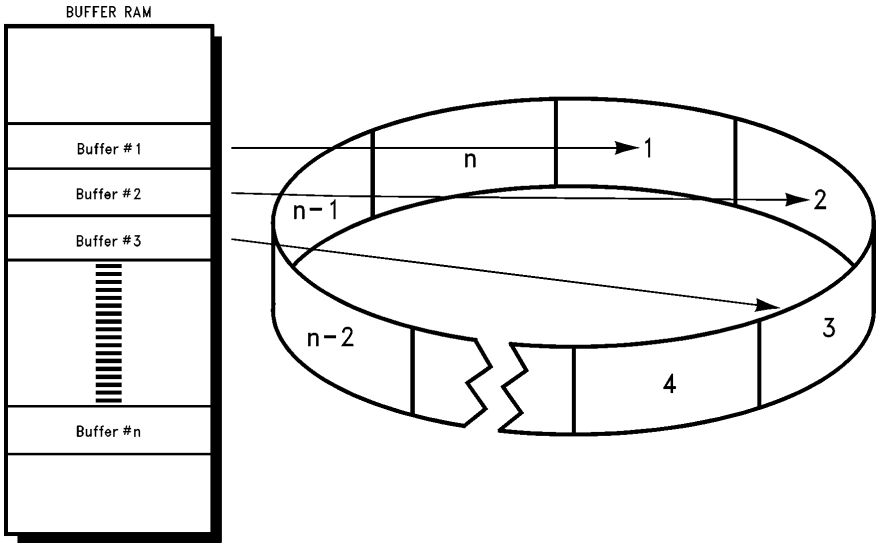
When transmitting data, the software driver must first assemble the packet in the transmit portion of the ring using DMA transfers. This information must include the destination address, the source address, the type/length of the packet, and the data itself. When transmission begins, the controller's local DMA channel transfers the data out of the ring and into the controller's FIFO. The controller sends out the data and appends a CRC field. The block of buffer memory used by the packet is then returned for reuse.

Linked List Packet Handling

In a linked list structure, packets that are received or transmitted are placed in buffers that are linked by lists of pointers. The advantage to this approach, as mentioned earlier, is that it should eliminate unnecessary packet copying. The software driver pre-allocates memory for receiving data and stores a list of pointers to available buffer pages in a Receive Resource Area *(Figure 14)*. Another list of pointers (Resource Descriptor Area) is created when packets are received. Each pointer in this list corresponds to the starting address in memory of the received packet. Multiple packets can be stored in the same buffer area as long as their total length does not exceed the buffer page size.

The transmit buffer management scheme uses two areas in memory for transmitting packets *(Figure 15)*. The Transmit Descriptor Area contains several fields that describe the packet to be transmitted and a pointer to the descriptor of the next packet to be transmitted. Quite often, operating systems store packet header information in one portion of memory and application data in another. Each of these fields is called a fragment.

The typical Ethernet packet contains multiple fragments, so the linked list buffering scheme provides pointers to each piece as well as a count of how many fragments are in the packet. In contrast, the buffer ring architecture would have required the driver to copy all of the fragments and the applications data into a contiguous area of local RAM prior to transmission. The buffer ring is a simple lower performance packet buffering scheme. The linked list structure adds some complexity, but will increase performance when properly tailored to the network operating system.



TL/F/11784–16

FIGURE 13. Buffer Ring DMA Structure

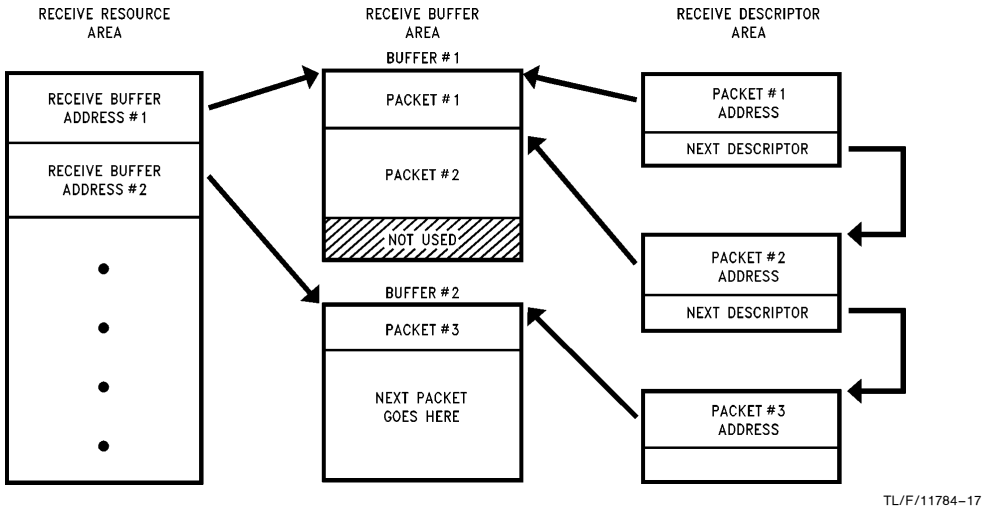


FIGURE 14. Linked List Packet Receive Buffer Structure

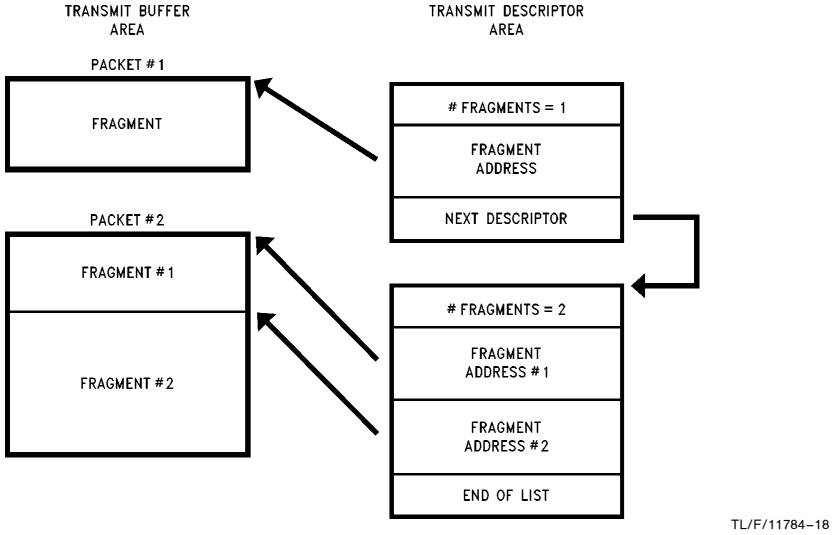


FIGURE 15. Linked List Packet Transmit Structure

SYSTEM APPLICATIONS

For a certain application, many architectures may prove adequate, but the *best* solution may not be obvious. The following section wil discuss the general design tradeoffs to each approach as they apply to personal computers and office peripherals.

PC/PC-AT Client Adapter

While many companies promote performance, most simple interfaces prove to be sufficient, therefore cost and compatibility are of greater importance. The I/O mapped design is by far the most prevalent architecture for PC client cards due to good packet throughput performance and low cost. I/O mapped cards also tend to be the easiest to install because the I/O space of PC's tends to be less crowded then the memory space.

Shared memory cards must map their local packet RAM into the PC's address space between 640k and 1M. In the DOS environment, this space is crowded with BIOS ROMs, EGA/VGA video RAM, and disk controller hardware. Depending on the machine, configuration address contention can result. Also, the PC-AT bus timing for dual ported RAMs is tricky and varies somewhat between clones, thus making compatibility a more difficult issue.

For bus mastering cards, arbitration is not well implemented on many PC-AT compatibles and is not available in the PC-XT. These types of cards seem to have the most trouble interfacing to PC-AT clones because they are sensitive to system timing and the addition of other add-in cards. An improperly designed card can interfere with DRAM refresh and thus cause catastrophic failure. Earlier Ethernet controllers are not suitable for this approach because the bus cycles are excessively slow, thus limiting CPU performance.

Note: PS/2's with a microchannel bus provide very good arbitration, so bus masters are much more suitable for this environment.

The other architectures previously mentioned tend to be too expensive for the minor performance gains that would be achieved.

Performance. Assuming that the I/O port design is the reference, then the relative performance of the shared RAM design offers between a 2%–7% packet throughput improvement. Faster 386 based machines tend to minimize any advantages when the NOS is taken into consideration. The bus master could offer up to 10% speed improvement, but when using older Ethernet controllers this improvement is less. The major advantage to this approach can be lower CPU utilization.

PC (ISA Bus) Server Adapter

Until very recently, servers have been mostly high performance ISA Bus PC-ATs. Due to multiple users, the traffic on the server is much higher than the client, and so bottlenecks in packet transfer will be more noticeable. Servers may also have to support multiple network cards to allow for the internal bridging of networks. The weakest link in these systems tends to be the relatively slow ISA bus.

For ISA bus servers, I/O mapped, shared RAM, and bus master designs all have their advantages and disadvantages. The best overall solution could be the I/O mapped design in spite of its slightly lower performance; it is compatible to industry standards, lower cost, and offers reasonable performance. Multiple cards can easily be employed since this interface does not put a burden on the bus, nor tie up needed RAM space.

The Dual-Ported RAM design offers slightly better performance, and won't tie up the bus, but does use precious memory space. This solution may prove less suitable if several cards are required in the Server.

Simple bus masters can provide the best performance, but since multiple cards can tie up the PC-AT's ISA bus and prevent CPU and refresh from getting sufficient access to the bus, this could present a problem when multiple cards are placed on the bus. No standard exists for supporting arbitration among multiple bus masters. Some initial implementations will only allow installation of one card due to slow bus cycles.

Performance. Assuming that the I/O port design is the reference, then the relative performance of the shared RAM design offers between a 2%–7% packet throughput improvement. The bus master could offer up to 10% speed improvement.

PS/2 and EISA Server Adapter

The 32-Bit 386/486 PS/2 and EISA machines require the best performance, and cost tends to be a secondary issue. In addition, both these buses have intelligent bus arbitration schemes for bus ownership. The major difference between the two is that the arbitration scheme for EISA has the potential for having a relatively large bus latency, whereas the Micro Channel bus latency tends to be lower. This difference affects the bus mastering approaches taken.

The I/O mapped scheme is not optimal since cost is less of an issue and performance is more important. The Dual Port RAM scheme is a better choice as bus transfer speeds can be optimized by using fast RAMs, but the cost of the associated RAMs and logic is more (4-32kx8 SRAM are typically used for on-card buffering).

A simple bus master can be a very good choice if the desired bus latency is accommodated and an intelligent buffering scheme is implemented. This architecture can cost the same as a shared memory design, but provide faster packet throughput. Also multiple cards can easily be accommodated.

For the best performance, a well implemented intelligent board which does on board protocol processing is the best choice. However, the cost is prohibitive, and while overall server CPU usage can be minimized, typical implementations do not offer the best overall throughput.

PC Motherboard Applications

The goals in designing Ethernet connectivity onto PC motherboards differs from those of PC add-in cards. First, due to severe competition and a network oriented focus, add-in board designers tend to concentrate on both the cost and performance of an implementation. The primary concern for PC motherboard designers is CPU performance and compatibility to existing standards. The purpose of including Ethernet is to provide added value and a simple inexpensive connection. Board space and power consumption tend to be more critical on motherboards.

Applications on the motherboard fall into two design categories:

1. An adapter card design folded down onto the main system board.
2. A system bus interface (or local bus) directly connected to the CPU.

The best approach depends on whether the Ethernet's design goal is primarily cost or performance driven. The folded-down design offers compatibility with well established standards and thus the lowest risk. The system interface architecture offers better performance at the expense of complex system design considerations.

PC System I/O Bus Design

The easiest approach to embedded Ethernet is to simply graft an existing PC adapter's design onto the motherboard or daughter card. This places the controller in a less performance critical area of the overall system design (the I/O bus) and allows a migration path from a solution that is known to work. Since backward compatibility is achieved, investments in software and experience are preserved. This design can be applied across an entire line of PC's with no modification to the hardware or software.

The most common implementation would be to ''fold'' an ISA bus 16-bit Ethernet card design onto the motherboard and thus provide a common interface for both ISA and EISA. The growing popularity for ISA based adapters has led semiconductor suppliers to provide very highly integrated solutions for this environment. Unfortunately the same integration level is not yet available for EISA based 32-bit designs.

The only disadvantage to ''folding down'', an adapter card solution is slightly lower performance. Since the throughput of Ethernet is usually ''cable limited'' this approach is suitable for clients and most servers.

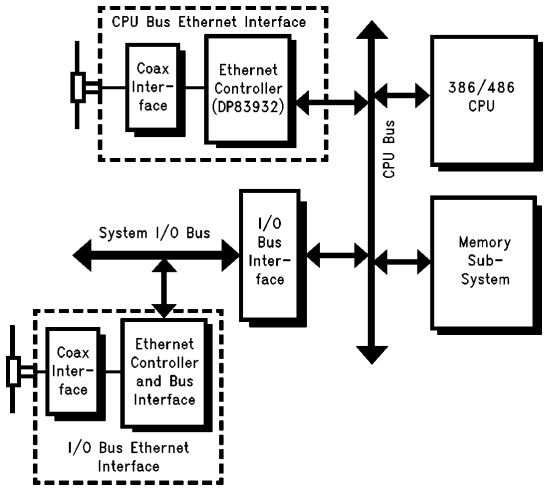
PC System Bus (CPU Bus)

In this implementation, the Ethernet controller is tightly coupled to the system CPU bus (386 or 486). This is illustrated by the top shaded block in *Figure 16*. In a PC, the highest performance bus is the CPU system bus. Ethernet controllers designed to operate in this environment can provide a relatively clean interface with a low parts count. The bus master architecture makes the most sense for this bus due to the simplicity of the interface.

The CPU system bus tends to be the most critical aspect of a PC's overall performance. Adding peripherals to this bus has generally been avoided because I/O functions can reduce bus efficiency. Embedded cache memories on some CPU's help to lessen this burden, but system performance will be affected. Another concern is that this bus was not designed to support the large fanout required for driving multiple I/O devices.

Interfacing to the CPU's bus presents many challenging design problems. The characteristics of this bus are determined by both the CPU and the memory controller. Changes in CPU type and frequency will cause the interface to vary for each PC model in a product line. The controller's operating frequency must be closely matched to that of the CPU to avoid timing problems. This can create problems for modular PC's that offer CPU upgradeability.

Table I summarizes the discussion on PC-Ethernet architectures. It should be noted that the ratings assume that each implementation is a good efficient design. For example, a simple bus master is an excellent architecture only for applications where it meets with the requirements of the bus; this may not always be true. Some qualitative performance references are given, but these should not be taken as valid for every case.



TL/F/11784–19

FIGURE 16. Two Choices for Ethernet on a PC Motherboard

TABLE I. Suitability of Architecture for PC Applications

Architecture (Note 1)	I/O Mapped	Dual Port RAM	Simple Bus Master	Buffered Bus Master	Intelligent
PC, PC AT (ISA), PS/2 (Client) (Note 2)	Excellent	Good	Fair	Poor	Poor
PC AT (ISA Bus) (Server) (Note 2)	Good	Good	Fair	Poor	Fair
PS/2 (Server) (Note 2)	Fair	Good	Excellent	Fair	Excellent
PC AT (EISA) (Client/Server)	Fair	Good	Excellent	Good	Good
PC Motherboard (System Bus) (Note 3)	Good	Good	Good	Poor	Poor

Note 1: The rating from best to worst: Excellent, Good, Fair, Poor.

Note 2: These applications assume that the Ethernet interface is supplied on an adapter card.

Note 3: This application places the Ethernet interface on the PC AT motherboard (system planar) interfaced to the system CPU bus. If the Ethernet interface is placed on the motherboard connected to an I/O bus the architectural choice is represented by the bus (like ISA).

CONCLUSION

The correct choice of an Ethernet controller must be a carefull balance of all of the design goals. In the majority of cases, the throughput of 16-bit Ethernet controllers is more than sufficient (with the exception of servers). For 386 or greater based PC's, throughput is limited by the network operating system and the 10 Mbit/sec. data rate of Ethernet. The decision to use a 32-bit controller should be based on the need for available CPU bandwidth, and to a much lesser extent, throughput.

LIFE SUPPORT POLICY

NATIONAL'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS WRITTEN APPROVAL OF THE PRESIDENT OF NATIONAL SEMICONDUCTOR CORPORATION. As used herein:

1. Life support devices or systems are devices or systems which, (a) are intended for surgical implant into the body, or (b) support or sustain life, and whose failure to perform, when properly used in accordance with instructions for use provided in the labeling, can be reasonably expected to result in a significant injury to the user.
2. A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.

National Semiconductor Corporation
2900 Semiconductor Drive
P.O. Box 58090
Santa Clara, CA 95052-8090
Tel: 1(800) 272-9959
TWX: (910) 339-9240

National Semiconductor GmbH
Livry-Gargan-Str. 10
D-82256 Fürstenfeldbruck
Germany
Tel: (81-41) 35-0
Telex: 527649
Fax: (81-41) 35-1

National Semiconductor Japan Ltd.
Sumitomo Chemical
Engineering Center
Bldg. 7F
1-7-1, Nakase, Mihama-Ku
Chiba-City,
Ciba Prefecture 261
Tel: (043) 299-2300
Fax: (043) 299-2500

National Semiconductor Hong Kong Ltd.
13th Floor, Straight Block,
Ocean Centre, 5 Canton Rd.
Tsimshatsui, Kowloon
Hong Kong
Tel: (852) 2737-1600
Fax: (852) 2736-9960

National Semiconductores Do Brazil Ltda.
Rue Deputado Lacorda Franco
120-3A
Sao Paulo-SP
Brazil 05418-000
Tel: (55-11) 212-5066
Telex: 391-1131931 NSBR BR
Fax: (55-11) 212-1181

National Semiconductor (Australia) Pty, Ltd.
Building 16
Business Park Drive
Monash Business Park
Nottinghill, Melbourne
Victoria 3168 Australia
Tel: (3) 558-9999
Fax: (3) 558-9998