The Architecture and Implementation of a High-performance FDDI Adapter

By Satish L. Rege

Abstract

With the advent of fiber distributed data interface (FDDI) technology, Digital saw the need to define an architecture for a high-performance adapter that could transmit data 30 times faster than previously built Ethernet adapters. We specified a first generation FDDI data link layer adapter architecture that is capable of meeting the maximum FDDI packet-carrying capacity. The DEC FDDIcontroller 400 is an implementation of this architecture. This adapter acts as an interface between XMI-based CPUs, such as the VAX 6000 and VAX 9000 series of computers, and an FDDI local area network.

Fiber distributed data interface (FDDI) is the second generation local area network (LAN) technology. FDDI is defined by the American National Standards Institute (ANSI) FDDI standard and will

The architecture and implementation presented in this paper are for the DEC FDDIcontroller 400, Digital's high-performance, XMI-to-FDDI adapter known as DEMFA. This adapter provides an interface between an FDDI LAN and Digital's XMI-based CPUs, presently the VAX 6000 and VAX 9000 series of computers.[1,2] DEMFA implements all functions at the physical layer and most functions at the data link layer.[3,4]

We begin the paper by differentiating between an architecture and an implementation. Then we present our project goal and analyze the problems encountered in meeting this goal. Next we give a historical perspective of Digital's LAN adapters. We follow this discussion by describing in detail the architecture and implementation of DEMFA. Finally, we close the paper by presenting some results of performance measurement

coexist with Ethernet, the first generation LAN technology.

at the adapter hardware level.

The Architecture and Implementation of a High-performance FDDI Adapter

Adapter Architecture and Implementation

Before we discuss the DEMFA architecture and its implementation, it is necessary to understand what is meant by an adapter architecture and an implementation of that architecture. An adapter architecture specifies a set of functions and the method of executing these functions. An implementation that incorporates all of these functions and conforms to the method of executing these functions becomes a member of the adapter architecture family. Thus, for a given architecture, many implementations are possible.

To grasp the concept presented in the previous paragraph, consider the VAX CPU architecture. This architecture defines the instruction set, which is composed of a set of arithmetic, logical, and other functions, and a format for the instruction set that a processor should implement to be classified as a VAX computer. Examples of VAX implementations are the VAX 11/780 and the VAX 9000 computers, which both conform to the VAX CPU architecture.

Our Goal and the problem Definition

Our goal was to define an architecture for an FDDI adapter that meets the ultimate performance goal of transmitting approximately 450,000 packets per second (packets/s). This goal is considered ultimate because 450,000 packets /s is the maximum packet-carrying capacity of FDDI. Note that this transmission rate is approximately 30 times greater than that of Ethernet, which can transmit approximately 15,000 packets/s.

The Architecture and Implementation of a High-performance FDDI Adapter

Before defining the problem, the basic properties of XMI and FDDI must be understood. XMI is a 64-bit-wide parallel bus that can sustain a 100-megabyte-per-second (MB/s) bandwidth for multiple interfaces.[5] Each interface attached to the XMI bus is referred to as a commander when it requests data or a responder when it delivers data. XMI is an interconnect that can have transactions from several commanders and responders in progress simultaneously.

FDDI is a packet-oriented serial bus that operates using the token ring protocol and has a bandwidth of 100 megabits per second (Mb/s).[6] FDDI is capable of transmitting packets as small as 28 bytes, which take 2.24 microseconds to transmit. Therefore, FDDI can carry approximately 450,000 minimum-size packets /s. The largest packet that FDDI can carry is 4508 bytes. The ANSI/IEEE 802.5 standard defines the FDDI operation; Digital has developed its own implementation of the FDDI base technology as a superset of the ANSI standard.[3]

Our problem was to architect an adapter that could interface XMI, i.e., ANSI defines the protocol for interfacing an adapter to an FDDI LAN.[6] But we had to define the protocol between the adapter and the VMS and ULTRIX operating systems used by most VAX computers. Thus, solving the problem required us to architect a data link layer adapter that would satisfy both protocols and meet the FDDI maximum packet transfer capability.

Historical Perspective

The computer industry has built many LAN adapters since the inception of Ethernet ten years ago. The first LAN adapter built by Digital was the UNIBUS-to-NI adapter (UNA). (NI is Digital's alias for Ethernet.) The Digital Ethernet-to-XMI network adapter, known as DEMNA, is Digital's most recent Ethernet adapter.[7]

Let us choose the maximum throughput rate expressed in packets per second as a performance metric for LAN adapters. The historical perspective shows that the first adapter to meet the Ethernet packet-carrying capacity is the DEMNA. Therefore, it took approximately eight years and six generations for an Ethernet adapter to achieve this throughput

a parallel high-bandwidth CPU bus for VAX computers, to a serial fiber-optic networking bus. To avoid being the bottleneck in a system, such an adapter must be able to transmit or receive 450,000 packets/s.

rate. Consequently, many designers thought that our goal of meeting the ultimate FDDI packet-carrying capacity was impossible.

But the DEMFA architecture, a first generation FDDI data link layer adapter architecture, can meet the maximum FDDI packet-carrying capacity. In this sense, the DEMFA architecture is ultimate.

Traditional Adapter Architectures

In this section, we analyze the traditional adapter architecture and show that by using this architecture we could not meet our performance goal. Figure 1 is a block diagram of a traditional adapter, e.g., DEMNA. In such a design, a CPU in the adapter operates on every transmitted and received packet. Thus, using this traditional architecture to build an ultimate FDDI adapter would require a CPU capable of handling 450,000 packets/s. To predict the performance of such a CPU, we extrapolated from the performance data of the CPU used in DEMNA.[7] This traditional adapter can handle approximately 15,000 packets/s using a CPU rated at 3 VAX units of performance (VUPs).

If we assume a linear model to extrapolate the performance of a CPU from DEMNA to DEMFA, an ultimate FDDI adapter would require at least a 90-VUP CPU. Such a CPU was neither available nor cost-effective for timely shipment of our adapter. Besides, it would be extravagant to use a 90-VUP CPU in an adapter whose host CPU may have a performance as low as 3 to 4 VUPs. Therefore, we looked for a different solution.

DEMFA Architecture

The DEMFA architecture is characterized by the following specifications for functionality and the means to achieve this functionality:

- As mentioned earlier, the DEMFA architecture implements all functions at the physical layer and a major subset of the functions at the data link layer.
- The architecture requires that this functionality be implemented in pipelined stages, which are used to receive and transmit packets over stages to proceed in an asynchronous fashion.
- The architecture requires a packet-filtering capability in the pipelined stage nearest to the FDDI ring; this capability helps to minimize adapter and host resource utilization.
- The architecture specifies the DEMFA port, which minimizes the information transfer required to interact with the host operating system. This interaction takes place during both initialization and the normal operation of receiving and transmitting packets.

In the following sections, we elaborate on different features of the DEMFA architecture.

Pipelined Architecture with No CPU Interference

Once we determined that the traditional architecture of a CPU processing the packets could not meet our performance goal, we began to investigate alternative architectures. The requirement was to either process one receive packet or queue

the FDDI ring without CPU interference.

- The DEMFA architecture specifies a ring interface for communicating between the pipelined stages. Rings operate as queues that allow buffering between pipelined stages, enabling these one transmit packet in a time period less than or equal to the time it takes to transmit on an FDDI ring. Thus, the device we architected must process 28-byte packets in less than 2.24 microseconds. A little thought will show that if we are able to meet the requirements for

processing small packets at the FDDI bandwidth, then the requirements for larger packets can be easily met. Our final choice was a three-stage pipeline approach which broke down the complexity of

implementation while meeting our performance goal. As shown in Figure 2, the three stages of the pipeline in the adapter are the FDDI corner and parser (FCP) stage, the ring entry mover (REM) stage, and the host protocol decoder (HPD) stage. Figure 2 also shows two other functions required of the adapter: the buffering of packets, which requires a memory called the packet buffer memory (PBM) and a memory interface called the packet memory interface (PMI); and the local intelligence, also called the adapter manager (AM).

DEMFA Functions

This section presents brief descriptions of the DEMFA functions and the pipelined stages in which these functions are performed. This, according to our definition, is the DEMFA architecture. A later section, One Implementation of the DEMFA Architecture, describes an implementation in detail.

is also responsible for capturing the token on the FDDI ring, transmitting packets, and implementing the physical layer, e.g., media access control (MAC), functionality required by the FDDI standard.

The REM stage is responsible for distributing packets received over the FDDI ring to the host computer and to the AM. This stage also collects the packets from the host and the AM to queue for FDDI transmission.

The HPD stage interfaces with the XMI bus to move received packets from PBM to the host memory and to move transmit packets from the host memory to the PBM.

The PBM stores the packets received over the FDDI ring and the packets to be transmitted over the FDDI ring. It also stores the control structures required for accessing these packets. The PMI arbitrates the requests made by the three pipelined stages and the AM to access the PBM.

The FCP stage converts serial photons on the FDDI ring into packets and then writes the packets into PBM longwords, 32 bits at a time. The parser implements the logical link control (LLC) filtering functionality. This stage

The AM implements the functionalities of self-test and initialization in the adapter and also a subset of the SMT function required by the ANSI FDDI specification.[8] The adapter manager performs no function in either the receipt or transmission of individual packets to the host.

We use ring interfaces to communicate within the adapter and between the adapter and the host. These interfaces are described in detail immediately following the next section.

Performance Constraints on the Pipelined Stages

Consider the three pipelined stages and their ring interfaces. At any time, the three independent stages are processing different packets. Thus, if the HPD stage is processing received packet 0, the REM stage may be working on received packet 4 and the FCP on received packet 7. Note that packets 1, 2, and 3 wait on a ring between the REM stage and the HPD stage. Similarly packets 5 and 6 wait on a ring between the FCP stage and the REM stage. The PBM must have enough bandwidth to service the three stages. It also must service them

By dividing the processing of a packet over the three stages and the ring interfaces used to queue packets between these stages, we reduced the complexity of the total adapter functionality. Any implementation of this architecture specification would consist of three loosely coupled designs that use ring interfaces to communicate with one another.

Each stage must process a packet in less time than it takes to transmit the packet on the FDDI ring. As we mentioned previously, this transmission time is 2.24 microseconds for the smallest packet. A larger packet may take longer to process than a small packet, but such a packet also takes longer to transmit on the FDDI ring.

Thus, to meet our performance goal, we architected a three-stage pipeline implementation, with each stage meeting a packet-processing time dependent upon the packet size. In addition, our architecture specified a PBM with sufficient memory bandwidth to service the asynchronous requests from the three stages with minimal latency.

Ring Interface-The Core of

with low latency so that the first-in, first-out (FIFO) buffers in the FCP stage do not overflow.

the DEMFA Architecture

The ring interface forms the core of the DEMFA architecture. An interface is necessary to exchange data between the adapter and the host computer and also between the different

stages and functional units of the adapter. Such an interface usually consists of a data structure and a protocol for communication. We evaluated various data structures, including a linked list or queue data structure, and found that a ring data structure is efficient to manipulate and would be easy to implement in state machines, if desirable.

Implementation of Ring Structures. Ring structure implementation requires a set of consecutive memory addresses, as shown in Figure 3. The ring begin pointer and the ring end pointer define the beginning and end of a ring. Two entities, the transmitter and the receiver, interface with a ring to exchange data. The transmitter interface delivers data to the receiver interface using the ring structure. This data resides in memory that is managed by one of the two interfaces. If the transmitter interface manages the memory, the ring is called a transmit ring. If the receiver interface manages the memory, the ring is called a receive ring.

Rings are divided into entries that consist of several bytes each; the number of bytes in an entry is an integral multiple of longwords. A ring, in turn, must contain an integral number of entries. The entry size and the number of entries in a ring determine the ring size. We chose an entry size that is a power of two in bytes and the number of ring entries

to be divisible by two, as well. These choices helped to simplify the hardware implementation used to peruse these rings.

Each entry consists of

- An ownership bit, which indicates whether the transmitter interface or the receiver interface owns the entry
- Buffer pointers, which point to transmitted or received data
- A buffer descriptor, which contains the length of the buffers, and status and error fields

The definitions of these fields in an entry and the rules for using the information in these fields constitute the ring protocol.

Only the interface that owns an entry has the right

to use all the information in that entry. This right includes using the buffer pointers to operate on data in the buffers. Both interfaces have the right to read the ownership bit, but only the interface with

ownership may write this bit.

The two interfaces can exchange entries by toggling the ownership bit. After toggling this bit, the transmitter and receiver interfaces need to prod each other to indicate that the ownership bit has been toggled. This is accomplished using two hardwired Boolean values, by means of an interrupt, or by writing a single-bit register. Hardwired Boolean values are used when both the transmitter and the receiver are on the adapter. Either the interrupt scheme or the method of writing a single-bit register is used when the transmitter and receiver converse over an external bus, e.g., an XMI bus.

The word "signal" is used henceforth to represent the prodding of one interface by the other. A transmitter interface uses "transmit done" to signal the receiver interface that data has been transmitted. A receiver interface uses "receive done" to signal the transmitter interface that the data has been received. Note that we have defined the DEMFA port protocol in such a way that the number of interrupts used to signal the host

The unit of data exchanged between the transmitter interface and the receiver interface is a packet. A packet may be written in a single buffer if the packet is small or over multiple buffers if the packet is large. In this paper, we use the term buffer to refer generically to buffers in the adapter or in the host. The buffers in the adapter are always 512 bytes in size and, when referred to specifically, are called pages. The buffers in the host may be of different sizes.

An exchange of data requires single or multiple buffers, depending upon the packet and buffer sizes. One field of two bits in the buffer descriptor is used to designate the beginning and end of packet. These bits are called the start of a packet (SOP) and the end of a packet (EOP). Thus, for a one-buffer packet both the SOP and the EOP are asserted. For a multiple-buffer packet, the first buffer has the SOP asserted, the middle buffers have both the SOP and the EOP deasserted, and the last buffer has the EOP asserted. The buffer descriptor also contains fields that we do not describe in this paper.

across XMI is minimized to reduce the host performance degradation caused by interrupts.

Data Exchange on a Transmit Ring. Data exchange between a transmitter interface and a receiver interface is accomplished in a similar manner on both transmit and receive rings. Therefore, we discuss the exchange in

detail for a transmit ring; for a receive ring, we note only the dissimilarities.

The events that occur during the data exchange on a transmit ring are shown in Figure 4. The process is as follows. The transmitter interface manages the memory used to exchange data and has two pointers to the ring entries, i.e., the fill pointer and the transmitter free pointer. The transmitter interface uses the fill pointer to deliver data to the receiver interface. The transmitter interface uses the transmitter free pointer to recover and manage the buffers freed by the receiver interface. The receiver interface uses only one pointer, i.e., the receive pointer, which points to the next entry that the receiver interface interrogates to receive data.

To understand how data is transmitted, assume that the pointers move from top to bottom, as shown in Figure 4. Initially, all the pointers designate the location indicated by the begin pointer.

A transmitter that has data to transmit to a receiver uses the entry indicated by the fill pointer. First, the transmitter verifies interface writes a single entry and then toggles the ownership bit and signals the receiver interface.

For multiple buffers, the transmitter interface increments the fill pointer and repeats the two steps described in the previous paragraph to write all the buffer addresses and the length and status information. Then the transmitter interface toggles the ownership bits of all later entries of the multiple buffers before toggling the ownership bit of the first entry. This protocol preserves the atomicity of the packet transfer between the transmitter and receiver interfaces. Then the transmitter interface signals the receiver interface that a packet is available on the transmit ring. This signal alerts the receiver interface, which then examines the entry pointed to by the receive pointer. The receiver interface operates on the entry data if it owns the entry.

The receiver interface returns the entries to the transmitter interface by toggling the ownership bits and then signals receipt of data to indicate the return of the entries

that it owns the entry by checking the ownership bit. Second, the transmitter writes the buffer address and the remaining fields in the entry. In the case of a single buffer packet, the transmitter (and hence the free buffers). Note that there is no need to return these free buffers in a packet, atomic fashion. The transmitter interface uses the transmitter free pointer to examine the

ownership bits in the entry and to reclaim the buffers.

The interfaces operate asynchronously, since each one can transmit or receive data at its own speed. If the transmitter interface can transmit faster than the receiver interface is able to receive, the transmit ring fills up. Under such circumstances, the receiver interface owns all the entries in a transmit ring, the fill pointer equals the transmitter free pointer, and data transmission stops. Conversely, if the receiver interface is faster than the transmitter interface, the transmit ring will be nearly empty. In this case, the transmitter free pointer and the receive pointer are almost always equal.

Note the following invariants that apply to the pointers when data is exchanged on a transmit ring: the fill pointer cannot pass the transmitter free pointer; the transmitter free pointer cannot pass the receive pointer; and the receive pointer cannot pass the fill pointer.

Data Exchange on a Receive Ring. As also shown in Figure 4, the operation of data exchange on a receive ring is similar to that operation on the transmit ring, with the following differences. The receiver interface manages the memory used for exchanging data. Consequently, the receiver interface has two pointers, the receiver free pointer and the receive pointer, and the transmitter interface has only one pointer, the fill pointer.

Table 1 shows the various DEMFA rings and the transmitters and receivers that interface with each ring.

Table 1

DEMFA Rings and Their Transmitter and Receiver Interfaces

Rings	Transmitter	Receiver	Remarks
Rings_in_Packet_Buffer_Memory			
RMC Receive Ring	FDDI Corner and Parser Stage	Ring Entry Mover Stage	Contains data that originated on the FDDI ring.
RMC Transmit Ring	Ring Entry Mover Stage	FDDI Corner and Parser Stage	Contains data that originated at the host or the AM, destined for the FDDI ring.
HPD Receive Ring	Host Protocol Decoder Stage	Ring Entry Mover Stage	Contains data that originated at the host, destined for the FDDI ring.
HPD Transmit Ring	Ring Entry Mover Stage	Host Protocol Decoder Stage	Contains data that originated at the FDDI ring, destined for the host.
AM Receive Ring	Adapter Manager	Ring Entry Mover Stage	Contains data that originated at the AM,

			destined for the FDDI ring or the host.
AM Transmit Ring	Ring Entry Mover Stage	Adapter Manager	Contains data that originated at the FDDI ring, destined for the AM.

Table 1 (Cont.)

DEMFA Rings and Their Transmitter and Receiver Interfaces

Rings	Transmitter	Receiver	Remarks
Rings_in_Host_Memory			
Host Receive Ring	Host Protocol Decoder Stage	Host	Contains data that originated at the FDDI ring or the AM, destined for the host.
Host Transmit Ring	Host	Host Protocol Decoder Stage	Contains data that originated at the host, destined for the FDDI ring.
Command Ring (Transmit Ring)	Host	Adapter Manager	Contains commands that originated at the host for the AM; Note that the AM replies in the same ring.
Unsolicited Ring (Receive Ring)	Adapter	Host Manager	Contains unsolicited messages from the AM to the host.

Subsystem Level Functionality

The basic functions that an FDDI LAN adapter is required to perform are receiving and transmitting packets over the FDDI ring. The adapter must be able to establish and maintain connection to the FDDI network. The connection

The implementation of the complex CMT algorithm in an adapter requires an intelligent component, such as a microprocessor, that can receive, interpret, and transmit packets. Note that the number of CMT packets that flow over the FDDI ring constitutes only a small fraction of the

management (CMT) protocol, a subset of the station management (SMT) protocol, specifies the rules for this connection.[8]

normal traffic. Therefore, a low-performance CPU is adequate to implement connection management. The CPU in the DEMFA device is called the adapter manager.

The packets in the receive stream that originated on the FDDI ring and are addressed to this host or adapter (together called the node) can take one of the following paths:

- Packets not addressed to this node are forwarded over the FDDI ring.
- Packets addressed to this node are delivered to the host computer.
- Packets addressed to this node are delivered to the AM.

The delivery of packets to the host computer implies that the adapter has a pointer to a free memory buffer in which to deposit the received packet. The DEMFA port, described in the next section, specifies the rules for extracting free buffer pointers from the host memory.

For each packet that the host needs to transmit, the adapter must know the buffer address or addresses and the extent of each buffer. The DEMFA port defines the method to exchange this buffer information. In addition, the host and the adapter microprocessor must be able to exchange information.

The DEMFA port specifies the data structure and protocol used for communication between the adapter and the host computer. Rather than invent a new protocol, we modified the DEMNA port specification.[7] The data structure used to pass information between the host and the adapter is a ring structure. Such structures are more efficient to traverse than queue structures.

The DEMFA port defines the four separate host rings listed in Table 1:

- The host receive ring, which contains pointers to free buffers into which a packet received over the network can be deposited
- The host transmit ring, which contains pointers to filled buffers from which packets are removed and transmitted over the FDDI ring by the adapter
- The host command ring, which sends commands to the AM
- The unsolicited ring, which the AM uses to initiate communication with the host CPU

The DEMFA port defines the protocol for this communication also.

DEMFA Port

By using four host rings, we differentiated between the fast and frequent data movement to and from the FDDI ring and the comparatively slow and infrequent data movement required for communication with the AM.

One Implementation of the DEMFA architecture

Previous sections specified the DEMFA architecture. The remainder of this paper describes an implementation of the DEMFA architecture. In the following sections, we present details of the implementation for the packet buffer memory and the packet memory interface; the three pipelined stages, FCP, REM and HPD; and the adapter manager.

Packet Buffer Memory and Packet Memory Interface

The packet buffer memory stores the data received over the FDDI ring before delivering this data to the host. The PBM also stores data from the host before transmitting over the FDDI ring.

PBM consists of two memories: the packet buffer data memory and the packet buffer ring memory. Virtually, the packet buffer data memory divides into seven areas-one used by the AM and three each for data reception and data transmission to and from the three external interfaces. These three interfaces are the FCP stage, the HPD stage, and the AM. The areas are accessed and managed by the

The three pipelined stages and the memory refresh circuitry use the packet memory interface (PMI) to access PBM. The PMI arbitrates and prioritizes the requests for memory access from these four requesters. Physically, the PMI has three interfaces: the FCP stage, the REM stage, and the HPD stage. Virtually, the PMI has four interfaces; the HPD interface multiplexes traffic from both the host and the adapter manager. The PMI also has the functionality to refresh the dynamic memory and to implement a synchronizer between the 80-nanosecond FDDI clock and the 64-nanosecond XMI clock.

All interfaces request access to the memory by invoking a request/grant protocol. Some accesses are longword (4-byte) transactions that require one to two memory cycles; others are hexaword (32-byte) transactions and require a burst of memory cycles.

The interfaces have the following priorities: (1) refresh memory circuitry, (2) the REM stage, (3) the FCP stage, and (4) the HPD stage. The refresh memory circuitry has the highest priority because

six rings residing in the packet buffer ring memory and listed in Table 1. Note that the division is considered virtual because the physical memory locations of the areas change over time.

data loss in the dynamic memory is disastrous. Also the refresh circuitry makes a request once every 5 to 10 microseconds, thus ensuring that the lower priority requesters always have access to the memory. The REM has the second

highest priority because it always requests one longword, which requires one memory cycle. Once the REM receives data, by design it waits at least two cycles before making the next request. Thus, the REM does not monopolize the memory, and the FCP can always get its requests serviced. The FCP stage requires guaranteed memory bandwidth with small latency to avoid an overflow or underflow condition in its FIFOs. Finally, the HPD interface has the lowest priority because no data loss occurs if memory access is denied for a theoretically infinite amount of time. Our adapter design has mechanisms that guarantee memory access to the HPD.

FDDI Corner and Parser Stage

The FCP stage, illustrated in Figure 5, provides the interface between the FDDI ring and the packet buffer memory. This stage can receive or transmit the smallest packet in 2.24 microseconds, as required by our performance constraints.

The receive stream in this stage converts the incoming stream of photons from the FDDI ring into a serial bit stream using the fiber-optic transceiver (FOX) chip. The clock and data conversion chip then recovers the clock and converts the incoming code from 5 to 4 bits. The MAC chip converts this electronic serial bit stream to a byte stream. The MAC chip implements a superset of the ANSI MAC standard.[9] Digital has a specific implementation of the MAC chip.[3] The ring memory controller (RMC) interfaces with the byte-wide stream from the MAC, converts the bytes into 32-as a stream of photons. This stage can generate and append 16 bytes of cyclic redundancy code (CRC) to every packet before transmitting.

The parser component of this stage interfaces with the RMC bus to generate a forwarding vector that has a variety of information including the data link user identity and the destination of the packet, i.e., the host or the AM. The parser extracts packet headers from the RMC bus and operates on the FDDI and the LLC parts of the packet headers. The parser then processes this information in real time, using a content-

bit words, and writes these words to the PBM, using the RMC receive ring and the ring protocol.

The transmit stream accesses a packet from the PBM, waits for the token on the FDDI ring, and transmits the packet

addressable memory (CAM) that stores the profiles of data link and other users. As a result, the parser generates a forwarding vector that contains the destination address of either the host user or the AM user. The forwarding

vector destination field is given a "discard" value, if the packet header does not match any user profile. Note that the forwarding vector is a part of the buffer descriptor field in the RMC receive ring.

Ring Entry Mover Stage

The REM moves filled packets from receive rings to transmit rings by copying pointers rather than copying data. (Copying pointers is a much faster operation than data copy.) Note in Figure 6 that for a given interface, no filled packet moves from its receive ring to its transmit ring. For example, no filled packet moves from the RMC receive ring to the RMC transmit ring. Also, in this design there is no need for a path from the HPD receive ring to the AM transmit ring.

A second function performed by the REM stage is to return free packets from the transmit rings to the proper receive rings. Transmit rings point to free packets after the receiver interface has consumed the information in the packet. The REM, which

The ring entry mover stage performs four major functions: (1) moving filled packets from receive rings to transmit rings, (2) returning free packets from transmit rings to receive rings, (3) managing buffers, and (4) collecting statistics. Figure 6 shows the various rings, the ring entry mover, and the movement of filled and free packets.

the color field, a subset of the buffer descriptor field. The color field contains color information that designates the receive ring to which the buffers belong. This color information is written into the buffer descriptors of the free buffers during initialization. Note that during initialization, the adapter free buffers in the PBM are allocated to the three receive rings with which the REM interfaces.

The REM also performs buffer resource management. Note that a reserved

pool of buffers exists for traffic arriving over the FDDI ring. This FDDI traffic has two destinations, namely the host CPU and the adapter manager. To ensure that one destination does not monopolize the pool of buffers, the pool is

is a transmitter interface on all transmit rings in the PBM, owns these buffers after the appropriate receiver interface toggles the ownership bit. The REM returns the buffers to the original receive ring by using information in

divided into two parts: host allocation and AM allocation. The REM delivers no more than the allocated number of buffers to one destination.

The fourth major function that the REM performs is to collect statistics. The REM collects statistics in discard counters for packets that cannot be delivered due to lack of resources. The REM interrupts the AM when these counters are half full. The AM reads, processes, and stores these counters for statistical purposes. The AM read operation resets these counters. There are a number of other counters in REM.

Host Protocol Decoder Stage

HPD Receive Pipeline. The receive pipeline has three stages: (1) the fetch and decode host receive entry stage, (2) the data mover stage, and (3) the receive buffer descriptor write stage. Most pipelines work in a lockstep fashion; that is, each stage takes the same amount of time to process input. In our design, the processing time varies for each stage in the pipeline. For example the data mover stage will

The host protocol decoder interfaces with the XMI bus, fetches and interprets entries from the host receive and transmit rings, and moves data between the host and the PBM. This stage also acts as a gateway for the AM to get to the host memory or to the PBM.

Figure 7 is a block diagram of the HPD stage. The receive and transmit pipelines store and retrieve receive and transmit data from the host memory. The two pipelines work in parallel. We now explain the operation of the receive pipeline in detail. The transmit pipeline operates in a similar manner; thus, we highlight only the differences.

interlocks to signal the completion of work.

The fetch and decode host receive entry stage has knowledge of the format and size of the ring and sequentially fetches host receive ring entries. If the adapter does not own an entry, this stage waits for a signal from the host before fetching the entry again. If the adapter does own the entry, this stage decodes the entry to determine the address

take a much longer time to transfer 4500-byte packets than to transfer 100-byte packets. The fetch and decode host receive entry stage, on the other hand, may take the same amount of time to decode entries for packets of either size. Consequently, stages use

of the free buffer in the host memory and the number of bytes in the buffer. The stage then passes this buffer information to the data mover stage and the address of the host entry to the receive buffer descriptor write stage. In addition, this stage

prefetches the next entry to keep the pipeline full, in case data is actively received over the FDDI ring.

In parallel, the PMI interface stage part of the HPD chip fetches the next entry from the HPD transmit ring. Decoding this entry determines the address of the buffer in the PBM and the amount of data in the buffer. The packet buffer bus interface passes the buffer address and length information to the data mover stage and the address of the HPD transmit ring entry to the receive buffer descriptor write stage.

Now, the data mover stage has pointers to the host free buffer and its extent and to the PBM filled buffer and its extent. The stage proceeds to move the data from the PBM to the host memory over the XMI bus. Depending on the XMI memory design, this transfer involves octaword or hexaword bursts. The process of moving data continues until the depletion of packet data in the PBM.

The data mover stage signals the receive buffer descriptor stage when the packet moving is complete. The receive buffer descriptor stage return the free buffer to the ring of origin.

HPD Transmit Pipeline. The HPD transmit and receive pipelines are symmetrical.

The HPD receive pipeline delivers data from the HPD transmit ring to the host receive ring. The HPD transmit pipeline delivers data from the host transmit ring to the HPD receive ring.

There is one exception to the symmetry. The transmit pipeline does not fetch an entry from the HPD receive ring in PBM to determine if there are enough free buffers available. A hardware interface between the PMI and the HPD, i.e., a Boolean signal, indicates whether there are enough buffers to accommodate the largest possible size transmit packet. This exception is an artifact of our implementation; we wanted to reduce the accesses to the PBM, since its bandwidth is a scarce resource.

Adapter Manager

The local intelligence, also known as the adapter manager, implements various necessary adapter functions including self-test and the initialization. The AM also implements part of the CMT code that manages

writes in the status fields of the host receive ring entry and the HPD transmit ring entry. This stage also gives ownership of the filled buffer to the host

and of the free buffer to the REM. The REM can then

the FDDI connection.[10] In addition, the AM interfaces with the host to start and stop data link users by dynamically manipulating the parser data base.

Tracing a Packet Through the Adapter

The major steps for data transfer incorporate the subfunctions previously discussed. This section traces the path of a packet P through the adapter, first on the receive stream and then on the transmit stream. We assume that adapter initialization is complete and that all data structures in the packet memory and parser data base are properly set. In this example, we further assume that packet P is small enough to fit into a single buffer. Large packets require multiple buffers.

Receive Stream

A packet destined for the host passes through the three major pipelined stages in the adapter. A brief description of the intrastage operation and details of the interstage functioning follow. The four parts of Figure 8 illustrate the receive process.

FDDI Corner and Parser Stage. Figure 8(a) shows packet P on the FDDI ring; the packet is actually a stream of photons. This stage converts the stream of photons into a packet. At this point, a free

The stage determines if packet P is addressed to this node, forwards the packet on the FDDI ring, and copies the packet for this adapter if it is addressed to this node. This stage also generates a CRC for the packet. The FCP stage then deposits the copied packet into the free buffer in the RMC receive ring entry shown in Figure 8(b).

After depositing the complete packet, this stage writes the buffer descriptor and toggles the ownership bit. The ring entry mover now owns packet P. The FCP stage is free to receive the next packet, which is stored in the next buffer in the RMC receive ring.

Ring Entry Mover Stage. The REM extracts the packet buffer descriptor and determines the number of pages in packet P. This stage also has an account of the number of pages outstanding on the HPD transmit ring. The REM delivers packet P to the HPD transmit ring provided the host resource allocation is not exceeded.

The REM delivers the packet by copying page pointers from the RMC receive ring to the HPD transmit ring, as shown in Figure 8(c).

buffer is available for packet P in both the RMC receive ring and the host receive ring. The FCP stage owns the free buffer in the RMC receive ring.

Note that the HPD transmit ring is large enough to write all pointers from the RMC receive ring and the AM receive ring. The REM then transfers ownership of the

HPD transmit ring entry to the HPD stage and the RMC

receive ring entries to the FCP stage.

HPD Stage. The HPD receive pipeline operates on a packet it owns in the HPD transmit ring. As shown in Figure 8(d), after fetching the address of the free host buffer, this pipeline moves packet P from the PBM to the host memory and toggles the ownership bit of the host entry. Simultaneously, the HPD returns ownership of the free buffers in the HPD transmit ring to the ring entry mover stage. The REM returns these buffers to the RMC receive ring as free buffers.

Transmit Stream

To transmit data from the host transmit ring to the FDDI ring, the packet must pass through the same three stages as for the receive stream, but in the reverse direction.

HPD Stage. For the receive stream, the HPD receive pipeline prefetches the free buffer from the host receive ring. In contrast, the HPD transmit pipeline must wait for the host to fill the transmit buffer and transfer ownership to the host transmit ring. The HPD stage then moves the data from the host memory

Ring Entry Mover Stage. The REM moves the packet from the HPD receive ring to the RMC transmit ring. Again, the REM copies pointers from ring to ring and toggles the ownership bit on the RMC transmit ring.

FDDI Corner and Parser Stage. Although the packet is available in PBM for transmission, the FCP stage must receive a token before transmitting over the FDDI ring. Once the transmission is complete, the buffer on the RMC transmit ring is now free. The FCP stage returns ownership of the buffer to the REM, which then returns the free buffer back to the HPD receive ring or the AM receive ring, depending upon the origin. Again, the free buffers are returned by copying buffer pointers.

The receive and transmit streams for the adapter manager are similar to those for the host; therefore, we do not describe these processes.

Hardware and Firmware Implementation

to the PBM if the hardwired signal between the REM and the HPD indicates that a sufficient number of pages is available. Finally, the HPD transfers ownership of the host transmit ring entry to the host and the HPD receive ring entry to the REM.

The hardware implementation of DEMFA consisted of four large gate arrays, custom very large-scale integration (VLSI) chips, dynamic and static random access memories (RAMs), and jelly bean logic. Figure 9 is a photograph of the DEMFA board.

Photo of DEMFA Board

The four gate arrays specified and designed by the group are the parser, the adapter manager interface (AMI), the host protocol decoder, and the packet memory controller (PMC), which incorporated the function of the packet memory interface and the ring memory controller. We now describe aspects of the gate array development. Note that we used the Compacted Array technology developed using LSI logic for our implementation. The gate arrays have 224-pin surface mount packaging.

Table 2 shows various gate arrays, the total gate count for each gate array, and the percentage of control gates and data path gates. Control gates are defined as gates required for implementing state machines used for control. Data path gates are gates required for registers and multiplexors, for example. Note that the complexity of gate arrays is proportional to the percentage of control gates. The gate arrays in Table 2 were fairly complex because they consisted of approximately 50 percent control gates.

Table 2

Gate Counts for DEMFA Gate Arrays

Gate_Array	Total_Gates	Data Gates (Percent of total)	Control Gates (Percent of total)
Parser	20296	39	61
PMC	61537	40	60
HPD	81265	34	66
AMI	15002	49	51

Module Implementation

We used the 11-by-9-inch XMI module for implementing the adapter. Early in the project we defined the pin functions for various gate arrays. Once these were defined we could design our module. SPICE modeling helped in arriving at a correct module design with the first fabrication. The design was thorough and completed early in the project.

Firmware Implementation

The DEMFA firmware has three major functions: self-test, FDDI management (using Common Node Software), and adapter functional firmware. The DEMFA team implemented the adapter functional firmware while other groups designed the two remaining components. The DEMFA functional firmware can initialize the adapter and then interact with the host to start and stop data link layer users, as well as perform other functions. The firmware is implemented in the C language for the Motorola 68020 system. The total image size is approximately 160 kilobytes.

Performance

The graph presented in Figure 10 shows the adapter performance for the receive and transmit streams at the adapter hardware level for this implementation. The data represents throughput measured in megabits per second as a function of packet size measured in bytes. Figure 10 illustrates that the receive and transmit streams meet the 100-Mb/s throughput when the packet size is approximately 69 bytes. The bottlenecks in this implementation of the DEMFA architecture timely fashion. For more detailed performance data, see the paper entitled "Performance Analysis of a High-speed FDDI Adapter" in this issue of the Digital Technical Journal.[11]

Conclusion

The goal of the DEMFA project was to specify an architecture for an adapter that would be at least 30 times faster than any previously built adapter. The architecture also had to be easy to implement. This paper describes the architecture and an implementation of DEMFA. Performance measurements of the adapter show that this first implementation successfully meets close to the maximum FDDI throughput capacity; thus, the DEMFA performance can be considered ultimate. Already, a number of adapters have been designed based on ideas borrowed from the DEMFA architecture and implementation. In a few years, architectures similar to this one may become the norm for data link and even transport layer adapters, rather than the exception.

Acknowledgements

I wish to acknowledge and thank my manager,

are (1) the PMI and (2) the combination of the XMI interface, bus, and memory. We implemented these interfaces in a conservative manner to reduce our risks and to produce the product in a

Howard Hayakawa, who out of nowhere presented me with the challenge of defining an architecture and an implementation for an FDDI adapter that would have a performance 30 times that of any existing adapter.

I must have taken leave of my senses to take on such a challenge. But the end results were worth the effort.

I would also like to thank Gerard Koeckhoven, who agreed to be the engineering manager for this adapter project. He took on the consequent challenge and the risk and supported me all along the way. In addition, I want to recognize Mark Kempf for the many hours he spent helping us during the conceptualization period and for chiseling our design. The TAN architects were of great assistance in making sure that our adapter met the FDDI standard.

I also wish to acknowledge the following members of the DEMFA project team for their contributions:

Santosh Hasani and Ken Wong, who designed the parser gate array; Dave Valley and Dominic Gasbarro, who designed the AMI gate array; Andy Russo and John Bridge, who designed the HPD gate array; Ron Edgar, along with the other PMC designers, Walter Kelt, Joan Klebes, Lea Walton, and Ken Wong; Ed Wu and Bob Hommel, who designed and implemented the module; adapter was ready. Also, I would like to thank VMS and ULTRIX group members Dave Gagne, Bill Salkewicz, Dick Stockdale, and Fred Templin.

References

1. B. Allison, "An Overview of the VAX 6200 Family of Systems," Digital Technical Journal, no. 7 (August 1988): 10-18.
2. D. Fite, Jr., T. Fossum, and D. Manley, "Design Strategy for the VAX 9000 System," Digital Technical Journal, vol. 2, no. 4 (Fall 1990): 13-24.
3. H. Yang, B. Spinney, and S. Towning, "FDDI Data Link Development," Digital Technical Journal, vol. 3, no. 2 (Spring 1991): 31-41.
4. A. Tanenbaum, Computer Networks (Englewood Cliffs, NJ: Prentice Hall, Inc., 1981).
5. B. Allison, "The Architectural Definition Process of the VAX 6200 Family," Digital Technical Journal, no. 7 (August 1988): 19-27.
6. Token Ring Access Method and Physical Layer Specifications, ANSI /IEEE Standard 802.5-

the team that implemented the functional firmware, Ed Sullivan, David Dagg, Da-Hai Ding, and Martin Griesmer; and Ram Kalkunte, for designing and building a simulation model to accurately predict the performance well before the

1989 (New York: The Institute of Electrical and Electronics Engineers, Inc.; 1989).

7. R. Stockdale and J. Weiss, "Design of the DEC LANcontroller 400 Adapter," Digital Technical Journal, vol. 3, no. 3 (Summer 1991, this issue): 36-47.
8. FDDI Station Management (SMT), Preliminary Draft, Proposed American National Standard, ANSI X3T9/90-X3T9.5/84-49, REV. 6.2 (May 1990).
9. Token Ring Media Access Control (MAC), (International Standards Organization, reference no. ISO 9314-2, 1989).
10. P. Ciarfella, D. Benson, and D. Sawyer, "An Overview of the Common Node Software," Digital Technical Journal, vol. 3, no. 2 (Spring 1991): 42-52.
11. R. Kalkunte, "Performance Analysis of a High-speed FDDI Adapter," Digital Technical Journal, vol.3, no.3 (Summer 1991, this issue): 64-77.