

FDDI Data Link Development

By Henry S. Yang, Barry A. Spinney, and Stephen Towning

Abstract

The fiber distributed data interface (FDDI) data link is based on the ANSI X3T9.5 FDDI standards with Digital's enhancements to provide greater performance, reliability, and robustness. The FDDI project team encountered significant challenges, including the evolving ANSI X3T9.5 FDDI standards and the development of the technology to implement

the data link, coupled with time-to-market pressure. Appropriate considerations and design trade-offs were made to design complexity, performance, risk, cost, and schedule, when deciding functional partitioning and semiconductor technology. Extensive simulations and a novel test approach were used to verify the algorithms, the functional models comprising the chips, and the physical chips themselves.

Introduction

The proliferation and importance of distributed

and efficiency. The advent of fiber optics, large-scale integrated circuits, and related technologies makes it possible to provide a relatively low-cost, high-speed local area network (LAN) with large physical extent and connectivity. One LAN standard is the fiber distributed data interface (FDDI), a 100-megabit-per-second token ring that uses an optical fiber medium.

The scope of FDDI spans the data link layer and the physical layer. The FDDI data link provides its users with communication services on a multiaccess LAN for transmitting and receiving frames with best-effort delivery service (also called the datagram service). The development of the FDDI data link encountered several significant challenges, including the instability of the standard, unproven technology and protocols, and an order of magnitude increase in speed from the International Standards

system applications place
special requirements on
networks in terms of
topological flexibility,
performance, reliability,
scalability, robustness,

Organization (ISO) 8802-
3 carrier sense multiple
access with collision
detection (CSMA/CD) LAN.
The design of the FDDI data
link involved performance

Digital Technical Journal Vol. 3 No. 2 Spring 1991

FDDI Data Link Development

considerations such as throughput, latency, data integrity, and reliability.[1,2]

In this paper we discuss the development of Digital's FDDI data link and present some of the key algorithms developed by Digital. We then describe the design and

development of the FDDI data link technology and its implementation in the FDDI, focusing on the FDDI data link's very large-scale integration (VLSI) chip set. Finally, we describe the development methodology used in simulation, verification, and testing.

FDDI Data Link Overview

The FDDI data link provides an upward multiplexing, datagram service to support multiple data link users concurrently within the same computer system. The FDDI data link incorporates the ISO 8802-2 logical link control (LLC) and FDDI standards. Also, the FDDI data link provides a mapped Ethernet service, defined by the Internet RFC 1103 standard, to map an Ethernet frame onto an ISO 8802-2 LLC frame for transport over the FDDI LAN.[3]

The FDDI data link consists of an LLC sublayer, a

media access control (MAC) sublayer, and their management. The LLC sublayer provides LLC services, the mapped Ethernet service, and multiplexing/demultiplexing

services for multiple users. The key functions provided by the MAC sublayer include the FDDI token ring protocol, frame transmission and reception, initialization and error recovery for the token ring, address recognition and filtering on receive, and frame error detection. Figure 1 is an example of the FDDI architecture model, showing a dual attachment station (DAS) or dual attachment concentrator (DAC) with a single data link entity. A DAS or DAC may have zero, one, or two link entities; two or more physical (PHY) port entities; and control of their management. A link entity is an instance of data link that contains an LLC and a MAC entity. A data link user accesses the data link services through the port entity. As shown in Figure 1, multiple data link users may use the same link entity.

The FDDI data link uses the FDDI MAC protocols to provide fair access to a multiaccess channel, which is built of individual point-to-point physical links.[4] A token ring consists of an ordered, cyclic set of MAC protocol

entities called MACs. It operates by passing a token sequentially from MAC to MAC around the ring. Only the MAC with the token may transmit frames onto the ring, and only one token can be present on the ring at any instant. At the end

of transmitting its frames, the MAC transmits the token onto the ring. Frames circumnavigate the entire ring and are subsequently removed (stripped) by the originating MAC after one and only one rotation. The FDDI MAC protocol is different from the ISO 8802-5 token ring protocol.[5]

The FDDI MAC protocol uses a timed token protocol, whereby MACs on the token ring cooperatively attempt to maintain a specified

token rotation time by using the observed network load to regulate the amount of time a MAC may transmit.[6] The specified token rotation time is called the target token rotation time (TTRT). The TTRT is negotiated using a distributed algorithm called the claim token algorithm, which is invoked each time the ring is initialized. The FDDI MAC protocol also includes fault detection and recovery functions to aid in the restoration of ring operation in the presence of transient faults.

As shown in Figure 1, each instance of a station (e.g., DAS or DAC) contains a set of station management functions. Some of the key station management functions included are

and a set of frame-based protocols. The set of frame-based protocols for station management includes duplicate address detection, neighbor notification protocol for ring map generation, and loopback protocol. The scope of the station management frame-based protocols is limited to a single ring. These functions are implemented in the Common Node Software (CNS) and the FDDI chip set.[7]

FDDI Data Link Algorithms

Digital's realization of the FDDI data link includes major enhancements and value-added features. The development of the FDDI data link encountered several architecture and implementation issues. The key issues included design for high performance, error characteristics and data integrity, removal of frames by bridges, cleaning the ring of unwanted frames and long fragments, and stability and reliability of the ring.[2] Digital provided the impetus to resolve these issues as well as the solutions to be incorporated into the ANSI FDDI standards. Some of these solutions are

initialization, observation

described below.

and control of link
and PHY port entities,
control of the insertion
and removal of the
station from the ring,
topology control, fault
detection and recovery,

Frame Content Independent
Stripping (FCIS) Algorithm

Digital Technical Journal Vol. 3 No. 2 Spring 1991

FDDI Data Link Development

The underlying logical topology of all token rings is a closed loop structure, which inherently has the property of continuously circulating frames transmitted on the ring. Because of this property, all token rings have algorithms for removing frames transmitted by MACs on the ring. The frame removal algorithm is called a frame stripping algorithm. When stripping a frame, the fragment size (remnant of the frame) must be less than 17 bytes. Frames that are not properly stripped can remain and traverse the ring repeatedly, wasting bandwidth and resources within systems on the ring and causing severe congestion in the systems due to delivery of duplicate frames.

The FDDI MAC protocol uses a frame stripping algorithm in which all MACs on the ring continually strip received frames whose source address matches their own MAC address. Limitations of this algorithm arise when implementing bridges or systems that need to transmit frames with source addresses which are different from each MAC's own address.[8] For example, a bridge may forward frames with

of stations in the extended LAN.[9] If a bridge were to use the source address match algorithm, the bridge would have to complete the address match operation within one microsecond from the beginning of the frame reception. Therefore, the use of the source address match algorithm by a bridge imposes significant costs and implementation complexity. The design of the frame stripping algorithm is made more difficult by the fact that there can be up to 560 frames outstanding (i.e., transmitted but yet to be stripped) and the fact that there is less than one microsecond to decide whether to strip or to repeat the frame.

Digital developed an algorithm called the frame content independent stripping (FCIS) algorithm, which is implemented in the MAC chip.[10,11] The algorithm is based on stripping the same number of frames that the MAC transmitted on the ring independent of the content of the frame. After the MAC captures the token for frame transmission, a local count is incremented each time a frame is transmitted. After transmitting all its frames, the MAC transmits a void frame,

no modifications, and, therefore, the forwarded frames contain source addresses which are different from the bridge address. Also, a bridge may support tens of thousands

which is a minimum size frame (i.e., 17 bytes), before transmitting the token.[10,12] On receive, if the count is greater than zero, the received frame is stripped; and

for each error-free frame received and stripped, the count is decremented. After transmitting the token, the MAC does not strip frames when the count is equal to zero, except for frames with a source address matching the MAC's address. When receiving an error-free void frame with the source address matching the MAC's address, the count is unconditionally reset to zero. As a result, the algorithm uses a count that is kept locally to track the number of outstanding frames for stripping, and it uses a transmitted void frame as a backup mechanism to indicate the end of stripping.

The operation of the FCIS algorithm is seen in the space-time diagram of Figure 2, which shows three stations on the ring while station B is the only station participating in the FCIS algorithm. Time increases from top to bottom, and the position of downstream stations on the ring goes from left to right in this diagram. Station A first receives the token T and transmits frames A1, A2, and the token. Station B then transmits frames X1, Y1, and Z1, which have source addresses X, Y, and Z that are not

transmitting frame Z1, station B transmits its void frame, VB, and then the token. When station B receives its void frame, the count is reset to zero, which causes station B to stop frame stripping. Subsequently, an entirely new epoch of transmission and frame stripping can begin with the next token capture.

The combination of the count and the void frame provides robust frame stripping that is independent of the content of the transmitted frame. The FCIS algorithm also has the desired property of operating transparently with other MACs which

are not implementing the algorithm. Implementing the FCIS algorithm in the MAC chip greatly reduces the cost and complexity of products, which otherwise may need additional hardware components to perform similar functions. Ring Purging Algorithm

One of the well-known problems on a token ring is the circulation of frames or long fragments that are not stripped by the transmitter. The frame or long fragment not stripped by the transmitter is called a no-owner frame (NOF). On an idle ring,

the same as station B's MAC address. As shown in Figure 2, each time station B transmits a frame, it increments its count; and each time station B strips an error-free frame, it decrements the count. After

NOFs can circulate around the ring, along with the token, continuously at the speed of the ring. NOFs may be received by one or more systems on the ring repeatedly at an extremely high rate, which can lead

FDDI Data Link Development

to severe congestion and waste of system resources. For example, a single NOF on an idle ring can create a frame arrival rate of 2,700 frames per second (for maximum size frame) to about 290,000 frames per second (for minimum size frame). In the best case, the NOFs are removed when they arrive at a MAC that is transmitting (i.e., holding the token).

Digital developed an algorithm called the ring purging algorithm to remove NOFs. The ring purging

algorithm ensures that NOFs do not traverse the ring more than twice. The implementation of ring purging consists of two related, but different, algorithms. The first one is the purger election algorithm, which is a distributed election algorithm to select a designated MAC to be the purger for the ring. The second algorithm is the purging algorithm, which is executed by the designated MAC to clean the ring of NOFs. We describe the purging algorithm in this section.

The purging algorithm

adopted for Digital's FDDI data link purges the ring transparently each time a token is received by the purger. When the

purge cycle. Once the purge cycle has started, the purger unconditionally removes all frames or fragments received. The purge cycle is terminated when the purger receives one of its error-free void frames, a token, or a ring initialization frame. In order to increase the probability of correctly terminating the purge cycle, the purger transmits two void frames; but it terminates its purge cycle based on receiving only one error-free void.

Figure 3 shows the operations of the ring purger in removing an NOF, during an idle ring and during a busy ring. Each time station S3 (the purger) receives the token, it may transmit its frames, followed by two void frames and then the token. It purges the ring until it receives one of its error-free void frames. As shown in the example, the NOF was purged by station S3 on its second traversal around the ring. Also, the example shows that the purging of the ring is transparent (i.e., there is no disruption to the ring).

purger receives a token,
it begins a purge cycle by
transmitting two special
frames, called void frames.
If the purger has frames
to transmit, it completes
the transmission of its
frames before starting the

The impact of ring purging

on ring performance is negligible, because the ring purger only initiates the purge cycle when it has the right to use a token. In the worst case, the ring purger's effect on usable bandwidth is less than 0.22 percent. For implementations compliant to the ANSI FDDI MAC standard, these void frames have no effect since the standard prohibits copying void frames.

The ring purging algorithm removes NOFs, including long fragments, without disrupting the operation of the ring, and it removes NOFs within two traversals of the frame. In addition, it has an important property that permits more than one purger to operate in the same ring at any time. This property allows the purger election algorithm to be more optimistic (i.e., when in doubt during election, one can start purging) during the transition period when the distributed election algorithm is stabilizing. The purger election algorithm is implemented in the Common Node Software (CNS), and the purging algorithm is implemented in the MAC chip.

FDDI Data Link Chips

The FDDI MAC sublayer functions are implemented by the three FDDI data link chips: the ring memory controller (RMC), media access control (MAC), and content addressable memory (CAM). The RMC interfaces between the frame buffer memory on the system side and the MAC chip on the network side. It consists of a direct memory access (DMA) engine designed to supply the MAC chip with frames to send and to store frames received from the MAC chip. The interface on the system side provides gathered reads on transmit and scattered writes on receive. Although the RMC's interface to the MAC chip was custom designed for FDDI operation, the RMC can, in principle, be used for other data links that run at 100 megabits per second or less. The MAC and CAM chips implement the FDDI MAC protocol functions. The functions implemented by the MAC chip include the token access protocols, frame delineation, frame parsing, address recognition, frame check sequence generation and verification, frame insertion, frame repetition, frame removal, token generation, and error detection and

recovery algorithms (e.g.,
the beacon and claim
algorithms). The CAM chip
provides the destination
address filtering, which
determines if a received
frame is to be received or
discarded, and the setting

Digital Technical Journal Vol. 3 No. 2 Spring 1991

FDDI Data Link Development

of the A-indicator, which is part of the frame status field.

RMC Chip

The RMC chip is a high-performance coprocessor intended for full-duplex data transfer between the buffer memory and the MAC chip. It uses a pair of circular buffer queues for transmit and receive to manage DMA data transfers to and from the buffer memory. Two independent on-chip, first in, first out (FIFO) buffers, for receive and transmit, are provided to decouple the buffer memory from the real-time nature of the MAC interface. A fragment and frame filter is provided to reduce unnecessary memory accesses caused by the reception of fragments or frames not addressed to this station. As shown in Figure 4, the RMC chip has three interfaces: the processor interface, the MAC chip interface, and the buffer memory interface. The processor interface allows the initialization, control, and observation of the RMC. The MAC chip interface consists of high-speed transmit and receive data paths for transfer of data and control information between the MAC chip and the RMC's FIFO

buffers. The buffer memory

interface provides a burst mode DMA for read/write from/to the buffer memory on a single bus arbitration cycle, where the burst size can be up to four or eight longwords. The RMC can provide a maximum data transfer rate of about 44 megabytes per second.

The RMC is implemented using a 1.5-micron-drawn, two-metal-layer custom complementary metal oxide semiconductor (CMOS) technology. It uses roughly 87,000 transistors, a large number of which are used for the two FIFO buffers, where the receive FIFO buffer is 256 bytes and the transmit FIFO buffer is 128 bytes. The RMC uses 102 signal pins and is available in a 132-pin cerquad package. It is also a fully synchronous design [some self-timed logic is used in the FIFO random access memory (RAM) devices] using a 12.5-megahertz (MHz) primary clock and 25-MHz clock for sampling incoming signals. The FIFO RAM was implemented using a full custom methodology, and the remainder was implemented using an automated standard cell methodology.

The RMC interfaces to buffer memory using a data and address multiplexed bus, which is a 32-bit-wide bus plus the four parity signals and additional control signals. A bus transaction consists of an address cycle driven

by the RMC followed by a burst of data cycles, either driven by the RMC on receive or driven by the buffer memory on transmit. One of the unique features of this chip is that it is able to use a 32-bit-wide buffer memory composed of

low-cost [100-nanosecond (ns) access time] dynamic random-access memory (DRAM) chips, whereas many of the other FDDI memory controllers available on the market require a 64-bit buffer memory or require very fast DRAM or static random access memory (SRAM) chips. To achieve these reduced memory and chip requirements, the RMC's buffer memory accesses were done in bursts of between one and eight longwords. Then, by making use of the DRAM's fast page mode, in which subsequent, sequential reads/writes are faster than the first one, the needed buffer memory bandwidth is attainable.

The RMC directly accesses a transmit ring and a receive ring, each consisting of a circular queue of descriptors. Each descriptor supplies the buffer memory address of a transmit buffer or a receive buffer. An OWN bit mechanism is used in each descriptor to determine if the descriptor and its buffer is owned by the

RMC or not. It supports gathered read and scattered write in which frames to be received or transmitted can use one or multiple buffers (and hence multiple descriptors), but a specific buffer/descriptor

one for the frame, then the RMC writes the receive status and the frame byte count into the descriptor. Transmit buffers are also 512 bytes long and the RMC reads the size (in bytes) from the first descriptor for the frame.

MAC Chip

The MAC chip implements the FDDI MAC protocols, and it interfaces between the RMC or equivalent chip and the FDDI physical layer chip.[13] As shown in Figure 5, the MAC has four interfaces: the processor interface, the RMC interface, the physical layer chip interface, and the CAM interface. The processor interface allows the initialization, control, and observation of the MAC. The RMC interface is custom designed for FDDI operation and allows the MAC to interface to either the RMC or to equivalent chips implementing the RMC interface. The physical layer chip interface allows the MAC to receive and transmit on the FDDI ring.

The MAC chip is implemented using a 1.5-micron-drawn, channelless, two-metal-layer CMOS gate-array technology and uses roughly 49,000 transistors (12,000 used gates). The MAC chip uses 86 signal pins and

can only be used by one frame. Each receive buffer is required to be 512 bytes long. The RMC rewrites each descriptor in the receive ring to indicate the number of bytes actually used; and if the buffer is the last

is available in either a 120-pin pin grid array (PGA) package or a 120-pin plastic quad flat pack (PQFP) package. This fully synchronous design uses primarily a single 12.5-MHz clock (80-ns cycle time);

FDDI Data Link Development

this operation is applied to the microprocessor bus as well. An additional double-speed clock is used in some of the peripheral interface logic to sample incoming signals and prevent hold time problems associated with clock skew between different chips.

The internal structure of the MAC chip has a full-duplex architecture. No logic is shared between the receive and transmit portions of the chip. Hence this chip can receive and transmit simultaneously for an indefinite period. This capability complies with the ANSI FDDI MAC standard that implementations be able to receive, parse, and validate certain frames (e.g., claim frames and beacon frames) even while transmitting. Two separate frame check sequence (FCS) checker/generators are required for transmit and receive functions. The MAC chip calculates the FCS, which is specified as a specific 32-bit cyclic redundancy check, eight bits at a time. A one-bit implementation is much smaller, but requires a 100-MHz clock. Even with such a clock, it is not easy to implement one bit at that speed. A byte-wide implementation requires considerably more exclusive OR (XOR) gates,

but exploits more of the inherent parallelism of the algorithm and hence can be implemented using slower clock speed.

One of the important features provided by the

MAC chip is the support of a 3-byte packet request header for transmission. The use of the packet request header construct allows simple, pipelined processing of the transmit descriptor along with the transmit data at high speed. This construct allows a software device driver to build a transmit descriptor to precede and identify the frame, which is then passed through the system bus and DMA data movers for delivery to the MAC chip. Every frame transmitted by the MAC chip must first contain the packet request header, which is used as the transmit descriptor and is not transmitted as part of the frame. The packet request header is used to instruct the MAC chip on how and when to transmit the frame. For example, it instructs the MAC chip on whether to append the FCS to the frame or not, or on the type of token to use when transmitting the frame.

CAM Chip

The CAM chip provides a 64-entry content addressable memory where each entry is a 48-bit address. Typically, the entries in the CAM consist of

multicast addresses used by upper layer protocols and the data link management protocols. The CAM is used to parse the destination address field of each frame received to decide whether

Development

the destination address received matches one of the entries in the CAM. The result of the address match is then provided in real time as input to the MAC chip, which decides whether to receive or to discard the frame. In the worst case, the MAC protocol requires that the destination-address-match decision be completed in less than 10 bytes of time (i.e., 800 ns), starting from the end of the destination address field.

The CAM chip is implemented using a 1.5-micron-drawn, two-metal-layer custom CMOS technology and uses roughly 44,000 transistors-34,000 of which are used for the core array of 64 words of 48 bits (plus a valid bit). The CAM chip uses 37 signal pins and is available in a 44-pin cerquad package. It also is a fully synchronous design using the same 12.5-MHz primary clock (plus the 25-MHz clock for sampling incoming signals).

FDDI Data Link Development

As shown in Figure 6, the CAM consists of three interfaces: the processor interface, the MAC chip interface, and the physical layer chip interface. The physical layer chip interface consists of an 8-bit bus which transfers data bytes from the physical layer chip to the MAC chip. Every 80 ns, one byte is loaded into

the appropriate position in the 6-byte internal compare register. Once the last byte of a 48-bit address has been clocked into the CAM chip's compare register, a match/no match indication is given to the MAC chip within 120 ns by the MAC chip interface.

The other interface to the CAM is the processor interface, which is used by the processor to load and change the CAM entries. The processor does not directly read or write the CAM array, but instead reads and writes (16 bits at a time) to three 16-bit data registers and one 16-bit command register. By clearing the appropriate bit in the command register, the processor requests a read, write, or compare from the CAM array. The arbiter ensures that the CAM array is idle (i.e., not used by the compare

one transfer or compare at a time. The arbitration mechanism guarantees that the compare register access is never delayed, since the MAC chip requires the match indication to be valid during a specific clock cycle. The CAM is also designed to allow entries to be added and removed at any time (even while the FDDI ring is running).

FDDI Simulation and Verification

Digital's FDDI technology development method was a top-down approach, starting with high-level system models of FDDI behavior and progressing to more detailed behavioral and

structural models as confidence in functionality increased. Several simulation models and analytical models were developed to study and model the FDDI at the architecture and system levels. Using these models, studies were done on the error characteristics and robustness of FDDI, stability of the ring topology, performance and operational behaviors, and correctness of the protocols.[1]

Every chip was partitioned into logical subblocks and a DECSIM (Digital's simulation tools and

operation) before allowing the processor's request to complete. In order to ensure atomicity for read, write, and compare, this arbitration is required because the actual CAM array can only perform

language) block-behavioral model was developed for each subblock. The external interfaces and internal structure of the model accurately represented some unit or subunit of a chip.

Development

When the model interfaced to some nonexistent model (e.g., unwritten behavioral model or external interface to another chip), a transactor was provided. The transactors provided accurate models of interfaces (timing, control, and data signals), but lacked the internal detail of a behavioral model. The use of the transactors was particularly important when modeling the buffer memory interface, since the design of the interface is implementation dependent (designed according to product needs by the development group using the FDDI chip set).

Each block-behavioral model was tested in isolation; then all were combined to produce a behavioral model of the target chip. When the chip model was successfully tested, each behavioral subblock was replaced by a corresponding structural model representing gate /transistor logic. The new chip model was then retested until the structural model behaved identically to its behavioral counterpart. As the model of each chip was completed, the transactors driving its external interfaces were

between chips and was repeated until all chips in the chip set had been tested together as a system.

After chip layout was completed, the structural models were enhanced to reflect the more accurate timing data. The test vectors were again applied using a checker model, which consisted of one block-behavioral model and one structural model of the same chip. The test vectors were applied to each model simultaneously, and external and internal signals of both models were compared for consistency. Any discrepancy between the monitored signals was

thoroughly investigated and corrected as necessary.

For test and debug, we planned to develop a dedicated hardware tester to test the physical FDDI chips. Unfortunately bugs found by using such a tester occur too late in the process-the chips are already built. In order to meet our time-to-market goal, we needed to maximize activity in the simulation environment. Rather than waiting for the hardware, we decided to develop and apply as many of the tester-based tests as possible in the simulation environment.

replaced with the model
for the next adjacent chip.
The resulting combination
was then tested together
using the remaining
transactors and test
vectors. This process
tested interoperability

Simulation Test Bed
The version of DECSIM
we were using provided
a C language interface
capability. By writing the
tests in C language and
making use of a common but

FDDI Data Link Development

small environment-specific interface library, it was possible to simulate the system behavior using the chip models. By rewriting the interface library, the same tests were run unchanged on the tester and were used in regression testing of the chips during the fabrication process. The tests, interface library, and chip models became known as the simulation test bed. This was the first attempt to use this DECSIM capability as a cornerstone of a development strategy.

All the chip models were combined to construct an entity resembling a single attachment station (SAS)

which was then tested as a full system, that is, a single operational FDDI node (in loopback). Extensive use of mixed mode simulation (mixing transactors and behavioral and structural models) aided test bed performance

because the level of model detail could be varied, depending upon the area being tested. Time was saved by substituting higher level models in areas peripheral to those under test.

The simulation cluster was a cluster of four VAX 8840 systems, each system having

(196,560 VAX 11-780 CPU hours equivalent).

- o For the single-node test bed, there were 827 MAC and 384 RMC tests. Using the 8840 cluster, the test suites required 336 (2,184 VAX 11-780 CPU hours) and 192 hours (1,248 VAX 11-780 CPU hours), respectively, for completion.

The individual tests varied in complexity from those requiring a few CPU minutes to those requiring days to run. For example, one MAC test which loops back ten 512-byte packets within the single-node test bed required 36 CPU hours to complete.

The importance of the test bed cannot be overstressed, as it is the major innovation in Digital's development methodology. Some CNS firmware was also developed in this environment. The benefits provided include:

- o The tests assisted the chip designers to discover bugs in the chip designs at the correct stage of development-in design rather than after silicon.
- o It was far easier to debug the tests in the simulation environment

four processors. Some idea of the extent of the effort expended can be conveyed by the following statistics:

- o The total number of CPU hours used for the design and verification effort was 30,240

rather than on the physical hardware. During simulation we could observe and

control chip behavior. Test results were easy to determine with clear pass or fail

Development

- indications, and no human interpretation of large strings of 1's and 0's was required.
- o Since all test code was developed in C, no specialized test language was required, and standard support tools were readily available. Libraries of reusable (debugged) test-support functions were available, providing such functions as create frame, compare frame, write control/status register, and read control/status register.
- o Easy transition back and forth between the physical and simulation environments was important. Any bugs found in the physical environment had to be reproduced in the simulation environment in order to test bug fixes.

FDDI Tester

In parallel with chip design, another development group was assigned to test the chip set in a prototype FDDI system. Their approach was to design an FDDI tester that used the FDDI chip set. We also wanted to make this tester

configurable as various

The main value of the tester was its capability to perform long-term, steady-state testing, using billions of frames. It was also required to test complex topologies using multiple testers (as FDDI stations) in large rings. These activities are prohibited in the simulation environment because of the excessive amount of compute time required. The tester had to be capable of driving the data at full FDDI bandwidth, introducing controlled error conditions on the fiber, and accurately monitoring activity on the ring at full FDDI bandwidth. The tester itself was constructed so that it could be controlled via an external Ethernet link and multiple testers could be synchronized via external clock and control lines.

Testing the First Chips

The first pass of chips tested free of major defects. Most tests applied in the test bed passed the first time; the few exceptions were due to unrealistic timing expectations of the tester environment.

The next stage of testing

FDDI entities, e.g., an SAS, a wiring concentrator, or an Ethernet-to-FDDI bridge, so that we could use the FDDI tester to build an FDDI ring to investigate the behavior of the ring.

involved combining testers into multinode FDDI configurations and exchanging data at full speed over extended periods. This stage was also successful; no additional bugs were discovered, and the data

FDDI Data Link Development

link performance and ring stability exceeded the expectations. The steady-state testing was then augmented by introducing pathological conditions into the ring such as stations with duplicate addresses and noise on the fiber. The promiscuous capture modes and frame/event time stamping proved invaluable in analyzing subsequent behavior in these cases as the effects were often complex. Station management implementation was highly stressed and performed without error.

Finally, the same testers were configured as bridges, and prototype Ethernet-to-FDDI bridging firmware was introduced. At this stage several minor deficiencies with the control algorithms required for bridging were detected. These deficiencies were due mainly to insufficient analysis of FDDI bridging requirements by the test team; therefore, the test cases were correspondingly inadequate. All problems were repeatable on the simulation test bed, and bug fixes with new tests were developed. No deficiency was severe enough to prevent testing the prototype FDDI bridges and the development of more efficient algorithms for the FDDI bridge products.

Conclusion

The development of the FDDI data link and the chip set represents a major accomplishment and technical breakthrough in the high-speed LAN area. Significant contributions were made by Digital in the area of FDDI MAC algorithms and protocols to improve the performance and robustness of the FDDI LAN. The FDDI data link chips described in this paper are used in all members of Digital's FDDI product line, including bridges, wiring concentrators,

and adapters. These products have benefited tremendously from the verification and test method adopted. Digital has built on its knowledge and experience in systems, networks, computer-aided design/simulation, and semiconductors to provide FDDI design, development, and methodology. Again, Digital has shown industry leadership by producing the FDDI chip set and products.

Acknowledgments

The authors acknowledge the technical contributions of Raj Jain, Brian Myrick, Chuck Lee, and K.K. Ramakrishnan. The authors would like to thank Jerry Hutchison, Bill Cronin, and Raj Jain for their review

and several useful comments
on this paper.

References

1. R. Jain,
"Characteristics of
Fiber Distributed Data
Interface (FDDI),"
IEEE Transactions on
Communications, vol. 38,
no. 8 (August 1990).
Overview of the Common
Node Software," Digital
Technical Journal, vol.
3, no. 2 (Spring 1991,
this issue): 42-52.
2. R. Jain, "Performance
Analysis of FDDI Token
Ring Networks: Entity
Effect of Parameters and
Guidelines for Setting
TTRT," Proceedings of
the ACM SIGCOMM '90
(September 1990).
8. MAC Bridge
Specifications,
ANSI/IEEE Standard
802.1d (unapproved
draft), P802.1d
/D8 (New York: The
Institute of Electrical
and Electronic
Engineers, Inc., 1989).
3. Standard for the
Transmission of IP
Datagrams Over FDDI
Networks, Internet
Engineering Task Force
(IETF) RFC 1103 (June
1989).
9. B. Stewart, W. Hawe, and
A. Kirby, "Local Area
Network Connection,"
Telecommunications
Magazine (April 1984).
4. M. J. Johnson, "Fairness
of Channel Access
for Non-Time-Critical
Traffic Using the FDDI
Token Ring Protocol,"
NASA Ames Research
Center, Research
Institute for Advance
Computer Science, RIACS
TR 86.9 (March 1986).
10. H. Yang and K. K.
Ramakrishnan, "Frame
Content Independent
Stripping for Token
Rings," Proceedings
of the ACM SIGCOMM '90
(September 1990).
5. Token Ring Access
Method and Physical
Layer Specifications,
ANSI/IEEE Standard
802.5-1989 (New
York: The Institute
of Electrical and
the 15th Conference on
Local Computer Networks
(October 1990).
11. K. K. Ramakrishnan
and H. Yang, "Frame
Content Independent
Stripping for Token
Rings," Proceedings of
12. Fiber Distributed Data
Interface (FDDI) Media
Access Control, American
National Standard, ANSI

Electronic Engineers,
Inc., 1989).

X3.139-1987 (June 1988).

6. R. M. Grow, "A Timed
Token Protocol for
Local Area Networks,"
Electro/82, Token Access
Protocols (17/3) (May
1982).
7. P. Ciarfella, D. Benson,
and D. Sawyer, "An

13. J. Hutchison, C.
Baldwin, and B.
Thompson, "Development
of the FDDI Physical
Layer," Digital
Technical Journal, vol.
3, no. 2 (Spring 1991,
this issue): 19-30.

Digital Technical Journal Vol. 3 No. 2 Spring 1991

17

=====
Copyright 1991 Digital Equipment Corporation. Forwarding and copying of this
article is permitted for personal and educational purposes without fee
provided that Digital Equipment Corporation's copyright is retained with the
article and that the content is not modified. This article is not to be
distributed for commercial advantage. Abstracting with credit of Digital
Equipment Corporation's authorship is permitted. All rights reserved.
=====