1

MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS
STANDARD REFERENCE MATERIAL 1010a
(ANSI and ISO TEST CHART No. 2)

National Library
of Canada

Bibliothèque nationale
du Canada

Canadian Theses Service

Service des thèses canadiennes

Ottawa, Canada
K1A 0N4

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

An ATM to FDDI Bridge: Protocol Design and Implementation

by

**Sanjeev Agrawal, B. Eng.
M. N. R. Engg. College, Allahabad, India**

**A thesis submitted to the
Faculty of Graduate Studies and Research
in partial fulfillment of the requirements
for the degree of
Master of Engineering**

**Ottawa-Carleton Institute for Electrical Engineering
Faculty of Engineering
Department of Systems and Computer Engineering
Carleton University
Ottawa, Ontario, Canada
December 1990
© Sanjeev Agrawal, 1990**

National Library
of Canada

Bibliothèque nationale
du Canada

Canadian Theses Service Service des thèses canadiennes

Ottawa, Canada
K1A 0N4

The author has granted an irrevocable non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of his/her thesis by any means and in any form or format, making this thesis available to interested persons.

The author retains ownership of the copyright in his/her thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without his/her permission.

L'auteur a accordé une licence irrévocable et non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées.

L'auteur conserve la propriété du droit d'auteur qui protège sa thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

ISBN 0-315-68896-3

Canada

**The undersigned recommend to the Faculty of Graduate Studies
and Research acceptance of the thesis**

**" An ATM to FDDI Bridge: Protocol Design and Implementation"
submitted by Sanjeev Agrawal, B.Eng.
in partial fulfilment of the requirements for
the degree of Master of Engineering**

Thesis Supervisor

Thesis Supervisor

**Chairman, Department of Systems and
Computer Engineering**

**Ottawa-Carleton Institute for Electrical Engineering,
Department of Systems and Computer Engineering
Faculty of Engineering
Carleton University
15 January, 1991**

Abstract

The networking devices that interconnect high-speed LANs, are required to execute the interconnection protocol at a high-speed so as to match the performance potentials of interconnected LANs. The conventional design of network controllers, such as those used by state of the art bridges, needs improvement in order to accomplish the task. A minimal functionality bridge-interface protocol, intended to support an interconnection of multiple FDDI LANs with an ATM backbone network, is presented in the thesis. The bridge uses two approaches to overcome the data processing bottleneck: Minimizing the functionality of the protocol, and use of a special memory architecture which allows parallelism in protocol processing and data copying. An architecture for a custom VLSI implementation of the BI protocol is proposed. Throughput analysis is presented for general-purpose processor and for the custom VLSI architecture.

Acknowledgments

I would like to thank my thesis supervisors, Dr. A. R. Kaye and Dr. S. A. Mahmoud for their guidance and encouragement during the course of this research.

I gratefully acknowledge the funding support provided by Carleton University, Telecommunications Research Institute of Ontario, and National Sciences and Engineering Research Council.

Finally, I would like to thank my brother Rajiv Agrawal for his continuous encouragement.

To My Parents

Table Of Contents

Acceptance Sheet.	ii
Abstract.	.iii
Acknowledgments.	iv
Table of Contents.	vi
List of Tables.	ix
List of Figures.	x
List of Acronyms.	.xii
 1 Introduction	
1.1 Background.	1
1.2 Objective.	2
1.3 Scope.	2
1.4 Literature Survey.	4
1.5 Thesis Lay-out.	6
 2 Bridge Interface Protocol.	8
2.1 Overview	8
2.2 Architectural Issues.	8
2.2.1 Bridges	8
2.2.2 Appropriate Layer for Bridging.	10
2.2.3 Desirable Characteristics of a bridge.	11
2.2.4 Bridge Functionality	11
2.2.5 Connection oriented vs. connectionless	12
2.2.6 Address Resolution and Mapping.	13
2.3 Protocol Descriptions	13
2.3.1 The FDDI Protocol	14
2.3.2 ATM Protocol.	17

2.4 The Proposed BI Protocol.	21
2.5 BI Protocol Implementation.	26
2.5.1 State Machine Implementation	26
2.5.2 Processing Steps.	33
3 RISC Processor Implementation and Performance Assessment.	35
3.1 Overview.	35
3.2 Method of Analysis.	35
3.3 Performance Model Description	36
3.3.1 Performance Model basis	36
3.3.2 Performance Model Approximations.	39
3.3.3 Performance Model Parameters.	40
3.3.4 Performance Model Equations	41
3.3.5 Performance Model Results	43
3.3.6 Performance Improvements Using Special Copying Hardware	48
3.4 Performance Assessment Conclusions.	49
4 Architecture for VLSI Implementation and its Performance Analysis.	50
4.1 Factors Related to VLSI Architecture.	50
4.1.1 Need of a VLSI architecture for the bridge processor.	50
4.1.2 Factors related to VLSI design.	51
4.2 Design Approach to the Architecture.	51
4.3 Buffer Management	52
4.3.1 Memory Architecture	52
4.3.2 Factors in Buffer Management.	55
4.3.3 The Buffer Structure.	55
4.3.4 Addressing Mechanism.	59
4.3.5 Service Mechanism.	60
4.3.6 Alternative Mechanisms for Buffer Management.	60
4.4 Architecture for VLSI Implementation: ATM to FDDI.	63
4.4.1 Components of VLSI Architecture.	63
4.4.2 Event Sequences and Timing.	70
4.4.3 Summary: Protocol Function Execution Timings.	72

4.4.4 Throughput Analysis.	73
4.4.5 Delay Analysis	74
4.5 Architecture for VLSI Implementation: FDDI to ATM.	75
4.5.1 Components of VLSI Architecture.	75
4.5.2 Event Sequence and Timings.	78
4.5.3 Summary: Protocol Function Execution Timings.	79
4.5.4 Throughput Analysis.	80
4.5.5 Delay Analysis.	81
4.6 Alternative Architecture.	82
4.6.1 Functions needing improvement.	83
4.6.2 Bridge Protocol Processor Architecture.	83
4.6.3. BPP Operations for ATM-FDDI Bridge.	87
4.6.4 Performance Assessment of BPP	88
4.7 Summary.	90
 5 Traffic Related Performance Issues.	 92
5.1 Comparison of bridge-processing and MAC delays.	92
5.2 Need of higher priority for the bridge on FDDI ring.	93
5.3 Effect of 4 byte BI header on segmentation and reassembly.	94
5.4 Frame loss probability.	98
5.5 Requirement of additional MAC components.	98
 6 Conclusions.	 99
 Appendix I FDDI Supernet Chipset	 103
Appendix II Sparc Implementation: C Code.	106
Appendix III Details of VLSI Architecture: ATM to FDDI.	120
Appendix IV Details of VLSI Architecture: FDDI to ATM.	131
References	138

List Of Tables

Table 3.1 Protocol Function Execution Characteristics: ATM to FDDI	44
Table 3.2 Protocol Function Execution Characteristics: FDDI to ATM	46
Table 4.1 Address Selector Operation.	59
Table 4.2 Execution Timings for Protocol Functions: ATM to FDDI	72
Table 4.2 Execution Timings for Protocol Functions: FDDI to ATM	80
Table 5.1 Effect of 4 byte header on number of cells needed for a frame	96

List Of Figures

Figure 1.1 Wide Area Network	2
Figure 2.1 Bridging of two stations on remote networks	10
Figure 2.2 FDDI MAC Frame Format	16
Figure 2.3 B-ISDN ATM Protocol Reference Model.	18
Figure 2.4 ATM Cell and Header Structures	20
Figure 2.5 Functional Architecture of the Bridge.	21
Figure 2.6 (a) Segmentation of a MAC Frame into ATM Cells	25
Figure 2.6 (b) BI Protocol Data Unit.	25
Figure 2.6 (c) BI Header Formats for Different Cells	25
Figure 2.7 State Diagram: FDDI to ATM data flow	27
Figure 2.8 State Diagram: ATM to FDDI data flow	30
Figure 3.2 The Effect of Pipelining for ATM to FDDI data flow.	38
Figure 4.1 Triple port Memory Architecture for 1 Megabit Module	53
Figure 4.2 Two Bank Interleaved Memory	54
Figure 4.3 Buffer Organization	56
Figure 4.4 Flow Chart: Buffer Management	57
Figure 4.5 VCI Activity table	58
Figure 4.6 Alternative Buffer Mechanism Approach 1	61
Figure 4.7 Alternative Buffer Mechanism Approach 2	62
Figure 4.8 Types of FDDI Frames at the Sender Bridge.	63
Figure 4.9 Block Diagram: ATM to FDDI direction of VLSI Architecture	64
Figure 4.10 The VCI Sorter Unit	66
Figure 4.11 Address Generator	67
Figure 4.12 The Frame Transfer Unit	69
Figure 4.13 Minimum Inter-arrival time of ATM cells	71
Figure 4.14 Event Sequence for a cell processing	71
Figure 4.15 Components of delay across the bridge for ATM to FDDI direction	73

Figure 4.16 Block Diagram: FDDI to ATM direction of VLSI Architecture	76
Figure 4.17 The Cell Transfer Unit (CTU)	77
Figure 4.18 Event Sequence for frame processing for FDDI to ATM direction	79
Figure 4.19 Components of delay across the bridge for FDDI to ATM direction	81
Figure 4.20 Organization of Bridge Interface Using a BPP	84
Figure 4.21 The Bridge Protocol Processor Architecture	85
Figure 4.22 Special Register Set for BPP	86
Figure 5.1 Comparison of MAC and Bridge Processing Delays	93
Figure 5.2 Throughput versus Arrival Rate per Priority Level	94
Figure A.1 SUPERNET Chipset interconnect block diagram	104
Figure A.2 The Alignment Mechanism	122
Figure A.3 Timing Diagram: SOF Cell	124
Figure A.4 Timing Diagram: COF Cell	125
Figure A.5 Timing Diagram: EOF Cell	126
Figure A.6 Timing Diagram: SCF Cell	127
Figure A.7 (a) ATM to FDDI: Frame Processing Component State Diagram	129
Figure A.7 (b) ATM to FDDI: Frame Transfer Component State Diagram	130
Figure A.8 Timing Diagram: Frame Processing Component	133
Figure A.9 Timing Diagram: Cell Transfer Component	134
Figure A.10 (a) Frame Receive State Diagram	136
Figure A.10 (b) Frame Processing State Diagram	136
Figure A.10 (c) Cell Transfer State Diagram	137

List Of Acronyms

AAL	ATM Adaptation Layer
ADR	Alignment Data Register
AF	ATM to FDDI
ALU	Arithmetic Logic Unit
ATM	Asynchronous Transfer Mode
BI	Bridge Interface
BISDN	Broadband Integrated Services Digital Network
BPP	Bridge Protocol Processor
CBR	Continuous Bit Rate
CISC	Complex Instruction Set Computer
COF	Continuity Of Frame
CPP	Communication Protocol Processor
CT	Cell Type
CTQ	Cell Transmit Queue
CTQPP	Cell Transmit Queue Process Pointer
CTQRP	Cell Transmit Queue Read Pointer
CTQWP	Cell Transmit Queue Write Pointer
CTU	Cell Transfer Unit
CU	Control Unit
DA	Destination Address
DPC	Data Path Control
EA	End Address
EOF	End Of Frame
FA	FDDI to ATM
FC	Frame Control
FCS	Frame Check Sequence
FDDI	Fibre Distributed Data Interface

FEA	FIFO End Address
FIFO	First In First Out
FORMAC	Fibre Optic Ring Media Access Controller
FSA	FIFO Start Address
FSM	Finite State Machine
FSRP	Frame Segment Receive Pointer
FTQ	Frame Transmit Queue
FTQPP	Frame Transmit Queue Process Pointer
FTQRP	Frame Transmit Queue Read Pointer
FTQWP	Frame Transmit Queue Write Pointer
FTU	Frame Transfer Unit
GFC	Generic Flow Control
HEC	Header Error Check
IC	Integrated Circuit
ISO	International Standards Organization
LAN	Local Area Network
LI	Length Indicator
LLC	Logical Link Control
MAC	Media Access Control
MAN	Metropolitan Area Network
MAR	Memory Address Register
Mbps	Mega bits per second
MBR	Memory Buffer Register
MID	Multiplexing ID
ms	milli-second
ns	nano-second
OSI	Open System Interconnect
PDU	Protocol Data Unit
PMD	Physical Medium Dependent
PT	Payload Type
RAM	Random Access Control

RBC	RAM Buffer Controller
RISC	Reduced Instruction Set Computer
RPTR	Read Pointer
RX	Receive
SA	Start Address
SAR	Segmentation And Reassembly
SCF	Single Cell Frame
SDU	Service Data Unit
SMT	Station Management
SOF	Start Of Frame
SPARC	Scalable Processor Architecture
SSA	Segment Store Address
T&C	Timing and Control Unit
TCP	Transmission Control Protocol
THT	Token Holding Timer
TRT	Token Rotation Timer
TTR	Timed Token Rotation
TTRT	Timed Token Rotation Timer
TX	Transmit
VBR	Variable Bit Rate
VC	Virtual Connection
VCi	Virtual Channel Identifier
VLSI	Very Large Scale Integration
VMTP	Versatile Message Transaction Protocol
VP	Virtual Path
VPI	Virtual Path Identifier
VRAM	Vedio RAM
WAN	Wide Area Network
WPTR	Write Pointer

Chapter 1

Introduction

1.1 Background

Local Area Networks (LANs) are rapidly becoming the standard communication medium for interconnecting distributed facilities within corporate premises. The growing number of packet switching networks and LANs allow the computer user to have access to resources beyond those available on a single system. However, the resources of a single LAN are often inadequate for meeting users' needs, and a need arises to interconnect several LANs in order to extend the geographical limitations of a LAN.

From a users' point of view, an interconnected set of networks may appear simply as a large network. The interconnection of networks is carried out by special devices. There are three type of devices which are normally used to interconnect networks: bridges, gateways and routers. These devices provide a communications path and necessary logic for the data movement across interconnected networks. These devices are employed at different layers of the Open System Interconnect (OSI) model and are distinguished on the basis of functionality carried by them.

The growing speeds of the LANs and the emerging standards for Broadband Integrated Services Digital Network (BISDN) present a challenge for the high-speed network interconnection, because of the fast processing required for the high-speed data movement. One such scenario, an interconnection of 100 Mbps Fibre Distributed Data Interface (FDDI) high-speed LANs [1], through a backbone based on Asynchronous Transfer Mode (ATM) protocol [2], is discussed in this thesis. Figure 1.1 shows basic configuration of the Wide Area Network (WAN) environment where bridges are used to carry out the interconnection. Because the frame formats used in the end networks are identical, minimal processing is feasible at the bridge.

Figure 1.1 Wide-Area Network

1.2 Objective

The current state-of-art bridges can not handle data movements at rates of 100 Mbps or more. The protocol processing to execute the data transfer functions in the bridge needs to be handled in hardware as software implementation would clearly be inadequate for the purpose. Specialized architecture such as the use of multiprocessor configuration may be a reasonable choice for the bridge implementation.

The objective of this thesis is to investigate the practical feasibility of a high-speed implementation of an FDDI to ATM bridge that permits us to match the performance potentials of the interconnected LANs.

1.3 Scope

The objective is achieved by breaking the problem into different steps as given below.

- (1) A Bridge Interface (BI) protocol is designed which is compatible with ATM concepts and standards and has minimal functionality so as to lend itself to a high-speed implementation, capable of data transfers up to 100 Mbps or more.

- (2) The potential performance of the BI protocol is assessed by analyzing a C language code implementation on RISC based SPARC processor. The performance bottlenecks are determined by investigating the components that effect the throughput of the C code implementation of the protocol.
- (3) It has been tried to explore from the SPARC based C code analysis, whether a conventional processor with a traditional memory architecture is capable of doing the needed processing. It was found that a conventional processor is clearly incapable for the purpose.

A VLSI architecture with a special memory architecture is proposed as a solution for the bridge interface implementation. The objective of this architecture has been to exploit the protocol inherent parallelism so as to achieve a fast processing and allow data movement across the bridge with a least possible transit delay.

The VLSI architecture proposed, is however specific to the BI protocol implementation. Although it meets the required goal, a more general protocol processor architecture could be developed which would allow other similar applications to be handled.

- (4) Other issues such as the traffic characteristics related factors that affect the performance of the bridge are discussed in the end.

The performance of the bridge interface has been analyzed from the point of view of data movement and the effects of processing speed limitations; other issues such as station management activities have not been discussed. The BI protocol considered here has functionality sufficient for only asynchronous type of data movements (i.e. FDDI-I) across the bridge; further analysis and modifications to the BI protocol may be needed in order to implement it for data movements which are synchronous in nature (i.e. FDDI-II).

The VLSI architecture proposed in the thesis is at the block level and not at the lower hardware level; it thus details only functional elements. The architecture proposed is scalable: an improvement in hardware clock rate will thus proportionally enhance the performance of the architecture. Although the architecture is specific to the BI protocol implementation, its concepts are strong enough to accomplish solutions to other similar implementation problems.

The performance of the bridge greatly depends on the traffic characteristics handled by it. Unfortunately no appropriate analysis is available on the traffic characteristics for internetworking environment which involves a mix of service types as envisaged

in BISDN. Thus, certain performance issues such as buffer requirement at the bridge, allocation of higher priority to the bridge on the FDDI ring etc, can not be addressed appropriately.

1.4 Literature Survey

Transparent bridging is becoming popular as a mechanism for interconnecting LANs. Several authors have emphasized the importance of transparent bridging as a means for optimizing the functionality of a networking device [4,5,7]. Hawe et al. [5] classify the transparent bridges on the basis of method of forwarding the data. The two types classified are: *Pass through* and *translation*. The pass through mechanism usually allows interconnection of similar LANs and passes the protocol data units intact, from one network to the other. The bridge considered in the thesis is essentially a pass through bridge. The translation mechanism usually interconnects LANs which are sufficiently similar so as to allow a translation of address and control fields from one network to another.

Backes [7] outlines the functions of a transparent bridge. Crowcroft and Riddoch [10], Pitt et al. [6] and Backes have described and explained the addressing and routing algorithms for bridges. They identify a problem in interconnected bridges, known as looping, which may occur because of existence of multiple paths in bridged LANs. The spanning tree algorithm, which ensures that there is only one path of bridges and LANs in a wide area network, is suggested by authors as a solution to the problem. However, in a wide area network, which has ATM as the backbone, this kind of problems will not arise because of the uniqueness of the virtual circuit.

A similar topology of wide area network, as considered in this thesis, has been discussed by Vallee et al. [30] for the interconnection of IEEE 802.5 token ring networks with an IEEE 802.6 MAN as backbone. Vallee uses an LLC level bridge and finds the throughput and delay characteristics in a window flow control environment. The bridge analyzed here however, is different in architecture and approach of analysis. Perlman et al [3] have discussed the selection of an appropriate layer for LAN interconnection.

The high speed protocol implementation needs to exploit the inherent parallelism within

the protocol [9,22]. Kanakia and Cheriton [9] have discussed a possible implementation of a high-speed transport layer protocol - Versatile Message Transaction Protocol (VMTT). They indicate that the current host to network adapter interfaces impose excessive processing, system bus and interrupt overhead on a host processor. These adapters are either limited in function, wasting key resources such as system bus, or else intelligent but too slow, because of the protocol complexity and inadequate memory architecture. The authors indicate that checksumming and encryption dominate the protocol processing of a large packet. Moreover, for internetworking applications where high speed data movements in two directions are involved, the system bus and the memory bandwidth are often the bottleneck. Dang et al [21] have compared different microprocessors working at different clock rate, and determined the network throughput which can be obtained by them. They have proposed a dual port memory as the solution to the problems of system bus and memory bandwidth. The idea of using a dual port memory is also supported by Kanakia and Cheriton and Zitterbart [20].

The performance of the protocol processing can be increased by reducing the number of copying operations. The activities such as removing the headers, reassembly and segmentation etc. may need additional copying [9]. It has been an objective in the VLSI architecture design, to minimize the amount of copying. Giarrizzo [31] and Clark et al. [15] have emphasized a similar approach for the processing of packets in an environment which includes different OSI layers. A need of a scatter-gather DMA, which permits protocol processing while not incurring additional copying, has also been stated by Giarrizzo.

The flexibility of an architecture is important to prove its usefulness. A general purpose communication protocol processor has its own advantages over a special purpose VLSI architecture. A communication protocol processor (CPP) is analyzed by Mandalia et al. [22]. The authors have used special instructions to carry out the protocol functions more efficiently than by a general purpose processor. A similar approach is used in this thesis to explore a bridge processor architecture that uses specific instructions to carry out certain protocol activities in a fast manner, and is more general purpose than the VLSI architecture.

Prior to designing a VLSI architecture for the BI protocol, in this work, the throughput of its implementation on a general purpose processor is analyzed to reach a ball park for the performance requirements. Clark et al. [15] have used a similar approach to analyze the

processing overhead of a transport layer protocol - Transmission Control Protocol (TCP). The authors concluded after a study of the protocol that it is the operating system related functions such as buffer management which are the real source of overhead and not the TCP protocol itself. They modified the TCP protocol so as to remove the operating system related functions and wrote special functions to replace them. A common path of the protocol was determined and the number of assembly instructions needed to execute the common path were obtained by compiling a protocol implementation code on an Intel 80386 processor. The authors indicated that an efficient implementation of the TCP could even achieve a throughput of 800 Mbps, but this was not substantiated.

Martini et al. [26] indicate that there are two components which may be identified as possible bottlenecks in a high speed communication system, namely medium access protocols and processing of protocols. The authors compare the media access delay of a 140 Mbps token ring backbone LAN to the station delays of the bridges which are used to interconnect 2 Mbps token ring LANs. They conclude that for state-of-art bridges which can not forward more than a few thousands packets per second, station delay is the major cause of bottleneck. However, if the bridge processing is increased to deliver about 10,000 packets per second, the impact of station delay reduces and becomes smaller than the MAC delay. This proves the requirement of high-speed processing in the ATM to FDDI bridge considered in the thesis.

Another study with respect to the traffic characteristics is done by Wainwright and Myles [27]. They studied different network sizes under different traffic conditions and compared the delay characteristics of FDDI and IEEE 802.6 metropolitan area networks (MAN) network-access mechanisms. The authors concluded that the maximum throughput of both the networks is adversely affected by the presence of a high proportion of short packets in the traffic, and that this effect is greater in IEEE 802.6 network. Kanalia and Cheriton have presented a similar discussion for their Network-adaptor-board architecture. However, the work done in this thesis has not addressed the traffic related issues in great detail.

1.5 Thesis Layout

This thesis will be concerned with the data processing speed limitations of high speed

network interfacing with an ATM to FDDI bridge as the example scenario. Chapter 2 discusses the architectural issues of the bridge and proposes a minimal functionality protocol. The minimal functionality protocol is intended to accomplish a throughput which should match the performance potentials of interconnected LANs. The FDDI and ATM protocols are also described in this chapter.

The throughput of the bridge is assessed by analyzing a common data path for the two directions of data flow across the bridge and is described in Chapter 3. A C code implementation on a RISC processor is used to carry out the analysis and to determine the bottleneck area.

Chapter 4 provides a detailed architecture description of a VLSI implementation for the BI protocol and discusses its performance. It will be shown that the VLSI architecture matches the needed throughput requirement. A more flexible architecture which may also be used for other similar applications is discussed in the end of Chapter 4.

Chapter 5 discusses some issues which are related to the effect of traffic characteristics related factors on the bridge performance. It is felt that a more exhaustive study of these issues is needed to reach more precise conclusions. This requires a precise traffic model for the wide area network environment.

Finally, Chapter 6 summarizes the major conclusions as well as provides some suggestions for future work.

Chapter 2

Bridge Interface Protocol

2.1 Overview

Most of the existing protocols at the data link layer or above are slow compared to the throughput needed to match the increasing speed at the physical layer. The two possible approaches to improve the throughput of the existing protocols are:

- (1) Design new protocols with an objective to have a minimal possible functionality
- (2) To explore a better way of implementing protocols.

Clark et al [15] have studied the overhead for the transport protocol TCP and showed that it is not the protocol that is the major cause of the low throughput; instead it is the overhead caused by the operating system that slows down the protocol. The authors say that the TCP throughput could even reach 800 Mbps if implemented properly.

This chapter discusses a similar approach to the design of a minimal functionality bridge interface (BI) protocol. The architectural issues related to the protocol are discussed first. Choosing an appropriate layer for LAN interconnection and the connection mode play an important role in determining the functionality carried by the interconnecting device.

The three protocols, the FDDI, the ATM and the BI protocols are then described followed by a detailed explanation of the state machine implementation of BI protocol.

2.2 Architectural Issues

2.2.1 Bridges

When two systems are not connected directly to each other and the connectivity is through an intermediary, the intermediary is known in International Standards Organization (ISO)

terminology as a *relay* [3,4]. There can be one or more relays between the two systems. The ISO has defined in their OSI reference model, seven layers within a network architecture. A relay can be defined on the basis of the layer at which it functions. If the relay shares a common layer n protocol with other systems, but does not participate in a layer $n+1$ protocol in the process of relaying information, it is known in ISO terminology as *layer n relay*.

Commonly implemented types of relays are:

- (1) Physical layer relay — known as a *repeater*
- (2) Data link layer relay — known as a *bridge*.
- (3) Network layer relay — known as a *router*.
- (4) Any higher layer than network layer relay — known as a *gateway*.

Unfortunately no terminology is used universally. Thus, the term gateway, when used in networking literature, may refer to a device functioning at any layer from data link layer to upper layers.

The relay discussed in this thesis, essentially operates at the data link layer. It is basically a MAC layer bridge which is the simplest of all types of relays. The bridge discussed in the thesis, which is used to interconnect two similar networks, uses the transparent bridging concept as given below.

Transparent Bridging

A bridge only relays user information. It does not forward data link specific control information such as a token. If the bridge passes the user information from one network to another in an intact manner, and it does not modify or affect the protocol of each of the individual end networks in the process of transferring the user information, then it is known as transparent bridging. Another property of transparent bridging is that it does not require any modification to the existing protocol at the end stations. This is in contrast to other bridging techniques that require modifications to the MAC sublayer of existing stations [5,6]. Thus in transparent bridging, hosts need no special knowledge to transfer information through a bridge, whereas they normally need to know the address of the relay while sending a message through a routing device. Transparent bridges are transparent to protocols operating at or above the Logical Link Control (LLC) sublayer. They support both connectionless and connection oriented services [7]. Basic functions

of a transparent bridge are : (1) Frame forwarding, (2) Learning of station address and (3) Resolving possible loops in topology.

LLC: Logical Link Control MAC: Media Access Control
 ATM: Asynchronous Transfer Mode Layer
 PIHY: Physical Layer PMD: Physical Medium Dependent Layer

Figure 2.1 Bridging of two stations on remote networks

In general, some of the useful properties of a bridge are that, it increases reliability, performance, security and geographical extent of the interconnected networks.

2.2.2 Appropriate Layer for Bridging

The selection of an appropriate OSI layer is an important factor in designing a relay. The functionality carried out by the bridge varies according to the specific layer at which the networking is done. The objective for the design of the bridge here, is to achieve a minimal functionality. This can be achieved only if the internetworking is done at lower layers, that is at the data link or the network layer. The bridge-to-bridge routing functions are an inherent part of the ATM network and this work discusses connection orientation (see Section 2.2.5). Further, by doing the bridging at MAC layer, no explicit flow-control is required at the bridge. Thus, it is concluded that in order to support a minimal functionality of the bridge interface implementation, MAC layer is the most appropriate choice for the bridge.

2.2.3 Desirable Characteristics of a bridge

The desirable characteristics of a bridge can be summarized as below.

- **Minimize traffic:** only the traffic generated by the user stations exists on the backbone; control messages local to end networks, are not sent on it.
- **Minimize transit delay:** the higher layer protocols are designed to have flow control using special timers. The bridge transit delay above a certain limit will cause the protocol to malfunction. The transit delay through the bridge therefore must be the minimum possible.
- **No Duplicates:** the bridge should not cause duplicate frames to be delivered to the destination(s).
- **Sequentiality:** the combination of LANs and bridges should not permute the frame ordering as transmitted by the source station.
- **Low error rate:** LANs typically have a low effective bit error rate. Higher layer protocols are often designed with this in mind. This allows the protocols to operate more efficiently since they can assume that errors are infrequent. Extended LANs should not increase this error rate substantially.
- **Generalized Topology:** For purposes of traffic splitting and reliability, it would be useful to allow arbitrary interconnection of LANs via bridges.
- **Low Congestion Loss:** Individual LANs minimize congestion by employing access control schemes which prevent excessive traffic from entering the LAN. Extended LANs are more vulnerable to congestion because of limited buffers compared to the traffic handled. This phenomenon should be minimized by a proper design.

2.2.4 Bridge Functionality

The functionality of a bridge varies according to the types of services supported by it. A bridge which supports isochronous or time critical services will require functions to minimize the transit delay. The bridge considered here however supports only asynchronous services (i.e. FDDI-I) which are not time critical but may be bursty in nature.

There exists a set of core functions [8] that are required for all asynchronous/bursty data services, whereas optional functions are selected by applications according to the need. Examples of core functions are segmentation and reassembly, buffer management and those of optional functions are flow control and retransmission on error detection. The

bridge executes most of the core functions. The functions are carried out by the bridge in form of a bridge-interface (BI) protocol. The functionality of the proposed BI protocol is summarized below.

(a) Functions carried out by the BI protocol:

- Segmentation and reassembly
- Address translation
- Frame identification and sorting
- Cell-loss detection

(b) Functions not done explicitly by the BI protocol:

- Sequencing
- Flow control
- Error control and retransmission
- Routing

The latter functions are executed inherently as part of the ATM protocol. The ATM protocol is described in detail in section 2.3.2. According to Kanakia and Cheriton [9], checksum, encryption etc. are the components that dominate the protocol processing of any transport protocol. The BI protocol minimizes the protocol functionality by not doing these functions, thus achieving significant saving in protocol execution time.

2.2.5 Connection oriented vs. connectionless

Networks based on packet switching use two alternative modes for data transfer: connection-oriented and connectionless. The mode may be different at different levels of OSI layers; a transport layer may be connectionless whereas the underlying MAC layer may be connection oriented or any other combination. The selection of a particular mode for the bridge will depend on the factors such as the type of the backbone network and the modes supported by it, and the functionality to be carried out by the bridge.

The details of the functionality needed by the connection-oriented and the connectionless modes for the ATM network are given in section 2.3.2. The connectionless mode requires

more functionality than the connection-oriented one. From the point of view of very high speed networks, such as the one being considered in this thesis, more functionality may result in a significant increase in the protocol processing time. The connectionless approach requires in addition, convergence sublayer functions which are not required by the connection-oriented approach. The latter assumes the orderly receipt of cells, thus it does not need frame-segment-identification and hence needs less processing at the receiving as well as the sending end. Not only does a connectionless approach increase transmission overhead but, more significantly, it increases the address processing at different nodes in the ATM network.

2.2.6 Address Resolution and Mapping

The address resolution scheme directly affects the bridge performance; a scheme which results in a large address-map table will result in a longer search time. For proper bridge-to-bridge data transfers, the sending bridge must know the VCI/VPI field value corresponding to the destination bridge. This needs a proper translation of the MAC address field into the VPI/VCI field. The bridge maintains a database which contains the mapping between the two addresses for all the stations within the WAN. This however, may cause a large number of values to be searched whenever a source bridge wants to send a frame. An approach to deal with this situation is to use a hierarchical addressing for different stations on each FDDI ring. In this scheme all stations on an FDDI LAN are allocated contiguous address values. The address bits which identify an individual station is not of concern to the sender bridge; only the bits that identify the LAN are needed to identify the remote LAN. This requires a fewer bits to be compared for the search and results in a faster execution.

Creation and updating of the database is done at the time of connection set-up.

2.3 Protocol Descriptions

This section describes the protocols in the order as given below:

- (1) FDDI protocol
- (2) ATM protocol

The FDDI and ATM protocols are not described in great detail; only the salient features and the components which are related to the thesis work are discussed.

2.3.1 The FDDI Protocol

FDDI [1] is a 100 Mbps token ring with a maximum configuration of 100 kilometers and 500 stations. It consists of data-link and physical layers, and station management (SMT) plane. The data-link layer has two sublayers: Logical Link Control (LLC) and the Medium Access Control (MAC).

LLC provides a common protocol to provide the required data assurance services between MAC and the network layer. It can be a standard one such as IEEE 802.2. The heart of the protocol is the MAC. It provides fair and deterministic access to the medium, address recognition, and generation and verification of frame check sequences. Its primary function is the delivery of frames, including frame insertion, repetition and removal. The physical layer does the coding/encoding of data being transmitted/received on the physical media and provides synchronization among various stations. It moves a stream of 4 bit symbols to and from the MAC layer. SMT plane provides the control necessary at the station level to manage the processes underway in the various FDDI layers such that a station may work cooperatively on a ring. SMT provides services such as control of station initialization, configuration management, fault isolation and recovery, and scheduling procedures.

FDDI MAC protocol

The FDDI MAC protocol in concept is similar to the IEEE 802.5 token ring except the token release mechanism. In the FDDI protocol, the token is released immediately after the last data frame is sent by the transmitting station. In the token ring protocol, the token is released as soon as the transmitted frame comes back to the transmitting station. This is because of the fact that the frame transmission time in FDDI is much less than the ring propagation delay; thus enabling more than one frame to exist simultaneously on the ring. The protocol is described as follows. The stations are arranged in a ring and they control access to the channel by passing a token around the ring. Only the station which has seized the token is permitted to transmit across the channel. Each station removes its own transmitted MAC frame from the ring and can thus receive acknowledgment, appended

by the destination as a trailer to the frame. The transmitting station issues the token immediately at the termination of its transmission, so that the next busy station along the ring can seize it following the latest transmission which crosses its interface. An idle station repeats the token, while a busy one strips it from the ring.

The access of nonisochronous frames onto the medium is controlled by a timed token rotation (TTR) protocol. When the ring is initialized, the stations negotiate a target token rotation timer (TTRT). The protocol guarantees that the average token rotation time will be less than TTRT and the maximum token rotation time will be less than $2 \cdot \text{TTRT}$. Thus, stations with tight delay constraints will request a small TTRT. The smallest requested TTRT is selected and is used by all of the stations. The choice of TTRT greatly affects the utilization of the ring.

Each station has a token-rotation-timer (TRT) and a token-holding-timer (THT); these are used to control the amount of time that the station is allowed to transmit asynchronous messages. The TRT is used to measure the time between successive repetitions of the token. When a station receives the token, it checks its TRT. If the time elapsed exceeds TTRT, the station is not allowed to transmit any asynchronous frame. Otherwise, the THT is loaded with the value of TRT, and the station begins transmitting asynchronous frames. After each frame transmission, it checks to see whether the THT-threshold has been crossed. Once it has been, the token is passed to the next station. THT-threshold thus controls the time for which a station can transmit the maximum number of asynchronous frames on a particular token capture; up to eight message priority levels can be established by using different threshold values for different message types. The maximum threshold value is TTRT.

Frame format and description of fields

The MAC frame format is shown in Figure 2.2, and its various fields are described below.

- **Frame Control (FC)** field is an 8 bit field which carries information on frame type and class, and on the address length. Thus synchronous, asynchronous classes and LLC, SMT or control types of frames are distinguished with the help of this field.
- **Destination Address (DA)** and **Source Addresses (SA)** are two separate fields which can be either 16 or 48 bits in length. On a ring, a mix of both types of addresses are permissible. Group addressing is also possible along with broadcast facility.

FC = Frame Control

DA = Destination Address

SA = Source Address

INFO = Information field

FCS = Frame Check Sequence

Figure 2.2 FDDI MAC Frame Format

- **Information (INFO)** field contains user data and can be of any length from 0 to 4500 octets.
- **Frame Check Sequence (FCS)** is used to detect erroneous data bits within the frame as well as erroneous addition or detection of bits to the frame.

Primitives

The primitives for movement of data unit between MAC and LLC sublayers are given below.

(a) For transfer of a data unit from MAC to the local entity:

```
MAC_UNITDATA.indication    (
    FC_value
    destination_address
    Source_Address
    M_SDU
    reception_status
)
```

The FC_value and M_SDU contain the FC and INFO fields respectively, as described above. Reception_status contains the results of the error check performed by the MAC hardware on the FCS value and the frame status information such as whether the frame was copied by the destination or not.

(b) For transfer of one or more SDUs from local LLC to single peer LLC entity or multiple LLC entities in the case of group address.

```

MAC_UNITDATA.request      (
    FC_value (1)
    destination_address (1)
    M_SDU (1)
    requested_service_class (1)
    stream (1)
    .
    .
    .
)

```

Requested_service_class parameter indicates to the MAC layer that whether the data unit being sent is of type synchronous or asynchronous. For the latter, the priority level can be optionally provided. The stream parameter when set, indicates that multiple data units are being transferred; when reset it indicates the end of data for that destination address.

2.3.2 ATM Protocol

Asynchronous Transfer Mode (ATM) [2] is the transfer mode for implementing Broadband Integrated Digital Services Network (B-ISDN). The protocol, at the time of writing this thesis is in a process of being standardized. It is a packet oriented transfer mode using asynchronous time division multiplexing technique: the multiplexing information flow is organized in fixed size blocks, called cells. An ATM cell has a header and a payload. The generic protocol model of ATM protocol shown in Figure 2.3 has three major components:

- (1) ATM Adaptation layer (AAL)
- (2) ATM layer
- (3) Physical Medium Dependent (PMD) layer

The PMD layer presents the data received from ATM network to ATM layer in an acceptable form, possibly 8-bit or 32-bit. The ATM and adaptation layers provide the transfer functions and are described below in details.

Figure 2.3 B-ISDN ATM Protocol Reference Model

ATM layer functions

The functions of ATM layer are the same for connection oriented and connectionless types of services. For the ATM to FDDI bridge, a connection oriented mode is used. Functions of ATM layer are given below.

- Cell delineation: it merges appropriately cell boundary information and the cell flow into a data flow to be transmitted, and recovers the cell boundaries from the received data flow.
- Cell header verification: it computes and appends Header error check (HEC) to outgoing cells, checks and discards incoming cells if header errors can not be corrected.
- Cell multiplexing-demultiplexing: it multiplexes cells coming from individual VC/VPs into a non-continuous composite flow of cells and reverse. It provides therefore, buffering to outgoing cells for contention resolution.
- Cell routing: performs VCI/VPI processing and routing according to routing information of the header.

While moving the data from AAL to and from PMD layer, the ATM layer does not buffer them; it just passes them through FIFO.

Adaptation layer functions

The AAL supports higher layer functions of the user, control and management planes. It also supports connections between ATM and non-ATM interfaces. The AAL has not been

standardized completely as of this time. There are two broad classes of services provided by the AAL: Variable Bit Rate (VBR) and Constant Bit Rate (CBR). The analysis for the ATM to FDDI bridge is being done only for the asynchronous i.e. VBR types of services. For connection-oriented and connectionless data services the functions carried out by AAL are given below.

Connection oriented mode

- detection of information blocks from the higher layers; and sending information blocks to higher layers.
- handling of partially filled cells
- segmentation & reassembly of information blocks
- action on lost and inserted cells
- flow control

Connectionless mode

- detection of information blocks from the higher layers; and sending information blocks to higher layers.
- handling of partially filled cells
- segmentation & reassembly of information blocks
- message identification (MID)
- per cell error detection

ATM cell structure and description of fields

The cell structure has been standardized to have 5 bytes of header and 48 bytes of payload. It is shown in Figure 2.4.

- **Generic Flow Control (GFC)** field is used for flow-control at the user-network interface.
- **Virtual Channel/Virtual Path Identifier (VPI/VCI)** identifies a source-destination pair on an ATM virtual circuit. The VPI/VCI is also used to route a cell from a source to a destination on the ATM network.
- **Payload Type (PT)** is under standardization.
- **Reserve (Res.)** field is reserved for future use.
- **Header Error Check (HEC)** contains error checking information for the header.

ATM cell structure

ATM header structure

GFC: Generic Flow Control (4 bit)

VCI/VPI: Virtual Channel/Virtual
Path Identifier (24 bit)

PT: Payload Type (2 bit)

Res.: Reserved for future (2 bit)

HEC: Header Error Check (8 bit)

Figure 2.4 ATM Cell and Header Structures

Primitives

The primitives used by ATM layer for data unit transfers to and from the AAL (or the equivalent) are as follows:

(a) For data unit transfer from ATM layer to AAL:

```
ATM_UNITDATA.indication      (
                                VCI/VPI value
                                Payload_type
                                ATM_payload
                                )
```

The VCI/VPI value identifies a source-destination pair. The format of the VPI/VCI field is still under standardization and it is not definite whether the ATM layer passes the VCI/VPI value or a translated form to the AAL. However for the purpose of the analysis in the thesis, the primitive format is the one that is shown above. The payload-type field may not necessarily be needed, if it has a significance only for the ATM layer and not for the AAL.

(b) For data unit transfer from AAL to ATM:

```

ATM_UNITDATA.request      (
    destination_address (1)
    payload_type (1)
    ATM_payload (1)
    stream (1)
    .
    .
    .
)

```

The `destination_address` field is used by the ATM layer to determine the VCI/VPI field value. The payload type field again may not be necessary if it has no significance to AAL. The structure of the primitive is similar to the one that of `MAC_UNITDATA.request`, in that it uses a parameter stream to indicate a multiple data unit transfer.

2.4 The Proposed BI Protocol

The BI protocol is intended to interface a standard implementation of FDDI MAC layer on one side and ATM Adaptation layer (AAL) on the other by using the architecture shown in Figure 2.5. The essential data transfer functions of the bridge interface are to:

Figure 2.5 Functional Architecture of the Bridge

- segment the FDDI frames into ATM cells for transmission.
- construct the ATM cell header.

— sort and reassemble incoming ATM cells into FDDI frames.

Since several remote FDDI LANs may be simultaneously sending frames to any receiver LAN, the constituent ATM cells of these different frames will be interleaved as they arrive at the bridge.

For the purpose of transferring FDDI frames between bridges, a BI protocol has been designed which has the following objectives:

- (a) Ensuring proper transfer of frames between LANs.
- (b) Ensuring transparency of the inter-bridge transfer to the FDDI MAC protocols at each edge of the WAN.
- (c) Not duplicating any FDDI MAC or ATM or higher layer functions in edge-to-edge communicating stations.
- (d) Minimizing the functionality required in the bridge.

Objectives a, b, and c essentially implement a pass-through mechanism [5] between the two LANs. Given those objectives, objective d is intended to lay the basis for high speed implementation. The protocol design features are discussed in detail below.

- **Connection control:** The connection mode for data transfers between any two bridges is connection oriented one. It is assumed that an ATM connection has been established at some previous time between each pair of bridges. The connection mechanism which connects any two bridges through the ATM backbone for the purpose of data transfers, is the one adopted by the standard ATM protocol. The request for connection for the purpose of data transfers, is made by the higher layers in the end stations. However, the bridge itself might make connection requests in certain cases when it requires to do some station management activities.

It is noted that a connection-oriented approach requires much less functionality than a connectionless approach and hence is more suited to high-speed implementation of the protocol. Moreover, a pre-established connection ensures minimum delay in bridge-to-bridge data transfer and reduces the requirement of buffering at the sending bridge.

- **Addressing:** each bridge-to-bridge link is identified by a unique virtual path/virtual channel identifier (VPI/VCI) in whatever standard form eventually adopted for the use

in ATM cell headers. It is assumed that the addresses are unique within the WAN. The correct VPI/VCI field at the sending end is determined by consulting a look-up table at the entry corresponding to the sending MAC address field. The look-up table is established at the time of connection establishment. This approach is similar to that of Tanaka [11]. It is also assumed that MAC addresses are unique within a WAN. To this end the bridge will support both 16 and 48 bit addresses as defined by the standard FDDI MAC protocol.

- **Sequencing:** inherent to the ATM protocol, a connection-oriented mode ensures sequential delivery of ATM cells between any two end stations. Thus, various frame segments sent by a source bridge arrive in order at the destination bridge which makes the protocol processing simpler and faster.

- **Segmentation and reassembly:** segmentation and reassembly requires identification of various frame segments. Further, since a receiver bridge may be receiving more than one frame at a time in the form of interleaved cells from different sources, a frame-identification is needed to separate the incoming frame segments in order to get them reassembled. As noted in section 2.3.2, the ATM layer passes VCI/VPI value (or an equivalent) as part of the primitive for passing a data unit to the above layer. The VPI/VCI field in each cell header is used to identify the LAN bridge from which the cell has come. Since each bridge can only be transmitting one FDDI frame at a time, the VPI/VCI field also identifies the correct frame to which the cell should be reassembled at the receiving bridge.

The sender bridge makes 47 byte segments of a MAC frame in order to create ATM payloads. These frame segments are appended with a one-byte BI protocol header to form a 48 byte ATM payload. This has been depicted in Figure 2.6(a). The structure of the BI protocol data unit (PDU) which constitutes the full payload of an ATM cell is shown in Figure 2.6(b). The BI header provides the identification of the frame segments; it uses a 2 bit Cell_Type field to identify start-of-frame (SOF), end-of-frame (EOF), continuity-of-frame (COF) and single-cell-frame (SCF). The individual consecutive intermediate cells are not distinguished from each other as it is assumed that all the frame segments are delivered in order. Another identification provided by the Cell_Type field is the single-cell-frame which carries a MAC frame less than 47 byte. The format of the BI header for different types of cells is also shown in Figure 2.6(c). The remaining 6 bits of the BI header are utilized to carry different information for different types of cells.

For an SOF cell, it carries the cell-count (CC) information with a granularity of two. This information is used by the receiver bridge for local buffer allocation. For EOF and COF cells, these 6 bits of BI header carry the length indicator (LI) value which indicates the number of user data bytes in the last cell of a frame. These partially filled cells are taken care by the receiver bridge at the time of reassembly. The segmentation-and-reassembly (SAR) policy for the BI protocol involves less functionality than that carried by the standard AAL.

- **Error Control:** The MAC and the ATM protocols do their respective error checks and the BI protocol does not implement any additional error checking. The BI protocol checks for the loss of either the EOF or SOF cells in a frame only by detecting violations of the order in which these should arrive. If a frame can not be reassembled because of such loss, it is abandoned.

Another error condition may occur when the EOF cell of a frame is lost followed by the SOF cell loss of the next frame. This may force the frame segments of the two frames to be appended together with the buffer allocated to the first frame is overwritten by the segments of the second frame. The bridge takes care of this by doing a check on the number of cells that go to make a frame. It keeps looking for the last expected cell of a frame, and tests whether it is an EOF cell or not. If it is not, it assumes that the EOF cell has been lost due to error. It then tests the next cell-which may be either the SOF, COF or the EOF cell of the next frame. If it is an EOF cell of the next frame, the bridge completes the frame and sends it to upper layer although with error. If it is an SOF or a COF cell it starts a new buffer block to store the next frame. The cell-count information provided in the SOF cell is used to determine the instance of the expected arrival of an EOF cell.

No further error checking is carried out by the bridge because the edge-to-edge MAC entities will detect any erroneous frames and treat them as they would treat any other erroneous frame. Similarly, lost frames will be detected by edge-to-edge transport of higher layer protocols as though the frames had been transmitted locally. This helps in minimizing the BI protocol functionality.

- **Congestion control:** it will depend on the traffic characteristics and the memory being used at the bridge. There is no rigid policy at the bridge regarding congestion control;

Figure 2.6 (a) The segmentation of a MAC frame into ATM cells

Figure 2.6 (b). BI Protocol Data Unit

Figure 2.6 (c). BI Header Format for different cells.

it simply discards the excess data when buffer overflow occurs. A congestion control policy used by the ATM protocol standards may be implemented as an alternative.

- **Flow control:** there is no specific policy laid-out for the bridge interface. The flow control policy adopted by the AAL standard protocol may be implemented for a bridge

to bridge link flow control.

- Routing: is done as an inherent function of the ATM protocol.

Minimal functionality: connectionless vs connection oriented mode

The connectionless mode has following disadvantages over the connection oriented mode, with respect to the minimal functionality carried by the bridge.

- requires additional table search for the message identification (MID) field.
- requires additional processing for sequencing related mechanism.
- requires additional convergence sublayer functionality on top of AAL.
- frame by frame addressing and routing requires additional processing.
- routing through the ATM network for connectionless mode requires more time than connection-oriented mode.

2.5 BI Protocol Implementation

The protocol functions described in section 2.3.3 can be implemented using a finite state machine (FSM) approach. State machines are useful in defining the protocol functions in a systematic and efficient manner, and have been used for the design of other protocols as well [1,12,13,14].

2.5.1 State Machine Implementation

The bridge consists of the following two data flows:

- (1) Receive from FDDI, transmit to ATM, and
- (2) Receive from ATM, transmit to FDDI.

These data flows are referred to as FDDI to ATM and ATM to FDDI directions respec-

tively, in further discussions. The state machines described here are for a single bridge. For the ATM to FDDI direction, on the virtual circuit, the data being received is that from a single source LAN. Frames from different source LANs will be received in the form of interleaved cells, but cells from separate sources are identified by separate virtual circuits. The details of the state machine implementations for the two data flows are given below.

(a) FDDI to ATM Direction

The state machine is shown in Figure 2.7. It has four states:

FA0: Await_New_Frame (wait for indication of a new frame)

FA1: Receive_and_Analyze_Frame (copy a frame from MAC and analyze it)

FA2: Discard_Bad_Frame (discard a frame with error)

FA3: Process_Frame_and_Transfer_Cells (segment the frame, create the cells and transfer them to ATM layer)

Figure 2.7. State Diagram: FDDI-to-ATM data -flow

The details of various states and their associated transitions are given below.

FA0 (state 0): Await_New_Frame

This state represents the condition when the bridge interface (BI) is waiting for an indication from the MAC of the arrival of a new frame. The local MAC entity generates a primitive `MAC_UNITDATA.indication` to tell the

upper layer of the arrival of a new frame. The primitives are discussed in section 2.3.1.

The transitions from and to the state FA0 are as follows:

FA (0 to 1)

— occurs as soon as the BI starts receiving a frame, i.e. basically the transition occurs on the receipt of a MAC_UNITDATA.indication from the MAC or a similar indication from the MAC controller IC.

FA (2 to 0)

— occurs when a bad frame has been received and has been discarded.

FA (3 to 0)

— occurs when BI has sent all the ATM cells of a particular frame and is now available to receive a new frame.

FA1 (state 1): Receive_and_Analyze_Frame

This state represents the condition when the BI receives a frame from the MAC. The BI puts this frame into the local buffer for further processing. While the frame is still being received, the BI starts processing it so as to get some pipelining effects.

The BI analyzes the Frame Control (FC), Destination Address (DA) fields of the received frame. It decodes and translates them to get the Payload-Type and the VCI/VPI fields of the ATM protocol. These fields are discussed in section 2.3.2.

The transitions from and to the state FA1 are as given below.

FA (1 to 2)

— occurs if after decoding the FC and DA fields, there is no match from the look up tables, thus leading to the event of discarding the frame just received.

FA (1 to 3)

— occurs after valid FC, DA fields have been received i.e. when there is no error in the header of the frame.

FA (0 to 1)

— occurs as soon as the BI starts receiving a frame, i.e. basically the transition occurs on the receipt of a MAC_UNITDATA.indication from the MAC or a similar indication from the MAC controller IC.

FA2: (state 2) Discard_Bad_Frame

After the BI finds a mismatch in the DA and/or FC fields, this state is reached. The mismatch could take place because of the occurrence of an error while the frame was being transferred from MAC to BI.

Following are the transitions from and to the state FA2.

FA (2 to 0)

- occurs when a bad frame has been received and has been discarded.
- occurs if after decoding the FC and DA fields, there is no match from the look-up tables, thus leading to the event of discarding the frame just received.

FA3: (state 3) Process_Frame_and_Transfer_Cells

Represents the processing of a frame so as to segment it into ATM cells. This processing requires the following steps.

- translate DA of FDDI frame to VCI/VPI field of ATM cell header
- translate FC field of FDDI frame to determine PT field of ATM cell header
- construct BI header for SOF cell
- construct BI header for COF cell(s) if needed
- construct BI header for EOF cell taking care of the useful user data. This is achieved by using the LI field of the BI header.

The transitions from and to the state FA3 are as follows.

FA (3 to 0)

- occurs when BI has sent all the ATM cells of a particular frame and is now available to receive a new frame.

FA (1 to 3)

- after a valid FC, DA fields have been received i.e. when there is no error in the header of the frame, this transition takes place.

Figure 2.8 State Diagram: ATM-to-FDDI data-flow

(b) ATM to FDDI Direction

The ATM to FDDI data flow state machine is shown in Figure 2.8. It has the following five states:

AF0: Await_Cells (wait for indication of a cell from ATM layer)

AF1: Decode_Cell_Type (decode the type of cell)

AF2: Receive_and_Analyze cells (copy cell payloads and analyze BI header)

AF3: Process_Cells (assemble a frame from cell payloads)

AF4: Transfer_Frame (send the frame to MAC layer)

The various states and the related transitions are described below.

AF0 (state 0): Cell_Allocate

The state represents the condition when BI is waiting for the indication of an ATM cell from the ATM layer. It ensures that buffer is available for the reception of frame segments from the ATM network.

Following are the transitions from and to the state AF0.

AF (0 to 1)

— occurs as soon as a cell is received from ATM layer, i.e. on the reception of ATM_UNITDATA.indication

AF (1 to 0)

— occurs when the SOF cell is not received from the ATM layer.

AF (4 to 0)

— occurs after the BI has transferred an assembled frame to MAC and is now available to receive cells from the ATM layer.

AF1 (state 1): Decode_Cell_Type

Represents the condition when the BI has just received a cell from the ATM layer and it tests to find if it is an SOF, COF, EOF or SCF cell.

The transitions to and from the state AF1 are as follows.

AF (1 to 0)

— occurs when the SOF cell is not received from the ATM layer.

AF (1 to 2)

— occurs after the cell-type has been decoded.

AF (0 to 1)

— occurs as soon as a cell is received from ATM layer, i.e. on the reception of ATM_UNITDATA.indication

AF (2 to 1)

— occurs as soon as BI receives an EOF cell. After this event, BI expects an SOF cell from the same destination.

AF2 (state 2): Receive_and_Analyze_Cells

Represents the condition when the BI has just received a cell from the ATM layer and it tests to find if a loss of a cell has occurred. Thus if the received cell is of type SOF, it is checked whether the EOF cell of the previous frame had been received or not. In case the received cell is COF, the BI tests to see whether the SOF cell had been received or not. These are done to ensure the proper receipt of a frame. If a loss is found, the frame is marked to be discarded.

The following transitions occur from and to the state AF2.

AF (2 to 1)

— occurs as soon as BI receives an EOF cell. After this event, BI expects an SOF cell from the same destination.

AF (2 to 3)

— occurs after all the cells of a particular frame have been received and put into a separate buffer by the BI.

AF (1 to 2)

— occurs after the cell-type has been decoded.

AF3 (state 3): Process_Cells

Represents the condition when the BI is ready to construct an FDDI frame from the ATM cell payloads. The steps needed to reassemble a frame are as follows.

- strip the BI header
- extract the payload, add descriptors and pointers to various frame segments so as to ensure their proper reassembly.
- take care of the partially filled payloads in the EOF cell payload
- mark the end of the frame (to handle partially filled cell payload)

Transitions to and from state AF3 are as follows.

AF (3 to 4)

— occurs after the frame segments pertaining to a particular frame have been assembled, i.e. various segments have been tagged with pointers and descriptors. BI then generates a signal MAC_UNITDATA.request in order to send a frame to the MAC layer.

AF (2 to 3)

— occurs after all the cells of a particular frame have been received and put into a separate buffer by the BI.

AF4 (state 4): Transfer_Frame

Represents the condition when the BI is ready to move a frame to the MAC layer.

AF (4 to 0)

- after the BI has transferred an assembled frame to MAC and is now available to receive cells from the ATM layer.

AF (3 to 4)

- occurs after the frame segments pertaining to a particular frame have been assembled, i.e. various segments have been tagged with pointers and descriptors. BI then generates a signal `MAC_UNITDATA.request` in order to send a frame to the MAC layer.

2.5.2 Processing Steps

The protocol processing for the two data flows consists of sets of processing steps. These steps are used to lay-out the protocol in a structured form in order to get a C language implementation. The programmed implementation of the protocol is used to assess the throughput of the protocol and is described in Chapter 3. Given below are the processing steps required for each type of data flow.

(a) FDDI to ATM

- (1) Preallocates a FIFO buffer to an expected incoming frame (FA0).
- (2) Detects the primitive `MAC_UNITDATA.indication`. The processing is initiated by an indication primitive sent by the MAC layer, i.e. the MAC Controller IC [16] (FA0).
- (3) Decodes frame control (FC) field. In this step, the one bit subfield --- "address-length" is decoded to determine the length of the destination address of the MAC frame. The information needed for the Payload-type field of the ATM cell header may also be obtained from the FC field (FA1).
- (4) Translates the destination address (DA) field of the MAC frame to the VCI/VPI field of the ATM cell header, using the look up tables (FA3).
- (5) Tests for a bad frame. If the values obtained from the translation of the FC and DA fields is found invalid, the frame is declared as a bad frame. The BI flushes the frame from the FIFO and initializes the start address of the FIFO (FA2).

(6) Prepares the informations for ATM and BI headers. Calculates the cell-count and the length-indicator fields of the BI header (FA3).

(7) Takes first 47 bytes of the MAC frame, appends the appropriate BI header and passes the payload to the ATM layer along with ATM header related information. Similarly, BI sends all other payloads to the ATM layer (FA3).

(b) ATM to FDDI

(1) Preallocates FIFO buffers for the incoming ATM payloads (AF0)

(2) Tests for the SOF cell. The bridge, when initialized, expects for an SOF cell in order to start the reassembly of a possible frame (AF1).

(3) Stores the cells in the FIFO buffer (AF2).

(4) Tests for end-of-frame cell. Arrival of an EOF cell causes the end of a frame. If a new SOF cell arrives instead of an EOF cell, the BI assumes that the frame is in error and discards it (AF2).

(5) The end of a frame is marked depending on the test carried out in step (4) (AF3).

(6) The LI field in the EOF cell is examined and the end-of-frame is marked accordingly (AF3).

(7) Transfers the frame to MAC layer (AF4).

Chapter 3

RISC Processor Implementation and Performance Assessment

3.1 Overview

The major function of the bridge is to transfer data from one network to the other. Thus, the performance goal of the bridge is to process the data transfer protocol, for each direction of data flow, fast enough to get the maximum possible throughput with least transit delay. Hence, for each direction of data flow, the desired throughput is 100 Mbps. Meeting these throughput values would ensure minimum queueing delay in either direction. The objective of this performance analysis is to assess the throughput capability of the bridge interface protocol and to determine the bottleneck of the protocol processing. The following sections describe the two approaches for the analysis of BI protocol execution characteristics: one approach reflects the implementation of the bridge interface on a uniprocessor; the other approach assumes a multiprocessor architecture which executes the protocol in a modular manner.

3.2 Method of Analysis

The ATM to FDDI direction data transfer requires the reassembly of various ATM payloads to construct a MAC frame at the receiving bridge. Similarly, for FDDI to ATM direction, segmentation of a MAC frame is done into various ATM payloads. For the analysis of these data transfer protocol functions, they have been implemented in a C language code. The code implements the state machines as described in Chapter 2. This code can be compiled to an assembly language code and the assembly level

instructions that are needed to implement the individual parts of the protocol can be identified. For example, if the number of instructions for state AF1 and AF2 in the ATM to FDDI direction data flow state machines can be identified then their equivalent execution times can be found and hence the processing time needed for each cell can be determined. Following a similar approach, the processing characteristics for different frame lengths can be obtained for the two directions of data transfers, and finally, the average throughput of the bridge for the two directions of data flow can be found. In order to carry out this analysis, either of the two processor architectures: RISC or CISC, can be used [15]. For the analysis carried out in this thesis a RISC architecture has been used, because it has a simpler assembly language construct as compared to the other. Another advantage of RISC architecture is that all the instructions except load and store, are single cycle instructions which makes the analysis much simpler.

The C code written here provides a close approximation to the protocol function execution characteristics, and may further be improved to get more precise results. The assembly code has been optimized using the optimization facility of the compiler. It has been an objective of the code implementation that the effects of the overhead caused by the operating system should be minimized. The copying of data from one memory to the other has been implemented as a separate assembly language routine which is linked to the main program.

3.3 Performance Model Description

3.3.1 Performance Model basis

The performance model is similar to the functional architecture shown in Figure 2.5. It comprises the following components:

- (1) Bridge interface (BI) processor with its local memory.
- (2) FDDI MAC layer protocol processor with its dual-port frame buffer memory (i.e. the SUPERNET chip set from AMD Inc. [16], described briefly in Appendix I)
- (3) ATM layer protocol processor with its local FIFOs.

The BI processor is responsible for the data transfers across the bridge. It has two separate independent modules which handle the respective data flows. The BI processor in the

analysis presented here is a SPARC processor. The local memory of the BI processor is needed for copying the data only for the ATM to FDDI data flow. The segmentation of FDDI frames is done in such a way that it does not need to be copied into the BI processor. The segments of data are moved directly from the MAC memory to ATM FIFOs after an appropriate BI header has been appended to each of them. The local memory of the BI processor copies frame segments received from the ATM network in order to get them reassembled. It is noted here that the BI processor has access to the FIFO structures of the MAC and the ATM layers. The RISC architecture uses single cpu cycle instructions except for load and store instructions. A static RAM of appropriate memory access time, i.e. an effective memory access time equivalent to a single cpu instruction cycle time would thus result in a two-cycle load instruction and three cycle store instruction for the SPARC based assembly code.

The queuing of MAC frames occurs at MAC local memory. Queuing of those coming from remote bridges occurs at the BI local memory. The ATM layer simply converts the incoming serial bit stream from the physical layer below it to either 8-bit or 32-bit parallel data, and uses only an elastic FIFO buffer.

The performance models for each of the two directions of the data transfer protocol are analyzed with the following approaches:

- (a) Sequential execution of the code for a single frame.
- (b) Modular execution of the code with pipelining among processing on different frames.

The details of these approaches for the two directions of data transfer are given below.

3.3.1.1 ATM to FDDI direction

For the ATM to FDDI direction of data flow, the model of frame processing with sequential execution of the code, has three major components:

1. The time for minimum essential processing on each frame; this component is needed for every frame, irrespective of the number of cells a frame might consist of.

2. The time needed for protocol processing related to a cell. This includes the copying time per cell.

In a sequential approach, a single state machine does the entire protocol processing on a frame. Processing on a frame can start only after its previous one has been processed and transferred to MAC layer. For a sequential approach, the total frame processing time is determined from the equations given in section 3.3.4.1(a).

The sequential approach however may not achieve the desired throughput required by the BI processor with the current state-of-art technology. A better performance can be achieved by exploiting parallelism and pipelining among different states of the data-flow state machine shown in Figure 2.8. The five states can be visualized as a five stage pipeline. The processing of different states is inherently independent. However, a state as a whole is dependent on the previous one as far as order of execution is concerned. Thus, the execution of a particular state can start only after the execution of the one previous to it has finished.

Figure 3.2 The effect of pipelining for ATM to FDDI data flow. The full exploitation of pipeline occurs when there are at least five frames being processed together, i.e. when the pipeline is full.

The pipelining and parallelism effect can be described as follows. It is assumed that there are five different processes, one for each state, which run concurrently. It could be said that for one particular frame all the five states must be processed in order but it is possible that a particular process can switch to another frame as soon as it has finished execution on its current one. For example, process AFD when finished with execution on frame n , can switch to frame $n+1$. This achieves parallelism as processing of frame $n+1$ starts before the processing of frame n is over. Considering an ideal scenario where all the five states take equal execution time, the throughput of the data flow can be boosted five times. The effect is depicted in Figure 3.2. However in practice, context switching and synchronization among various processes cause additional overhead and throughput will be less than the ideal value. The overall performance of this modular approach would be bounded by the slowest stage of the pipeline.

3.3.1.2 FDDI to ATM direction

The major components of the processing for this data flow are:

1. Processing of MAC frame header and preparation of BI header of individual ATM cell payloads.
2. Copying of cells from MAC memory to the ATM layer FIFOs.

For sequential approach, the components are executed in the order as if they were a single process. The throughput of the sequential approach is improved by exploiting parallelism and pipelining among the two components as described above. It is similar in concept to that of ATM to FDDI data flow described in section 3.3.1.1.

3.3.2 Performance Model Approximations

The protocol code written in C language has been written with maximum modularity in mind. Efforts have been made to minimize the effects of overhead related to the operating system. The program which copies data from one memory to the other, if written in a conventional way would involve memory routines of the operating system and would result in unnecessary overhead. For the analysis carried out in the thesis the copying program has been written in assembly language and then it has been linked to the main program. In the C-language code the context switching part has not been implemented.

The overhead caused by the context switching mechanism will be discussed later in the details of the VLSI architecture.

To determine the bottleneck, the worst case scenario has been considered in these analyses. Thus, the incoming traffic from the ATM network has been considered as a continuous stream of MAC frames coming in the form of interleaved ATM cells on a virtual circuit at 150 Mbps. At the FDDI transmit end, the largest possible token rotation time has been considered.

The code written here, has been analyzed only for its static behavior. Certain conditions like execution of a repetitive loop, can not be tested unless the code is used dynamically i.e. in a running condition. This might seem as a critical approximation but a careful look at the code suggests that in fact there are not many conditions of that type; the only loop repetition which is critical is the copying part. However, in doing the calculations for frame processing-time, the total copying time of cells is determined by multiplying the number of cells to copying time needed per cell; thus the resultant assembly language code contains a collection of all the assembly-instructions needed for individual cells. This way, the static code execution time becomes equivalent to that of a running code. Thus, it can be assumed that the code would produce similar results if executed in a dynamic manner. The C-code is given in Appendix II.

3.3.3 Performance Model Parameters

The performance of the bridge interface will be affected by a number of independent parameters that can be separated into two groups:

1. FDDI-MAC layer interface parameters.
2. ATM layer interface parameters.

The data transfer across the bridge passes both of these interfaces and therefore, they may be termed as two of the possible bottlenecks. Since the objective of this analysis is to set the desired performance characteristics of the VLSI architecture for the bridge interface, most of the parameters given below would be used with their respective worst case values.

3.3.3.1 MAC layer interface parameters

- clock rate at which the MAC layer circuitry works.

- memory access time of the MAC layer local frame buffer memory.
- data bus width of the frame buffer memory.
- number of ports in the memory.

3.3.3.2 ATM layer interface parameters

- clock rate at which the ATM layer circuitry works.
- memory access time of the ATM layer local cell buffer memory.
- data bus width of the cell buffer memory.
- number of ports in the memory.

3.3.4 Performance Model Equations

3.3.4.1 ATM to FDDI direction

(a) Without parallelism or pipelining

Frame processing related component

Essential processing per frame = Cell loss detection time + EOF cell processing time
+ Buffer management

Cell processing related component

Copying time per word = (Bus arbitration + address decoding time) + ATM layer memory access time + MAC layer memory access time

Cell copying time = 12 x (Copying time per word)

Cell processing time = Cell detection time + header decoding time + VCI table search time

Cell Processing related component = Cell copying time + Cell processing time

Throughput for sequential approach

Single frame processing time = (Frame processing related component) + N x (Cell processing related component), where N = number of cells in a frame

Throughput = $(47 \times 8 \times N) / \text{single frame processing time}$

(b) With parallelism and pipelining

The above approach is a sequential one where no parallelism or pipelining has been exploited. In fact with multiple FIFOs with their individual frames, it is possible to pipeline various states of the state machine. The approach is discussed in section 3.3.1.1.

The execution characteristics of this approach can be found from the equation given below. The same equation is used to analyze the characteristics of all the states.

The Processing time = (Total number of instructions) x (cpu cycle time) + (number of load and store instructions) x (memory access time)

3.3.4.2 FDDI to ATM direction

(a) Without parallelism and pipelining

Frame processing related component

Essential processing per frame (excluding cell processing component) = MAC-indication detection time + frame copying time into MAC memory (overlaps the DA-VCI processing) + Descriptor processing time

Cell processing related component

Copying time per word = (Bus arbitration + address decoding time) + ATM layer memory access time + MAC layer memory access time

Cell processing related component = 12 x (Processing time related to copying a word + copying time per word)

Throughput for sequential approach

Single frame processing time = (Frame processing related component) + N x (Cell processing related component), where N = number of cells in a frame

Throughput = $(47 \times 8 \times N) / \text{single frame processing time}$

(b) With parallelism and pipelining

(b) With parallelism and pipelining

The throughput for this approach is found by finding the slowest state and assuming that the pipeline is full. The execution characteristics of this approach can be found from the equation given below. The same equation is used to analyze the characteristics of all the states. The details of the timing characteristics for various states are shown in Table 3.1.

The Processing time = (Total number of instructions) x (cpu cycle time) + (number of load and store instructions) x (memory access time)

3.3.5 Performance Model Results

The results given below have been obtained from the optimized assembly language code on SPARC architecture on a Sun 4 machine.

3.3.5.1 ATM to FDDI direction

For simplicity, the two memory access times are assumed equal and of the order of the processor cycle time, and that the frames are multiple integrals of cell payloads. The calculations, using the execution-time values from the Table 3.1 give the following results.

(a) Without parallelism and pipelining

Frame processing related component

Cell Loss detection time (As shown by state AF2 in Table 3.1(c)) = 5480 ns

EOF cell processing time (As shown by state AF3 in Table 3.1(d)) = 4560 ns

Buffer Management time (partially in state AF4) = 1600 ns

Frame processing related component = $5480 + 4560 + 1600 = 11640$ ns

Cell processing related component

Cell detection time (As shown by state AF0 in Table 3.1(a)) = 3560 ns

Cell header decode time (As shown by state AF1 in Table 3.1(b)) = 3960 ns

Cell copying time = $12 \times [(40 + 40) + 17 \times 40]$ ns = 9120 ns, where access-times of the two memories are 40 ns each, and 17 additional cycle are needed for processing related to the copying (including alignment activities while doing reassembly) of each word. Each cell consists of 47 bytes of frame-segment; however for simplicity, the calculations are done for 12 words or 48 bytes. The macro implementing the cell copying is given in Appendix II.1.

Per cell search time (needed for VCI search) = $10 \times 20 \times 40$ ns = 4000 ns, it is assumed that the number of entries searched is 10 and the number of instructions needed to process a search is 20.

Cell processing related component time = $3560 + 3960 + 9120 + 4000 = 20640$ ns

Throughput for sequential approach

Throughput is calculated for effective data moved on per cell basis.

Throughput = $(47 \text{ bytes} \times 8) \times N / (11640 + 20640N)$, where N = number of cells.

The average value of the throughput comes out as 13 Mbps.

Using the final relationship, the throughput of the data transfer protocol for the ATM to FDDI direction can be calculated for different lengths of MAC frames. Out of the three

**Table 3.1 Protocol Function Execution Timing Characteristics:
ATM to FDDI**

(a) State AF0

Instruction Type	Execution Frequency	CPU cycle count	Total cycles	Total execution time (ns)
Load	10	2	20	800
Store	5	3	15	1600
Others	54	1	54	2160
Total Execution time		3560 ns (89 cycles)		

(b) State AF1

Instruction Type	Execution Frequency	CPU cycle count	Total cycles	Total execution time (ns)
Load	14	2	28	1120
Store	5	3	15	600
Others	56	1	56	2240
Total Execution time		3960 ns (99 cycles)		

(c) State AF2

Instruction Type	Execution Frequency	CPU cycle count	Total cycles	Total execution time (ns)
Load	19	2	38	1520
Store	10	3	30	1200
Others	69	1	69	2760
Total Execution time		5480 ns (137 cycles)		

(d) State AF3

Instruction Type	Execution Frequency	CPU cycle count	Total cycles	Total execution time (ns)
Load	16	2	32	1280
Store	7	3	21	840
Others	61	1	61	2440
Total Execution time		4560 ns (114 cycles)		

components, the effect of the frame processing component on the throughput is prominent only for smaller frame lengths of approximately 700 bytes, and for frame lengths larger

(e) State AF4

Instruction Type	Execution Frequency	CPU cycle count	Total cycles	Total execution time (ns)
Load	18	2	36	1440
Store	10	3	30	1200
Others	77	1	77	3080
Total Execution time		5720 ns (143 cycles)		

than that, the sum of the cell processing and the cell copying components dominate the processing per frame. For different lengths of MAC frames, the throughput values have been calculated and the throughput of the bridge for the ATM to FDDI direction is found to be 13 Mbps.

(b) With parallelism and pipelining

Using the modular approach, i.e. with the implementation of parallelism and pipelining, the throughput of the bridge can be improved. The state AF4 turns out to be the slowest state. It thus determines the effective throughput for this approach.

Processing time of state AF4 = $5920 + 9120N$, where N is the number of cells.

Throughput = $(47 \text{ bytes} \times 8) \times N / (5920 + 9120N)$

The average value of throughput is 38.5 Mbps.

3.3.5.2 FDDI to ATM direction**(a) Without parallelism and pipelining**

The calculations for the timings of frame processing related component are given below.

The related equations are described in section 3.3.4.2

Frame processing related component

The MAC-indication detection time (As shown by state FA0 in Table 3.3 (a)) = 1320 ns

Frame copying time into MAC memory = $12 \times (2 \times 160 \text{ ns}) \times N = 3840 \times N \text{ ns}$

where, N = number of cells, MAC memory access-time per word = $4 \times 40 \text{ ns}$
and each cell comprises of 47 bytes or 12 words (for simplicity of calculations).

Descriptor processing time = (As shown by state FA2 in Table 3.3(c)) = 2360 ns

Frame processing related component = $(1320 + 3840N + 2360)$ ns = $3680 + 3840N$

The cell processing related component

Total time for cell processing related component = $12 \times [(40 + 40) + 16 \times 40]$ ns = 8640 ns, where access-times of the two memories are 40 ns each, and 16 cycles are required for processing related to the copying of each word (including alignment activities). For simplicity of calculations, the number of words of a frame-segment (equivalent to 47 bytes) in a payload, are assumed to be 12. The macro implementing the cell copying is given in Appendix II.2.

Throughput for sequential approach

Throughput is calculated on the basis of per cell data moved by the bridge.

Throughput = $(47 \text{ bytes} \times 8) \times N / (3680 + 3840N + 8640N)$

The average value of throughput comes out to be an average of 23 Mbps.

**Table 3.2 Protocol Function Execution Timing Characteristics:
FDDI to ATM**

(a) State FA0

Instruction Type	Execution Frequency	CPU cycle count	Total cycles	Total execution time (ns)
Load	2	2	4	160
Store	3	3	9	360
Others	20	1	20	800
Total Execution time		1320 ns (33 cycles)		

(b) State FA1

Instruction Type	Execution Frequency	CPU cycle count	Total cycles	Total execution time (ns)
Load	11	2	22	880
Store	1	3	3	120
Others	48	1	48	1920
Total Execution time		2920 ns (73 cycles)		

(c) State FA2

Instruction Type	Execution Frequency	CPU cycle count	Total cycles	Total execution time (ns)
Load	3	2	6	240
Store	4	3	12	480
Others	41	1	41	1640
Total Execution time		2360 ns (59 cycles)		

(d) State FA3

Instruction Type	Execution Frequency	CPU cycle count	Total cycles	Total execution time (ns)
Load	13	2	26	1040
Store	4	3	12	480
Others	33	1	33	1320
Total Execution time		2840 ns (71 cycles)		

(b) With parallelism and pipelining

The throughput for parallelism approach is bounded by the slowest state which is state FA3. The processing of state FA3 essentially consists the cell copying component.

Processing time of state FA3 = $(2840 + 8640N)$ ns, where N = number of cells

Throughput = $(47 \text{ bytes} \times 8) \times N / (2840 + 8640N)$

The average value of throughput comes out as 45 Mbps.

The throughput obtained for the two data flows with parallelism approach needs more than one RISC processor. The timing characteristics of the state machines for the two directions, can be used to determine the number of processors needed to get the throughput as achieved by the parallelism approach. For example, for ATM to FDDI data flow, the slowest state is AF4; the other major time consuming state is the state A10 including the VCI search time. The rest of the states take only a fraction of the time needed by these two states. Thus, effectively, only two processors are needed to get the advantages of parallelism. Similarly for FDDI to ATM data flow, FA3 is the slowest state; FA0 along with frame copying time into MAC memory is the other most time consuming state. This shows a need of two processors in order to achieve the throughput obtained by parallelism approach, for this direction of data flow.

3.3.6 Performance Improvement Using Special Copying Hardware

Data movement from one memory to another in implementations requiring memory to memory data transfers is often done using a Direct Memory-Access (DMA) mechanism. The mechanism is efficient for relatively large blocks of data, and requires a very small number of instructions to do the operation. For the bridge interface being considered here, the data is copied from BI memory to FDDI-MAC memory for ATM to FDDI direction, and from MAC memory to the ATM FIFOs for FDDI to ATM direction. The data movement in both the cases however, requires the extraction and insertion of data bytes so as to take care of the one-byte BI header. For ATM to FDDI direction, the BI header of each ATM payload is removed, and a byte from the next 32-bit-word is extracted and added to the remaining 24-bits of the 32-bit-word which contained the BI header. Subsequent words of the payload are also aligned similarly. For the FDDI to ATM direction, a similar process incorporates the BI header into each 48-byte payload which contains 47 bytes of frame segment. Thus, in both the cases, data needs to be processed before it can be copied from one memory to another. This shows that a DMA operation is not possible in either case. However, in order to achieve an improvement in the throughput, special hardware could be designed, which would permit the alignment activities (as described above), for both data flows, to be carried out in least possible cycle counts.

It is assumed that the access times of both the memories are equal and equivalent to one clock cycle each; and that one clock cycle is needed per word for the processing carried out by the special hardware. This results in processing time for word-transfer (including access times of the two memories) equivalent to three clock-cycles and the copying time per cell turns out to be 1440 ns. (See below: item on cell copying related component).

3.3.6.1 ATM to FDDI direction using copying hardware

Following the calculations carried out in Section 3.3.5.1, the new value of throughput is calculated as below.

The Frame processing related component = 11640 ns

Cell copying time = $12 \times [(40 + 40) + 1 \times 40] = 1440$ ns, assuming the alignment activities need one clock cycle using a special hardware.

The Cell processing related component = $3560 + 3960 + 1440 + 4000 = 12960$ ns

The throughput = $(47 \times 8) \times N / (11640 + 12960N)$

The average value of the throughput is 36 Mbps.

3.3.6.2 FDDI to ATM direction using copying hardware

The Frame processing related component = $3680 + 3840N$

Cell processing related component = $12 \times [(40 + 40) + 1 \times 40] = 1440$ ns, assuming the alignment activities need one clock cycle using a special hardware.

The throughput = $(47 \times 8) \times N / (3680 + 5280N)$

The average value of throughput is 72 Mbps.

Thus, the use of a special hardware for copying (including alignment activities) improves the throughput to a value of 36 Mbps for the ATM to FDDI data flow and 72 Mbps for the FDDI to ATM data flow.

3.4 Performance Assessment Conclusions

Sequential processing is insufficient and can work only with a high processor clock rate which might be of the order of five to seven times the processor rate considered here of 25 MHz. This shows that a current general purpose processor, such as a RISC processor is inadequate or at best marginal, for an application such as bridge interface protocol implementation.

Using parallelism and pipelining, the needed improvement to get the desired throughput is of the order of three. It is therefore concluded that, a multiprocessor configuration is also inadequate for this implementation.

Copying and table search are two possible bottlenecks. In the next chapter, an architecture for a VLSI implementation of the BI protocol functions is discussed.

Chapter 4

Architecture for VLSI Implementation and its Performance Analysis

The high processing speed needed for the bridge interface can be achieved by using a VLSI approach to the bridge protocol processing. In this chapter, the factors related to the VLSI design are discussed first, followed by the approach and the assumptions for the design. The buffer management in the bridge is discussed in section 4.3. The architecture has been described in two parts: (1) ATM to FDDI direction and (2) FDDI to ATM direction. The timing characteristics and the state machines for both the directions are discussed in the respective sections of the two data-flows followed by a performance analysis.

4.1 Factors Related to VLSI Architecture

4.1.1 Need of a VLSI architecture for the bridge processor

The usual approach to a computer architecture design is to use a single processor first and then go for a multiprocessor based configuration if a single processor is found to be insufficient to carry-out the desired processing. In an application such as a bridge interface processor, a general purpose processor is not a proper choice from the point of view of the performance and utilization. A VLSI architecture is a better option for the purpose. Moreover, a multiprocessor based architecture with a design which properly utilizes the resources, can also be a reasonable choice. Furthermore, the architecture of a processor depends on the application it is intended to handle. A general purpose processor usually is suitable for applications which require intensive instruction and data processing. The bridge however, has a task of a different nature; it does more copying than protocol function execution and a general purpose processor will not be able to meet

the desired performance and utilization. This emphasizes the need of a specialized VLSI approach for the implementation of the BI-protocol.

4.1.2 Factors related to VLSI design

- Data paths within a processor are an important component of the processor organization [17]. The lesser the number of data paths, the smaller is the VLSI design area. However, the number of data paths directly effect the number of clock cycles needed for a particular operation. For example, a multiplication of two operands in a register bank will take one clock cycle if the register bank has two read and one write data paths connecting to the ALU; however it requires a minimum of three clock cycle if there is only one data path connecting the register bank to the ALU.
- Size of the register bank or the on chip static RAM also effects the price and performance of a VLSI design [17]. Larger memory on a chip increases its area and hence the cost, but helps enhance the performance by providing lower times for the data read and write. However, these elements can be optimized to get a reasonable price-performance ratio.

4.2 Design Approach to the Architecture

The objective of the design is to process and move the data from one direction to another for the two data flows, fast enough to match the high speeds of the two networks. In the previous chapter it has been shown that among the protocol functions, it is essentially the copying of data which slows down the overall data movement. It is therefore needed that the effect of copying be minimal so as to get the maximum possible throughput. This is achieved by using a dual port memory architecture. The details of the dual port memory are given in section 4.3.

The major components of a general purpose processor are the arithmetic logic unit (ALU), the control unit (CU), the register set and the data paths. The ALU and other similar units such as address-generation-unit are also termed as functional units. In the architecture proposed here, in order to get a minimum protocol execution overhead, several independent functional units have been used. These units are utilized to get parallelism and pipelining effects in protocol processing. They achieve a benefit over central processor unit which does the processing in a sequential manner. By using separate functional units, contention of resources such as a single register file with limited number

of registers, can be avoided even while the units continue to execute in parallel. The operations related to multiple functional units are discussed in more detail in Appendices II and III. The use of multiple functional units however tends to increase the number of data paths and the cost of the design. An attempt has therefore been made to optimize the number of data paths.

The address-mapping tables and context variables needed to maintain the transmit and receive queues for the two data flows should reside in high-speed local static RAM of the VLSI architecture so as to permit a minimum execution time. A minimum on chip RAM requirement is accomplished by providing storage to only the essential data; remaining data is stored in the dual port memory.

4.3 Buffer Management

The FDDI to ATM direction of data flow in the bridge does not need additional buffering provided the incoming data is processed fast enough to get a 100 Mbps throughput. The data received from the FDDI ring media can be processed while it is in the MAC memory and forwarded to the ATM layer in form of payloads consisting of segments of MAC frames, and hence does not need any additional copying.

For the ATM to FDDI direction, the ATM payloads from the ATM layer FIFOs are received into a buffer for re-assembly into MAC frames. However, the high speed of the incoming data, the interleaving of various MAC frames among ATM cells and the variable size of MAC frames make the buffer management complex.

Multi-port memory architectures provide a solution to these kind of high performance applications and allow overlapped data movement such that the new data transfers into the system even while the old data moves out. The buffer management strategy is described in the following sections.

4.3.1 Memory Architecture

The buffer in the bridge which receives the data from the ATM layer, requires accesses from three locations: the ATM layer FIFOs, the MAC layer frame buffer and the bridge interface processor. The two possible memory architectures are a triple port memory with two serial access ports and one random access port [18], and a dual port video random access memory (VRAM) [9,19]. The need for a special memory architecture

SAM = Serial Access Memory

(Note - A VRAM architecture can be obtained by removing the dotted portion, i.e. one of the two serial ports of the triple port memory)

Figure 4.1 Triple port memory architecture for 1 Megabit module

for high-speed communication processing has been emphasized by Zitterbart [20] and Dang et. al [21].

The architecture for a triple port memory is shown in Figure 4.1. The diagram represents a 1 Mbit module. A combination of eight similar modules gives a 32 bit wide memory of 1 Mword. There are 512 storing elements of 512 x 4bits each. The 4-bit data entering from the serial port first goes to a shift register of length 512. The row and column addresses supplied by the external processor result in a simultaneous transfer of 512 x 4 bits from the shift registers to the memory array. As a result the number of accesses

to the memory array is reduced and the saving can be utilized by the DRAM port. The architecture of the dual port VRAMs is similar to that as shown in Figure 4.1, with the only difference that it has only one serial access port. Thus, by removing the block of the triple memory, as shown in dotted lines, a VRAM architecture can be obtained.

The dual port VRAMs don't permit a full duplex data movement on the serial port like the triple port memory but they provide bidirectional data movement. The VRAMs require less control circuitry and are less expensive than the triple port memory. The current 40 ns access time VRAMs with 32 bit can attain a data transfer rate of 800 Mbps [19], fast enough for the ATM to FDDI data flow across the bridge. The serial access ports in VRAMs and triple port memories are bidirectional, thus a VRAM serial port may provide 400 Mbps data transfer rate to each direction of the data flow through the memory buffers: the data movements from the ATM layer to the memory and from the memory to the MAC layer buffer. The random access port of the VRAMs with 200 ns access time gives 160 Mbps transfer rate. interleaved memory architecture. The current

MAR = Memory Address Register

MBR = Memory Buffer Register

Figure 4.2 Two bank Interleaved Memory

triple port memories on the other hand have 25 ns access time for the two serial ports and 80 ns for the random access port [18].

Furthermore, the fact that a MAC frame can not be transferred unless it has been received completely by the bridge interface, indicates that the bridge does not need an overlapped data transfer over the serial ports. A triple port memory will thus remain under-utilized and thus a dual port memory would be a better option for the bridge.

A further improvement in the transfer rate can be achieved by using an interleaved memory architecture. The memory can be divided into two banks of equal size: one with even addresses and the other with odd addresses. The basic advantage in the case of the interleaved memory comes from the fact that the data can be transferred from one bank while the other is being refreshed. This will reduce the average latency by almost the amount of recovery or the refreshing time of the memory. The architecture is shown in Figure 4.2.

4.3.2 Factors in Buffer Management

The buffer management should take care of following factors:

- (1) Proper utilization of the memory. MAC frames of variable length may cause bad utilization of the memory.
- (2) Proper ordering of the received frames from different sources so as to render a FIFO service.
- (3) Efficient and simple address mechanism.

4.3.3 The Buffer Structure

This section presents a structure which accommodates first two of the above factors. The buffer memory receives data from the ATM network which includes more than one MAC frame being received in form of interleaved ATM cells. The first job of the buffer management is to separate all the cells of individual frames and arrange them so as to get their re-assembly properly and efficiently. In the bridge, for the ATM to FDDI data flow, the destination for all the incoming frames is just one ring; this means that separation of arriving frames into individual FIFOs is not necessary. Moreover, a memory structured as a single large FIFO simplifies the service mechanism and avoids the overhead of processing needed for FIFO separation. However, this needs a mechanism to allocate a buffer equivalent to the length of an incoming frame.

Figure 4.3 Buffer Organization

In the currently available FDDI-MAC hardware [16], the MAC layer protocol processor provides a descriptor field at the beginning of each of the frames received from the ring media. The descriptor field is written immediately after the receipt of the last byte of the frame in the MAC layer local buffer. This descriptor field consists of the length of the received MAC frame in bytes.

The sender bridge converts this length in bytes into an equivalent number of cells, as part of the protocol described in chapter 2. The 6 bits of the BI header of the SOF cell carry this cell count information, which can be decoded at the receiver bridge as soon as the SOF cell is received, and a buffer of equivalent size can be allocated to that frame. The cell count for a maximum FDDI frame length of 4500 octets is 96 which can not be accommodated in the available 6 bits of the BI header. Consequently it is proposed that the cell count be used with a granularity of two cells, which can be accommodated in the 6-bit field. Therefore, the receiver bridge can allocate a buffer to the incoming frame in a multiple of 2 cell payloads or 96 bytes.

The buffer organization is shown in Figure 4.3. The FIFO is a wrap around one, and as soon as it reaches the end, it starts all over again from the beginning. The pointers FSA and FEA represent the FIFO start and end addresses respectively. There are two other global pointers, the read pointer RPTR and the write pointer WPTR. The pointer RPTR keeps the start address of the next frame which is to be transferred to the MAC layer

memory, and WPTR keeps the address of the location where the next newly arrived frame from a remote bridge source is to be stored. Along with the global pointers described above, there are pointers individual to each of the frame being received from the remote bridges. These are called a frame-segment-receive-pointer (FSRP). An FSRP individual to a frame, keeps the address of the location where the next frame segment (in form of an ATM payload) is to be stored. The write pointer (WPTR) always points to the address where a next expected frame (i.e. a new SOF-cell) can be stored. It is updated every time a new SOF cell arrives (Conceptually, it is similar to a Program Counter being updated on every instruction fetch). The pointer WPTR therefore must be updated

Figure 4.4 Flow Chart: Buffer Management

VCI	Activity bit	FSR Ptr	Alignment bit	SOF	EOF
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.

Figure 4.5 The VCI-Activity table

within a time period less than the reception time of a cell. These pointers are part of the address generator described in section 4.4.1.

The flow chart in Figure 4.4 shows the operations involved in the buffer management. It uses the VCI—Activity table shown in Figure 4.5. The table consists of the VCI values corresponding to each of the remote bridges; an activity bit indicating whether a particular VCI has an incomplete frame with outstanding cells to its credit in process of being received. This activity-bit is set immediately on the receipt of the SOF-cell and reset after the receipt of the EOF-cell. This activity information is used to dynamically adjust the VCI-Activity table so as to achieve a minimum table search time.

Every time a cell arrives from the ATM layer, the VCI of the cell is compared against the activity bit in the table. An active VCI indicates that the cell payload is to be stored at a location shown by the individual FSR pointer (Frame Segment Receive pointer) associated with that VCI. Also shown in the flow chart is a possible error condition of the loss of an EOF cell. The VCI-Activity table is searched and if a match of the VCI occurs, the FSR pointer of the individual VCI is used to indicate to the address where the cell is to be stored; otherwise the cell is stored at the address as indicated by the WPTR. The above mechanism allocates a buffer amount to an incoming frame as soon as its SOF cell is received. If the next cell is an SOF cell of a new frame, it is sent to be stored in the memory at a location offset by the cell count to the SOF cell start

address of the previous frame. This permits an orderly frame reception and allows the frames to be serviced in the order in which they arrived, provided the FIFO is served from the top of the queue to the bottom.

The buffer management strategy described above, may not give an extremely efficient memory utilization, but it greatly simplifies the service mechanism and FIFO handling for the arriving frames and hence reduces the required processing time.

4.3.4 Addressing Mechanism

The address mechanism is shown in Figure 4.11. The function of address generator is to provide addresses in the dual port memory for reading and writing of data frames. For reading from the memory, RPTR is used. For writing into memory, the write address is chosen from one of the two possibilities — WPTR or FSRP. The address selector decides using the Table 4.1 that, which of the two should be sent to the dual port memory as the store address of the frame-segment just received. As obvious from the Table 4.1, whenever an SOF cell is received, the value of current WPTR is used as cell-store-address and after this value is read it is overwritten by the new value of the WPTR. The adder is used to update the WPTR value. The size of the adder has been shown for a 256 Kword buffer memory with 16 bit address length and needs to be selected appropriately according to the buffer requirement of the bridge.

Table 4.1 Address Selector Operation

VCI match (Y/N)	SOI (Y/N)	Action on the current cell
N	N	Discard the cell
N	Y	Store the cell at WPTR
Y	N	Store the cell at the individual FSR pointer.
Y	Y	Discard the previous frame (EOF cell loss) Store the cell at WPTR (start a new frame)

4.3.5 Service Mechanism

The frames received from the remote bridges are served from the top to the bottom of the FIFO. The pointer RPTR ensures a first-in-first out order.

4.3.6 Alternative Mechanisms for Buffer Management

Two alternative mechanisms were considered for buffer management. They are described in the following sections.

4.3.6.1 Alternative Mechanism — Approach 1

In the first approach as shown in Figure 4.6, the memory is considered as a large FIFO consisting of equal size blocks of the size equivalent to the maximum FDDI frame length. One whole block is given to each incoming frame in the order of their arrival. The FIFO requires two global pointers; one for the address of the next expected frame to be received and another for the next frame to be transferred to the MAC layer. The frame requires an individual pointer to keep the address of the location, where the next frame segment

Maximum FDDI Frame length = 4.5 Kbyte
 n = number of frames received without wrap around of the FIFO.
 VCI = Virtual Circuit Identifier

Figure 4.6 Alternative Buffer Management Approach 1.

is to be stored. The incoming ATM cells are sorted according to their respective VCI values using the Frame ID table shown in Figure 4.6(b), and then the payload is stored at the address shown by the individual pointer of that frame. Once the FIFO is full, wrap around occurs. This may be described as a brute force approach, but it greatly simplifies the memory address mechanism. Simplicity and fast addressing mechanism are the main advantages of this approach although a similar effect is achieved by the approach as described in section 4.2.3, by pre-calculating the expected frame start address. This approach does not require the source bridge to pass the frame length information to the destination bridge. The disadvantage of this approach is a poor utilization of memory. Moreover, this approach needs a very simple mechanism to decide as to how the frames are to be served, since the service can be provided in simple FIFO manner by serving from the top of the queue to the bottom.

4.3.6.2 Alternative Mechanism — Approach II

The second approach is to use separate FIFOs for individual remote bridge sources. Selecting an improper size for a FIFO will however not provide a good memory utilization. A better way for proper memory utilization is to divide the memory into equal fixed size blocks, large enough to hold more than one FDDI frames of maximum length. In case, a block gets full, a chain of these buffer blocks for each of the FIFO can be formed. The mechanism for chain of buffer blocks for a FIFO is described later in this section.

The buffer organization is shown in Figure 4.7(a). The memory is considered as partitioned into fixed size buffers, large enough to accommodate N number of maximum length MAC frames. The number N can be selected on the basis of an approximate analysis of the traffic characteristics and the service rate provided by the bridge interface. At a particular instant of time, a FIFO might be receiving a new frame while a previously received frame is being transmitted to the MAC layer. Individual pointers are required for each of the FIFO for receive as well as transmit direction. Thus, in certain cases, when one memory block is not sufficient to hold the incoming data, a chain of buffers as shown in Figure 4.7(b) can be formed. This is achieved by using descriptors at the beginning of each of the memory block and a pointer at the end of it. The descriptor consists of a flag, which shows whether it is the last block of the chain. The pointer at the end of a block contains the address of the next memory block of the chain. Each time a new block of memory is filled, the descriptor-bit and the pointer field in the previous

(a) Buffer Organization

(b) Chained Buffers in a FIFO

Maximum FDDI Frame Length = 4.5 Kbyte

n = Number of buffer blocks

N = Number of maximum length FDDI frames in a buffer block

Figure 4.7 Alternative Buffer Management Approach 2

block are modified.

The second approach provides a better memory utilization than the first one, but needs higher number of pointers. As compared to the approach described in section 4.2.3, it needs more pointers and needs a mechanism for service, such as time-stamping or circular order of servicing each FIFO.

4.4 Architecture for VLSI Implementation: ATM to FDDI

In the analysis of the architecture, it is assumed that certain arithmetic and logical operations such as comparison of two words, shifting of a word and other similar operations are carried out in one clock cycle. This is a reasonable assumption considering that in most of the RISC processors, all the instructions are single cycle except those which are memory intensive such as load and store [17]. The clock cycle has been assumed as 40 ns and all the calculations are based on this value. This is in accordance with the analysis of the assembly language code as carried out in chapter 3.

The length of the FDDI MAC frame varies usually from 128 bytes to 4500 bytes although certain frames may be as small as or even less than an ATM payload length. A sender bridge will normally require to send a combination of SOF, COF and EOF cells in a way similar to as shown in Figure 4.8. In order to analyze the performance of the VLSI architecture for the ATM to FDDI direction of data flow, it is necessary that the processing characteristics of each of the four types of the cells be examined individually and then the bottleneck be determined.

4.4.1 Components of VLSI Architecture

The block diagram of the VLSI architecture is shown in Figure 4.9. The details of various functional units are given below followed by the details of the operations.

(a) An FDDI Frame may consist of zero or more COF cells

(b) An FDDI Frame may consist of only one cell

Figure 4.8. Types of FDDI frames at the sender bridge

Figure 4.9 Block Diagram: ATM to FDDI direction VLSI Architecture

- **Activity Detector** receives interrupts from the external elements, such as the ATM and the MAC layers and informs the Timing & Control unit correspondingly. The ATM_UNITDATA.indication from the ATM layer indicates the arrival of a cell. The other signal, the buffer full indication from the MAC layer, prohibits any further transfers to the MAC layer until the buffer regains a space equivalent to one maximum length MAC frame.

- **Barrel Shifter** is the unit which extracts the VCI and BI header from the incoming ATM cell. It is also used to copy the last byte of a payload to the corresponding entry in the VCI-Activity table. This byte is needed for the purpose of alignment of the successive frame segments which are received in the form of ATM payload. The barrel shifter uses a pass transistor, which has a property of allowing the signal to be passed in either direction of its output [17]. This permits a simultaneous movement of data words to the bidirectional latch and the multiplexor, from the barrel shifter, as shown in Figure 4.9.

The synchronization needed to extract a certain byte or a word is achieved with the help of the Timing & Control unit. This allows the VCI to be sent to the VCI Sorter unit, the BI header to be sent to the BI Decode unit and the last word of the payload to be sent to the VCI-Activity table. The multiplexor determines which word is sent to the decoders and to the VCI Sorter unit. In a way, its function is similar to that of a crossbar-switch.

- **Bi-directional Latch** allows a proper synchronization of the incoming and the outgoing data over the serial port of the memory. The maximum possible data rate on the serial port is divided equally among the ATM and the MAC layer data movements. The synchronization of the 32-bit data latch is done by the Timing & Control unit.

- **BI Header Decoder** consists of three units: Cell Type, Cell Count and Length Indicator decode units. As obvious from their names, the units decode cell type, find cell count and determine number of useful bytes in the EOF cell respectively.

All the three units get a simultaneous input of 8 bit from the barrel shifter via a multiplexer. The multiplexer ensures that the VCI value received from the barrel shifter is sent to the VCI Sorter, and the BI header value is sent to the header decoder units. The CT decode unit uses only the first two bits where as the CC and the LI decode units use the last 6 bits. The decoded information is utilized by other functional units depending on the type of cell. T & C unit manages the required coordination between the BI Decoder and other functional units.

- **VCI Sorter unit** has a comparator as shown in Figure 4.10, and a VCI-Activity table shown in Figure 4.5. The VCI Activity table is discussed later in this section. The

Figure 4.10 The VCI Sorter unit

table consists of entries which show all the possible VCIs and their corresponding frame segment receive (FSR) Pointers. A one bit entry against each VCI shows whether the VCI is in process of receiving a frame, i.e. whether it is currently active or not. On the receipt of a cell, the VCI sorter compares the VCI of the cell to those in the VCI Activity table. This activity is carried out in parallel with the decoding of the BI header. At the end of the two activities, the sorter unit decides the segment store address (SSA) where the currently received frame segment can be stored in the memory. The address generator is used to accomplish the task of determining the SSA value. It also determines as to how the VCI Activity table should be updated. For example, The SOF cell is stored at the address shown by WPTR (discussed in section 4.3), and the activity bit is set in the table. The COF and EOF cells are stored at the addresses corresponding to the FSR Pointer. The activity bit remains unchanged for COF cell and is reset on the arrival of an EOF cell. The Alignment Data Register (ADR) is used for temporarily storing the alignment byte. This is needed as part of the reassembly procedures to carry out the alignment of 47 byte frame segments.

- **Address Generator** provides the addresses of the dual port memory locations, for the storage of the incoming frame segments from the remote sources as well as to the MAC frames which are transferred to the MAC memory from the bridge memory. Figure 4.11 shows the basic blocks of the Address Generator. It determines the segment-store-address

(SSA) for writing MAC frame-segments into the bridge memory being received from the remote bridge. It also generates the addresses for reading a frame out of the bridge memory. For writing into the memory, the address generator outputs the segment-store-address by selecting one of the two possibilities which are presented to it from other functional units in form of two registers. One of the write register contains the address value provided by the FSR Pointer, the other contains the address value provided by the WPTR. The selection is made depending on the cell-type value. For reading from the bridge memory, the read address is provided by the Frame Transfer unit (FTU) into the Frame Read Address register. It is used to access the start of a frame in order to transfer it to MAC memory.

Figure 4.11 Address Generator

- **Dual port memory** acts as a buffer for incoming frame segments and is used for temporary storage during the reassembly of a frame. The frames are then transferred to the MAC layer and while in process, may undergo a queuing delay within the dual port memory. This happens in certain conditions when the FDDI ring medium is heavily loaded and the MAC local buffer is full.

The size of the memory can not be predicted unless an appropriate analysis of the traffic characteristics are available. For simplicity, it is assumed to be of 64Kword (256 Kbytes) with a 32 bit word.

- **Timing & Control unit** coordinates among various functional units. It is initialized by the Activity Detector which sends a signal on the receipt of an ATM_UNITDATA.indication. It also provides locking of certain functional units so as to avoid contention among them. The sequence of operations in the case of various types of cells are fixed and are carried out depending on the values of the SOF and EOF fields as provided by the CC Decode unit.

The Timing & Control unit is a hardwired one.

- **Cell Loss Detector unit** looks for the loss of SOF and EOF cells of a frame. On receipt of every COF and EOF cells of a particular frame, it tests if the SOF cell has already been received or not. It also does a test to detect the EOF cell loss, every time an SOF cell is received from the ATM layer. A loss leads to the discarding of the particular frame.
- **Frame Transfer unit** may also be termed as an intelligent DMA device. It has a transmit queue table (FTQ table) which serves as a queue of the frames which are ready to be transferred to MAC layer. These are the frames which have been processed after being received from a remote bridge.

The FTQ table is served by three pointers: one for writing into (FTQW Ptr), another for reading from the table (FTQR Ptr), and the third one to indicate the frame which is pending any processing (FTQP Ptr). The Frame Transfer unit (FTU) is shown in Figure 4.12. It has two registers which are loaded with the frame start and end addresses in order to transfer a frame from the bridge memory to the MAC memory. The SA Decrementer assists the FTU to carry out the adjustment needed while accessing the first word of the frame in order to transfer it from the BI memory to the MAC memory. The frame segment length is 47 bytes, thus it needs the adjustment for the BI header which has been removed from the 48 byte ATM payload. When the FTU needs to access a frame in order to transfer it to MAC layer, the SA Decrementer moves-back, the start address of the frame to be accessed, by one byte. This eliminates the BI header from being read along with the frame data. The adder unit is used to calculate the appropriate end-address value of the frame in memory by adding the length-indicator (LI) value to the EOF-cell-start-address.

Figure 4.12 The Frame Transfer unit (FTU)

- **VCI Activity table** is used for identifying a cell on the basis of its source. It also helps in determining the store addresses of the intermediate cells and the EOF cell. The table, as shown in Figure 4.5, has following elements:
 - VCI field keeps the VCI values of all the remote bridge sources.
 - Activity bit shows whether a VCI is currently in process of receiving a frame. It is set immediately on the arrival of an SOF cell and reset as soon as an EOF cell arrives; remains unchanged for COF cell.
 - FSR Pointer contains the address of the location where the next segment of the frame currently being received on the corresponding VCI, is to be stored.

- Alignment byte field contains the last byte of the current payload. It is used for the alignment of successive frame segments.
- SOF bit field is set to indicate the receipt of SOF cell for a frame. It is used to test the loss of an SOF cell.
- **Frame Transmit Queue (FTQ) table** is used as a transmit queue. It also serves to determine the loss of the EOF cell of a frame. The table, as shown Figure 4.12 (b), has the following entries:
 - VCI is used as frame identification and helps to determine the EOF loss of a frame.
 - SA contains the start address of the frame received on the corresponding VCI. It is loaded into a register of the FTU when a frame is to be transferred to MAC memory.
 - EA contains the end address of the frame received on the corresponding VCI. It is loaded in the FTU along with the value of SA when a frame transfer occurs.
 - RX is set if all the processing of a frame is over. This indicates that a frame is ready for transfer and is used by the FTU whenever it wishes to transfer a MAC frame.
 - TX bit is set after the transfer of the frame. It is checked by the FTU before the transfer of a MAC frame. It also helps the queue management after the wrap around of the table takes place.
 - EOF bit is set to indicate the receipt of the EOF cell for a frame. It is also used to test for EOF cell-loss.

4.4.2 Event Sequences and Timing

The operations involved in the ATM to FDDI direction data flow with respect to the architecture shown in Figure 4.9, are given below. The minimum inter-arrival time of the ATM cells on a virtual circuit (at 155 Mbps) is 2730 ns as shown in Figure 4.13. This indicates that the bridge processor should carry out the protocol functions for all the four types of cells (SOF, COF, EOF and SCF) fast enough to be available to attend each of the incoming cell in a real-time manner. The primary aim of the processor is therefore to minimize the cell processing time. The processing of various cells have been analyzed in Appendix III.

The sequence of events for a cell receipt is shown in Figure 4.14. The event sequence starts with the cell arrival indication from the ATM layer. The first thing the BI needs

to do is to sort the cell on the basis of its VCI.

The VCI is received into the VCI Sorter and the VCI Activity table search is carried out. While the VCI Activity table search continues, the BI is sent to the decoder units and the

Figure 4.13 Minimum inter-arrival time of ATM cells

Figure 4.14 Event sequence for cell processing

header decoding takes place in parallel. Once these two activities are over, the SSA, i.e. the address where the cell-payload should be stored in the memory, is determined. All these activities are done in parallel to the reception of the cell into the serial port of the dual port memory. This is shown in the diagram at the bottom. The figure also shows the events which can be executed in parallel and those which are sequential in nature. The events shown in thick lines are essential for each of the cell, and those shown in thin outline are specifically needed for selected cells. The individual events are explained in more details later in Appendix III.

4.4.3 Summary: Protocol Function Execution Timings

Summary of execution timings of the protocol functions for the VLSI design is given in Table 4.2. The table contains a list of protocol functions and their execution times for the VLSI design, in numbers of clock cycles; the corresponding clock cycles for the SPARC implementation of the BI protocol are also listed to give a comparison of the two implementations. The numbers shown against the clock cycles of the VLSI

Table 4.2. Execution timings for protocol functions: ATM to FDDI

Type of Operation	Number of clock cycles	
	for VLSI architecture	on SPARC
VCI processing	6 + T	42 per entry
BI header processing	3	47
Determining segment-store-address	2	13
Updating FTU	4	4 ⁹
Cell-loss testing	2	55
Alignment	5	40
Throughput (Mbps)	100	38.5

T = Table Search time : VCI-Activity table + FTQ table

implementation, indicate the effective number of clock cycles and for some of the functions, the execution cycles that overlap with other functions have been excluded.

The throughput and delay analyses are presented in the following sections.

4.4.4 Throughput Analysis

Table 4.2 shows that the VLSI architecture outperforms the SPARC implementation. As mentioned in the previous section, the table search time is a major factor in determining the throughput of the data flow. To achieve a throughput limited by data copying operations, it should be limited to such a value that the protocol processing time on any cell does not exceed the copying time of a cell. Thus, effectively, it is the per word movement from BI memory to MAC memory that determines the throughput for the data flow. The throughput of the VLSI architecture, based on the copying operations, is calculated below. Figure 4.15 shows various components of data movement from ATM to FDDI network.

Figure 4.15 Components of delay across the bridge for ATM to FDDI Direction

Throughput Parameters

MAC memory access time per 32 bit word, $T_3 = 160$ ns

Dual port memory access time per 32 bit word, $T_5 = 40$ ns

Delay added to the dual port memory-access-time, in order to synchronize it to the MAC memory access time, $T_6 = 3 \times 40 = 120$ ns

Throughput calculation

Maximum possible ATM to FDDI data flow throughput = $32/(4 \times 40 + 160)$
 $= 100$ Mbps

From the throughput analysis, it is concluded that the table search may be the new bottleneck. The maximum possible search time for the VCI-Activity and FTQ tables put

together is determined by subtracting the maximum possible protocol-processing-time on any cell from the minimum cell-inter-arrival time. The value turns out to be 2090 ns.

Although the VLSI architecture meets the required throughput of 190 Mbps, a further improvement in throughput is possibly achievable, provided a faster MAC memory is employed at the MAC controller (i.e. at the SUPERNET chipset implementation). As shown in the calculations above, three idle cycles of 40 ns are added to the bridge memory access time to get timing synchronization with the MAC memory read cycle of 160 ns. The MAC memory has an access time of 80 ns and thus it can permit a maximum data transfer rate of 200 Mbps for reading and writing individually. However, normally this bandwidth is shared equally by reading and writing. Furthermore, while moving data to and from the ring medium, the data transfer rate is restricted by the network speed and not the memory speed. However, if a 40 ns access time MAC memory were used at the bridge interface, a 100% enhancement of the throughput value could be achieved for this data-flow. In this case, a 40 ns idle cycle needs to be added to the bridge memory access time. This however, may not improve the delay characteristics very significantly, but helps in minimizing the receive buffer allocation in the dual port memory since the data can be moved out of it at a much faster rate. Moreover it can be concluded that the architecture proposed here can support network interfacing at even higher speeds.

4.4.5 Delay Analysis

Various components of the processing across the bridge are shown in Figure 4.15. These are used to determine the component of the end-to-end delay, caused by the bridge. The delay calculations given below are with respect to each 32 bit word transfer from the ATM network to the FDDI ring. The three delay components are: cell reception to the BI memory, transfer of data from BI memory to MAC memory and the transmission of data from MAC memory to FDDI ring. It is assumed for these calculations that enough buffers are available at the MAC memory to receive data from the BI memory and hence there is no queuing delay involved in the data transfers from the latter to the former. The delays incurred for the three components are calculated below.

Delay Parameters

ATM network physical layer transmission time, $T_1 = 32 \text{ bit}/155 \text{ Mbps} = 100 \text{ ns}$ (approximately)

FDDI physical layer reception time, $T_2 = 32 \text{ bit}/100 \text{ Mbps} = 320 \text{ ns}$

MAC memory access time, $T3 = 160$ ns

ATM FIFO movement time = $T4 = 60$ ns

Dual port bridge memory access time, $T5 = 40$ ns

Delay added to the dual port memory-access-time, in order to synchronize it to the MAC memory access time, $T6 = 3 \times 40 = 120$ ns

Calculations of delay components

Delay for cell reception component = $T1 + T4 + T5 = 200$ ns

Delay for component-transfer from BI to MAC memory = $T3 + T5 + T6 + \text{Queueing Delay} = 320 + \text{Queueing delay}$, where Queueing delay is the access delay of the FDDI MAC protocol.

Delay for component-transmission of data to FDDI ring = $T2 = 320$ ns

Total delay across the bridge per 32 bit word for ATM to FDDI data flow = 840 ns + Queueing delay.

4.5 Architecture for VLSI Implementation: FDDI to ATM

4.5.1 Components of VLSI Architecture

The block diagram of the VLSI architecture is shown in Figure 4.16. The details of various functional units are given below followed by the details of the operations.

- **Cell Count (CC) Decode unit** gets the descriptor field from the MAC memory for the frame pointed by the Cell Transmit Queue Process Pointer (CTQPP) of CTQ table. The descriptor is also copied into the length field of the CTQ table, in parallel. The CC decode unit converts the frame length in bytes to an equivalent number of cell payloads. The cell count is then adjusted to the nearest multiple of two, which allows the frame length to be represented with a granularity of two cell payloads. The cell count value written in the CTQ table can be used by the Cell Transfer unit (CTU) later when the latter prepares the BI header for the SOF cell.

- **Frame Control (FC) Decode unit** gets the FC field from the MAC memory for the frame pointed by the CTQ Process Pointer. After decoding the FC field, the FC Decoder writes the relevant information into Cell Transfer unit.

Figure 4.16 Block Diagram: FDDI to ATM direction VLSI Architecture

Although the FC decode unit gets a 32 bit data from the MAC memory, it uses only first 8 bits to decode the FC field.

• **Destination Address (DA) Decode unit** receives the DA field from the MAC memory of the frame pointed by the CTQP Pointer. The DA field is either 16 or 48 bit. The DA decode unit gets two words from the MAC memory and uses the relevant bits for the search of the corresponding VCI. The FC decode unit informs the DA decode unit about the length of the DA field. The DA-VCI table which is used for the VCI search may be dynamically updated to get a minimum search time overhead. After a DA match occurs, the corresponding VCI is written into the CTQ table in the entry as pointed by the CTQP Pointer.

Figure 4.17 The Cell Transfer unit (CTU)

• **Timing and Control (T & C) unit** provides the synchronization among various functional units. It also locks certain units in order to avoid contention for resources

such as the bus or the memory.

- **Address Generator** produces the address of the MAC memory locations which are needed to read the data (For example SA, FC, DA and frame etc.). The address generator is loaded by the Cell Transfer unit (CTU) with the start address of a frame, whenever a frame is to be transferred from the MAC memory to ATM layer.

- **Cell Transfer Unit (CTU)** gets a segment of 47 bytes from the MAC memory, appends the appropriate BI header and despatches the payload to the ATM layer FIFOs. Figure 4.17 (a) shows the details of the CTU. The CTQ table holds the variables in separate registers which are needed to transfer a frame from the MAC to the ATM layer. Three pointers, the CTQ Write Pointer (CTQWP), the CTQ Process Pointer (CTQPP) and the CTQ Read Pointer (CTQRP) are used to manage the operations on the CTQ table. The pointer CTQWP is used to update information in the CTQ table corresponding to the frames which are being processed currently and are pending for transmission. CTQRP is used to access the entries corresponding to the frames which have been processed and are ready to be read out of the MAC memory.

Other elements such as the comparator, length counter and barrel shifter are used in coordination with the SA Decrementer and the BI header register to align a frame segment received from the MAC, add a BI header and transfer the cell to the ATM layer.

4.5.2 Event Sequence and Timings

The sequence of events for a frame processing for the FDDI to ATM direction is shown in Figure 4.18. Parallelism is achieved for the Frame Control decoding and the DA-VCI table search by doing it in parallel with the frame reception.

The Cell Count field which is to be added to the header of the SOF cell, can be determined only after the descriptor field of the frame has been written in the MAC memory. However, the descriptor of the received frame is written after the complete reception of the frame in the memory. This causes sacrificing some pipelining advantages which could have been gained by starting the frame transfer even while it has not been received completely from the ring and is in the process of doing so.

The bridge processing of the FDDI to ATM direction can be divided into two components:

- (1) The protocol processing on the frame and
- (2) The cell transfer to ATM FIFOs.

The two parts work in sequence. The protocol processing part can be carried out in a real-time and hence only one such process is needed for the bridge processing for incoming

Figure 4.18 Event sequence for frame processing for FDDI to ATM Direction

frames; however in certain cases the second component for a particular frame may not be finished while the next frame has been received, processed and is ready for transfer. This situation is more prominent when a large frame is followed by many small length frames from the FDDI ring media. Thus, two sets of state machines working in parallel should be sufficient to carry out the protocol function execution in an efficient manner. The details of the operations of the two components are described in Appendix IV.

4.5.3 Summary: Protocol Function Execution Timings

Table 4.3 summarizes the execution timings for protocol functions - the timing are given in numbers of clock cycles needed to execute an individual function. Corresponding values of SPARC implementation are also listed to give a comparison of the two implementations. Similar to the ATM to FDDI data flow, the numbers shown against the clock cycles of the VLSI implementation, indicate the effective number of clock cycles and for some of the functions, the execution cycles that overlap with other functions have been excluded.

The throughput and delay analyses are presented in the following sections.

Table 4.3 Execution timings for protocol functions: FDDI to ATM

Type of Operation	Number of clock cycles	
	for VLSI architecture	on SPARC
DA processing	$5 + T$	$33 + (21/\text{entry})$
FC processing	3	16
Descriptor processing	7	62
Updating CTU	2	17
Bad frame discard	2	35
Alignment	1 per word	12 per word
Throughput (Mbps)	100	32

T = Table Search time : DA-VCI table

4.5.4 Throughput Analysis

Table 4.3 shows that the SPARC implementation is unable to reach the throughput requirements — an indication that a general purpose processor is not suitable for these kind of applications. The throughput of 100 Mbps can be reached only when the table search requirement is met. As shown in the timing diagrams, the only component that causes a delay additional to the copying of the frame, is the descriptor processing time. This results in a buffer latency. This affects the overall timing characteristics and hence the throughput value of the data flow. However its effect is very small and is considered negligible in the throughput calculations. This buffer latency is of importance for short frames but is insignificant for large frames. The throughput for the VLSI architecture based on the copying operations, for this data flow is calculated below. Figure 4.19 shows various components of the data movement for this data flow.

Throughput Parameters

MAC memory access time per 32 bit word, $T_3 = 160$ ns

Barrel Shifter alignment time per word (for FDDI to ATM), $T_4 = 40$ ns

Delay added to ATM FIFO access time, in order to synchronize it to MAC memory access time, $T_5 = 20$ ns

ATM FIFO access time per 32 bit word, $T_6 = 100$ ns

Figure 4.19 Components of delay across the bridge for FDDI to ATM Direction

Throughput Calculation

Maximum possible FDDI to ATM data flow throughput = $32/(160+40+100+20)$
 = 100 Mbps

The search time for the DA-VCI table is the bottleneck for this data flow. The table search is carried out in parallel to the reception of data from FDDI ring into the MAC memory. The permissible time for the table search thus, may be variable, depending on the length of the frame being received. However, in order to determine the worst case value, the smallest length of FDDI frame is considered to determine the limitations on the table search time. The table search time is found by subtracting the maximum time for protocol processing on any frame, from the reception-time of the minimum length frame. The value is 560 ns and is a potential bottleneck.

4.5.5 Delay Analysis

Figure 4.19 shows various components for the processing delay of each 32 bit word. The three components are: frame reception from the FDDI ring, cell transfer from MAC memory to ATM FIFOs (includes appending of the BI header), and cell transmission time at the physical layer. It is assumed that there is no queuing delay for the data transfer to the ATM network. The delays incurred for the three components are calculated below.

Delay Parameters

ATM network physical layer reception time, $T1 = 32 \text{ bit}/155 \text{ Mbps} = 100 \text{ ns}$ (approximately)

FDDI physical layer reception time, $T2 = 32 \text{ bit}/100 \text{ Mbps} = 320 \text{ ns}$

2

of/de

2

MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS
STANDARD REFERENCE MATERIAL 1010a
(ANSI and ISO TEST CHART No. 2)

MAC memory access time, $T3 = 160$ ns

Barrel Shifter alignment time per word, $T4 = 40$ ns

Delay added to ATM FIFO access time, in order to synchronize it to MAC memory access time, $T5 = 20$ ns

ATM FIFO access time, $T6 = 100$ ns

Calculations of delay components

Delay for frame reception component = $T2 = 320$ ns

Delay for component-cell transfer from MAC = $T3 + T4 + T5 + T6 = 320$ ns

Delay for cell transmission component = $T1 = 100$ ns

Total delay across the bridge per 32 bit word for FDDI to ATM data-flow = 740 ns.

4.6 Alternative Architecture

The VLSI architecture proposed above, meets the desired performance of the protocol implementation. It however, is not flexible enough to be adopted easily for other similar applications. An alternative architecture for a protocol-processor can be drawn which uses special instructions and registers to facilitate the protocol functions execution in minimum possible clock cycles. The architecture is general and can be used for the implementation of any protocol which has a functionality similar to that of BI protocol. The architecture proposed below for a general purpose Bridge Protocol Processor (BPP) uses a RISC processor as the base design; a few registers are added to it to get a faster and more efficient processing of certain specific protocol operations — e.g. copying of data from one memory to other along with the counter management activities.

The advantages of using an approach similar to RISC, over CISC, are outlined below.

- RISC uses a simpler instruction set and smaller, simpler addressing mode. Most of the instructions are single cycle except load and store.
- RISC processor normally use a larger register file than those used by CISC processors. This helps in a faster execution of programs as lesser number of memory accesses are required for program execution.
- Require smaller chip area, less effort and time for design.

The main objective of the BPP design has been to optimize the per byte related processing of the protocol execution. For example, copying a 32 bit word from one memory to

another requires counter management and address calculations. These activities need a relatively large number of assembler instructions if implemented on a general purpose processor, hence incurring a lot of processing overhead for the per word copying. The BPP architecture however is designed to carry out these activities in a much more efficient manner.

The operations needed once per cell or once per frame are not considered very critical to the throughput of the bridge.

In order to illustrate the architecture, the protocol functions which need improvement for the two data flows are described first. These factors are based on the analyses carried out in chapters 3 and 4. Following that is a description of BPP architecture. Finally, the operations of the processor with respect to the two data flows are described.

4.6.1 Functions needing improvement

The functions which need improvement in order to enhance the throughput for each of the data flow are described below.

ATM to FDDI direction

1. Table search for the determination of VCI, in order to reassemble the frame segments carried by the ATM cells.
2. The copying of data from one memory to the other, along with the counter related activities.

FDDI to ATM direction

1. Table search for the determination of VCI/VPI corresponding to the DA field of a MAC frame.
2. The copying of data from one memory to the other, along with the counter related activities.
3. Alignment activities needed per word during the segmentation procedure.

4.6.2 Bridge Protocol Processor (BPP) Architecture

Figure 4.20 shows the organization for bridge interface using a BPP. The architecture for the BPP is shown in Figure 4.21. The organization uses two BPPs, BPP-FA (FDDI to ATM) and BPP-AF (ATM to FDDI), one for each data flow. Each of the processors has its individual local memory. The memory used by the BPP-FA is dual port, and that used by the BPP-AF is a triple port one. Three special units — interrupt control logic, bus arbiter and table search unit are used for specific purposes as described below.

Figure 4.20 Organization of bridge interface using the bridge protocol processor.

- Interrupt control logic receives the frame arrival indication for FDDI to ATM direction, and cell arrival indication for ATM to FDDI direction. It informs the respective BPP accordingly.
- Bus arbiter receives control signals from the two BPPs and allows a proper bus connection for BPP-memory, and memory-memory data movements.
- Table search unit gets independent access to the triple/dual port memory which are local to a BPP, and do the table search for respective data flow. These tables are the same as those described earlier in the VLSI architecture of each of the data flow. A separate search unit relieves the processor of the processing overhead and reduces the processor-memory bus traffic.

The BPP uses the architecture of a SPARC RISC processor. The architecture considered for here is SPARC MB86901 (Fujitsu Ltd. Copyright © 1988) high performance RISC processor. In MB86901, the processor execution unit consists of a three port ALU,

a data aligner, and a barrel shifter. These units provide fast arithmetic, logical and shift operations. A separate Address Generation Unit calculates load/store and branch addresses, enhancing processor performance by allowing the ALU to operate concurrently on data. An internal register file comprised of 120 registers organized into seven windows minimizes accesses to memory during procedure linkages and facilitates passing of parameters and alignment variables. By reducing external memory accesses, the register file significantly improves SPARC throughput.

Figure 4.21 The Bridge Protocol Processor Architecture
(Uses the architecture of MB86901 Sparc RISC Processor
by Fujitsu Ltd. Copyright 1988)

In order to achieve the functionality of BPP, a special register-set is added to the MB86901 architecture. The description of these registers is given below. Their operations for respective data flows are described later in section 4.6.3.

The set consists of specific purpose autoincrement and autodecrement registers. Thus, when loaded with the addresses of source and destination memories, the autoincrement registers help copying of 32 bit words from one memory to the other without incurring undesired assembly instructions. To minimize the cycles for counter related activities, a special autodecrement register — length counter which is decremented automatically after a 32 bit word is moved from one memory to the other. Alignment Activity Register is used to store temporarily, the BI header values which are needed to be appended to the beginning of frame segments, for the FDDI to ATM data flow. Similarly, it is used for temporarily storing the alignment byte, while reassembling the data being received from the ATM to FDDI data flow. Barrel Shifter is used to do the appending activity.

Figure 4.22 Special Register Set for BPP

Instruction Set

All the instructions used by the BPP are 32 bit words. They are divided in four categories:

- (1) Load/Store.
- (2) Arithmetic/logical/shift.
- (3) Control transfer.
- (4) Copying.

The first three categories essentially are the same as that of the SPARC processor. The fourth category includes an additional instruction — “copy”. The copy instruction causes the contents of the memory location indicated by source-address-register to be copied to the memory location indicated by destination-address-register. It assumes the respective addresses to be pre-loaded. These addresses can be loaded through the program which executes the protocol functions. The copy instruction needs only to specify the identification of the source and destination memories.

4.6.3. BPP Operations for ATM-FDDI Bridge

The BPP operations with respect to each of the data flow are described below.

ATM to FDDI

The protocol processing is initiated by the interrupt control logic which sends a signal to the BPP, after receiving a cell arrival indication from the ATM layer. The BPP gets the VCI/VPI into its local memory using the copy instruction. It sends a control signal to the table search unit in order to get the active VCI entry from the VCI Activity table. The BPP then does the reassembly of the incoming frames; the operations related to reassembly are similar to those described with respect to the VLSI architecture presented in Chapter 4. After a frame has been reassembled and is ready to be transferred to the MAC layer, the registers in the special register-set are loaded with appropriate parameters related to the frame being transferred. Thus, the source-address-register is loaded with the starting address of the frame in BPP local memory and destination-address-register is loaded with the location of the MAC memory where the frame is to be stored. The length counter is loaded with the length of the frame in bytes. The bus arbiter is instructed to disable other accesses to the MAC memory (or waits for the currently going operation to complete) , and then the data is moved from the BPP local memory to the MAC memory. After the transfer of each word, length counter is checked to determine the number of words left to be transferred. This ensures a proper transfer of the frame. After a frame has been transferred, the control of the MAC memory is relinquished in order to allow accesses by other units.

The local memory used by the BPP for this data flow is a triple port one. Data from ATM receive FIFOs can be moved to the local memory while the data from the local memory is being moved to the MAC memory on the other port. The third port is used by the table search unit for accesses to the table elements.

FDDI to ATM

The processing for this data flow is initiated by a signal from the interrupt control logic, which after receiving a frame arrival indication from the MAC layer, informs the BPP accordingly. The BPP then gets the start address of the frame from MAC memory into its local memory by executing a copy instruction. It then uses the frame start address to access the fields FC and DA of the frame in the MAC memory. These fields are found at the beginning of a MAC frame. The BPP, after getting the DA field, instructs the table search unit to do a DA-VCI table search. The table search can be carried out independently, using a separate port of the dual port BPP local memory. The other port is used by the processor for execution of the program. While the frame is in the process of being received in the MAC memory, the table search continues in parallel. After a frame is completely received, its descriptor value is fetched to the local memory from the MAC memory and the cell count (CC) and length indicator (LI) values are calculated. At this time, the frame is ready to be segmented and transferred to ATM layer. The source-address register is loaded with the start-address of the first frame segment and the length counter is loaded with a value of 12 which is equivalent to the ATM payload length. The copy instruction is used to move data from MAC memory into the barrel shifter which appends the BI header in the beginning of a 47 byte frame segment. This is achieved by a byte insertion - done by the barrel shifter. The length counter is autodecremented after each word movement, and is used to determine the end of a frame segment.

4.6.4 Performance Assessment of BPP

The assessment of BPP performance is done using an approach similar to that in the analysis of the SPARC processor implementation (described in Chapter 3). For the configuration shown in Figure 4.20, the throughput is calculated in a way similar to that of sequential approach for SPARC implementation. The throughput calculations are given below.

ATM to FDDI

The calculations given below are based on the throughput analysis described in section 3.3.5.1. The components which undergo improvement, as a result of using the BPP special registers, are described below.

Frame Processing related component = 11640 ns.

Cell copying related component = $12[(40+40) + 2 \times 40] = 1920$ ns, assuming that 2 instructions are needed for processing related per word of copying.

Cell Processing related component = $3560 + 3960 + 1920 + 4000 = 13440$ ns

Throughput = $(47 \times 8 \times N) / (11640 + 13440N)$, where N is number of cells.

The average throughput comes out to be 26 Mbps.

FDDI to ATM

The calculations given below are based on the throughput analysis described in section 3.3.5.2. The components which undergo improvement, as a result of using the BPP special registers, are described below.

Frame Processing related component = $3680 + 3840N$, where N is number of cells.

Cell copying related component = $12[(40+40) + 2 \times 40] = 1920$ ns, assuming that 2 instructions are needed for processing related per word of copying.

Throughput = $(47 \times 8 \times N) / (3680 + 3840N + 1920N)$

The average throughput comes out to be 52 Mbps.

The throughput values indicate that an improvement is achieved for both the data flows, as a result of using the special register set in the BPP. However, the throughput is still much less than the desired value. The reason for the low throughput is that for a sequential approach, the processing of other protocol functions became the bottleneck after the copying related bottleneck was overcome by using the special register set. This, however, can be improved if a second processor is used: two processors for each data flow direction. This approach is based on the conclusions of the SPARC processor based analysis described in Chapter 3. It was shown there that the state machines for each of the data flow can be implemented using two processors for each of the data flows. The processor used in addition to the BPP, may be a general purpose processor such as a RISC processor. The protocol functions execution is shared by the two processors so as to achieve the maximum possible throughput. The throughput improvement is shown in the calculations below.

ATM to FDDI

The functions of the protocol processing are divided between the two processors such that each of them gets equal load. The processing components related to a cell are important in determining the throughput; to minimize their effect on the throughput,

these components are executed distributedly by the two processors — the BPP does cell detection and cell copying and the general purpose processor does the cell header decoding. This distribution is determined on the basis of the timing values described in Section 3.3.5.

Cell processing related component = $3560 + 1920 = 5480$ ns, consists of cell detection time and cell copying related component.

Frame processing related component = 5480 ns, is essentially the cell detection time.

Useful user data per cell = 47 bytes = 47×8 bits

Throughput = $(47 \times 8 \times N) / (5480 + 5480N)$

The average throughput comes out to be 68 Mbps.

FDDI to ATM

The copying related component is reduced to 1920 ns per cell using the special register set, as shown in the calculations above. However, other functions such as frame processing related component now become the bottleneck; in particular the copying of frame into MAC memory which requires 3840 ns per cell. The two processors divide the protocol functions between them and result in a better throughput than that achieved by a single bridge protocol processor. The descriptor processing component and the frame copying into the MAC memory determine the throughput.

Descriptor processing time = 2360 ns

Frame copying into MAC memory = $3840 \times N$, where N = number of cells.

Useful user data per cell = 47 bytes = 47×8 bits

Throughput = $(47 \times 8 \times N) / (2360 + 3840N)$

The average throughput comes out to be 107 Mbps.

Thus, from the throughput calculations given above, it can be concluded that a single BPP is not adequate in order to achieve the desired throughput, even if copying operations are minimized; however, by adding one more processor, a significant throughput improvement is obtained. In the latter case, the actual throughput will be a little less (10% to 20%) because of the additional processing required for context switching and interprocessor communication activities.

4.7 Summary

A general purpose processor such as a RISC processor can not meet the requirements of applications which require intensive data copying together with the processing of protocol functions. It is obvious from the performance analysis presented in Chapter 3 that even for a minimal functionality bridge interface protocol, a RISC processor can only achieve one third of the desired throughput. A VLSI design is an appropriate choice for such an application. The VLSI architecture proposed in this chapter uses multiple functional units and gains parallelism over a central processing unit such as a general purpose processor. The separate ports for copying of data further enhance the parallelism in protocol processing by offering the copying of data and the protocol function execution in parallel. Although it needs additional control circuitry and is less economical than a DRAM, still the advantages gained from its use outweigh the cost factor. The throughput analysis shows that the potential bottleneck has been shifted from the processing of user data to the table search by means of the VLSI approach. Subject to the ability to meet the targets for table searching speed that has been specified, the potential throughputs in each direction is 100 Mbps. The targets may be met by using efficient search hardware, such as multiple comparator units.

The effects of buffer-latency for each of the data flows are discussed. It is concluded that with the advancement in technology, the architecture presented here can work at double the throughput of the values described in the performance analysis sections, provided MAC hardware is enhanced to render a 40 ns memory access time instead of the existing 80 ns. Essentially, the MAC memory access time is derived from the underlying physical layer speed — thus for a 100 Mbps FDDI ring, a MAC memory access time of 80 ns (32 bit) is sufficient. However, the proposed architecture for VLSI, provided the table search time requirements are met, can achieve a throughput of 200 Mbps with a 40 ns MAC memory. Thus, effectively, it can support interconnection of networks at much higher speed, such as 200 Mbps.

The VLSI architecture proposed here is specific to the FDDI to ATM bridge. A more general architecture for a bridge processor, presented later in the chapter is also an appropriate choice for the bridge interface implementation. This architecture uses special registers and instructions which are general in nature, and can be used for any protocol processing which has a functionality similar to that of the BI protocol presented in this thesis.

Chapter 5

Traffic Related Performance Issues

The performance analysis of the bridge interface is incomplete without considering the effect of traffic on the protocol processing characteristics. For the purpose of analysis, it is normally assumed that in an internetworking environment approximately one tenth of the traffic on a LAN goes out on the backbone network intended for other remote LANs. However, a short term sequence of frames might all be destined for remote LANs. Also, the overall traffic received by a LAN from the backbone may sometimes be high when the number of interconnected LANs is large. If the FDDI protocol of the local bridge is not able to handle this high traffic, data may be lost at the bridge because of buffer overflow. This gives rise to certain issues as to - what are the buffer requirements at the bridge, does the bridge need a higher priority on the FDDI ring etc.? This chapter briefly discusses some of these problems. A full study of traffic characteristics is beyond the scope of this thesis

5.1 Comparison of bridge-processing and MAC delays

Martini et al [26] have done a study on station delays of a bridge for the token rings at 2 Mbps. They showed that the bridge processing delay remains constant with the increasing load while the MAC delay increases and finally reaches a region of instability. A similar comparison can be done for the bridge processing and the FDDI MAC delays. Figure 5.1 shows the two delays: the MAC delay for the FDDI ring, and the bridge processing delay for ATM to FDDI direction. The graph is not drawn to a proper scale. It represents the delay in terms of bits, on the Y-scale, and ring utilization on the X-scale. The bridge processing delay is negligible compared to the MAC delay, and remains

constant with an increase in the load. The MAC delay characteristics have been taken from Wainwright and Myles [27].

Figure 5.1 Comparison of MAC and bridge processing delays. (The MAC delays have been taken from Wainwright and Myles [27]).

For the FDDI to ATM direction, there is no queuing delay and the delay caused by the bridge processing is 74 bits (or 740 ns) only.

5.2 Need of higher priority for the bridge on FDDI ring

Under heavy inter-LAN traffic conditions, the traffic received by a bridge from remote bridges, needs to be forwarded to the local ring medium as fast as possible. This will ensure a reasonable buffer requirement with an acceptable probability of frame discard because of buffer overflow. Dykeman and Bux [25] give a relation of various MAC

priorities for asynchronous data and the normalized throughput on an FDDI ring. The relation is used to assess the requirement of priority for the bridge on the FDDI ring. The authors show that the total maximum throughput remains high as long as the target token rotation time (TTRT) is large with respect to ring latency. (The TTRT is explained in chapter 2, section 2.3.1). They say that the FDDI parameters can also be configured so that the highest priority level receives all the ring's bandwidth when it is required. Figure 5.2 shows a copy of the results as obtained by the authors. They have calculated throughput as a function of arrival rate for an FDDI ring in which there are eight stations, each attempting to transmit frames at one of the eight asynchronous priority levels. The results show that at low arrival rates, frames of each priority level are transmitted. However, as arrival rate increases, the throughput of low priority frames begins to decrease and eventually becomes zero. Even the throughput of the highest priority level gets saturated after the arrival rate exceeds a certain limit. (in this example-65 Mbps).

Other similar results presented by the authors indicate that delay characteristics of these priority levels also get unbounded after certain limits of arrival rates. (in the range of 60 Mbps).

(From Dykeman and Bux [25])

Figure 5.2 Throughput versus arrival rate per priority level.

Thus, for the bridge, for ATM to FDDI data flow, it may be necessary that the ring access should be done at highest priority level. This will however depend on the inter-LAN traffic characteristics. This higher priority is provided by the upper layers at the end station. The higher priority reduces the number of possible retransmissions by ensuring low frame loss probability.

5.3 Effect of 4 byte BI header on segmentation and reassembly

The BI protocol uses a one byte header within a 48 byte ATM payload to carry out the protocol related functions as discussed in section 2.3.3. Use of a small size BI header — such as 1 byte, helps in keeping the reduction in bandwidth to a least possible value. However, as observed in Chapter 4, alignment is needed for every 32 bit word movement from one memory to other, because of the odd numbered size of the BI header. Thus, a 4 byte header will require less processing per word thus resulting a faster processing, but will cause a higher reduction in useful bandwidth. The gains and disadvantages of using a 4 byte header over a 1 byte header, are discussed in this section.

As described earlier in section 4.3, the data received from one network needs to be copied to the other in blocks of 4 bytes. This is needed while doing segmentation and reassembly procedures. Because of the BI header being of 1 byte, every 4 byte word transfer needs realignment incurring an extra clock cycle of processing on every word. This however can be saved if a 4 byte header is used. An analysis is presented below so as to see the gains and losses of using a 4 byte header over a one byte header.

In order to determine the loss in bandwidth because of using a 4 byte header instead of 1 byte header, a comparison of number of cells required for various MAC frame lengths for each case is done. The comparison is given in Table 5.1.

From the values shown in the table, it is obvious that there is a loss of upto 9% in the bandwidth because of additional 3 bytes used in the header. The effective user data rate on a virtual circuit is 137.91 Mbps for a 1-byte BI header and 129.11 Mbps for a 4-byte BI header. This results in a bandwidth loss of 8.8 Mbps.

The FDDI frame lengths are of variable lengths and so the values of the frame lengths shown in the table have been picked randomly.

Table 5.1 Effect of 4 byte header on number of cells needed for a frame.

FDDI Frame length (bytes)	Number of ATM cells required		For 4 byte header	
	with 1byte	with 4byte	increase in number of cells	% increase in number of cells
47	1	2 (1.1)	1	100
50	2 (1.1)	2 (1.2)	0	0
100	3 (2.1)	3 (2.3)	0	0
200	5 (4.3)	5 (4.6)	0	0
500	11 (10.6)	12 (11.4)	1	8.3
1000	22 (21.2)	23 (22.7)	1	4.4
1500	32 (31.9)	35 (34.1)	3	8.6
2000	43 (42.5)	46 (45.5)	3	6.5
3000	64 (63.8)	69 (68.1)	5	7.2
4500	96 (95.7)	103 (102.2)	7	6.8

The advantages of using a 4 byte header are however many. It simplifies the segmentation procedure by allowing minimum processing of successive payloads belonging to a frame. While doing segmentation, the VLSI processor fetches 32 bit blocks from the MAC memory and prepares an ATM payload by appending a BI header to each frame segment. Each 32 bit word needs alignment in order to accommodate the 1 byte BI header. A 4 byte header however allows the processor simply to fetch 32 bit words from the MAC memory and append the BI header, and pass the payload to the ATM layer without any complexity

of alignment. It requires less control circuitry as well as simplicity of processing. The use of 4 byte header also permits a DMA operation where a block of 44 bytes of frame segment can be moved to and from the memories. Similarly while doing reassembly, the successive ATM payloads are stripped of the BI header and concatenated to each other, in order to achieve the original frame. As discussed in section 4.4, reassembly requires a complex mechanism for the alignment of successive frame segments. This can be avoided by the use of a 4 byte header.

To illustrate the advantage of a 4-byte BI header with respect to segmentation, the throughput of the FDDI to ATM direction is recalculated. The approach for calculations is similar to that presented in Chapter 3.

(a) ATM to FDDI

Please refer Section 3.3.5.1 for more details.

Frame Processing related component = 11640 ns

Copying time per cell = $[11 \times (40+40) + 15 \times 40] = 1480$ ns, where the number of useful user data in a cell payload is equivalent to eleven 32-bit words. The number of instructions needed to do the DMA operation for a cell is 15.

Cell copying related component = $3560 + 3960 + 1480 + 4000 = 13000$ ns

Useful bits per cell = $44 \times 8 = 352$

Throughput = $352 \times N / (11640 + 13000N)$

The average value of throughput is 26 Mbps.

(b) FDDI to ATM

Please refer Section 3.3.5.2 for more details.

Frame processing related component = $3680 + (11 \times 2 \times 160) \times N = 3680 + 3520N$ ns, where each ATM payload carries eleven 32-bit words of frame segment.

Copying time per cell = $11 \times (40 + 40) = 880$ ns

Cell processing related component = 880 ns + $15 \times 40 = 1480$ ns, where the number of instructions needed to do the DMA operation for a cell is 15.

Throughput = $352 \times N / (3680 + 3520N + 1480N)$

The average value of the throughput is 66 Mbps.

Thus, a 4-byte header results in a faster processing by eliminating the alignment related steps for both data flows. Moreover, another indirect advantage of a 4 byte header is that it leaves 3 additional bytes in the ATM payload, which can be utilized for any additional BI protocol function such as congestion control.

5.4 Frame loss probability

One of the desirable characteristics of the interconnected LAN transparency is that the bridge should not increase the frame loss probability beyond that which is acceptable to the end stations. Most of the LANs maintain a reasonable frame loss probability because of error at physical medium or errors that occur while processing is being done. The bridge should keep the effective error rate and frame loss rate within acceptable limits. There are three sources for information corruption leading to a frame loss:

- (1) Fiber Transmission System.
- (2) Buffer overflows at different nodes and cell loss.
- (3) Cell loss due to congestion.

The frame loss probability in the case of FDDI to ATM bridge will mainly depend on the error characteristics of the ATM backbone network. The fiber transmission system provides a very low error rate; thus 2 and 3 of the above listed sources will be the major cause of frame loss. These two in general, tend to make the cell loss bursty in nature.

5.5 Requirement of additional MAC components

As discussed in previous sections, it seems that MAC delay may prove to be a bottleneck for the bridge to bridge data transfers. Under heavy traffic conditions, the arrival rate at a bridge may be high enough to leave the bridge saturated congested. In this circumstance, in order to keep a low frame loss probability, a solution may be to provide an additional MAC controller at the bridge. This however, is only a possible solution and a detailed study is needed to substantiate the proposal.

Chapter 6

Conclusions

The bridge interface processor described and analyzed in this thesis represents a major advance in high-speed network interface implementation. With abundantly increasing bandwidth at the physical medium, the OSI layers need to speed up and avoid becoming a bottleneck; especially in an internetworking environment where a networking device is to handle high-speed data movement. The BI processor emphasizes a fundamental approach to the problem: minimal functionality of the protocol.

There are a number of conclusions that influenced the development and analysis of the bridge interface implementation. These conclusions are summarized below:

- Finite state machine implementation of the protocol functions is an efficient way for exploring protocol inherent parallelism. State machines are suitable for hardware implementation, therefore are an appropriate choice for the type of analysis such as BI protocol implementation.
- A general purpose processor with a conventional memory architecture is inadequate for a high speed protocol implementation such as BI protocol. Operating system related overhead and the memory bandwidth limit the throughput. A general purpose processor such as a RISC processor can not meet the requirements of applications which require intensive data copying together with the processing of protocol functions. It is obvious from the results that even for a minimal functionality BI protocol, a present day RISC processor can only achieve one third of the desired throughput.
- Copying of data units from one entity to the other is a major cause of protocol overhead.

Copying along with other protocol functions, when implemented in hardware, will result in a better performance.

- Specialized memory architectures are a better choice for these types of applications. A memory with a bidirectional serial port and a random access port facilitates data movement to and from the network in parallel with the protocol processing.
- The VLSI architecture proposed for the BI protocol implementation meets the requirements of the throughput, provided an appropriate table-search mechanism is used. The careful use of multiple functional units working in parallel gains parallelism benefits without significantly increasing the chip area and the cost. Fast and efficient buffer management is reached through the VLSI architecture.
- The proposed VLSI architecture is scalable. An improvement in underlying hardware technology will proportionally enhance the performance of the architecture. Similarly, the communication protocol processor architecture is also scalable. Thus, these architectures with minor modifications are suitable to handle interconnection of networks with higher speeds than it is currently intended to. The speed can be improved with further advancements in VLSI technology.
- The traffic related analysis suggests that the bridge will need a higher priority at the FDDI ring to deal with the traffic being received from the ATM network.

Future Research Work

Following are some possibilities for further research work.

- Further optimization in the architecture for VLSI implementation: In this thesis, an attempt has been made to maximize the throughput of the VLSI implementation while keeping the number of data paths and the functional units to an optimal value. Optimization from the points of view of simplicity and cost of the implementation have

not been attempted but hold the promise of further improvement. Elements such as a barrel shifter usually require a combinations of circuits within them and need a lot of silicon space. As part of the future research work, it would be interesting to see how such elements can be replaced by simpler ones which would meet the requirements albeit with less functionality and requiring less silicon space.

Similarly, the dual port memory proposed for the bridge interface implementation is currently very expensive; much more than single port memories with much lower access-times. It may be possible to modify the proposed architecture by replacing the dual port memory with a single port memory with smaller access-time. A possible approach would be to do cycle stealing to permit protocol processing concurrent with the reception of data.

- **Isochronous traffic based BI protocol design:** The analysis of the BI protocol has been done for an asynchronous traffic environment. The protocol design was done with high throughput as the primary objective, and a least possible delay. The FDDI-II protocol supports isochronous traffic which put tight constraints on delays. It may be interesting to develop a BI protocol which can support both of the FDDI protocols. It is felt that the concepts employed in the bridge interface design can well be suited to a similar implementation for delay sensitive applications. It might however, need some more exploitation of pipelining effects. For example, for the ATM to FDDI data flow, in the BI protocol presented in the thesis, a frame is moved from one memory to the other only after its entire processing is over; however, if it is possible that a frame starts moving to another network, even while it is in the process of being received, then a better timing can be achieved. This however results in a problem: while receiving from the ATM network, the frames which are longer than one ATM cell payload consist of more than one cell — if one of the later cells is lost the frame is discarded. However, this will not result in a major problem as long as the frequency of such occurrence is low, since the higher layers can always detect them and ask for a retransmission. It may be noted here that the isochronous traffic may need a higher priority on the FDDI ring.

- **Traffic related analysis:** the traffic related analysis carried out in this thesis is preliminary and a more detailed analysis with an appropriate traffic model for internetworking environment will yield better results. Buffer requirements in the bridge can be more accurately predicted when a proper traffic model is known. Same is true with the congestion and flow control policies for the bridge. The requirement of alternatives

for the bridge architecture, such as need of two MAC components with a single virtual channel may be explored as well.

- **Improved MAC hardware:** The available MAC hardware could provide more parallelism if certain modification were done to it. The SUPERNETTM MAC hardware [Appendix I] from Advanced Micro Devices Inc. is the first available one. With advancement in technology, better MAC hardware implementations are becoming available. It may be possible in future, with newer hardware, to analyze and explore further advantages in pipelining and parallelism, and achieve a better performance.

- **IEEE 802.6 based analysis:** The analysis done in the thesis can be further extended to a WAN environment which uses an IEEE 802.6 metropolitan area network (MAN) [29] as the backbone to interconnect multiple FDDI LANs. Another possible configuration may include a bridge interconnecting multiple FDDI LANs with another FDDI LAN as the backbone. All the analyses however emphasize minimal functionality protocols as the basis for achieving a high speed implementation. Throughput and delay analysis for end-to-end data movement can be done for the three scenarios and a comparison could be done to realize their feasibility.

In the 802.6 MAN protocol, the major functionality is carried by the Distributed Queue Dual Bus (DQDB) layer which interfaces the physical and the MAC layers. The user data is in form of a MAC frame sent to the DQDB layer. In order to transmit a MAC frame, the DQDB layer prepares an initial-MAC protocol data unit (IMAC PDU) by appending header and trailer to the original MAC frame. The IMAC PDU is further segmented into smaller blocks and sent to the physical layer. This is basically a convergence sublayer function. The header and trailer are used by the DQDB layer of the receiver for the proper reassembly of the segmented slot units. It would be interesting to see whether in order to reach a minimal functionality BI protocol for an FDDI to 802.6 MAN bridge, whether it is possible to modify this convergence sublayer function to a less complex and optimum functionality one. It may be noted here that this will not infringe with the existing protocols of other stations on the 802.6 MAN since this protocol will be needed only for bridge-to-bridge transfers on the 802.6 network. IEEE 802.6 is basically a connectionless mode protocol but because of the ring property at the physical medium, the segments do not need any explicit sequencing.

Appendix I

The SUPERNETTM Family for FDDI

The SUPERNET family of chipset meets the proposed standard of the FDDI MAC. The family consists of 5 chips:

1. RAM Buffer Controller (RBC)
2. Data Path Controller (DPC)
3. Fiber Optic Ring Media Access Controller (FORMAC)
4. 2 chips - Encoder/Decoder

The chipset interconnect block diagram is shown in Figure A.1.

The primary function of SUPERNET chipset is to act as an interface between a host computer and the network medium. The description of the operations of the chipset is given below.

A1.1 Data Movements

The serial data stream is received by the encoder/decoder as shown in Figure A.1, and sent to the FORMAC. The FORMAC converts it to a parallel 8 bit data and sends to the DPC. The DPC converts the 8 bit data into 32 bit before storing it into the memory. The RBC is used to generate the addresses of the dual port memory for reading as well as writing. The host processor can communicate with the buffer memory by issuing a request to the RBC and using the DPC controlled 32 bit bus, or through software instructions to the RBC and accessing data in buffer memory through the DPC's internal registers. The memory arbitration among the two ports, is done by the RBC. The RBC also handles buffer management. It makes the buffer RAM appear like a wrap-around FIFO by manipulating its various internal pointers, and uses a linked-list structure for transmitting frames.

The DPC is a half-duplex device. At any time it can receive or transmit, but not simultaneously. During frame reception, the DPC generates write requests to the RBC to write the frame in the buffer memory.

While transmitting a frame, the DPC issues a request to the FORMAC. The transmission of two types of frames, the synchronous and the asynchronous, is done in coordination with the FORMAC. The FORMAC, after capturing a token, determines which type of frame can be transmitted on the ring, and informs the DPC accordingly.

Figure A.1 SUPERNET Chipset interconnect block diagram

A1.2 Buffer Memory

The dual port memory, consisting of SRAM, is used for intermediate storage of frames. At 100 Mbps data rate, a 70 ns access time is generally adequate. The buffer memory interface is designed to support the equivalent of 200 Mbps and 100% utilization of the network by one node, only half the memory is used by the DPC for the reception or transmission of frames. The remaining 100 Mbps bandwidth on the memory is then available to the host processor. When the DPC is neither receiving nor transmitting, the complete 200 Mbps bandwidth can be made available to the host.

This ample availability of memory bandwidth is provided to minimize the buffer space required. SUPERNET allows frames to be emptied as they are received. Buffers can be small and the node interface can be simple.

Appendix II

Sparc Implementation: C Code

AII.1 C Code: ATM to FDDI Direction

The C code written for this data flow assumes separate FIFOs for individual remote bridges. All states are mutually independent in execution, but for a particular frame the sequence of execution of various states is necessary. Flags are used for each frame in order to allow a state to determine if the state previous to it has finished processing or not. The programs use three macros which are described later in this section.

/* This program corresponds to the state AF0 of the ATM TO_FDDI data flow. This part of the program waits for an "interrupt", i.e. an ATM_UNTIDATA.indication from ATM layer. This interrupt sets the MSB of BIR to 1. Thus BIR=80 (Hex) or 1000 000. After that it finds the number of the FIFO of that cell by searching a look up table (VCI-FIFO_#) and declares that FIFO as non-empty. Then it loads a pointer with the start address of the ATM cell as given by the ATM layer. It sets the AF0 flag of the BI_header to 1 and increments the pointer to point to the next possible cell's start address. */

```
main()
{
    int k, N, J, MaxFIFO, VCI_Address,
    int BIR, AF0P, ACAP, RFVP, BI_header_value;
    struct FIFO_Table
    {
        unsigned int VCI;
        unsigned short int number, empty;
        unsigned int AF0P, AF1P, AF2P, AF3P, AF4P;
        unsigned int SOF[25],
        EOF[25],
        SOF_Address[25],
        EOF_Address[25],
```

```

    Transferred[25];
    }; struct FIFO_Table FIFO[20];
N=0; J=0;
    if ((BIR[0x80] == 0x80) /* ATM_UD.indication; A=1 in BIR */
        FIFO[N].AF0P = ACAP; /* Gets the start address of the payload */
        BI_Header_value = ContentsOf(FIFO[N].AF0P); /* reads BI header */
        ModifyOf(FIFO[N].AF0P, 0x20); /* Set AF0=1 */
        {
            for (k=0; k<=MaxFIFO; ++k) /* VCI table search starts */
            {
                if (VCI_Address == FIFO[k].VCI) /* it enables sorting of cells to their */
                    N = k; /* respective FIFOs */
            }
            FIFO[N].empty=1; /* This FIFO is set non-empty */
            FIFO[N].AF0P+=48; /* Modifies individual pointer to AF0 state */
        }
    }

```

/* The program given below implements the state AF1. It tests for different cell types. It uses macro — ContentsOf() to get BI header value from the bridge memory into the processor registers. The macro ModifyOf() is used to access the flags af0 and af1. */

main()

```

{
    unsigned short int N, J, I, RFVP;
    struct FIFO_Table
    {
        unsigned int VCI;
        unsigned short int number, empty;
        unsigned int AF0P, AF1P, AF2P, AF3P, AF4P;
        unsigned int SOF[25],
        EOF[25],
        SOF_Address[25],

```

```

EOF_Address[25],
Transferred[25];
); struct FIFO_Table FIFO[20];
N=0; J=0;
FIFO[N].AF1P = FIFO[N].AF0P; /* Initialization*/ if (FIFO[N].empty==1)
{
for (I=0; FIFO[N].AF1P<=RFVP; ++I) /* All Payloads in a particular FIFO, which have
been processed by AF0 are processed by AF1 */
if ((ContentsOf(FIFO[N].AF1P)&0x20)==0x20) /* Checks to see if AF0=1 */
{
switch ( ( ContentsOf(FIFO[N].AF1P) & 0xD0 ) ) /* Gets the BI header */
{
case 0x00: /* compares the BI header value for SOF cell */
J=J+1;
FIFO[N].SOF[J]=1; /* Updates the FIFO record accordingly, if */
FIFO[N].SOF_Address[J]=FIFO[N].AF1P; /* it is an SOF cell */
ModifyOf(FIFO[N].AF1P,0x10); /*Sets AF1=1*/
break;
case 0x40: /* No activity needed if its a COF cell */
break;
case 0x80: /* compares the BI header value for EOF cell */
if (FIFO[N].SOF[J]==1)
{
FIFO[N].EOF[J]=1; /* Updates the FIFO record accordingly, if */
FIFO[N].EOF_Address[J]=FIFO[N].AF1P; /* it is an EOF cell */
ModifyOf(FIFO[N].AF1P,0x10); /*Set AF1=1*/
}
}
else
{
FIFO[N].SOF[J]=0;
FIFO[N].EOF[J]=0;
FIFO[N].SOF_Address[J]=0;
FIFO[N].EOF_Address[J]=0; /*FIFO[N].AF1P remains same */ }
break;

```

```

    case 0xD0: /* compares the BI header value for SCF cell */
        J=J+1;
        FIFO[N].SOF[J]=1; /* Updates the FIFO record accordingly, if */
        FIFO[N].EOF[J]=1; /* it is an SCF cell */
        FIFO[N].SOF_Address[J]=FIFO[N].EOF_Address[J]=FIFO[N].AF1P;
        ModifyOf(FIFO[N].AF1P,0x10); /*Set AF1=1*/
        break;
    }
    FIFO[N].AF1P+=48; /* advances the AF1 pointer for the individual FIFO to */
    /* indicate to the next possible payload for processing */
    }
    }
}

```

/* The program given below is for state AF2 of ATM to FDDI data flow. It detects the cell loss and updates the FIFO record accordingly. Only SOF and EOF cell losses are detected. */

```

main()
{
    struct FIFO_Table
    {
        short int J,N,I,Init_value;
        unsigned short int BI_Header_value1, BI_Header_value2;
        unsigned int VCI;
        unsigned short int number, empty;
        unsigned int AF0P, AF1P, AF2P, AF3P, AF4P;
        unsigned int SOF[25],
        EOF[25],
        SOF_Address[25],
        EOF_Address[25],
        Transferred[25];
    }; struct FIFO_Table FIFO[20];
}

```

```

N=0; Init_value = 0; /*Initialization*/
if ( (ContentsOf(FIFO[N].SOF_Address[J])&0x30) &&
(ContentsOf(FIFO[N].EOF_Address[J])&0x30)) /* Checks if SOF and EOF cells */
for (J=Init_value; J!=0; ++J) /* have been processed by AF1 */
{
    Init_value=J;
    FIFO[N].AF2P = FIFO[N].SOF_Address[J];
    if (FIFO[N].SOF[J] == 1) /* EOF cell loss detection */
    {
        if ( (FIFO[N].EOF[J] != 1)&&(FIFO[N].SOF[J+1] == 1) )
        {
            FIFO[N].SOF[J]=0; /* A frame is discarded if */
            FIFO[N].SOF_Address[J]=0; /* an EOF cell is lost */
            FIFO[N].EOF_Address[J]=0;
            FIFO[N].AF2P=FIFO[N].SOF_Address[J+1];
        }
        else if (FIFO[N].EOF[J] == 1) /* if EOF cell is received then */
        {
            /* Update the FIFO record accordingly */
            for ( ; FIFO[N].AF2P<=FIFO[N].EOF_Address[J]; ++FIFO[N].AF2P)
                ModifyOf(FIFO[N].AF2P, 0x08); /*Set AF2=1*/
        }
    }
    else if (( FIFO[N].SOF[J] == 0)&&(FIFO[N].EOF[J]==1))
        FIFO[N].SOF[J]=0; /* A frame is discarded if */
        FIFO[N].SOF_Address[J]=0; /* an SOF cell is lost */
        FIFO[N].EOF_Address[J]=0;
        FIFO[N].AF2P=FIFO[N].SOF_Address[J+1];
    }
}

/* The program below, is for state AF3 of ATM to FDDI. It processes the EOF cell. It
gets the length indicator value and modifies the EOF_Address accordingly in the FIFO
record */

```



```

main()
{
    unsigned short int N,J,k, BI_Header_value, Init_value, Length;
    unsigned short int *BI_Pointer;
    struct FIFO_Table
    {
        short int J,N,I,Init_value;
        unsigned short int BI_Header_value1, BI_Header_value2;
        unsigned int VCI;
        unsigned short int number, empty;
        unsigned int AF0P, AF1P, AF2P, AF3P, AF4P;
        unsigned int SOF[25],
        EOF[25],
        SOF_Address[25],
        EOF_Address[25],
        Transferred[25];
    }; struct FIFO_Table FIFO[20];
    N = 0;
    BI_Header_value = ContentsOf(FIFO[N].FOF_Address[J]);
    FIFO[N].AF3P = FIFO[N].SOF_Address[J];
    Init_value = J; /* program - for finding Init value which matches current Frame's
    pointer */
    for (J=Init_value; J!=0; ++J) /*J should be initialized to the value as given by the current
    pointer AF3P, which has SOF_Address[J]*/
    {
        Init_value = J;
        if (FIFO[N].EOF[J]==1)
        if ( ((ContentsOf(FIFO[N].SOF_Address[J])&0x38)==0x38) &&
        ((ContentsOf(FIFO[N].EOF_Address[J])&0x38)==0x38))
        Length = ContentsOf(FIFO[N].EOF_Address+47); /* accesses the LI field */
        FIFO[N].EOF_Address[J] += Length; /* Adds LI to EOF cell start address to */
        /* modify the EOF_Addres */
        FIFO[N].AF3P = FIFO[N].SOF_Address[J]; /* done to access the AF3 flag */
        ModifyOf(FIFO[N].SOF_Address[J],0x02); /*Set AF3=1 */
    }
}

```

```
ModifyOf(FIFO[N].EOF_Address[J],0x02); /*Set AF3=1 */
```

```
}
```

```
}
```

/* The program given below implements the state AF4. The program tests if a frame is processed in order to transfer it to MAC layer. The AF3 flags of SOF and EOF cells are tested to determine this. The macro Transfer_Payload() is used to carry out the transfer of a payload from the bridge memory to the MAC memory */

```
main()
```

```
{
```

```
unsigned short int J, N, I, BIR, RFRP, *RFRP_Pointer, t;
```

```
unsigned short int *BI_Pointer, Init_value;
```

```
struct FIFO_Table
```

```
{
```

```
unsigned int VCI;
```

```
unsigned short int number, empty;
```

```
unsigned int AF0P, AF1P, AF2P, AF3P, AF4P;
```

```
unsigned int SOF[25],
```

```
EOF[25],
```

```
SOF_Address[25],
```

```
EOF_Address[25],
```

```
Transferred[25];
```

```
}; struct FIFO_Table FIFO[20];
```

```
RFRP_Pointer = &RFRP;
```

```
FIFO[N].AF4P= FIFO[N].SOF_Address[J]; /* Initializes the pointer to */
```

```
N = 0; /* indicate to the cell to be processed */
```

```
for ( J=Init_value; J != 0; ++J) /* Test if AF3=1 */
```

```
{
```

```
Init_value = J;
```

```
if ((( BIR | 0x40) == 0x40) && (FIFO[N].Transferred[J] != 0))
```

```
/* if MAC is ready and frame not transferred*/
```

```
if (((FIFO[N].EOF[J]==1) &&
```

```
(ContentsOf(FIFO[N].EOF_Address[J])&0x3d)==0x3d)) /*if AF3 processing */
```

```
{
```

```
/* is over */
```

```

for (RFRP=FIFO[N].SOF_Address[J]; RFRP<=FIFO[N].EOF_Address[J]; RFRP+=
48)
{
    Transfer_Payload(RFRP); /* Transfers segments to the MAC memory */
}
}
FIFO[N].AF4P = FIFO[N].SOF_Address[J];
}
FIFO[N].Transferred[J]=1; /*Initializes frame record */
FIFO[N].SOF[J]=0;
FIFO[N].EOF[J]=0;
FIFO[N].SOF_Address[J]=0;
FIFO[N].EOF_Address[J]=0;
}

```

MACRO ContentsOf()

```

_ContentsOf: save    %sp, %l0, %sp
               mov     address, %l0 /* load the address to reg l0 */
               lduba    [%l0], %r0 /* get the contents of l0 */
               ret
               restore

```

MACRO ModifyOf()

```

_ModifyOf:  save    %sp, %l0, %sp
            mov     address, %l0 /* load the address to reg l0 */
            lduba    [%l0], %r0 /* get the contents of l0 */
            or       %r0, flag, %r0 /* modify the value */
            stuba    %r0, [%l0] /* store the flag value at the address */
            ret
            restore

```

MACRO TransferPayload

```

_TransferPayload:  ld R0, cell_count /* Number of payloads to be moved */
                   ld counter2, %r1 /* Number of words per payload */
                   ld start_address, %g0 /* address from where data is to be moved */
                   ld destination_address, %g7 /* address where data is to be moved */
                   ld EOF_Address, %g5 /* used to take care of partially filled cells */

payload_transfer:  cmp 1, %r0 /* tests if its an EOF cell */
                   be eof_cell

word_transfer:     lda [%g0], %g1 /* load the first data word */
                   add %g0, 1, %g0 /* inc the start_address */
                   lda [%g0], %g2 /* access the next word */
                   shl %g1, 8, %g1 /* get rid of BI header/surplus byte */
                   mov [%g2], %g3 /* make a copy of data word */
                   shr %g3, 24, %g3 /* get the first byte of second word */
                   or %g1, %g3, %g1 /* concatenate firsts' 24 bit with seconds' 8 bits */
                   sta %g1, [%g7] /* store the word in MAC memory */
                   add %g0, 1, %g0 /* inc source_address to access next data word */
                   add %g7, 1, %g7 /* inc destination address */
                   cmp 1, %g4 /* Check for eof cell */
                   be eof_cell /* If eof cell, take care of partially filled cell */
                   sub %r1, 1, %r1 /* dec counter */
                   cmp %r1, 0 /* is the counter 0?, i.e. all 12 words moved */
                   bne word_transfer /* if not, move next word */
                   ba stop /* else stop */

eof_cell:          cmp %g0, %g5 /* compare if source address = EOF_Address */
                   be stop /* if true, data transfer is over */
                   mov 1, %g4 /* %g4 is used as a flag */
                   ba word_transfer /* transfer next word, if data is not finished */

stop: end

```

AII.2 C Code: FDDI to ATM Direction

/* This program is for state FA0 of the FDDI-to-ATM direction data flow. This detects for the MAC-UNITDATA.indication. It gets the starting address of the frame */

```
main()
{
    unsigned int RPR;
    unsigned short int N, MAC_UNITDATAindication;
    struct Frame_Record
    {
        unsigned int PSA, VCI;
        unsigned short int FA0, FA1, FA2, FA3, TX, Cells, LI;
    } Frame[4];
    N=0;
    do
    {
        if (MAC_UNITDATAindication == 1)
        {
            Frame[N].PSA = RPR; /* Set the Read-Pointer value of MAC frame i.e. the */
                               /* start address of the frame to payload segment start address */
            Frame[N].FA0 = 1; /* Set FA0 flag for this frame, to indicate that FA0 is over */
            Frame[N].TX = 0; /* Set TX flag to indicate that frame is not yet transmitted */
            ++N;
        }
    } while (N!=4); /* Maximum value of N=4, since there are four states */
}
```

/* This program is for state FA1 of FDDI-to-ATM direction. This tests the FC and the DA fields of the MAC frame The frame is discarded if any error is found in either FC or DA */

```
main()
```

```

{
    unsigned int DA_Higher, DA_Lower T_Var, DA_VCI[10], DA_Table_Element[10];
    long int DA;
    unsigned short int N, DA_Length, FC, k, No_of_Bridges;
    struct Frame_Record
    {
        unsigned int PSA, VCI;
        unsigned short int FA0, FA1, FA2, FA3, TX, Cells, LI;
    } Frame[4];
    N = 0;
    if ((Frame[N].PSA!=0)&&(Frame[N].FA0==1))
    {
        T_Var=Getword(Frame[N].PSA)>>24; /* Getword a macro—described later */
                                         /* 32 bits are read from MAC memory */
        FC = T_Var>>24; /* 8 bit FC is extracted */
        if ((FC&&40)==40) /* The Length bit is tested */
            DA_Length = 0;
        else DA_Length = 1;
        DA_Higher=Getword(++Frame[N].PSA); /*Higher 32bits of DA field are read */
        DA = DA_Lower + DA_Higher<<32; /* Complete DA is formed */
        for (k=0; k<=No_of_Bridges; ++k)
        {
            if (DA==DA_Table_Element[k]) /* DA-VCI table search occurs */
                Frame[N].VCI=DA_VCI[k]; /* DA values are compared to determine VCI */
        }
    }
}

/* This program is for State FA2 of the FDDI to ATM direction. It tests the end of the
receipt of a frame in MAC memory and calculates the cell-count and length-indicator
values */
main()

```

```

;
unsigned short int Frame_End, N;
unsigned int Frame_Length;
struct Frame_Record
{
    unsigned int PSA, VCI;
    unsigned short int FA0, FA1, FA2, FA3, TX, Cells, LI;
} Frame[4];
N=0;
do
{
    if (Frame_End == 1) /* test to see if frame has been received in MAC memory */
    {
        Frame_Length = GetWord((-Frame[N].PSA)&&0x00ff); /* Decrement frame */
        /* start address, in order to access the descriptor field of MAC frame */
        Frame[N].Cells = Frame_Length/47; /* Calculate cell-count */
        Frame[N].LI = Frame_Length % 47; /* Calculate length-indicator */
        Frame[N].FA2 = 1; /* Set FA2 flag to indicate that FA2 is over; done to */
        /* allow FA3 to start its processing on this frame */
        ++N;
    }
} while(N!=4); /* Maximum value of N=4, since there are four states */
}

```

/* This program is for state FA3 of FDDI to ATM direction. It does the transfer of frame into ATM Payloads; it segments the frame, appends appropriate BI header and transfers the ATM payload to ATM layer */

```

main()
{
    unsigned short int N,BI_Header,x,y,z;
    struct Frame_Record

```

```

{
    unsigned int PSA, VCI;
    unsigned short int FA0, FA1, FA2, FA3, TX, Cells, LI;
} Frame[4];

N = 0;

if ((Frame[N].TX != 1) && (Frame[N].FA2 == 1)) /* if frame not transmitted */
    /* and state FA2 has finished processing this frame */
{
    if ((Frame[N].Cells == 1) && (Frame[N].LI == 0))
        || ((Frame[N].Cells == 0) && (Frame[N].LI != 0))
        /* Test to see if it is an SCF cell */
    {
        BI_Header = x; /* x is a temporary variable for BI header of SCF cell */
        TransferCell(Frame[N].PSA);
    }
    else /* if not SCF, then SOF, EOF are required; COF may be required */
    {
        BI_Header = y; /* y is a temporary var for BI header of SOF cell */
        TransferCell(Frame[N].PSA); /* Create SOF cell and send to ATM layer */
        --Frame[N].Cells;
        Frame[N].PSA += 47; /* Increment Pointer to indicate to next payload */
    }
    do
    {
        BI_Header = x; /* Add BI header for COF cell */
        TransferCell(Frame[N].PSA); /* Send the COF cell to ATM layer */
        Frame[N].PSA += 47; /* Increment the pointer to indicate to next frame segment */
    } while (Frame[N].Cells != 0); /* Test to see how many COF cells are needed */
    BI_Header = z; /* z is a temporary var for BI header of EOF cell */
    TransferCell(Frame[N].PSA); /* Send EOF cell to ATM layer */
}

```



```

    }
    Frame[N].TX = 1; /* Update the frame record to show that its transfer is complete */
}

```

MACRO TransferCell

```

_TransferCell: ld cell_count, %r0 /* Number of cells to be transferred */
               ld counter2, %r1 /* Number of words per cell */
               ld start_address, %g0 /* Get start address of frame segments */
               ld destination_address, %g7 /* Get the place where data is to be moved */
cell_transfer: ld BI_Header, %g3 /* Get the BI header */
               shl %g4, 24 /* mov header value to MSB position */

word_transfer: lda [%g0], %g1 /* Get the first word in reg g1 */
               mov %g1, %g2 /* make a copy for local manipulations */
               shr %g2, 8, %g2 /* g2 has the 24 bits meant to be sent */
               mov %g3, %g5 /* mov previous surplus byte to reg g5 */
               mov %g1, %g3 /* extract surplus byte for this word */
               shl %g3, 24, %g3 /* reg g3 has the surplus byte */
               sta %g1, [%g7] /* send the word to ATM layer */
               add %g0, 1, %g0 /* access next word */
               add %g7, 1, %g7 /* increment the destination address */
               sub %r1, 1, %r1 /* Decrement the counter */
               cmp %r1, 0 /* Check if all words transferred */
               bne word_transfer /* if not, send next word */
               add %g0, 47, %g0 /* if all words transferred, access next cell */
               sub %r0, 1, %r0
               cmp %r0, 1
               bne cell_transfer

stop: end

```

Appendix III

Details of VLSI Architecture: ATM to FDDI

The details of individual events for the architecture presented in section 4.4.1 are explained in the following sections. To highlight the features of the VLSI architecture, detailed timing diagrams are presented in Figures A.3 to A.6. In these timing diagrams, different rows represent the functional units and the columns represent the clock cycles (each cycle of 40 ns). The description of events is given at the second row and the events and actions are briefly added to explain the operations carried out. The tail of an arrow shows the cause and the front shows the effect of an action. The timing characteristics for the SOF, COF, EOF and SCF are shown in the figures below and their details are given in that order.

AIII.1 Details of protocol execution operations common to all cell types

Operations which are common to all the four types of cell, are described in this paragraph. The operations individual to each cell-type are described in following sections. On an ATM cell arrival, the signal `ATM_UNITDATA.indication` from the ATM layer causes the Activity Detector to send a signal to the Timing & Control unit. The Activity Detector also takes care of any acknowledgment to be sent to the ATM layer in response to the signals received from the ATM layer. The Timing & Control unit now activated by the Activity Detector, signals the Barrel Shifter to transfer the next word (i.e. VCI) to be sent to the VCI Sorter unit. The VCI must be compared in the VCI Activity table in order to get the proper reassembly of the frame segments belonging to a particular frame. It may be noted here that at any instant of time, a sender bridge may only send one frame; hence a VCI match is enough to determine the correct frame to which the current segment be reassembled to. The VCI Sorter starts comparing the VCI against the entries in the VCI-Activity table to determine the FSR Pointer (FSRP) value. Although, the FSRP is not needed in the case of an SOF cell still, it brings some parallelism in processing if the cell turns out to be either COF or EOF. The next byte received from

the ATM layer after the VCI is the BI header. Thus, the BI header is extracted by the Barrel Shifter in the next cycle after the VCI, and is passed to all the three units of the BI header decoder. This way, a parallelism is achieved among the processing of the VCI and the three units of the BI header decoder.

For the time-being it has been assumed for simplicity of explanation that the VCI search time takes two clock cycles. This is done only to highlight the performance of the VLSI architecture although it is not realistic. This point is discussed later in Section AIII.2.

After the values of VCI and Cell-type are known, SSA is determined by the address generator. The segment is then stored at the SSA. In the case of SOF and COF cell, the last byte of their payloads is saved in the VCI Activity table in the entry called as Alignment byte. Another operation needed in the case of COF and EOF cell is writing back the Alignment byte; this is explained below.

At this point, a mechanism for the alignment of various frame segments is presented. Although it is a diversion from the main discussions, but it helps to explain subsequent details more clearly.

Alignment of frame segments

The transfer of segments (received in form of ATM payloads) to the dual port memory requires proper alignment of the successive segments so as to provide a properly reassembled frame to the MAC layer. There are two possible ways to do this alignment: It can be done (1) while the data is being received from the ATM FIFOs into the dual port memory or (2) while the frame segments are being transferred from the dual port memory to the MAC memory.

The first approach has two basic advantages in that it presents a complete contiguous frame to the MAC layer, and makes the transfer to the MAC memory very simple and fast. It allows the alignment to be done along with the other protocol functions, in parallel with the reception of a cell hence more efficiently.

The second approach described later in this section however needs proper synchronization with the MAC processor, which makes it slow and more complex than the first approach. The first approach is adopted for the reassembly of frame segments for the bridge discussed in this thesis. Figure A.2 depicts the alignment mechanism. The description of this approach is as follows. The alignment is achieved by using the FSRP, and the Alignment Byte fields of the VCI-Activity table, and the SA field of the FTQ table. A

47-byte frame segment when stored in a memory with a 32-bit word boundary, needs one more byte at the end to complete the last word. On receipt of the SOF cell of a frame the SA field is moved ahead by one byte, thus eliminating the BI header from the very beginning of the entire frame. To make the subsequent cells contiguous, their store addresses are generated in increments of 47 bytes from their respective previous segments. However, this forces the last byte of each of the segment to be overwritten by the BI header of the next segment. This byte needs to be recovered. The solution is reached by temporarily storing the last byte of a payload in the VCI-Activity table, against the corresponding VCI.

Note : Successive segments are written every 47 bytes.

Figure A.2 The Alignment mechanism

Each time a cell payload starts being copied, a Word Counter counts up to 11, and then generates an interrupt to the Timing & Control unit. The Barrel Shifter is signalled to copy the last byte of the next 32 bit word to the Alignment Byte field of the VCI-Activity table. At the same time the ADR is loaded with the last word of the previous frame segment. The overwritten byte is recovered by writing the alignment byte immediately after the next segment is transferred to the memory. This is done by using the random access port of the dual port memory. This mechanism perfectly aligns the whole frame into 32-bit contiguous words and eases the transfer to the MAC layer.

The second approach requires an additional pipeline or a shift register between the serial port output and the MAC layer. This pipeline has a barrel shifter which outputs a 32 bit word after it has received a minimum of two words from the serial port. The barrel shifter removes the BI header, shifts the bytes and delivers top four bytes. The procedure is repeated until the end of the frame is reached. This procedure is complex and slow.

AIII.2 Details of protocol execution operations for individual cells

The protocol processing carried out on each of the cells are described below. The state machine details are given in section AIII.3 and should be referred to while reading the following details.

- **SOF cell:** The operations common to each cell type are already described above. In this section the operations individual to SOF cell are explained. After the cell-type has been decoded, it is known that the WPTR value is the address for the cell storage (Since an SOF cell always goes to a new frame location). Please refer to section 4.3.3 for more details. At this time, the value of WPTR is put on the output of the Address Generator and is copied to the SA-Decrementer of the FTU. The value of the address is decremented by one byte and the resultant value is stored in the SA field of the FTQ table in the entry corresponding to the current VCI. This eliminates the BI header of the SOF cell. The FTU updates the FTQ table further by writing the VCI, and resetting the values of RX and TX. The VCI, RX, TX update is required every time a new frame arrives at the bridge. The mechanism for alignment is described in detail earlier in this section. After counting 11 from the first word of the current payload the Word Count unit sends a signal to the Timing & Control unit, which causes the Barrel Shifter to copy the last payload byte into the appropriate field of the entry in the VCI-Activity table. This also is needed for alignment.

Whenever an SOF cell arrives, there is a possibility that the EOF cell of the previous cell might have been lost. A reset EOF field bit indicates the EOF cell loss of the previous frame, since under normal circumstances, the EOF field bit is set as soon as the EOF cell is received. A loss of EOF cell will force the FTU to initialize the entry corresponding to that frame; indicating the discard of that frame.

Locks are provided by the Timing and Control unit whenever a condition occurs - when the functional unit is ready to execute but the resource is busy. For example at time T7 in Figure A.3, the Cell Loss Detector unit is ready to search the FTQ table, to determine the cell loss, but since the FTQ table is occupied by the FTU for the updating, a lock is put on the Cell Loss Detector unit.

Figure A.3 Timing Diagram: SOF Cell

The timing diagrams for the COF, EOF and the SCF cell are shown in the Figures A.4 to A.6. The details of these are very similar to those of the SOF cell; hence only the operations which are different from the SOF cell will be described here.

• **COF cell:** The COF cell has a very limited processing. The FTQ table does not need to be updated. The SOF cell loss detection is done in the case of COF cell by testing the SOF bit of corresponding VCI-Activity table entry. Another significant operation done as part of the COF cell protocol processing, is the updating of the FSR Pointer. It is incremented to point to the memory address location where the next segment of the individual frame would be stored. Finally as part of the alignment activities, the last byte of the previous frame segment is overwritten after this cell is stored into the memory.

Figure A.4 Timing Diagram: COF Cell

• **EOF cell:** The processing for the EOF cell requires VCI search, BI header decoding and the alignment activities needed for the previous payload. Furthermore, the EOF cell processing needs to update the FTQ table in order to take care of the useful data bytes in the last segment which may be partially filled; it modifies the end of frame address according to the number of bytes carried in the EOF cell. This is achieved with the help

of an adder which gets the information of the useful bytes of the last cell (that is the LI value) and adds it to the start address of the EOF cell store address of the memory. This modified value is stored in the EA field of the corresponding FTQ table entry and is used later when the frame is transferred to the MAC memory. The timing diagram shows how these activities are carried out with minimum possible execution cycles. The pointer FTQ Read Pointer is used to reach the queue of the frames which are pending for transfer to the MAC layer.

Figure A.5 Timing Diagram: EOF Cell

• SCF cell: The SCF cell processing needs almost all the steps needed by the SOF and the EOF cells. It does not however need to do any alignment activity, since there is only one segment in the whole frame. The Segment Store Address (as shown in the figure for the Address Generator) for an SOF cell is the address pointed by the WPTR. This is determined by the Address Generator unit which decides the address on the basis of

the cell-type field. In the case of an SCF cell, the BI header contains the LI information and not the cell count. The Timing & Control unit, upon knowing that it is an SCF cell, initiates the updating of the FTQ table. The values of VCI, RX, TX, SA and EA are written in a similar manner as in the case of SOF, EOF cells. The EOF cell loss of the previous frame occurs if the EOF field bit is found reset. Thus if the EOF is not found as set, the BI assumes that the EOF cell is lost and the previous frame is discarded by initializing the corresponding entry in the FTQ table.

Figure A.6 Timing Diagram: SCF Cell

Frame Service Mechanism

The frame service i.e. the frame transfer to the MAC memory is carried out in parallel to the cell reception from the ATM layer. It may be noted however, that a particular

frame can be transferred to MAC layer only after it has been completely received and processed in the BI local memory.

The frame transfer is handled by a separate independent part of the FTU, which uses the FTQ Read Pointer (FTQRP) to determine which frame can be transferred. FTU has to ensure before a transfer, that the frame is ready for that. The RX bit of the FTQ table is used for the purpose. As part of the frame processing, the RX bit is set by the FTU after the completion of processing of a frame in the dual port memory. The FTU after ensuring that $RX = 1$, loads the SA and the EA of the frame into two temporary registers and then starts transferring the frame block by block. The state diagram representing the frame service is given later in section AIII.1 in Figure A.7(c).

In the beginning of the details of the timing characteristics, the VCI-Activity table search time was assumed to be a minimum of 2 clock cycle. From the timing characteristics, it is known that the cell processing needs 16 cycles. As the inter-arrival time of cell is equivalent to 68 cycles (40ns cycle time), it is possible to devote the remaining 52 cycles (or equivalently 2080 ns) to the VCI search. Considering the fact that there may be a large number of VCIs supported by the bridge, it is likely that the VCI search may prove to be the bottleneck. The solution lies in an efficient search algorithm. Parallel search can be done by dividing the VCI table into several parts with separate comparator units working on individual parts, so as to render a search time of the order of 52 cycles.

AIII.3 State Machine details

The state machine details are shown in Figure A.7. There are two main components of the state machine: the frame processing component (Fig. A.7a) and the frame transfer component (Fig. A.7b). The first one consists of the details of protocol function processing related to each type of cell. The second one is executed only when a frame is ready for transfer and is waiting in the queue.

Figure A.7 (a). ATM to FDDI : Frame Processing Component state diagram

Figure A.7 (b) ATM to FDDI: Frame transfer component state diagram

Appendix IV

Details of VLSI Architecture: ATM to FDDI

In this section, the frame segmentation procedure is described first followed by the details of operations of the two components. The timing diagrams have been explained along with the state machines for the FDDI to ATM direction of data flow.

Frame Segmentation: The frame segmentation is done while the frame is being transferred to the ATM FIFO from the MAC memory. The data is copied in 32 bit words from the MAC memory into the Barrel Shifter by the Cell Transfer unit (CTU). The CTU passes the frame start address through the Address Generator to initiate a frame transfer. The start address is copied into the address generator from the SA entry shown by the CTQRP of the CTQ table. The length and the cell counters of the CTU are loaded with the Len (length) and the CC fields respectively, of the same table entry. Please refer to Figure 4.17 for details of CTU and CTQ table.

The data transfer from MAC memory is in form of 32 bit words. The length counter contains the length of the frame in bytes, and is decremented by 4 every time a word is fetched to the CTU. The first word of a frame is shifted by one byte and the BI header meant for the SOF cell is appended to it. The next word is shifted by one byte and the previous surplus byte is appended to it in the beginning. Thus, after every 47 byte payload transfer, one surplus byte will be left. This byte will keep on accumulating after successive payload transfers. A small Alignment buffer (for example 32 bit) is provided to store these surplus bytes temporarily. The surplus byte is pushed into the barrel shifter immediately before the first word of the next segment arrives at the barrel shifter. The barrel shifter then appends the surplus byte to the beginning of the next payload. It may be noted here that after every payload transferred to the ATM layer, one surplus byte is left. Thus, for every four payload transferred one word will be surplus. This will need a delay of one clock cycle after every four payload transfers, in order to clear the 32 bit Alignment buffer.

The length counter is used to determine the type of BI header to be appended to a segment. After every word transfer from the MAC memory, the length counter is decremented by four. It is checked after each payload transfer to see if there are more than 47 bytes left. If yes, the BI header contains a cell-type value for COF cell; if not the cell-type value is set to EOF cell type.

A second approach to handle the segmentation more efficiently is to manipulate the pointer RPR of the SUPERNET set. Whenever a frame is to be segmented into 47 byte payloads and sent to the ATM layer, the RPR is forced back by one byte for every frame segment. This makes a room for the BI header in the beginning of the payload. This way 48 bytes are read at a time from the MAC memory, of which the first byte is nothing but the last byte of the previous payload. This byte is duplicate and hence redundant; and is ultimately overwritten by the BI header value. A barrel shifter may be used whenever the first word is being passed to the ATM FIFO and the first byte can be overwritten by an appropriate BI header. This ensures a 32 bit data movement from the MAC memory and is very simple and efficient. Unfortunately, the SUPERNET does not allow the manipulation of the RPR, hence this mechanism can not be used.

- **Frame Processing:** In normal circumstances, a protocol processor which executes the functions of the layer just above the MAC layer would receive and process the frame after it has been received completely by the latter and an indication to this effect sent to the former. However, the bridge processor uses a different approach and achieves pipelining advantages. In the FDDI SUPERNET hardware, the signal RECEIVE goes high as soon as a frame reception starts from the ring media, and goes back to low as soon as the frame reception ends. The bridge processor uses this signal to determine the start of a frame. The Frame Detector as shown in Figure 4.16, gets this signal and signals the Timing and Control unit of the arrival of a frame from the ring media to the MAC layer. At this moment, the CTU gets the Frame start address and loads it into its CTQ table at the CTQWP. This has been depicted in the state machine in Figure A.10 (a). The frame start address is nothing but value of the pointer RPR of the SUPERNET set. The CTQPP is used to access the frame which should be processed next. The frame start address is used to access the FC and DA fields of this frame, which are read from the MAC memory and copied into the FC and DA Decoders respectively. The FC Decoder determines the DA-Length field and sends an indication to the DA Decoder which translates the DA to its corresponding VCI and writes into the FIQ table. If no

match for the DA occurs in the DA-VCI table the frame is discarded by initializing the corresponding entry in the FTQ table. After the processing of a frame is over, the RX bit is set to indicate that the frame is ready for transfer. CTQPP is then updated to indicate to the next frame in the queue. The timing characteristics are shown in Figure A.8.

Figure A.8 Timing Diagram: Frame Processing Component

The search time needed for a DA match in the DA-VCI table is the factor that affects the frame processing component. The upper threshold value for the DA-VCI table search will depend upon the minimum length of the MAC frame. The FDDI frame length can be a minimum of 12 bytes which happens for an information field of zero bytes. For a 12 byte frame, the reception time from the ring media into MAC memory is equivalent to 48 clock cycles of 40 ns (i.e. the equivalent of six 32-bit-words transferred every 320 ns as received from the FDDI ring at 100 Mbps speed) as shown in Figure A.8. The table search time is determined by subtracting the time needed for processing of protocol functions (i.e. the equivalent of 14 clock cycles) from that of frame reception (i.e. the equivalent of 48 clock cycles). Thus, for a particular frame, if the DA-VCI table search

can not be completed within 1360 ns, it needs another instance of the frame processing component in order to attend to the next possible frame from the ring.

As soon as the frame reception by the MAC layer ends, the RECEIVE signal goes low. This is detected by the frame detector, which conveys it to the Timing and Control unit. The Timing and Control unit now initiates the CTU to read from the CTQ table, the start address of the frame which is to be transferred. This address value is decremented by one word, in order to access the descriptor field of the MAC frame. The resultant address is loaded into the Address Generator and the descriptor field in the MAC memory can thus be accessed. The descriptor field is copied to the CC Decoder which finds the CC field as well as the LI field and writes them into the CTQ table. The RX bit of the corresponding entry in the CTQ table is set to indicate that the frame is ready for transfer.

• **Cell Transfer:** The cell transfer component works disjointedly with the frame processing component. The timing characteristics for the cell transfer has been shown in Figure A.9. The CTQ table entry for a frame which has been processed, contains the Start Address of that frame. The pointer CTQ Read Pointer (CTQRP) is used by the Cell Transfer unit (CTU) to access the next frame which is ready to be transferred. The CTQRP thus acts as a transmit queue pointer. The entries in the CTQ table represent

Figure A.9 Timing Diagram: Cell Transfer Component

the queue elements where the queue is served from top to bottom. The proper service of the frames in the queue is ensured by testing for the RX bit. This bit is set by the CTU after the frame processing is over.

In order to transfer cells, the segmentation of a frame into 47 byte payloads is done by using a barrel shifter. The segmentation procedure is described in the beginning of this section. The BI headers which are needed to be added to the payloads, are loaded into the respective registers of the CTU. The data is copied in 32 bit words from the MAC memory and after it passes through the Barrel Shifter, it is added at the end of the appropriate header and passed to the ATM FIFO.

AIV.2 State machine Diagrams

The state machine diagrams for this data flow are shown in Figure A.10. It comprises three components: The frame receive, frame processing and the cell transfer components. The first one, shown in Figure A.10 (a), only detects the start of a frame and gets the start address of the frame into a register. The second one, shown in Figure A.10 (b), when available for processing, uses that start address to access the FC, DA fields and processes them. After the processing is over it sets the RX bit in the corresponding entry of CTQ table to indicate that the cells are ready to be transmitted. The third one, shown in Figure A.10 (c), appends the appropriate header to the frame segments and despatches them to the ATM layer.

Figure A.10 (a) Frame Receive state diagram

Figure A.10 (b) Frame Processing state diagram

Figure A.10 (c) Cell-transfer state diagram

References

- [1] American National Standard, "FDDI Token ring Media Access Control (MAC)," ANSI X3.139 - 1989.
- [2] CCITT COM XVIII-11-E November 1989.
- [3] R. Perlman, A. Harvey, G. Varghese, "Choosing the appropriate ISO Layer for LAN Interconnection," *IEEE Network*, vol. 2, no. 1, pp. 81-85, Jan. 1988.
- [4] A. L. Chapin, "Standards for Bridges and Gateway," *IEEE Network*, vol. 2, no. 1, pp. 90-91, Jan. 1988.
- [5] B. Hawe, A. Kirby, and R. Stewart, "Transport Interconnection of Local Networks with Bridges," in *Advances in Local Area Networks. IEEE Press (NY)* 1987, pp. 482-495.
- [6] R. C. Dixon, D. A. Pitt, "Addressing Bridging and Source Routing," *IEEE Network*, vol. 2, no. 1, pp. 25-32, Jan. 1988.
- [7] F. Backes, "Transparent Bridges for Interconnection of IEEE 802 LANs," *IEEE Network*, vol. 2, no. 1, pp. 5-9, Jan. 1988.
- [8] S. Dravida, V.R. Saksena, "Error Performance of IEEE 802.6 MANs," *Proceedings of INFOCOM*, pp. 105-115, June 1990.
- [9] "H. Kanakia, D. R. Cheriton, "The VMTP Network Adapter Board (NAB): High-Performance Network Communication for Multiprocessors," *ACM SIGCOMM*, pp. 175-186, 1988.

- [10] J. Crowcroft, M. Riddoch, "Managing the interconnection of LANs," *Issues in LAN Management, IFIP, 1988, Proceedings of the IFIP TC6/W6.4 Workshop on LAN Management*, pp. 173-181, July 1987.
- [11] T.P. Tanaka, Y. Takiyasu, "A Consideration on ATM Technology in Private Networks," *Proceedings of GLOBECOM*, pp. 1835-1840, 1989.
- [12] K. Ohno, Y. Watanabe, M. Masuda, H. Mizuno, T. Wada, "Optical High Speed (100 Mbps) Token Ring System," *Proceedings of GLOBECOM*, pp. 0645-0652, 1988.
- [13] W. M. Kiesel, "A Multi-access Protocol for Local Area Networks with Dynamic Priorities and Recovery Mechanisms," *Fourth European Fibre Optic Communications and Local Area Networks Exposition*, pp. 183-191, June 1986.
- [14] XTP Protocol Definition, Revision 3.4, Protocol Engines Inc., Santa Barbara, CA, 17 July 1989.
- [15] D. D. Clark, V. Jacobson, J. Ramkey, H. Salven, "An Analysis of the TCP Processing overhead," *IEEE Communications*, vol. 27, no. 6, pp. 23-29, June 1989.
- [16] "The SUPERNET family for FDDI," 1989 *Data Book — Advanced Micro Devices*.
- [17] S. B. Furber, "VLSI RISC Architecture and Organization," Marcel Dekker Inc., 1989.
- [18] D. Bursky, "Triple-Port Dynamic RAM Accelerates Data Movement," *Electronic Design*, vol. 38, no. 10, pp. 37-43, May 24, 1990.
- [19] J. Mailloux, J. Mills, M. J. Seibert, "Speed Memory, Ease Timing Requirements with VRAM Functions" *Electronic Design*, vol. 37, no. 24, pp. 95-99 Nov. 23, 1989..
- [20] M. Zitterlart, "High-Speed Protocol Implementations based on a Multi-Processor Architecture," *Proceedings of IFIP WG6.4/6.1 Workshop on Protocols for High-Speed Networks*, May 9-11, 1989.

- [21] M. Dang, P. Rolin, L. Sponga, and G. Votsis, "Some effects of HSLANs on the design of MAC communicating circuits," *Proceedings of IFIP TC6/WG6.4 Intl. Workshop on HSLAN*, 1987, 37-52.
- [22] B. Mandalia, G. T. Davis, R. E. Landa, D. V. Voorhis, M. Ilyas, E. B. Fernandez, T. Koshgoftar, "Flexible Hardware Architecture for Multi-media Communication Processing," *International Communications Conference*, pp. 0961-0965, 1989.
- [23] M. Akoi, T. Uchiyama, S. Tonami, A. Hayakawa, H. Ichikawa, "Protocol Processing for High Speed Packet Switching Systems," *International Zurich Seminar*, pp. 125-130, 1986.
- [24] A. Krishnakumar, K. Sabnani, "VLSI Implementation of Communication Protocols — A Survey," *IEEE Journal on Selected Areas in Communications*, vol. 7, no. 7, pp. 1082-1090, Sept. 1989.
- [25] I. Rubin, J. E. Baker, "Media Access Control for high speed Local Area and Metropolitan Area Communications Networks," *Proceedings of the IEEE*, vol. 78, no. 1, pp. 168-203, Jan. 1990.
- [26] P. Martini, M. Rupprecht, "Designing High Speed Controllers for High Speed Local Area Networks," *Proceedings of GLOBECOM*, pp. 0170-0174, 1989.
- [27] N. Wainwright, A. Myles, "A Comparison of the Delay Characteristics of the FDDI and IEEE 802.6 MAC Layer Protocols," *Hewlett Packard Laboratories, Information Systems Centre, Bristol*, Mar. 13, 1989.
- [28] D. Giarrizzo, "High-Speed Parallel Protocol Implementation," *Proceedings of IFIP WG6.4/6.1 Workshop on Protocols for High-Speed Networks*, May 9-11, 1989.
- [29] Draft proposed, IEEE Standards 802.6 metropolitan area network (MAN), QPSX dual bus media access control and physical layer protocol documents. Nov. 1989.

- [30] R. Vallee, L. Orozco-Barbosa, N.D. Georganas, "Interconnection of Token rings by IEEE 802.6 Metropolitan Area Networks," *Proceedings of INFOCOM*, pp. 934-942, 1989.
- [31] S. Weber, "Speciality SRAMs are filling the speed gap," *Electronics magazine*, vol. 63, no. 5, pp. 85-92, May 1990.

END

03·03·92

FIN