HYBRID SWITCHING:
CONVERGING PACKET AND TDM FLOWS
IN A SINGLE PLATFORM

A Thesis Submitted to the College of

Graduate Studies and Research

In Partial Fulfillment of the Requirements

For the Degree of Master of Science

In the Department of Electrical and Computer Engineering

University of Saskatchewan

Saskatoon

By

Roshan Parajuli

PERMISSION TO USE

In presenting this thesis in partial fulfilment of the requirements for a Postgraduate degree from the University of Saskatchewan, I agree that the Libraries of this University may make it freely available for inspection. I further agree that permission for copying of this thesis in any manner, in whole or in part, for scholarly purposes may be granted by the professor or professors who supervised my thesis work or, in their absence, by the Head of the Department or the Dean of the College in which my thesis work was done. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of Saskatchewan in any scholarly use which may be made of any material in my thesis.

Requests for permission to copy or to make other use of material in this thesis in whole or part should be addressed to:

Department of the Department of Electrical and Computer Engineering
57 Campus Drive
Saskatoon, SK, S7N 5A9, Canada
Phone: (306) 966-5380
Fax: (306) 966-5407

ABSTRACT

Optical fibers have brought fast and reliable data transmission to today’s network. The immense fiber build-out over the last few years has generated a wide array of new access technologies, transport and network protocols, and next-generation services in the Local Area Network (LAN), Metropolitan Area Network (MAN), and Wide Area Network (WAN). All these different technologies, protocols, and services were introduced to address particular telecommunication needs. To remain competitive in the market, the service providers must offer most of these services, while maintaining their own profitability. However, offering a large variety of equipment, protocols, and services posses a big challenge for service carriers because it requires a huge investment in different technology platforms, lots of training of staff, and the management of all these networks.

In today’s network, service providers use SONET (Synchronous Optical NETwork) as a basic TDM (Time Division Multiplexing) transport network. SONET was primarily designed to carry voice traffic from telephone networks. However, with the explosion of traffic in the Internet, the same SONET based TDM network is optimized to support increasing demand for packet based Internet network services (data, voice, video, teleconference etc.) at access networks and LANs. Therefore the service providers need to support their Internet Protocol (IP) infrastructure as well as in the legacy telephony infrastructure. Supporting both TDM and packet services in the present condition needs multilayer operations which is complex, expensive, and difficult to manage. A hybrid switch is a novel architecture that combines packets (IP) and TDM switching in a unified access platform and provides seamless integration of access networks and LANs with MAN/WAN networks. The ability to fully integrate these two capabilities in a single chassis will allow service providers to deploy a more cost effective and flexible architecture that can support a variety of different services.

This thesis develops a hybrid switch which is capable of offering bundled services for TDM switching and packet routing. This is done by dividing the switch's bandwidth into VT1.5 (Virtual Tributary -1.5) channels and providing SONET based signaling for routing the data and controlling the switch's resources. The switch is a TDM based architecture which allows each switch's port to be independently configured for any mixture of packet and TDM traffic, including 100% packet and 100% TDM. This switch allows service providers to simplify their edge networks by consolidating the number of separate boxes needed to provide fast and reliable access. This switch also reduces the number of network management systems needed, and decreases the resources needed to install, provision and maintain the network because of its ability to "collapse" two network layers into one platform.

The scope of this thesis includes system architecture, logic implementation, and verification testing, and performance evaluation of the hybrid switch. The architecture consists of ingress/egress ports, an arbiter and a crossbar. Data from ingress ports is carried to the egress ports via VT1.5 channels which are switched at the cross point of the crossbar. The crossbar setup and channel assignments at ingress port are done by the arbiter. The design was tested by simulation and the hardware cost was estimated. The performance results showed that the switch is non-blocking, provide differentiated service, and has an overall effective throughput of 80%. This result is a significant step towards the goal of building a switch that can support multiprotocol and provide different network capabilities into one platform. The long-term goal of this project is to develop a prototype of the hybrid switch with broadband capability.

ACKNOWLEDGEMENTS

I would like to express my gratitude to my supervisor Dr. Carl McCrosky for his support, guidance, and encouragement throughout the course of this research work. I greatly appreciate his timely thesis revision and critical feedback. I would also like to thank Telecommunication Research Laboratories (TRLabs) for providing financial support and other resources for this research. Jack Hanson, Vera ljubovic, Andrew Kostiuk, and the rest of the students and staffs at TRLabs, Saskatoon, are all thanked for their time and help. I am also thankful to Dr. Eric Salt, Dr. Ron Bolton, Dr. Li Chen, Dr. David Dodds, and Dr. Daniel Teng who helped me at some point of time during this research. I also express sincere thanks to the committee members, Dr. Li Chen, and Dr. Wahid Khan for their valuable comments and suggestion in the preparation of this thesis. I would also like to thank Dr. Simone Ludwig for acting as my external examiner.

I am grateful to my friends in Saskatoon, as well as at the University of Saskatchewan for their constant support and encouragement. I would like to give special thanks to Prem Sharma, Nirmala Sharma, Sumit Paudyal, Dipendra Rai, Umesh Gyawali, Jeeva Paudyal, Aseem Sharma, and Edwin Gono Santosa for their help in many ways. I have a very special feeling for my lovely wife, Shrijana Dhakal, who in many ways helped me during my Master’s program. Her consistent support, encouragement, and understanding are highly appreciated.

I wish to dedicate my work to my loving parents, Balika Parajuli and Gyan Prasad Parajuli. Without their encouragement and support I would not have come this far.

TABLE OF CONTENTS

PERMISSION TO USE..i
ABSTRACT..ii
ACKNOWLEDGEMENTS...iv
TABLE OF CONTENTS...v
LIST OF FIGURES viii
LIST OF TABLES..x
LIST OF ABBREVIATIONS...xi

1) Introduction 1
1.1 Motivation for Research 3
1.2 Background of Hybrid switching 4
1.3 Objectives 6
1.4 Thesis organization 7

2) Hybrid Switching Architecture Overview 9
2.1 Hybrid Switching Concept...9
2.2 System Overview 10
2.3 Data Path...12
2.3.1 Class of Service (CoS) 13
2.4 Arbiter 14
2.5 Control Paths..16
2.5.1 Control Path for Carrying Flow Requests.....................................16
2.5.2 Control Paths for Carrying Time Slot Assignments 17

3) SONET Signaling, Data Paths and Control Paths 18
3.1 SONET Signaling 18
3.1.1 The SONET Frame 18
3.1.2 SONET Bandwidth 19
3.1.3 The STS-N Frame 20
3.1.4 SONET Virtual Tributaries (VTs) 21
3.2 Packet Queuing Issues and Virtual Output Queue (VOQ) 21
3.3 Data Path Architecture..23
3.3.1 Ingress Ports...23
3.3.2 Ingress Packet Queues 23
3.3.3 TDM Filler 24
3.3.4 Switch Core..25
3.3.5 TDM Extractor...26
3.3.6 Packet Assembler...27

3.3.7 Egress Packet FIFO....28
3.4 Control Paths 28
3.4.1 Flow Requests....28
3.4.2 Ingress Time Slot Assignment....29
3.4.3 Egress Time Slot Assignment....29
3.4.4 Switch Time Slot Assignment 30
3.4.5 Packet Channel Assignment 30
3.4.6 SONET Channel Assignment 30

4) Arbiter Bandwidth Allocation.... 31
4.1 Introduction....31
4.2 Arbitration Goal....31
4.2.1 Priority 32
4.2.2 Fairness 32
4.2.3 Efficient Bandwidth Utilization....32
4.3 Arbiter Block 33
4.3.1 Request Matrix....35
4.3.2 Fair Proportional Algorithm (FPA) 36
4.3.3 Time Slot Algorithm (TSA)....40
4.3.3.1 Goal....41
4.3.3.2 Ingress/Egress Open....41
4.3.3.3 Open Time Slot 42
4.3.3.4 Masking Slot 43
4.4 Arbiter Design Alternatives....46
4.4.1 Software Approach 46
4.4.1.1 NIOS II Processor....47
4.4.1.2 Hardware and Software Partitioning....47
4.4.1.3 Software Design Process....48
4.4.1.4 Software Arbitration Process 49
4.4.1.5 Advantages of Software Based Approach 50
4.4.1.6 Implementation Results 51
4.4.1.7 Limitations of Software Based Approach....52
4.4.2 Hardware Approach....52
4.4.2.1 Hardware Architecture....52
4.4.2.2 Advantages of Hardware Approach....53
4.5 Arbiter Implementation Summary....54

5) Design Verification and Testing.... 56
5. 1 Introduction....56
5.2 Simulation and Verification Process....56
5.3 Verification and Testing 57
5.4 Summary of the Verification 60

6) Performance Results and Discussions 58
6.1 Introduction....61

6.2 Simplified IP Packet Structure 61
6.3 Load Model 61
6.3.1 Distribution of IP Packets by Size 63
6.3.2 Distribution of IP Packets by Class of Service (CoS) 64
6.3.3 Distribution of IP Packets by Offered Load 65
6.3.4 Distribution of IP Packets by Hotspot Port 65
6.4 Performance Matrices 66
6.4.1 Throughput 66
6.4.2 Queue Depth Vs Arbitration Cycles 67
6.4.3 Non-blocking and Uniform Loads 67
6.5 Simulations and Performance Results 68
6.5.1 Objectives 68
6.5.2 The Simulations 68
6.5.3 Determining the Operating Load 69
6.5.4 Results and Discussions 69
6.5.4.1 Variation of Queue Depth with Offered Load for Uniform Traffic 70
6.5.4.2 Variation of Queue Depth with Packet Classes for Uniform Traffic 71
6.5.4.3 Variation of Queue Depth with Hotspot Port 72
6.5.4.4 Variation of Queue Depth with Hotspot Classes 73
6.5.5 Summary of the Results 74

7) FPGA mapping: Logic Utilization, Delay and Power Estimation 76
7.1 Introduction 76
7.2 Field Programmable Gate Arrays (FPGAs) 76
7.2.1 Logic Utilization 77
7.2.2 Propagation Delay 77
7.2.3 Power Consumption 78
7.3 FPGA Cost Estimation Methodology 78
7.4 Synthesis Result and Analysis 80
7.5 Summary of FPGA Mapping 82

8) Conclusions and Future work 83
8.1 Conclusions 83
8.2 Thesis Contributions 85
8.3 Future Work 85
8.3.1 Prototype Fabrication in FPGA 86
8.3.2 Simulation for TDM and Packet Traffic 86
8.3.3 Switch with Speedup 86
8.3.4 Broadband Switching by Scaling Up Granularity to STS-1 86
8.3.5 Search for an Improved Arbitration Scheme 87

9) References 88

LIST OF FIGURES

Figure 1.1 Traditional LAN and MAN Network 5

Figure 1.2 Hybrid Switching LAN and MAN Network 6

Figure 2.1 Hybrid Switching Model 9

Figure 2.2 Hybrid Switching Block 10

Figure 2.3 Block diagram of Hybrid Switch Architecture 11

Figure 2.4 Arbiter Block Diagram 15

Figure 3.1 SONET Frame 19

Figure 3.2 Formation of STS-3 frame by byte interleaving 20

Figure 3.3 Virtual Output Queues for N destinations and C classes 24

Figure 3.4 TDM Filler Block 24

Figure 3.5 Switch Core Block 25

Figure 3.6 TDM Extractor Block 26

Figure 3.7 Packet Assemblers and FIFO Memory Blocks 27

Figure 4.1 Arbiter Functional Block Diagram 33

Figure 4.2 Arbiter Flow Chart 34

Figure 4.3 Switch Flow Matrix Defining Time Slot Assignment Goal 40

Figure 4.4 Time Slot Assignment Logic Block 41

Figure 4.5 Ingress/Egress Open Vector 42

Figure 4.6 Formation of OTS Vector 42

Figure 4.7 Masking Slot Finding Process 43

Figure 4.8 Binary Tree 44

Figure 4.9 Binary Add Algorithm Block 45

Figure 4.10 Arbiter Module with Software and Hardware Partitioning 48

Figure 4.11 Detailed Arbiter Hardware/Software Partitioning 50

Figure 4.12 Arbiter Hardware Block 52

Figure 5.1 Simulation Process for Design Verification 57

Figure 6.1 Structure of IP Packet Generated by Load Model 62

Figure 6.2 Internet Packet Distributions: Relative Frequency of Various Sizes 64

Figure 6.3 Internet Packet Distribution: Accumulated Distribution 64

Figure 6.4 Ingress Queue Length for Various Load Factors 70

Figure 6.5 Egress Queue Length for Various Load Classes of Traffic at Operating Load.. 71

Figure 6.6 Egress Queue Length for Various Load Classes of Traffic at Operating Load.. 73

Figure 6.7 Egress Queue Length for Various Classes of Hotspot Traffic 74

Figure 7.1 RTL Area and Power Analysis in IC Design Flow 79

LIST OF TABLES

Table 3.1 SONET Virtual Tributaries..21

Table 4.1 Request Matrix for Class 1 Traffic 37

Table 4.2 Converting Request Matrix to Column Grant Matrix...39

Table 4.3 Converting Column Grant Matrix to Switch Flow Matrix 39

Table 4.4 Arbiter Timing Result for Software Implementation 51

Table 7.1 FPGA Resource Utilization Table for 4x4 Hybrid Switch.................................80

Table 7.2 Scale-up Cost for Hybrid Switch...81

LIST OF ABBREVIATIONS

ADM	Add-Drop Multiplexer
ASIC	Application Specific Integrated Circuit
ATM	Asynchronous Transfer Mode
CoS	Class of Service
CIR	Committed Information Rate
CLB	Configurable Logic Block
DWDM	Dense Wave Division Multiplexing
DUT	Device Under Test
EIR	Extended Information Rate
FPA	Fair Proportional Algorithm
FPGA	Field Programmable Gate Arrays
FIFO	First In First Out
FR	Frame Relay
HDL	Hardware Descriptive Language
HoL	Head of Line
IB-VOQ	Ingress Buffered Virtual Output Queue
IC	Integrated Circuit
IP	Internet Protocol
LAN	Local Area Network
LE	Logic Elements
LUT	LookUp Table
MSS	Maximum Segment Size
MTU	Maximum Transmission Unit
MAN	Metropolitan Area Network
MPLS	Multiprotocol Label Switching

OTS	Open Time Slot
OQ	Output Queue
POH	Path OverHead
POTS	Plain Old Telephone Service
PSTN	Public Switched Telephone Network
QoS	Quality of Service
RAM	Random Access Memory
RTL	Register Transfer Level
SLA	Service Level Agreement
SONET	Synchronous Optical NETwork
SPE	Synchronous Payload Envelope
STS	Synchronous Transfer Signal
TDM	Time Division Multiplication
TOH	Time OverHead
TSA	Time Slot Algorithm
TCP	Transport Control Protocol
VOQ	Virtual Output Queue
VT	Virtual Tributary
WAN	Wide Area Network

Chapter 1

Introduction

Communications of voice, video and data has experienced several technological revolutions in last two or three decades. One of the greatest inventions of recent years is the Internet. The Internet has had phenomenal success, growing from a small research network to a global network that we use on a daily basis. The Internet is a packet based network. When a host wants to communicate with other hosts, it uses the Internet Protocol (IP) to place information in packets, which are then sent to the nearest router. The router stores, and then forwards these packets to the next hop. Through hop-by-hop routing, packets find their way to the desired destination. This is called packet switching. With this communication technique, link bandwidth is shared among many information flows, and these flows are statistically multiplexed on the link.

To keep up with the explosive demand for bandwidth as well as to adhere to Service Level Agreements (SLAs) for a growing number of mature business applications on the Internet, network switches must be both faster and smarter. These switches must not only simply terminate high-speed optical connections but also must switch a large number of connections from Dense Wavelength Division Multiplexing (DWDM) transport systems [1]. They must provide guarantees on parameters such as bandwidth, latency, loss rate, and jitter, which are not supported by current best-effort switch architectures. Finally, they must provide a path to migrate to Multiprotocol Label Switching (MPLS) based networks without abandoning existing investments in the legacy networks such as Synchronous Optical NETwork (SONET) and Frame Relay (FR).

Time division multiplexing (TDM) is a multiplexing technique that divides a circuit into multiple channels based on time. The technique is associated with telephone company voice services. TDM was designed to deliver a steady stream of digitized voice. The data rate for each channel is exactly what is needed to carry a digitized voice which is 64 Kbps.

While companies have long used TDM circuits for both voice and data, TDM circuits are not ideal for data because data tends to be bursty. The repeating time slots do a good job at delivering the streaming bits of digitized voice, but data bursts fill up the slots unevenly. When there is no data to send, bandwidth goes unused. When data bursts, there is usually not enough bandwidth.

In order to support traditional TDM based telephone network and packet based internet network, the service provider must have a multi-layer architecture; one layer supporting the packet based internet traffic while the other layer supporting the TDM based voice traffic. To compete on market, the service providers must provision all these services by squeezing as much expense as possible out of serving provisioning and support. However, supporting both types of services in today's multilayer architecture is expensive, inefficient, complex, and difficult to manage and troubleshoot.

This thesis describes a hybrid switch architecture designed to address these problems and provide enhanced flexibility at significantly reduced cost. We propose a switching method suitable for mapping packet based traffic into TDM time slots and to consequently allow simultaneous switching of packets and TDM traffic on the same switching platform. In this architecture TDM or packet data is mapped to TDM channels, enabling TDM switches to be used in the switch fabric of a router. This switching architecture provides lower cost TDM and packet switching and protects the investment already made in the legacy technology (principally SONET) while enabling new investments in both TDM switching and packet switching. In addition, the proposed hybrid packet/TDM architecture is straightforward and offers high speed data transfer. Because of its multiservice capability of handling both packets and TDM traffic, scalability, and support for quality of service, the switch architecture is directly applicable to at least the following areas:

- simplified networks,
- unified packet and TDM based circuit switching platforms,
- cost-effective delivery of multiple services on one platform,
- capacity to increase bandwidth and add new services,

- optimized used of physical space,
- high end enterprise applications, and
- easy installation and operation.

1.1 Motivation for Research

Over the last decade, through the combination of telecommunication market deregulation, economic globalization, and the Internet revolution, we have witnessed the emergence of a range of novel communication services for both residential and enterprise customers. IP-based applications and the deployment of IP technology not only increased the demand for new communication services but also established the quest for greater transmission speeds [2]. Yesterday’s focus was purely on the transmission of telephone voices by circuit provisioning with SONET and DWDM. The data world was based on TDM circuits designed for voice transmission. However, with the introduction of many protocols, access services, and internet applications at the access edge, the trend in data services, stretching out of the local area into the metro and regional transmission domain is changing the requirements for switching and routing. Transmission networks and systems are therefore also evolving to include mapping of multiple services and signal types into SONET. However, the service providers face the challenge of connecting multiple data traffic and voice traffic into their high-speed backbone because supporting wide range of access interfaces is complex and expensive.

Many challenges are associated with provisioning voice and data services using today’s legacy architecture, as shown in Figure 1.1. Voice-data overlay networks results in duplication of equipments, demands more operational space, and yields high costs. Also running separate operations for each service is complex, and difficult to manage and troubleshoot. Tomorrow’s transmission networks must therefore efficiently manage both types of services-provisioned circuits for voice traffic and switched multipoint data services simultaneously in one platform. Investment is reduced by having only one converged network managing both traffic types, ultimately leading to lower operational expenditures.

The hybrid switch addresses the requirements of a single common minimum-cost platform from an inter-community prospective. The switch is capable of handling both TDM flows and packet flows on any of its ports, and on sub-channels of its various ports. Each switch port uses SONET STS-12 signaling with a bandwidth of 622.08Mb/s. Each VT1.5 component of an STS-12 is treated as a conventional SONET (TDM) flow, or as stream of IP packets. The switch then switches TDM flows as logical circuits with low latency and negligible switching jitter, and switches packet flows with fair virtual output queuing techniques. The advantage of this architecture is that it allows the normal use of two different switching technologies and 'boxes' to be collapsed into one 'box', thereby offering substantial savings and flexibility to the user in Local Area Network (LAN) and Metropolitan Area Network (MAN) environments. It also follows a new and unique design for multiservice provisioning by integrating the right amount of packets and TDM functionality and provides a dedicated fabric for each technology, thereby guaranteeing optimum performance of this truly converged platform.

1.2 Background of Hybrid Switching

Figure 1.1 shows a typical carrier application under today's network conditions. From the figure it is obvious that different types of switching equipments are being used. Traditional TDM traffic for Plain Old Telephone Service (POTS) for Public Switched Transport Networks (PSTN) and dial-up Internet traffic are mostly carried over TDM based T1 lines. These T1 circuits are terminated in a T1 multiplexer. Packet based IP, Asynchronous Transfer Mode (ATM), and FR traffic are terminated in different terminal devices. Both TDM and packet traffic are mapped into the SONET signal and carried over the service provider's SONET network. Add-Drop Multiplexer (ADM) provides access to the SONET network.

The requirements for different types of equipments and protocols lead to substantial network complexity. Rack space is needed for the various types of equipment. Each set of equipment usually comes with its own management system. This requires expertise in multiple technologies with equipment from multiple vendors to manage each transport –

network type. In addition to the expense of supporting this infrastructure, the provisioning of services requires co-ordination of the configuration of all the network elements that are involved.

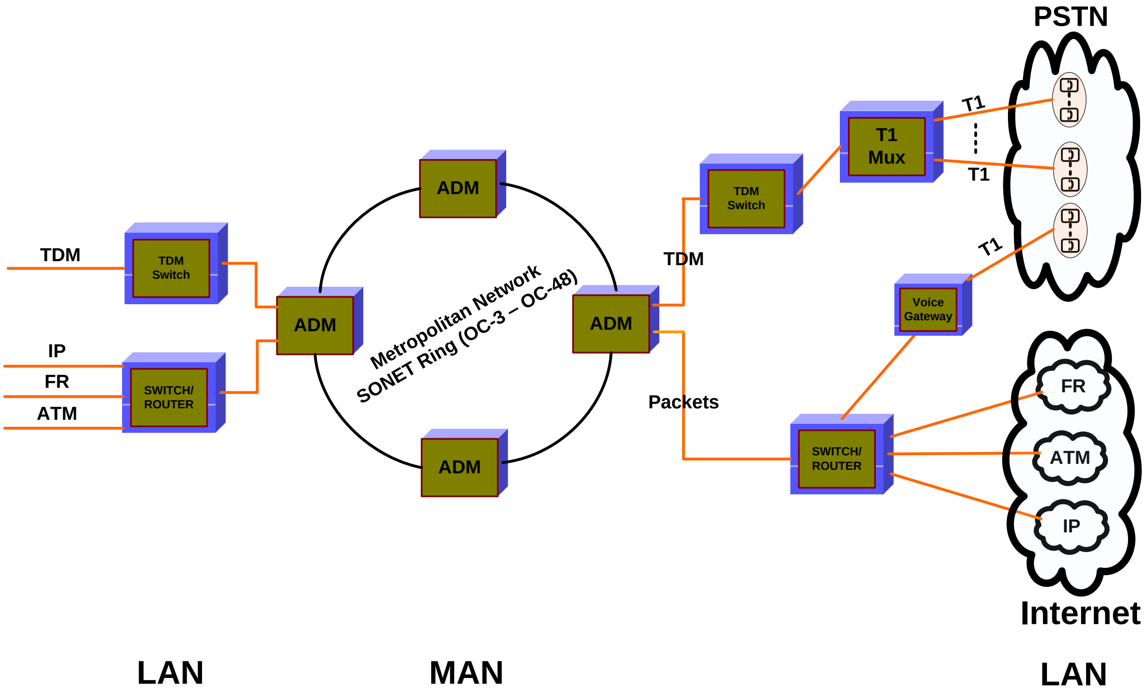


Figure 1.1 Traditional LAN and MAN Network

The hybrid switching concept combines the functions of a multiplexer with multiple service access devices and adds the capability to support multiple protocols in one platform. This new approach uses a rack-mounted chassis in which a wide variety of interface and function cards can be inserted to tailor the platform to specific roles. This new paradigm for building the network infrastructure converge the functions of time division backbone switches and packet switches into a single backbone making the infrastructure much simpler, cheaper, and easy to manage.

Figure 1.2 shows the hybrid switch application in a carrier network. The optical (or electrical) interface function to the hybrid switch has been combined with different access interfaces. All of this has been integrated into one platform and positioned strategically at the boundary between the MAN and the LAN. This new architecture removes barriers to operational efficiency and flexible provisioning for voice and data by creating a unified network that can be operated and managed easily.

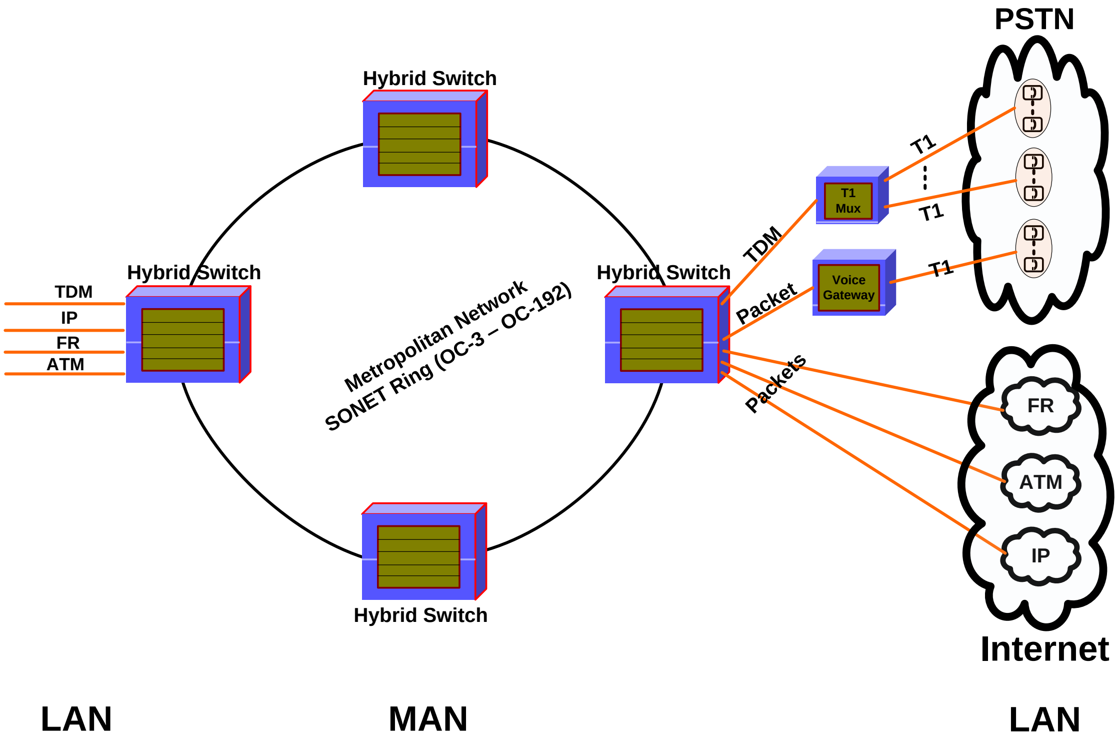


Figure 1.2 Hybrid Switching LAN and MAN Network

1.3 Objectives

The research objective is to design and evaluate the performance of a hybrid switching architecture for LAN and MAN networks. The key is to design a switch that is capable of accepting both TDM flows and packet flows on any of its ports, and forwarding it to its destination. Design, performance characterization, and simulation of the switch will be done in order to study the cost and benefits of having such a switching architecture on the current internet backbone network. More specifically, the research work is done in order to:

(1) study the feasibility of supporting both TDM based traffic and packet based traffic on a single switching platform,

(2) develop new algorithms for arbitration and study the performance of switches using that arbitration scheme, and

(3) study the design alternatives, and examine the cost and performance issues; and find what heuristics are most useful in developing efficient hybrid switching.

Our challenges in order to meet the above objectives are:

- ❖ Design of a switching platform which integrates the functionality of a TDM based switch and a packet based switch, with the challenge of not having excessive implementation costs.

- ❖ Design of the data path that can carry both TDM and packet traffic.

- ❖ Design of the TDM based switch backbone with a suitable fast arbiter. The three stage time-space-time (TST) switch should have an arbitration cycle less than four SONET cycles.

1.4 Thesis Organization

This thesis is organized in the following way. Chapter 2 gives a brief introduction to the hybrid switch. It gives an overview of the data paths, control paths, and the arbiter unit, and defines the interconnection networks. It also introduces the TST (Time-Space-Time) switching nature of the hybrid switch. Different classes of service supported by the switch and their priority for providing the Quality of Service (QoS) for the subscribers are also explained. Chapter 3 reports the architectural design of the data paths and the control signals. It also gives a brief introduction to SONET and describes how SONET signaling is used on the communication link between port card and the line card. This chapter also introduces packet queuing issues and the use of Ingress Buffered Virtual Output Queue (IB-VOQ) memories in our hybrid switch. Chapter 4 is about the arbiter. It explains arbitration process for packet and TDM traffic. The bandwidth arbitration algorithm and the time slot assignment processes are also described. Finally the chapter describes the arbitration design alternatives and explains a software approach and a hardware approach and their advantages and disadvantages. Chapter 5 explains how the design was verified and tested to make sure it functions correctly. Chapter 6 describes our simulation experiments and results. It describes the load model used for simulating the design, and analyses of the obtained results. Chapter 7 explains the FPGA implementation of the switch with cost and area analysis. Chapter 8 concludes the thesis and discusses future work.

Chapter 2
Hybrid Switching Architecture Overview

This chapter provides an architectural overview of our hybrid switch. Section 2.1 describes the overall switching concept. A hardware system overview is introduced in Section 2.2. An introduction to the data path is given in Section 2.3. Arbiter and control signals of the switch are introduced in Section 2.4 and Section 2.5 respectively.

2.1 Hybrid Switching Concept

The hybrid switch switches both TDM and packet traffic carried by SONET STS-12 (Synchronous Transport Module level - 12) links on a single switching platform. Figure 2.1 shows the hybrid switching model.

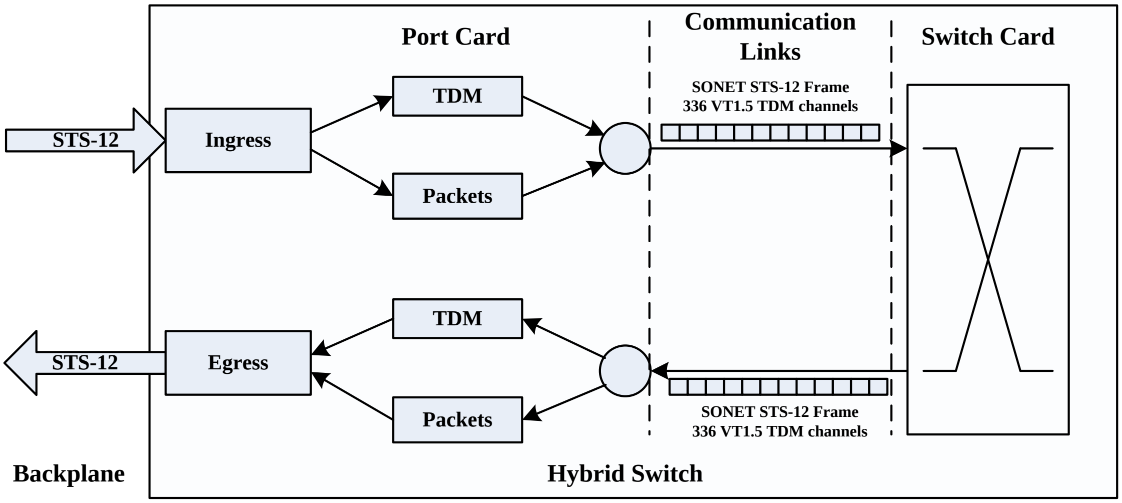


Figure 2.1 Hybrid Switching Model

Traffic carried by each SONET STS-12 link is differentiated into packet and TDM traffic at the ingress port. These packets are switched at level 3 (IP packets) or level 2 (ATM or Frame Relay). The remaining SONET TDM voice components are switched at their native TDM level. The switch's ingress and egress communication links are divided into 336 Virtual Tributary 1.5 (VT1.5) channels. The number of VT1.5s in one STS-12 frame is 336. Each channel can carry either packet or TDM traffic from ingress port to egress port. The traffic carried by the ingress VT1.5 channels is switched at the switch card. The VT1.5 channel allocation and crossbar set-up logic is provided by arbiter logic in the switch card. The traffic coming out of the switch card is extracted and differentiated into TDM and packet traffic at the egress side. Egress traffic is read, assembled, and stored at the egress port and sent as an STS-12 signal across the backplane.

VT1.5 channels are used to carry long packets by byte interleaving. Each channel can carry packet bytes from any queue. Each queue (flow) can be carried by one or more VT1.5 channels. The channel allocations may change at any STS-12 frame boundary. It is the job of the arbiter to announce such changes to the TDM filler, crossbar and TDM extractor so that the packets are switched and reassembled properly.

2.2 System Overview

Figure 2.2 shows a block diagram of an *NxN* hybrid switch which supports *C* priority classes. The switch contains *NxC* packet queues at each input, *NxC* output buffers at each output, a crossbar fabric and an arbiter.

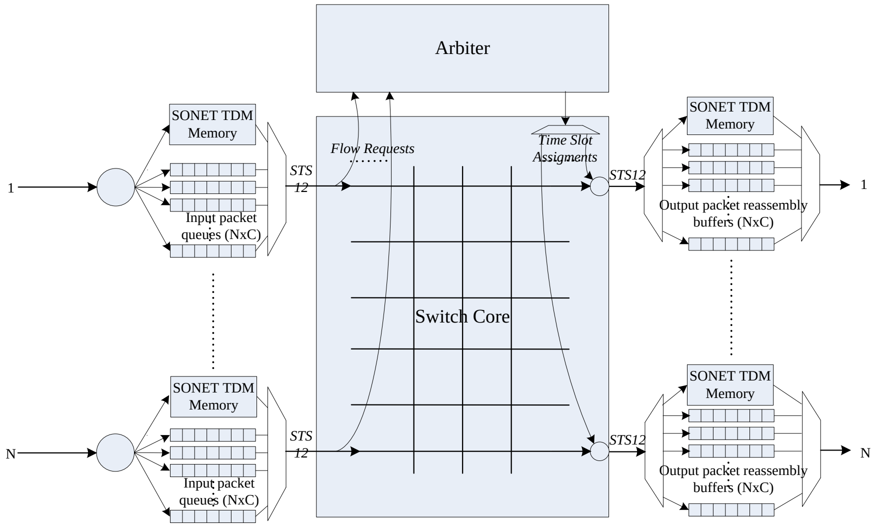


Figure 2.2 Hybrid Switching Block

The *NxC* input queues at each input port are used as IB-VOQ in order to respect priorities and eliminate head-of-line (HOL) blocking. N ingress ports by N egress ports are cross connected by the NxN crossbar which provides ingress lines for each input and egress lines for each output. This hybrid switch allows each port to be independently configured for any mixture of packet and TDM traffic, including 100% packet and 100% TDM.

Figure 2.3 shows the hybrid architecture in more detail. The communication paths from ingress ports, through the crossbar and to the egress ports carry the SONET STS-12 protocol which contains 336 VT1.5 channels. The switch connects inputs to outputs through these channels, each of which have an aggregate bandwidth of 1.544 Mbps. These VT1.5's can be used to carry standard virtual tributaries (SONET TDM) or byte streams which are used to carry the switch's packet traffic. Aggregations of VT1.5's can be used to carry STS-N's as broadband TDM.

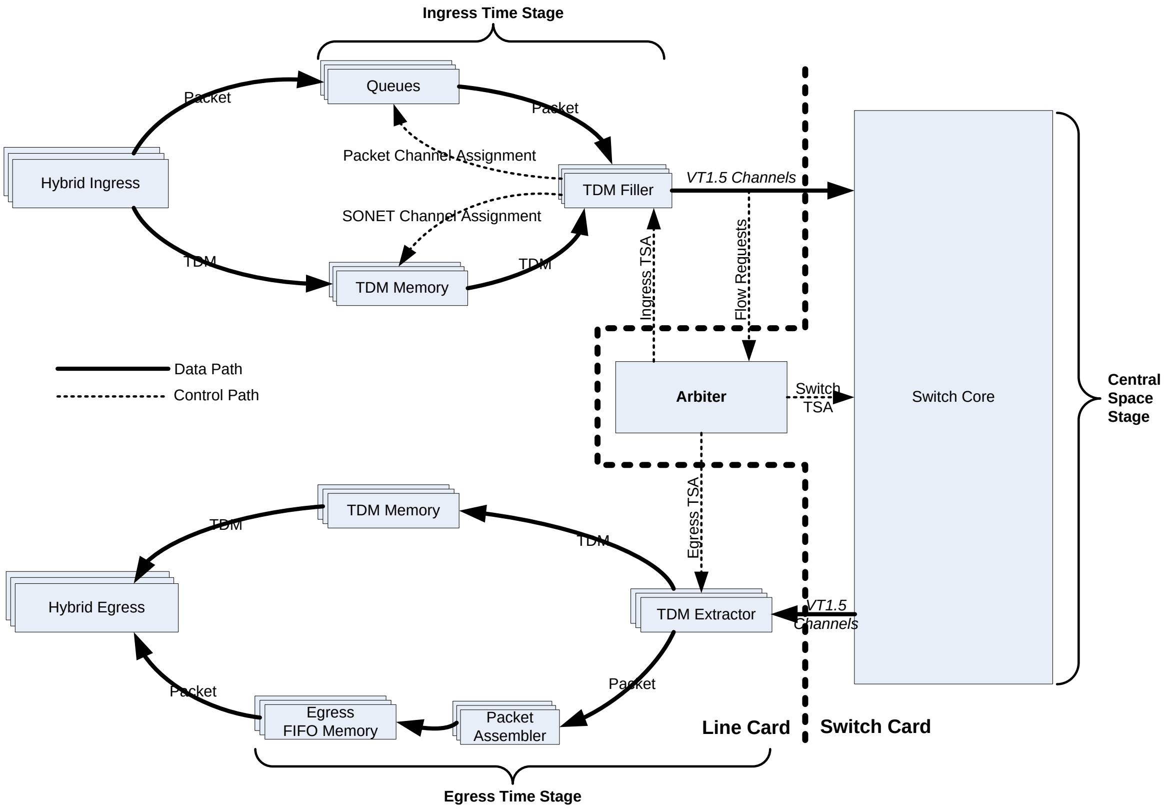


Figure 2.3 Block Diagram of Hybrid Switch Architecture

As shown in Figure 2.3, the hybrid switch architecture consists of three switching stages; two time switching stages on the line card and a space switching stage on the switch card. Traffic at the hybrid ingress ports is mapped to VT1.5 time slots within an aggregate SONET STS-12 by the TDM filler module, which is a time switch. The crossbar (switch core) acts as a space switch which connects STS-12 TDM signals from one input port to another output port. The crossbar makes separate connections for each VT1.5. The TDM extractor (the final time stage) reads the traffic carried by egress VT1.5 channels and uses it as a source to extract and reassemble packets and to extract SONET TDM traffic.

At each ingress port, there may be conflicting demands for resources such as buffer space and time through the crossbar. A control scheme for resource allocation is provided by the arbiter module. Its main objective is to resolve the conflicts and provide efficient and fair scheduling of these resources. The arbitration enforces three specific goals:

(1) respect for priorities,

(2) fairness among the ports, and

(3) maximum bandwidth utilization and maximal throughput.

By enforcing the above goals, arbiter maximizes bandwidth utilization and meets the QoS requirements as closely as possible.

2.3 Data Path

The solid lines in Figure 2.3 represent the data path. Each packet arriving at an ingress port is classified and placed in the appropriate ingress buffered queue based on its class and its destination. Each packet with distinct class and destination represents a separate flow. There are *NxC* distinct flows at each ingress port. The TDM (SONET) traffic is simply put in a SONET memory and each VT1.5 column of the SONET traffic represents one separate flow.

In each arbitration cycle each packet flow is assigned some set of VT1.5’s to carry packets from ingress to egress port. The collection of VT1.5's in aggregate form a byte stream into which the packets are fed in priority order. In each arbitration period a new set of VT1.5s may be allocated to any flow. The switch must be capable of hitlessly changing from one set of VT1.5's to the next set, without disrupting the traffic flow. For each output port, there is a choice of which of several input ports it will connect to. For each output grant, the crossbar arbitration logic selects one of the inputs and blocks the others. The TDM extractor on the output side selects the channel, reads the byte stream, and sends the data into the appropriate packet assembler. Once a full packet is assembled, it is sent to a FIFO buffer which is then read by the egress port. The packet assembler begins assembling the next packet with the next arriving byte.

For switching SONET TDM traffic, the switch carries out an N port SONET STS-12 time-space-time (TST) switch with switching occurring at every VT1.5 frame. The switch fabric exchanges the aligned STS-12 data streams at VT1.5 granularity through TST stages. Both ingress and egress time switch (TDM filler and TDM extractor) perform time slot interchange on the data stream while the space switch stage switches data from one SONET pipe to another. Each time slot is switched independently in the space switch stage. Switch control memories (TDM filler, TDM extractor and switch set up) are organized into pages of control words, which determine what permutations are implemented for each of the 336 VT1.5 positions per SONET STS-12 frame (in time) at each of the N ports (in space) for the switching stages. The switched TDM traffic is extracted and stored in TDM memory at the egress side and then transmitted through the egress port.

No frequency speed up is assumed for this switch architecture. As we shall see in the chapters on performance results and conclusions, some speed up is required in practice.

2.3.1 Class of Service (CoS)

CoS is a way of managing traffic in a network by grouping similar types of traffic together and treating each type as a class with its own level of service priority. It enables more predictable traffic delivery of priority data across IP-enabled networks. Our hybrid switch supports three classes of services: first (high priority), second (medium priority) and third (low priority).

The high-priority class supports strictly bounded delay and jitter applications such as VoIP and real-time video applications with non-preemptive bandwidth guarantees. The medium-priority class supports applications that are less sensitive to delay and jitter such as non-real-time video and VPN services. User-provisioned committed information rate (CIR) and extended information rate (EIR) guarantees are supported for different service requirements. The low-priority class of service supports best-effort applications that do not need bandwidth guarantees such as consumer Internet access via the IP protocol suite. With three unique classes of service, service providers can offer differentiated services with different price points to satisfy different customer requirements.

2.4 Arbiter

An *NxN* hybrid switch contains *NxNxC* queues at the ingresses which store different packet traffic flows based on sources, destination, and CoS assignment. Due to the presence of IB-VOQs at each ingress port (Figure 2.2), packets which are destined to different output ports may be transmitted through the switch in any order; regardless of their arrival time at the ingress port. It is the job of the arbiter to decide when the packet from each queue should be sent to the egress port.

Figure 2.4 shows the block diagram of the arbiter. The arbiter examines TDM requests for SONET traffic and flow requests for packet traffic and makes scheduling decisions for every VT1.5 channel, including which input buffers to read, which output buffers to write, and the configuration settings for the crossbar. The SONET control unit (microprocessor) embedded in the hybrid switch (not shown) carries TDM requests to the arbiter while SONET STS-12 Transport Overhead (TOH) and Path Overhead (POH) carry packet flow requests to the arbiter. TDM traffic has priority over all packet traffic and is served first. The remaining VT1.5 channels are allocated by the arbiter to packet traffic.

Channel allocation for SONET TDM traffic is static. All the channels are “open” for TDM requests and they get channels for all of their requests. However, the channel distribution for packet traffic is dynamic in nature and needs a rapid and flexible arbitration scheme in order to respond to the bursty, self addressing nature of packet traffic. In order to perform packet arbitration, the arbiter accepts bandwidth requests for the *NxNxC* input flows from the N ingress ports. These bandwidth requests are stored in a request matrix. The bandwidth requests combine information about current queue size (depth) and current traffic flow at ingress port (recent flow). The Fair Proportional Algorithm (FPA) scheme in the arbiter uses this information to allocate bandwidth to each ingress-to-egress flow. The Time Slot Algorithm (TSA) gives time slot assignments which map bandwidth allocations from the FPA to specific TDM channels. The time slot assignments computed by the TSA are passed to the switch core to configure the crossbar. The time slot assignments are also sent to port cards by embedding them in the SONET STS-12 signal’s overheads. Port cards use this information to fill the ingress TDM channels with packets and to extract packets out of STS-12 frame on the egress side.

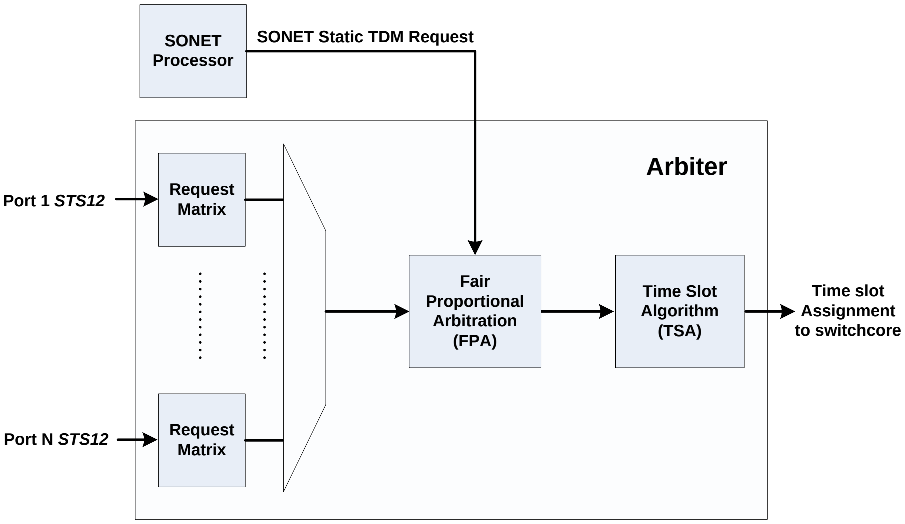


Figure 2.4 Arbiter Block Diagram

Unlike other packet switches which need only one grant per ingress port for full utilization of the channel bandwidth, our hybrid switch must have multiple grants per ingress port, one for each active ingress-to-egress VT1.5 flow. The total grant per port is the number of VT1.5 channels allocated per ingress by the switch's arbiter. Since there are 336 VT1.5 channels per ingress/egress link, we need to have 336 VT1.5 channel allocations per STS-12 frame to fully utilize the switch's bandwidth. Similarly, each output port contends for multiple input ports and there must be 336 VT1.5 channel allocations per STS-12 frame to fully utilize total switch's bandwidth. The arbitration task is thus symmetrical with respect to inputs and outputs. Furthermore, since the arbitration result for each port is dependent on the arbitration for other ports, the arbitration task cannot be performed by separate independent arbiters at the input ports or output ports. In order to maximize performance, the arbitration should result in switch configurations that allow for the maximum number of packets traffic to be transmitted simultaneously. The arbitration process itself must be fast relative to the rate at which packets are received and relative to the latency of packet transmission through the switch.

2.5 Control Paths

The dashed lines in Figure 2.1 represent the control paths. Control paths are used to carry the flow requests from port cards to the arbiter, and time slot assignment information from the arbiter to the port cards and from the arbiter to the crossbar. The control signals (paths) in our hybrid switch have the following main functions:

(1) Control the flow of packets from ingress queues to egress port.

(2) Configure switch crossbar to connect appropriate ingress port to egress port for each of 336xN VT1.5 time slots.

(3) Control egress port to select appropriate TDM channel in STS12 stream for packet and TDM recovery.

2.5.1 Control Path for Carrying Flow Requests

Flow requests from ingress queues are carried to the arbiter in SONET STS-12 signal's overheads. The entire SONET frame is passed to the arbiter module which parses the frame, extracts the flow requests and saves them as a request matrix. The request matrix is used to find the TDM channel arbitration for each flow at ingress port. The arbitration process is described in Section 2.3.

2.5.2 Control Paths for Carrying Time Slot Assignments

Time slot assignments are passed from the arbiter module to port cards by embedding them in SONET STS-12’s TOH and POH bytes being sent to the egress port. The time slot assignments are extracted at egress port card and used for controlling the TDM fillers and the TDM extractors as shown in Figure 2.2. The same time slot assignment at switch card is used by the crossbar (space switch) to cross-connect ingress and egress ports at VT1.5 granularity.

Chapter 3

SONET Signaling, Data Path and Control Paths

This chapter extends Chapter 2 by providing more detail about the data paths and the control signals. It begins with a brief introduction to SONET, and the formation of STS-x signal and VTs which are the basis for carrying data and control information across the switching platform. Section 3.1 describes the SONET frame and signaling. Section 3.2 describes the packet queuing issues. Section 3.3 gives an introduction to the hybrid switch at the architectural level and describes the data path block diagram. Section 3.4 describes the control paths and explains how the control signals are used to control data flow.

3.1 SONET Signaling

SONET is the basis for the transport of packet and TDM traffic in our hybrid switch. Ingress traffic and switch control information is carried via STS-12 signaling. Each SONET STS-12 frame is divided into multiple VT1.5 virtual tributaries (VTs) which provide the data path channels for ingress-to-egress traffic flow. The SONET overheads carry the control information for data flow and switch setup. The following subsections describe the SONET frame and VT sub-structure.

3.1.1 The SONET Frame

Figure 3.1 shows the fundamental SONET frame. This frame is known as a Synchronous Transport Signal, Level One (STS-1). It is 9 bytes tall and 90 bytes wide for a total of 810 bytes of transported data including both user payload and overhead. The first three columns of the frame are the Section and Line Overhead (SOH and LOH), known collectively as the Transport Overhead. The bulk of the frame itself, to the left, is the synchronous payload envelope (SPE), which is the container area for the user data that is being transported. The data, previously identified as the payload, begins somewhere in the payload envelope. The actual starting point will vary. The Path Overhead (POH) begins when the payload begins; because it is unique to the payload itself, it travels closely with the payload. The first byte of the payload is the first byte of the Path Overhead.

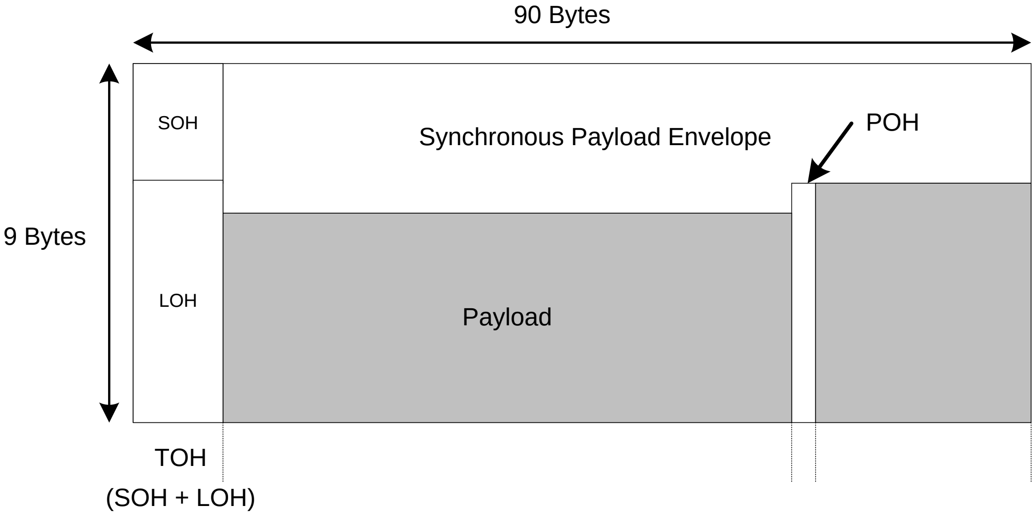


Figure 3.1 SONET Frame

3.1.2 SONET Bandwidth

The SONET frame consists of 810 bytes, and like the T-1 frame, it is transmitted once every 125µs (8000 frames per second). This works out to an overall bit rate of 810 bytes/frame x 8 bits/byte x 8000 frames/second= 51.84 Mbps, the fundamental transmission rate of the SONET STS-1 frame which is slightly more than a 44.736 Mbps DS-3, a respectable carrier level by anyone’s standard. The basic structure of the STS-1 frame is repeated for the higher rates. Three STS-1 frames are multiplexed to create the STS-3 which has a bandwidth of 155.52 Mbps, these in turn are multiplexed to create the STS-12, and so on. If the payload requires less than 51.84 Mbps, the Synchronous Payload Envelope (SPE) is subdivided into smaller components, known as *virtual tributaries*, for the purpose of transporting and switching payloads smaller than the STS-1 rate [3].

3.1.3 The STS-N Frame

In situations where multiple STS-1s are required to transport more payload information, SONET enables the creation of what is called STS-N frames, where N represents the number of STS-1 frames that are multiplexed together to create the frame. If three STS-1s are combined, the result is an STS-3. In this case, the three STS-1s are brought into the multiplexer and byte interleaved to create an STS-3, as shown in Figure 3.2. In other words, the multiplexer selects the first byte of frame one, followed by the first byte of frame two, followed by the first byte of frame three. Then it selects the second byte of frame one, followed by the second byte of frame two, followed by the second byte of frame three, and so on, until it has built an interleaved frame that is now three times the size of an STS-1: 9x270 bytes instead of 9x90 and is still generated 8000 times per second. The technique described above is called a single stage multiplexing process because the incoming payload components are combined in a single step. A two-stage technique is also commonly used. For example, an STS-12 can be created in two ways. Twelve STS-1s can be combined in a single stage process to create the byte interleaved STS-12; alternatively, four groups of three STS-1s can be combined to form four STS-3s, which can then be further combined in a second stage to create a single STS-12 [4].

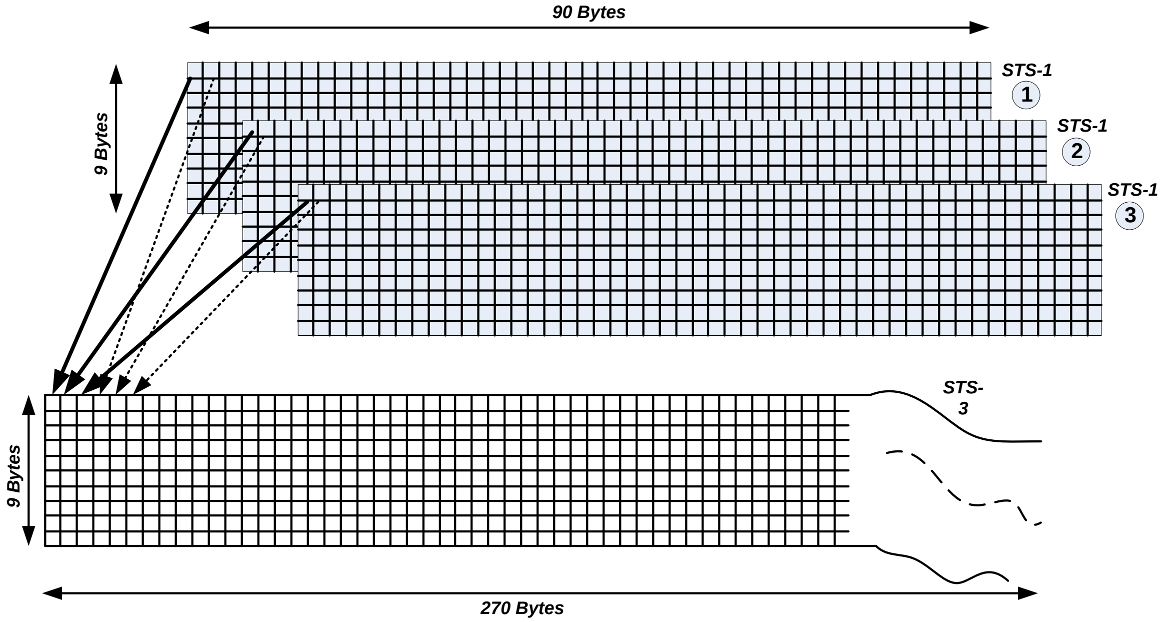


Figure 3.2 Formation of STS-3 frame by byte interleaving

3.1.4 SONET Virtual Tributaries

When a SONET frame is modified for the transport of sub-rate payloads, it is said to carry virtual tributaries (VTs) in which the payload envelope is chopped into smaller pieces that can then be individually used for the transport of multiple lower-bandwidth signals.

To create a VT, the SPE is subdivided. An STS-1 comprises 90 columns of bytes, four of which are reserved for overhead functions (Section, Line, and Path). This leaves 86 for actual user payload. To create virtual tributaries, the payload capacity of the SPE is divided into seven, 12-column pieces called “VT groups”. This leaves two unassigned columns which are called “fixed stuff”. Each of the VT groups can be further subdivided into one of four different VTs to carry a variety of payload types, as shown in Table 3.1 [4].

Table 3.1 SONET Virtual Tributaries

VT Type	Columns/VT	Bytes/VT	VTs/Group	VTs/SPE	VT Bandwidth (Mbps)
VT1.5	3	27	4	28	1.728
VT2	4	36	3	21	2.304
VT3	6	54	2	14	3.456
VT6	12	108	1	7	6.912

3.2 Packet Queuing Issues and Virtual Output Queue

The two main tasks involved in switching are: scheduling and data forwarding. Scheduling involves deciding for each input port which output port the packet should be sent to and arbitrating when more than one input port requests for the same output port while data forwarding involves sending the packet data from input ports to output ports according to the scheduling decision. Data is generally stored in a queue before sending it to output ports.

In general, packet switches can be divided into two categories: output queued (OQ) switches and input queued (IQ) switches, based on where delayed packets are queued. A typical OQ switch has a first-in-first-out (FIFO) queue at each output port to buffer the packets destined to that output port. OQ switches are shown to be able to achieve unity throughput and can easily meet different QoS requirements, such as delay and bandwidth, by applying various scheduling algorithms [5]. However, since there is no buffer at the input side, if the packets arriving at different input ports are destined to the same output port, all the packets must be transmitted simultaneously. Therefore, in order for OQ switches to work at full throughput, the switching speed of the internal fabric and the receiving speed of the output port must be N times faster than the sending speed of the input port in an N x N switch. This speedup requirement makes OQ switches difficult to scale. In particular, when the switch has a large number of input ports or the speed of a single input port increases to Gb/s, it is impractical to achieve the N speedup [6]. On the other hand, for IQ switches, the switching fabric and the output port only need to run at the same speed as that of the input port and, therefore, IQ switches have been the main research focus of high speed single stage switches. The single input queued switch, has a FIFO queue at each input port to store the incoming packets waiting for transmission. Since only the packet at the HOL of each input queue can participate in the scheduling, the packets behind the HOL packet suffer from so called “head of line” blocking, which means that, even though their destination output ports may be free, they cannot be scheduled to transfer because the HOL packet is blocked [7].

An efficient yet simple buffering strategy to avoid HOL blocking is to adopt a multiple input queued switch structure, which was introduced in [8]. A typical multiple input queued switch has a separate FIFO queue corresponding to each output port. It is the IB-VOQ structure since each queue stores those packets which have arrived at a given input port and are destined to the same output port. HOL blocking is eliminated because a packet cannot be held up by a packet ahead of it that goes to a different output. It is known that the VOQ switch structure can achieve 100 percent throughput for all independent arrival processes by using the maximum weight matching algorithm [9] or by using other maximum matching algorithms with speedup [10], [11], [12], [13].

3.3 Data Path Architecture

The hybrid switch is configured as a circuit switching platform which can switch both packets and TDM traffic. As described in Chapter 2, the switching process starts at the ingress port where traffic is differentiated into packet traffic and TDM traffic. Packet traffic is switched by request-grant process which is described in more detail in Chapter 4. SONET TDM traffic is switched by static channel allocation process which is also described in the next chapter. The switched TDM traffic and packet byte streams are extracted, assembled and transmitted through egress port. The following sub-sections describe the data path blocks of the hybrid switch.

3.3.1 Ingress Ports

Our experimental hybrid switch architecture is optimized for 32 ports. Each port supports 622 Mbps link bandwidth which is organized as 336 VT1.5 channels for carrying data from ingress to egress. Therefore, the aggregate duplex bandwidth of the switch is 19.9 Gbps (19.9 Gbps ingress, 19.9 Gbps egress). The communication backplane is SONET structured network. The packet traffic coming to the ingress port is differentiated according to class and sent to their appropriate queues (VOQs) and the TDM traffic is sent to a SONET memory.

3.3.2 Ingress Packet Queues

Figure 3.3 shows the ingress packet queues. There are *NxC* separate queues at each input port. Packets coming to the ingress port are stored in queues before they are transmitted to the egress port. Packets in the ingress queue are sent to the egress port through the crossbar in a controlled fashion by the arbiter. Scheduling of the crossbar and the arbitration of packet flows in queue are based on the queue depth and current traffic flow in the queue, as described in detail in Chapter 4.

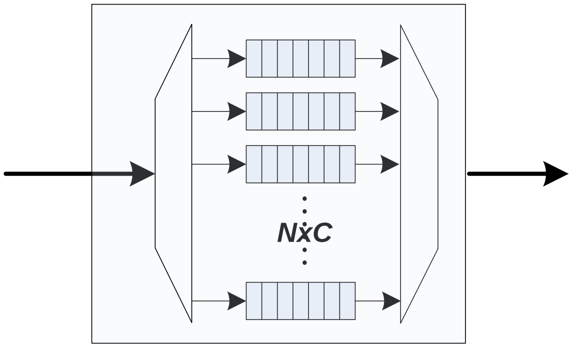


Figure 3.3 Virtual Output Queues for N destinations and C classes

3.3.3 TDM Filler

Figure 3.4 shows the block diagram of the TDM filler. The ingress (and egress) communication link is divided into 336 (28x12) VT1.5 TDM channels. Each channel can carry 9 bytes/column x 3 columns = 27 bytes. The TDM filler receives traffic from the ingress queues (VOQs and SONET memory) and fills ingress VT1.5 channels to carry data to the switch core. The channel assignment for each queue can change on any STS-12 frame boundary and it’s the job of the arbiter to make these changes to the TDM filler/extractor at a time. For the same priority traffic if more and more traffic are coming to the ingress port, the arbiter allocates more and more bandwidth to that flow.

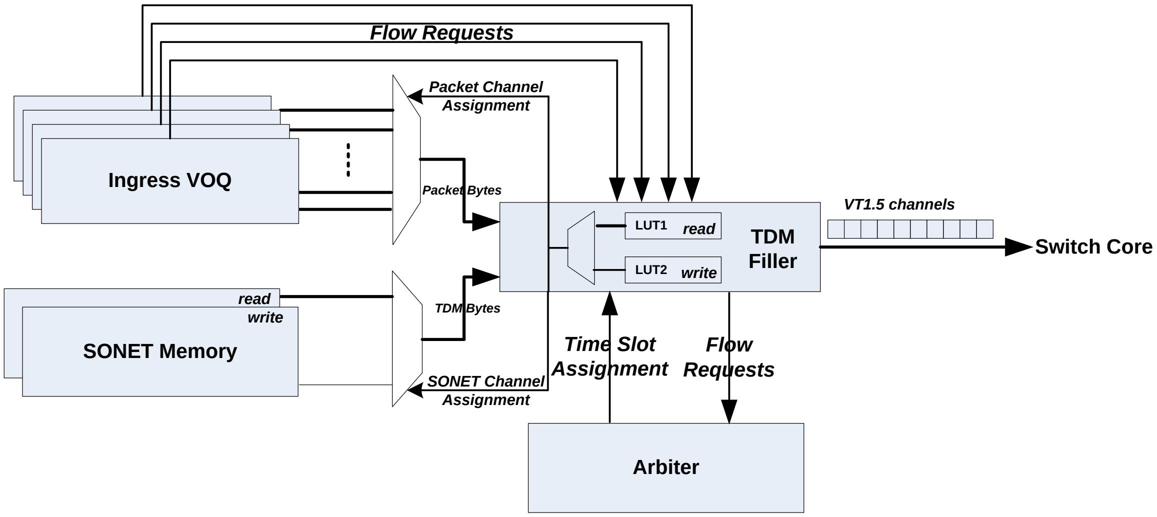


Figure 3.4 TDM Filler Block

There are *N* TDM Fillers, one per ingress port. The channel mapping decision for each TDM filler is based on the switch's arbitration process. The arbitration process generates time slot assignments that define which flow should use which channel at a time. The slot assignment is stored in a look-up table (LUT). There are two LUTs per TDM Filler block; one is used for writing TDM filling instruction for the next arbitration cycle and the other LUT is used for current VT1.5 channel filling instructions. In the next arbitration cycle, LUT roles are swapped.

3.3.4 Switch Core

Figure 3.5 shows the diagram of the switch core. The switch core located at the switch card connects all switch input ports to the output ports. It is the switch pipeline root, because it is the source of configurations that trigger the transmission of packets/TDM cells throughout the crossbar. The switch core is made up of a scheduler and a crossbar. The scheduler implements algorithms that provide efficient use of the crossbar bandwidth and the crossbar connects each ingress port to the egress port as directed. The crossbars and the arbiter connect to each port via high-speed serial links.

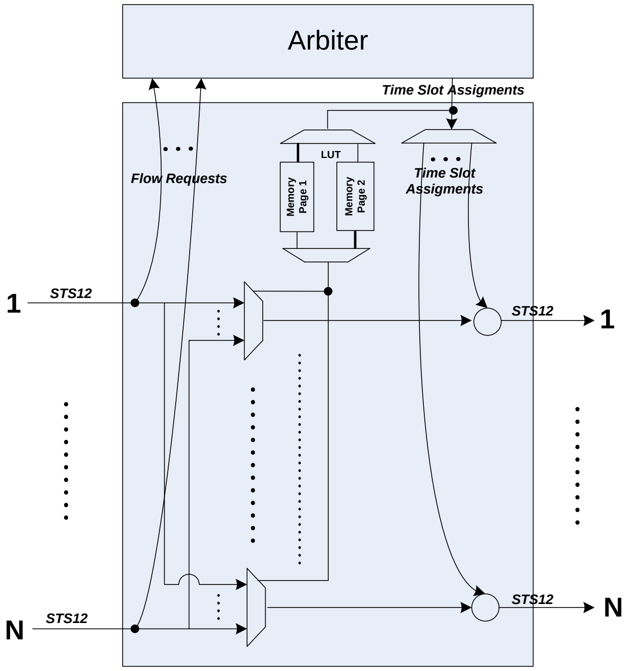


Figure 3.5 Switch Core Block

The digital crossbar in the switch core makes connections between the ingress ports and the egress ports. The arbiter configures the crossbar with time slot assignments generated by TSA module. The time slot assignments generated by TSA are stored in a LUT to be used in the next arbitration cycle. There are two LUTs; one for reading crossbar configuration and another for writing new configuration from arbiter. In every arbitration cycle the LUT's roles are swapped.

3.3.5 TDM Extractor

The TDM Extractor, as show in Figure 3.6, is used to extract packet and TDM traffic from STS-12 stream coming out of the switch core. The channel extraction information, which is stored in LUTs, is provided by the arbiter. The payload in the SONET STS-12 frame needs to be extracted and assembled at the egress port. Like in the TDM Filler and the Switch Core, there are also two LUTs in each TDM Extractor block (LUT1 and LUT2 in Figure 3.6); one for reading and one for writing. In each arbitration cycle, their roles are swapped.

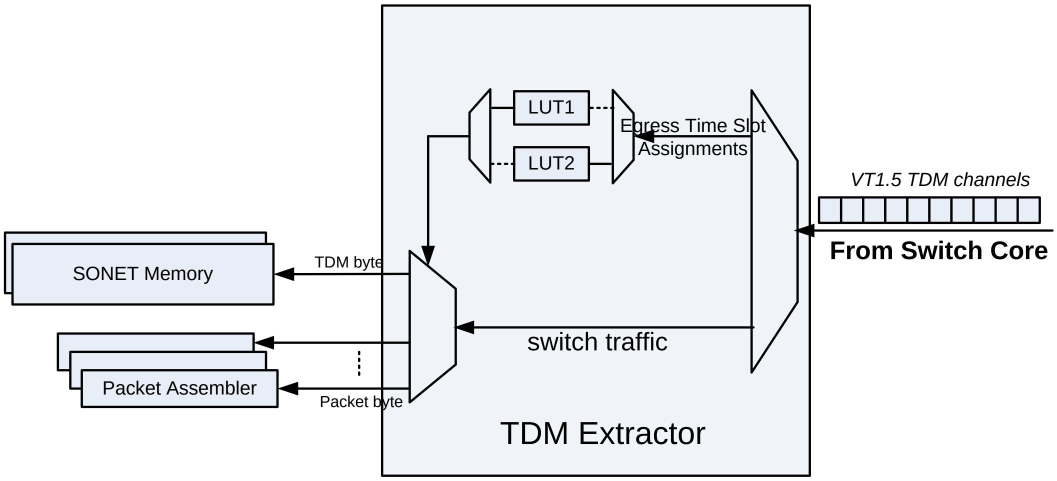


Figure 3.6 TDM Extractor Block

In order to extract the data from SONET signal, the extractor must be synchronized with the SONET frame. For synchronization, the TDM extractor monitors the incoming SONET signal and looks for the frame boundary signals (A1, A2). Once the boundary is found, it differentiates SONET data into control data (ingress and egress time slot assignments which are appended to SONET header) and payload data (actual traffic). The payload data is extracted from the VT1.5 channels and forwarded to corresponding traffic handler where as the control data is used to fill-up TDM filler/extractor LUTs as shown in Figure 3.6.

3.3.6 Packet Assembler

Before IP packets are sent to the output buffers, the packet's byte chunks which are carried by the SONET frame in the form of TDM cells (in SONET STS-12 payload) need to be assembled to re-form the original packets. Packet assembler modules are used for this purpose.

Packets from ingress VOQs are filled in VT1.5 channels by TDM filler and since one VT1.5 channel can accommodate only 9x3 =27 bytes, the egress port may have to wait for other channels in order to get all the bytes of a packet. During the waiting period, the transmitted bytes are stored in a memory inside the packet assembler. Once all the bytes of a packet arrive at the assembler, it is sent to an output FIFO memory as a single packet and then stored in a FIFO memory as shown in Figure 3.7.

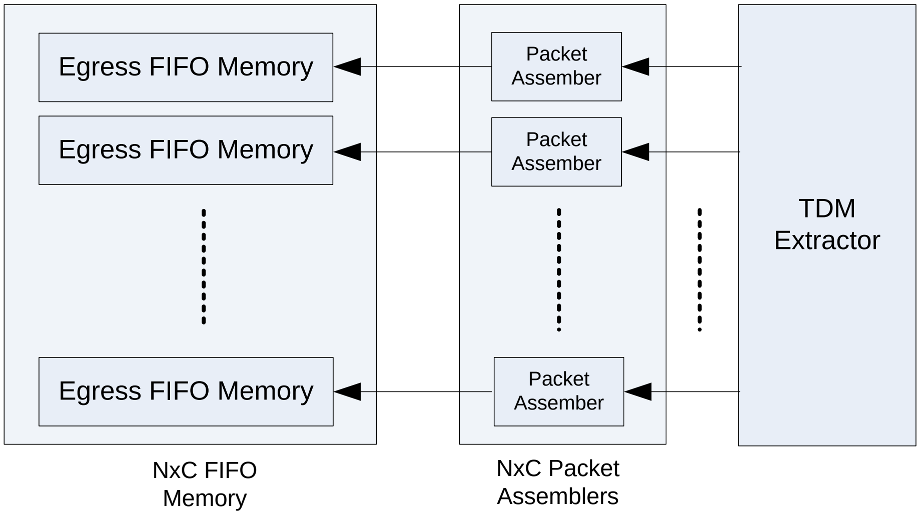


Figure 3.7 Packet Assemblers and FIFO Memory Blocks

3.3.7 Egress Packet FIFO

The assembled packet from the packet assembler is stored in egress FIFO memory (Figure 3.7). There are NxC FIFO memories, one for each source and class combination. The egress port transmits these packets to the backplane links.

3.4 Control paths

Our hybrid switch is equipped with data path and control signaling paths for cooperating with arbiter and other data path modules for high throughput packet routing. There are several control paths in the switch that carries control information. Following are the main control signals the control path carries:

(1) *Flow Request* from queues to the arbiter

(2) *Ingress Time Slot Assignment* from arbiter to port card.

(3) *Egress Time Slot Assignment* from arbiter to port card.

(4) *Switch Time Slot Assignment* from arbiter to crossbar.

(5) *Packet Channel Assignment* from TDM filler block to queue block.

(6) *SONET Channel Assignment* from TDM filler block to SONET memory.

3.4.1 Flow Request

In hybrid switch the arbiter allocates channel(s) for a particular flow at ingress port based on the channel requests for that particular flow. The Flow request is the number of VT1.5 channels requested by a queue at ingress port. It is calculated based on the current queue depth and recent traffic flow at the ingress port. The following equation is used to calculate the flow request:

$$BR = Q/N + F \tag{3.1}$$

Where,

BR	= Flow Requested.
F	= Recent Traffic Flow.
Q	= Current Queue Depth.
N	= Number of arbitration cycles to bring Q to zero.

The requested bandwidth information is carried to arbiter in SONET headers which are used by arbiter for crossbar scheduling and packet arbitration purposes.

3.4.2 Ingress Time Slot Assignment

The ingress time slot assignment signal carries the ingress link channel allocations information from the arbiter which tells the TDM filler how the ingress VT1.5 channels should be filled by ingress traffic. There are 336 VT1.5 channels in each ingress communication link and each of these channels may be carrying either packet traffic or TDM traffic or simply not carrying any traffic at all. Ingress TSA signal tells the TDM filler whether or not the channels are to be filled with packet traffic from ingress VOQ or from the SONET memory. The channel allocation is computed by the arbiter and the allocation result is carried to the port card by embedding it into SONET STS-12 (aggregation of 336 VT1.5 channels) signal's path and transport overheads.

3.4.3 Egress Time Slot Assignment

The egress time slot assignment signal from the arbiter module carries the information about how the communication channel's bandwidth is distributed among all the ports. This information is used to extract the TDM and packet traffic from the link and send them to their corresponding traffic handler. The egress time slot assignment is computed by TSA block in the arbiter and the signal is carried to the port card by embedding it into SONET STS-12 signal's path and transport overheads.

3.4.4 Switch Time Slot Assignment

The crossbar in the switch core is used to connect multiple inputs to multiple outputs at a time. A standard problem in crossbar switches is that of setting the cross-points when the particular ingress should be connected to specific output. In our hybrid switch the cross-points are closed and opened according to the time slot assignments provided by the arbiter. When the crossbar is enabled, the input is connected to the output and the traffic carried by VT1.5 channels in ingress communication link is switched to VT1.5 channels in egress communication link.

3.4.5 Packet Channel Assignment

The TDM filler divides ingress time slot assignments into two parts, one to allocate VT1.5 channels for packet traffic, and another to allocate channels for TDM traffic. Packet channel assignment is a control signal from the TDM filler to the ingress queue module that carries the information which queue should use which VT1.5 TDM channel to send packets to the egress port. This signal controls the flow of packets from ingress queue to the egress ports.

3.4.6 SONET Channel Assignment

This is a signal from TDM filler to the SONET memory that map is used to map the SONET TDM traffic directly into ingress VT1.5 TDM channels. This is also computed by the arbiter module.

Chapter 4
Arbiter Bandwidth Allocation

The arbiter controls and allocates access to the data path by controlling and allocating the shared resources. In our hybrid switch the shared resources are the ingress TDM channels, the egress TDM channels, and the crossbar paths. This chapter describes the resources, arbitration algorithms, design alternatives, implementation issues, and the timing results of the arbiter.

4.1 Introduction

The arbiter provides a mechanism to control and regulate the flow of traffic from the ingress ports to the egress ports. It manages the ingress traffic by distributing the available bandwidth resources fairly among ports and satisfying customer’s QoS requirements. This is accomplished by differentiating traffic in priority order. In order to perform the arbitration, the arbiter resolves bandwidth requests from the ingress ports according to some specifications. These specifications are defined in Section 4.3. The arbiter’s goal is to find an optimal set of switch settings through solving these requests.

4.2 Arbitration Goal

The goal of arbitration is to allocate bandwidth to all ports fairly, efficiently, and in a reasonable time. In order to achieve this goal, the arbiter allocates the switch’s bandwidth to the *NxNxC* flows according the following principles:

(1) priority of the packet flows,

(2) fairness between ingress ports and between egress ports, and

(3) efficient bandwidth utilization.

4.2.1 Priority

Our hybrid switch respects the customer's QoS requirement by allocating available bandwidth according to traffic types. Three classes of service are supported by the switch and each class represents traffic with different QoS requirements. Flow (bandwidth) requests from the ingress ports are served in strict order of priority classes. The higher priority traffic is allocated as much bandwidth as possible first and the residual bandwidth is passed on to the next lower priority traffic.

4.2.2 Fairness

The arbiter maintains fairness among all the ingress and all the egress ports by assigning bandwidth in proportion to their bandwidth requests, i.e., the channel requesting more bandwidth gets the higher bandwidth while the channel requesting less bandwidth gets the lower bandwidth. This fairness is accomplished by applying proportional bandwidth allocation scheme which is discussed in the next section.

4.2.3 Efficient Bandwidth Utilization

Our switch is bandwidth limited. A central issue in our hybrid switch is how to allocate this limited bandwidth to ingress flows efficiently while maintaining the fairness. The switch utilizes bandwidth efficiently by regulating the traffic on "per flow" basis. Our communication link is divided into multiple VT1.5 channels. The channel allocation is done by finding all the available channels first, and then assigning these channels to the flows. This scheme is efficient because bandwidth allocation for any flow may be realized along any available free channels and as long as there is any open (unassigned) channel. This process is called TSA which is discussed in the next section. The process of finding open channels and assigning them to different traffics in priority order forms a network policy that alleviates the application's performance and optimizes the bandwidth utilization.

4.3 Arbiter Block

Figure 4.1 shows the block diagram of the arbiter module.

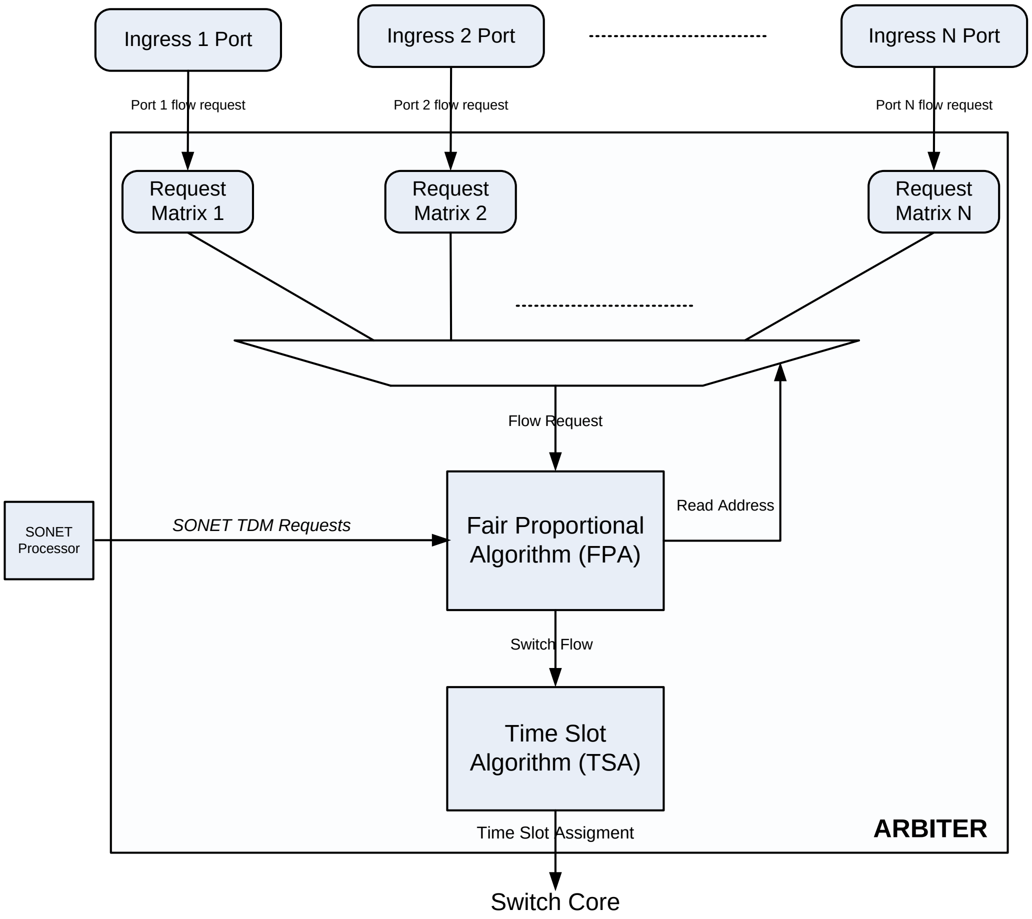


Figure 4.1: Arbiter Functional Block Diagram

The inputs to the arbiter are *flow requests* from port cards and *TDM requests* from SONET processor. The output from the arbiter is a set of time slot assignments which are passed to the switch core's LUT and to the port cards by appending them to SONET STS-12 frames. The entire arbitration process is completed in the following five steps:

(1) read TDM flow requests,

(2) allocate VT1.5 channels for TDM flow requests,

(3) read packet flow requests,

(4) allocate VT1.5 channels for packet flow requests, and

(5) update crossbar setting for next arbitration cycle and send time slot assignments in (2) and (4) to port cards.

These steps are described in reference to the flow chart in Figure 4.2. SONET TDM traffic has priority over packet traffic and thus bandwidth is allocated to TDM traffic first. The remaining bandwidth is allocated to packet traffic using a FPA scheme. In FPA, the flow request matrix is viewed separately as a set of column requests and a set of row requests. The arbitration is carried out in two separate steps: arbitration by column in step 1 and arbitration by row in step 2. In the first step, our arbitration algorithm checks the flow requests of each element of the matrix along columns and assigns bandwidth proportional to the request. This represents assigning bandwidth along the egress port so the bandwidth is fairly distributed along the egress port without violating the bandwidth threshold (336 VT1.5 channels). Similarly, in the second step bandwidth assignment is done along the "row" of the request matrix. This step will prevent bandwidth over-allocation on the ingress ports.

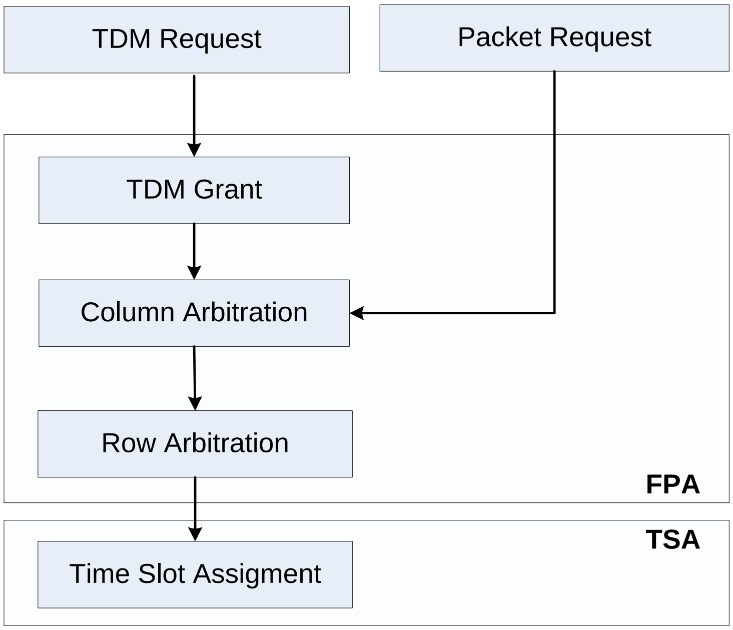


Figure 4.2 Arbiter Flow Chart

The bandwidth assigned to each flow using FPA must be assigned VT1.5 TDM channels such that there is no any ingress-to-egress violation; i.e. no more than one ingress flow should send traffic in one particular VT1.5 TDM channel and no more than one egress channel should get data from one particular VT1.5 channel. The process of “conflict free” slot assignment is done by time slot algorithm which is described in Section 4.3.3. After the time slot assignment by TSA, the arbiter’s decisions are then sent to the port cards and the switch core LUTs for crossbar configuration.

In our hybrid switch, the arbiter tries to find an “optimal” switch setting. However, the optimal result described here is not a “perfect” solution. The arbiter described here decomposes the arbitration process into two steps: arbitration among conflicting column requests (egress) followed by arbitration among conflicting row requests (ingress) that have won in the first step. In each step, the highest priority requests are served first in order to meet the QoS requirements. This solution is not optimal in the sense of finding perfect matches in switch settings because the arbitration in the second step may further reduce the optimal switch setting we found in the first step leaving switch capacity unused. In order to achieve the optimal result, arbitration would have to be an iterative process between step 1 and step 2 with alternating increasing and decreasing of individual request number until the optimal settings are found. This iterative process is heuristic in nature and may take a long time to converge. Instead, the arbiter described here is motivated by the need to find useful switch settings very quickly. In the other words, in this application there is an important trade-off between optimality and speed. A slow, optimal solution would be much less useful than a fast good but not-so optimal solution. This is discussed in more detail in the later parts of this chapter.

4.3.1 Request Matrix

Bandwidth (flow) requests from each traffic flow at the ingress ports are carried to the arbiter by appending the requests to SONET STS-12 overheads. The arbiter unit extracts these requests and saves them in bandwidth request memories in the form of a matrix. These requests are used to calculate the bandwidth different distribution to different ingress flows during the next arbitration cycle.

The bandwidth request for each ingress flow is calculated by using the following equation:

$$BR = Q/4 + F \tag{4.1}$$

Where,

BR = Bandwidth Requested (no. of VT1.5 channels)

Q = Queue Depth (no. of VT1.5 channels)

F = Recent Flow, i.e. total traffic in last arbitration period (no. of VT1.5 channels)

Equation 4.1 says the bandwidth requested is a function of queue depth and recent flow. The current queue depth (Q) and recent flow (F) are initially in the unit of bytes, but converted to the unit of VT1.5 by dividing it with number of 1B slots in one VT1.5 per SONET frame, which is 27.

The arbitration problem in the hybrid switch can be described as a problem of solving a matrix of requests from ingress ports, each one for access to an egress port via a cross-point of the crossbar. The priorities of the arbitration scheme include maintaining high link utilizations, small queuing delays, and fairness among competing sources. One way to achieve high utilization and low queuing delay is to vary the flow rate as a function of the queue length [14]. In our hybrid switch, the flow rate is a function of queue length; and it is varied by dividing the queue length into 2 parts: recent flow (F in bytes) and current queue depth Q (in bytes). The current traffic flow represents the amount of packet flow (in bytes) coming to the ingress port during last arbitration cycle and current queue depth represents total queue length minus current traffic flow. The above equation is derived from our need of having a switch that is very quick to react to the current traffic conditions while offering steady packet flow to the network. The switch tries to allocate bandwidth such that all the traffic flow that has accumulated during last arbitration cycle is drained in next arbitration cycle while the original queue depth is drained in four arbitration cycles (500 µs time). The request matrix changes in every arbitration cycle.

4.3.2 Fair Proportional Algorithm (FPA)

This algorithm is used for flow arbitration of packet requests which are stored in the request matrix inside the arbiter unit (Figure 4.1). The bandwidth for TDM requests is preallocated and thus arbitration on these TDM requests is omitted. As described earlier, the FPA arbitration is carried out in two steps: arbitration along column requests followed by arbitration along row requests that have won in the first step. The final result represents the number of VTs allowed for that particular flow request. In order to ensure the QoS requirements the two-step FPA is first applied to the highest priority traffic and then to lower priority traffic. As the arbitration has to be fair to every port and should give equal opportunity, arbiter should avoid one particular ingress/egress port getting more advantage over the other. Both the fairness and QoS requirement are fulfilled by allocating bandwidth proportionally. Proportional bandwidth granted for all the requests is calculated using the following equation:

$$BG = BR \text{ x } BA/SR \qquad (4.2)$$

Where,

BG = Bandwidth Granted for each flow request.

BA = Bandwidth Available (along ingress or egress port)

BR = Bandwidth Requested

SR = Sum of Bandwidth Requested (along ingress or egress port)

Table 4.1 Request Matrix for Class 1 Traffic

	Request Matrix					**Ingress SR**	**Ingress BA**
	120	**120**	**130**	**50**	Σ	**420**	**336**
	80	**50**	**140**	**120**	Σ	**390**	**336**
	90	**75**	**130**	**60**	Σ	**355**	**336**
	65	**70**	**125**	**100**	Σ	**360**	**336**
	Σ	Σ	Σ	Σ			
Egress SR	**355**	**315**	**525**	**330**			
Egress BA	**336**	**336**	**336**	**336**			

The process of flow arbitration is illustrated by an example. Table 4.1 shows a typical request matrix for class 1 traffic. Since the highest priority traffic gets access to all available bandwidth and when it does not use up all, the unused portion is shared by all low priority traffic, these requests have all the bandwidth available for grant. The bandwidth arbitration is carried out in the following two steps:

Step 1

Arbitration is carried out along all the columns of the Request Matrix.

Request Matrix ⇨ Column Grant Matrix

120	120	130	50
80	50	140	120
90	75	130	60
65	70	125	100

113	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

120	120	130	50
80	50	140	120
90	75	130	60
65	70	125	100

113	0	0	0
75	0	0	0
0	0	0	0
0	0	0	0

120	120	130	50
80	50	140	120
90	75	130	60
65	70	125	100

113	0	0	0
75	0	0	0
85	0	0	0
0	0	0	0

120	120	130	50
80	50	140	120
90	75	130	60
65	70	125	100

113	0	0	0
75	0	0	0
85	0	0	0
61	0	0	0

using the similar steps for other columns we get the following settings for column arbitration

Table 4.2 Converting Request Matrix to Column Grant Matrix

Request Matrix

	120	120	130	50
	80	50	140	120
	90	75	130	60
	65	70	125	100
	Σ	Σ	Σ	Σ
Egress SR	355	315	525	330

→

Column Grant

	113	120	83	50
	75	50	89	120
	85	75	83	60
	61	70	80	100
	Σ	Σ	Σ	Σ
Egress Grant	334	315	335	330

Table 4.2 shows the bandwidth allocated on the column of the request matrix. The sum of bandwidth allocated is given, which does not exceed the maximum capacity (336 channels).

Step 2

The grant matrix from Step 1 (Table 4.2) is used as a request matrix and the same procedure is followed as in Step 1. This step is to make sure the ingress port has not exceeded the maximum available link bandwidth.

Table 4.3 Converting Column Grant Matrix to Switch Flow Matrix

Column Grant Matrix

113	120	83	50
75	50	89	120
85	75	83	60
61	70	80	100

→

Row Grant Matrix (Switch Flow Matrix)

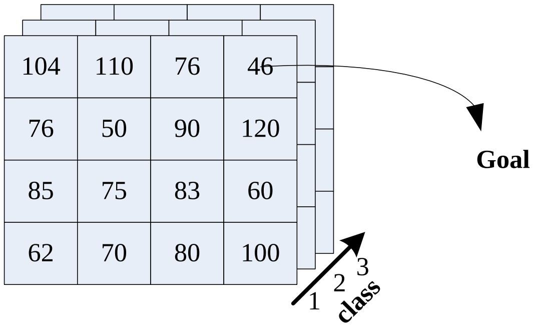
						Ingress Grant
	104	110	76	46	Σ	336
	75	50	89	120	Σ	334
	85	75	83	60	Σ	303
	61	70	80	100	Σ	311
	Σ	Σ	Σ	Σ		
Egress Grant	325	305	328	326		

The row grant matrix shown in Table 4.3 is the *Switch Flow Matrix* which represents the channel allocations by arbiter for different ingress flows. The row of the flow matrix represents the ingress port and the column represents the egress port. The sum of channels allocated at ingress and egress ports should not be more that the actual capacity of these ports i.e. the summations of each row and each column of the Switch Flow Matrix must not exceed 336. The above calculation shows that the arbiter does not violate this rule. This switch flow matrix is used by the TSA block in order to find the time slot assignment for each ingress flow.

4.3.3 Time Slot Algorithm (TSA)

The channel configuration in our hybrid switch is an interconnection pattern such that at most 336 VT1.5 channels can be utilized by an input port and at most 336 VT1.5 channels can be utilized by an output port in a single STS-12 frame time. A time slot conflict occurs if two or more channels from input are switched to the same output at the same temporal boundary. For a given *switch flow matrix*, an optimal TSA is an assignment that has all the conflict free slot assignment for all the elements of the flow matrix.

The time slot algorithm in our hybrid switching system is used to find channel settings (VT1.5 channels) of the links such that as many as possible of the channels are utilized without any conflicts. The process involves reading the switch flow table, finding the open channels, assigning channels to different flows, and sending slot assignment information to the switch core LUT and the port cards. Figure 4.3 shows the switch flow matrix that defines how many TDM channels should be allocated to a flow by the time slot assignment algorithm and Figure 4.4 shows the time slot assignment process.



104	110	76	46
76	50	90	120
85	75	83	60
62	70	80	100

Figure 4.3 Switch Flow Matrix Defining Time Slot Assignment Goal

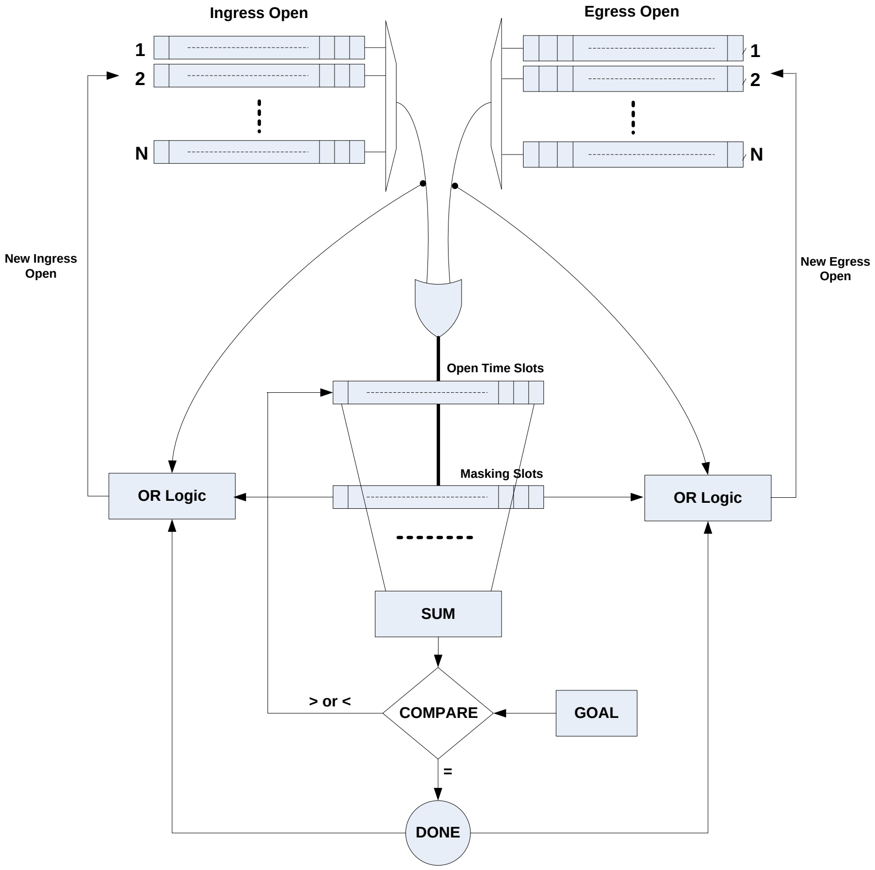


Figure 4.4 Time Slot Assignment Logic Block

4.3.3.1 Goal

The goal for each flow’s channel assignment is the ideal number of channels granted to a flow at an ingress port by the FPA module. The slot assignment algorithm reads the goal from the switch flow matrix and assigns VT1.5 channels to that flow. The time slot assignment process is done per flow basis, starting from high priority traffic to low priority traffic and the *goal* is selected randomly from the *switch flow matrix*.

4.3.3.2 Ingress/Egress Open

Before mapping any “grant” from FPA to TDM time slots, the arbiter needs to know which TDM channels are free at the ingress and the egress. Ingress/Egress open represents the numbers of VT1.5 channels, and their time positions, that are available for TDM channel mapping at ingress and egress port. As shown in Figure 4.5, this is a vector representation of VT1.5 channels in the form of bits. Each bit, if “0”, represents an open channel and if “1” represents an occupied channel. The bit position represents the time position of the VT1.5 channel. For NxN switch, there are N ingress open vectors and N egress open vectors.

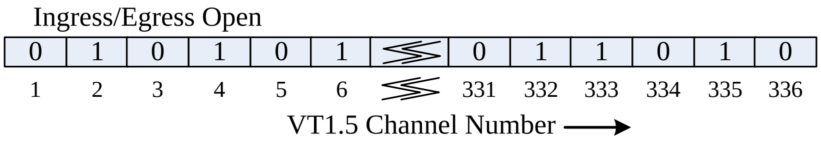


Figure 4.5 Ingress/Egress Open Vector

4.3.3.3 Open Time Slot

Open Time Slot (OTS) is the vector representation of open channels that are common to both, the ingress and the egress ports. OTS shows how many channels are available and how many are occupied and this must be known for making any connection from an ingress port to an egress port. It is calculated by doing an OR operation between ingress open and egress open. The figure below shows the process of finding OTS.

Figure 4.6 shows the OTS formation of OTS vector from ingress open and egress open vectors. The bit value ‘0’ in the OTS vector represents the free channel that can be mapped to carry traffic from ingress A to Egress B, while ‘1’ represents the channel which is occupied by another flow.

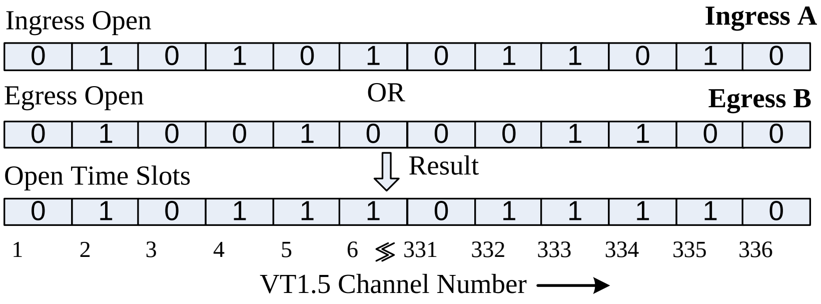


Figure 4.6 Formation of OTS Vector

4.3.3.4 Masking Slots

OTS is the representation of VT1.5 channels which are “available” or “occupied” for possible time slot assignments between particular ingress and egress ports. The number of TDM channels required to satisfy the goal is either greater than, or smaller than, or equal to the available channels represented by OTS. We need to find the open slots such that the time slots assignment is equal to the *Goal* or is as close as possible. The open channels in OTS vector after all the search is completed gives the *Masking Slots* vector. This is called *Masking Slots* because this vector is used to update the *Ingress/Egress Open* once the time slot assignment for a flow is done. Figure 4.7 shows the process of finding the *Masking Slots*.

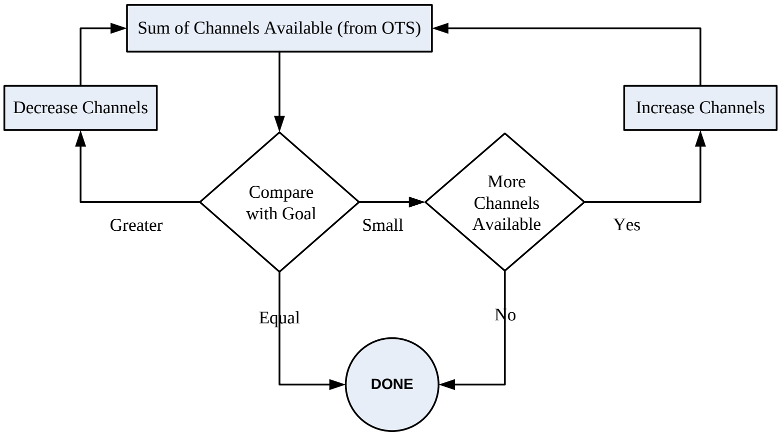


Figure 4.7 Masking Slot Finding Process

For finding the *Masking Slots,* we search the OTS vector such that the open channels are equal to the *Goal* (if there are enough “available” channels) or as close as possible to the *Goal* (if there are not enough “available” channels). This is done in order to avoid any channel over-allocation. The process of finding *Masking Slots* is done by applying *Binary Add Algorithm (BAA)* which is described below.

Binary Add Algorithm (BAA)

This algorithm is used to find the free time slots (VT1.5 channels) in OTS and match the time slots with our Goal. This is accomplished by a binary search.

Figure 4.8 shows a binary tree. A binary tree is made of nodes, where each node contains a left and right pointer and data element. The root pointer points to the topmost node in the tree. The left and right pointers recursively point to smaller sub-trees on either side. The minimum element of an ordered binary search tree is the last node of the left pointer and its maximum element is the last node of the right pointer. Therefore, the minimum and the maximum can always be found by tracking the left child and the right child respectively until an empty sub-tree is reached [15].

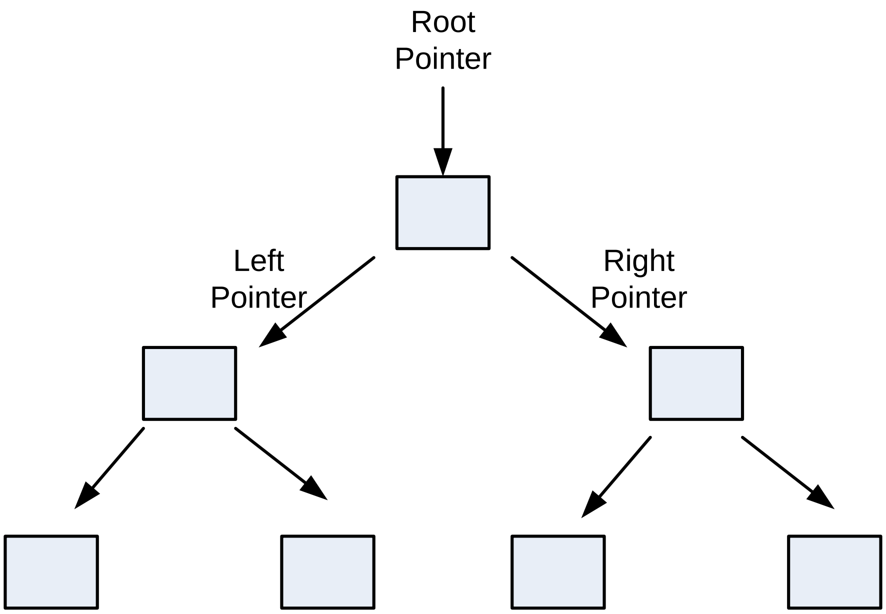


Figure 4.8: Binary Tree

BAA is a binary search process for finding the open VT1.5 channels such that the *Goal* is met without any channel over-allocation. There are 336 VT1.5 channels in each SONET STS-12 link and in order to provide the advantage of binary search, OTS vector of size 512 is used (factor of 2) instead of only 336. The block diagram for this algorithm is shown in Figure 4.9. Two pointers are used; Max and Depth. Max (M) points to the maximum slot number position on the open slots that matches the goal while D represents the incremental/detrimental depth in order to adjust the position of M to exactly match the goal.

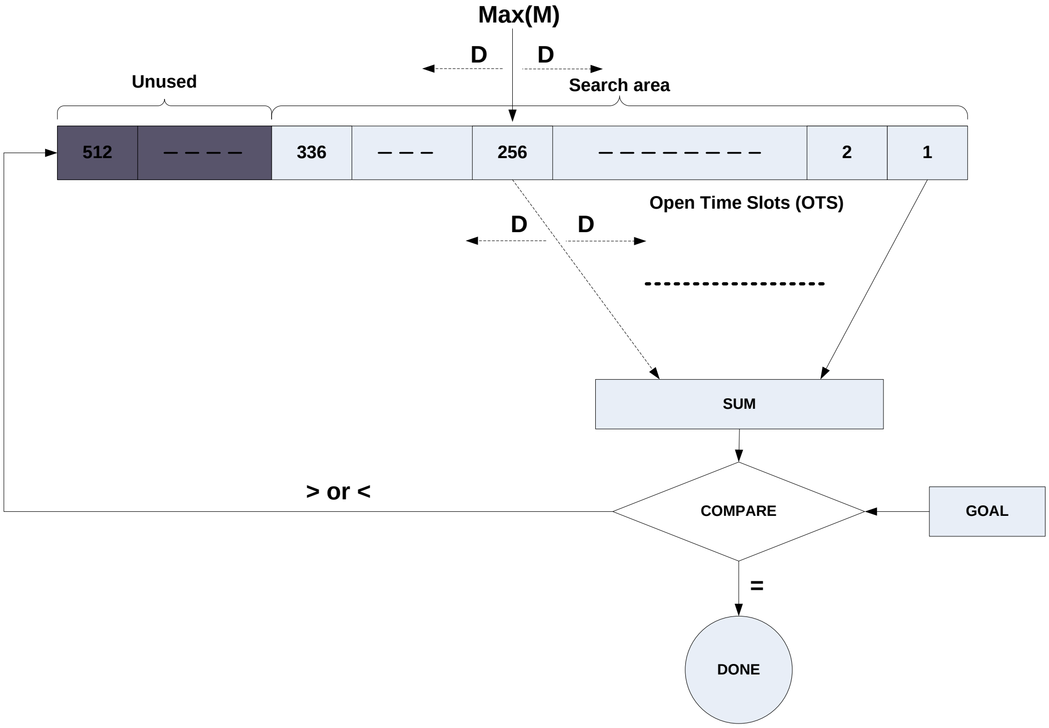


Figure 4.9 Binary Add Algorithm Block

Binary add algorithm for finding the maximal matching conflict free slot assignment from OTS is completed in the following 4 steps:

Step 1

M is chosen to be the middle of OTS (256^{th} slot). The value of D is initialized with 128 (M/2). The sum of open slots (binary "0") from bit position 1 to M is calculated by doing 'NOT' operation on OTS vector and summing all '1's.

Step 2

The current sum is compared with goal. If sum > goal then adjust new M=M-D and new D=D/2 and repeat step 1. Else if the sum is less than goal then make new M=M+D and new D=D/2 and repeat step 1. Else if the sum is equal to goal then TSA is done. Else go to Step 3.

Step 3

If Max (M) >336, set M at 336 because that is the maximum number of channels each ingress or egress port has. Repeat step 2.

Step 4

Any time if D= 1/2, BAA is assumed to be completed because the entire search for open channels has been finished.

All the 0's in OTS vector from channel number 1 to M represents the VT1.5 channels granted for a particular flow. The Masking Slots is found by inverting all the bits of OTS vector from 1 to M and resetting all the other bits (making '0'). All the 1's in Masking Slots represents the channels that are allocated for a particular flow in this arbitration cycle. The row number and the column number of the *Goal* in the *switch flow matrix* give the information about ingress/egress port number: where the traffic is coming from and where it should be routed. Based on all these information the time slot assignment "code" is generated. The code represents which ingress flow to read, in which egress port to send data and which VT1.5 channel carries the information. This code is then passed to the switch core LUT and the port cards. The *Masking Slots* vector is then used for updating the *Ingress Open* and the *Egress Open* OR-ing it with original *Ingress Open* and *Egress Open* vectors and the process is repeated for another *Goal*.

4.4 Arbiter Design Alternatives

There are two design alternatives for the arbiter. The first one is a software approach where arbitration is implemented in software (written in C) and downloaded to a processor implemented on an FPGA (Field Programmable Gate Array) board. The second approach is a hardware approach which is written in a Hardware Descriptive Language (HDL). In hardware approach the arbiter logic is directly downloaded to logic units on an FPGA board.

4.4.1 Software Approach

The software implementation run in the NIOS II soft processor from Altera and the coding is done in C language. The next sub-sections give brief introduction about the NIOS II processor and the software design process.

4.4.1.1 NIOS II Processor

The hybrid switch can be thought of as an embedded system where the arbiter is built to constantly respond to external events and to generate control outputs as a function of their current state and inputs. Embedded-system specification and design consists of two tasks, the first is describing a system's desired functionality and the second is mapping that functionality for implementation by a set of system components such as processors, FPGAs, memories, and buses [16]. Altera's Nios II embedded processor tool provides the platform and tools needed to integrate an entire system on a single programmable logic device (PLD).

The Nios II processor is a soft core processor which offers flexibility, scalability and low absolute cost. Many applications that require moderate performance are fit by the soft cores and they immediately benefit from process enhancements to their target hardware platform. The Nios II processor's parameterizability allows users to make the performance/cost tradeoff quickly, without needing to be a processor architect. Regardless of the configuration, the same instruction set allows Altera to deliver fully-verified cores and industry-standard software development tools such as C/C++ compilers. All the tools necessary to develop embedded designs are provided by Altera, including an industry-standard C/C++ compiler and debugger, peripherals, and drivers, the Quartus II software for design development, and download cables for device programming and verification. These tools provide a system-centric approach to development and allow hardware and software to be created concurrently.

4.4.1.2 Hardware and Software Partitioning

Figure 4.10 is a block diagram of the switch with software and hardware partitioned in the arbiter. In this hardware software co-design, the job of software and hardware are divided as follows: the software does the initialization and configuration (packet arbitration) and the hardware does the buffering of arbitration parameters (flow requests) and communicating TSA configuration to the port cards and the crossbar. In the other word software is used for flow arbitration (FPA unit) and slot assignment (TSA unit) while hardware is used for memory buffers for flow requests and switch flow and logic to pass TSA to port cards and crossbar.

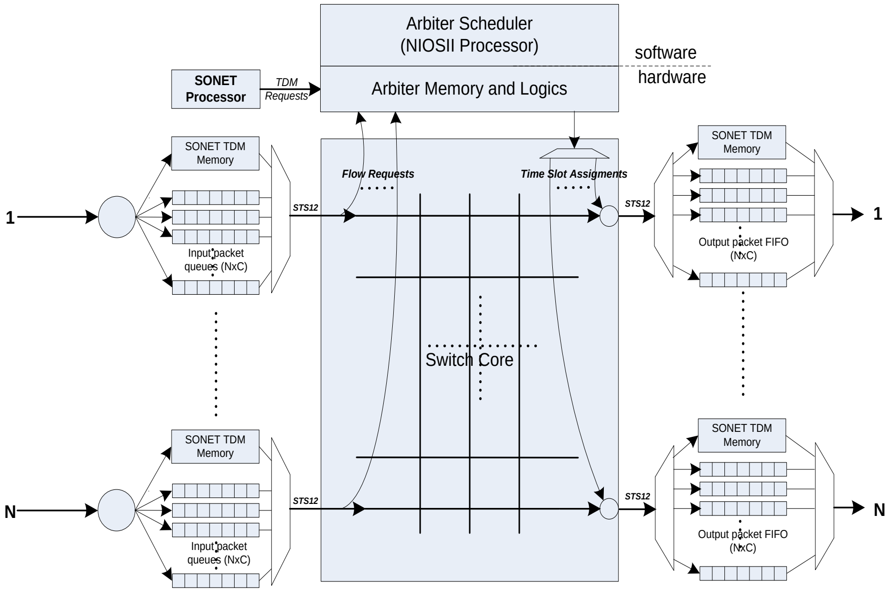


Figure 4.10 Hybrid Switch with Arbiter Software/Hardware Partitioning

4.4.1.3 Software Design Process

The HW/SW co-design process for arbitration can be summarized in three main steps:

(1) writing program in software,

(2) reading requests,

(3) detecting critical software parts, and

(4) hardware/software optimization of the algorithm.

The first step is implementing the algorithm in software. The ANSI C and the assembler programming language are supported by NIOS II IDE (Integrated Design Environment). Generally, the preferable choice is the implementation of the software code using ANSI C. In this way, instead of rewriting the code from scratch, the use of an already existing code for the algorithm, available in NIOS II IDE, shortens the design cycle. The portability of ANSI C allows also the code to be created and tested for functionality on other platforms. Once the software code has been tested for functionality and implemented into the target platform, the performance analysis is applied. For this purpose the bandwidth requests from packet flow and TDM flow are read and the performance is examined. The next step is to check if the required constraints have been met. If not, critical software parts have to be detected and optimized. To have precision on the time processing of the software code corresponding to a focused part, we can either use a logic analyzer to make our measurement directly on the FPGA output pin or use Signal Tap from Altera to capture the signal in PC. The final step is the software code refinement and optimization of critical software parts using hardware description. The general idea is to implement parallel structures in hardware for fastest data processing.

4.4.1.4 Software Arbitration Process

The functional block diagram of the architecture proposed for arbitration is as shown in Figure 4.11. The architecture has three main components: memory unit, controller unit and computational unit (scheduler). The memory unit consists of one dual port RAM per port to store the *flow request* from each port. The controller consists of a round robin counter to control the memory read. The computational unit processes the bandwidth requests stored in the RAM to generate time slot assignment. The time slot assignment defines the distribution of available bandwidth to different flows.

The bandwidth requests from ingress ports are stored in memories inside the arbiter. The shift register, TDM counter, SYNC (synchronizer), and WEN (write enable) in Figure 4.11 are used to monitor the incoming byte stream from ingress port, find the start of STS-12 frame (A1, A2 bytes), and enable the memories to store the flow requests from each ingress port. The software module reads the contents of request memories and uses this information to arbitrate available bandwidth to different ports. The software part acts as a master port and the hardware part acts as a slave port. When the scheduler is ready to compute the next arbitration, software sends *scheduler ready* signal informing hardware software part is ready to compute next arbitration. Since the hardware and software are run at different time (speed) there is another signal from scheduler (*send next byte*) to synchronize the software/hardware timing. These request flow control signal goes to the round robin counter which controls the flow of requests from memories to processor. The *queue status counter* signal from RR counter tells the software to which destination and class the current flow request belongs to. FPA and TSA are computed by processor and configuration signals (*switch configuration signal)* are passed to switch core and then to port cards.

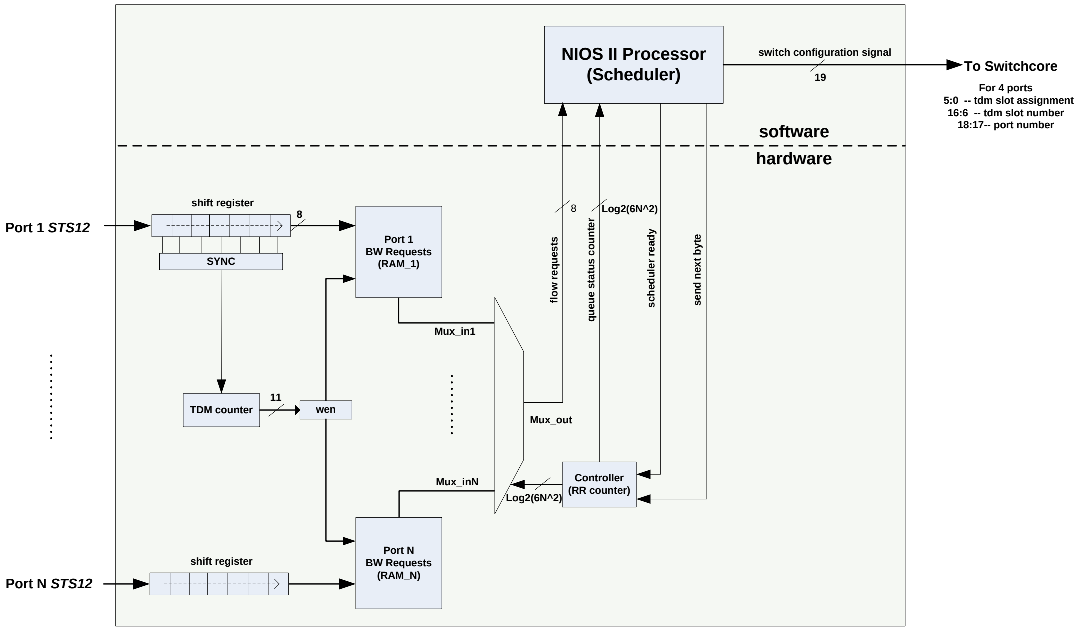


Figure 4.11 Detailed Arbiter Hardware/Software Partitioning

4.4.1.5 Advantages of Software Based Approach

Taking various factors such as flexibility, development cost, power consumption and processing speed requirement into account there exists a trade-off between hardware and software implementation. Hardware implementation is generally better than software implementation in processing speed and power consumption while software can give a more flexible design solution.

Programming in the NIOS II processor provides flexibility in the system implementation such an as: chose the exact set of CPUs, peripherals, and interfaces needed for the application; increase performance without changing your board design, accelerating only functions that require it; eliminate the risk of processor obsolescence; lower overall cost, complexity, and power consumption combining many functions into one chip [17]. This processor is very flexible and has some configurations that can be made in the design stages. These configurations allow the user to optimize the processor for his application. The most relevant are the clock frequency, the debug level, the performance level and the user defined instructions. The performance level configuration enables the user to choose one of the three performances available: The NIOS II/f (fast), which results in a larger processor that uses more logic elements, but is faster and has more features, such as multiplication and others; The NIOS II/s (standard), which creates a processor with balanced relationship between speed and area, and some special features; The NIOS II/e (economic), which generates a very economic processor in terms of area, but very simple in terms of data processing capability [18]. The user-defined instructions allow the user to import hardware designs and attach them to the processor hardware, making them accessible by means of customizable instructions.

4.4.1.6 Implementation Results

The C code and the Verilog HDL code for the arbiter were simulated on NIOS-II IDE. The switch's size was varied from 4x4 to 32x32 and the time period for the arbitration was recorded. Following are the timing results for different switch sizes:

Table 4.4 Arbiter Timing Results for Software Implementation

Number of Ports	Clock Ticks	Time (ms)
4	185497	1.55
8	670771	5.59
12	1327503	11.06
16	2452516	20.43
32	8838006	73.65

4.4.1.7 Limitations of Software Based Approach

The arbitration time for a 32x32 port switch was 73.65 ms (~35 STS12 frames) as compared to our target of 4 SONET STS12 frames time (500 µs). Although the implementation of the arbitration algorithm in software is easy and more flexible, it fails to meet the critical time for arbitration. An arbiter running that slowly would cause unacceptable ingress queue build up when traffic bursts arrive at the switch. Thus, the software based approach is impractical for our switching applications.

4.4.2 Hardware Approach

In order to overcome the limitation of software based approach, a hardware based approach was developed. In the hardware based design the function of processor in software is directly implemented in hardware. The Verilog HDL was used for the whole logic circuit design and simulated for testing. Verification and testing procedure and results for the simulations are discussed in Chapters 5 through 7.

4.4.2.1 Hardware Architecture

Figure 4.12 shows the block diagram of arbiter implemented in hardware.

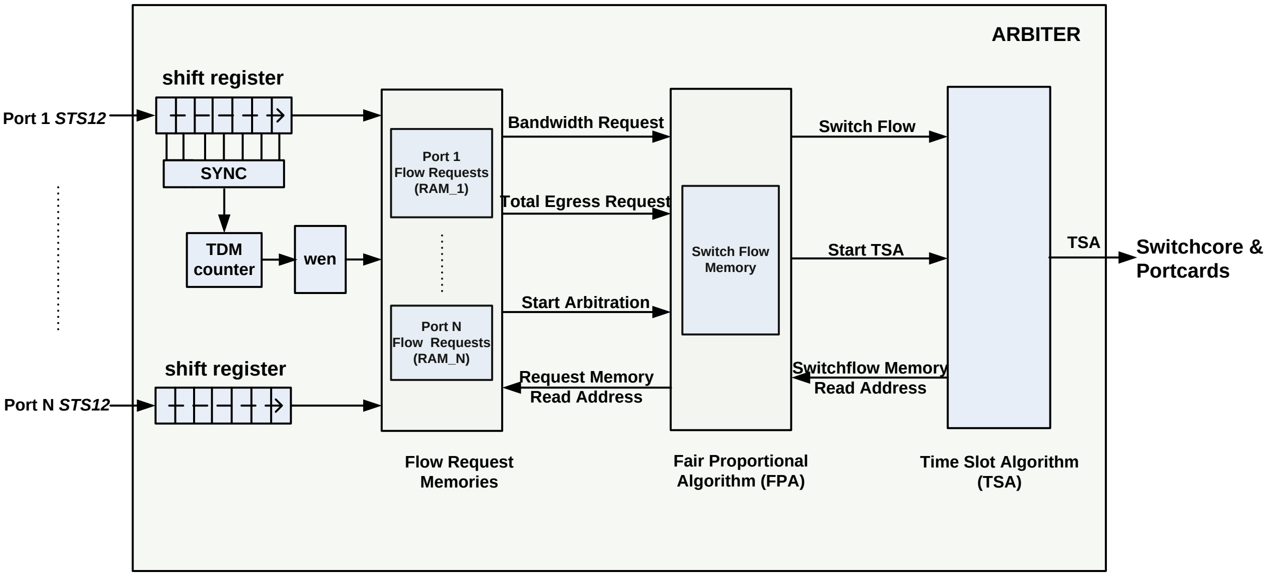


Figure 4.12 Arbiter Hardware Block

The arbiter contains three main functional modules: Flow Requests Memories, FPA and TSA. The functional block has *N* dedicated unidirectional 8-bit wide data buses for receiving the requested bandwidth from each ingress port. The width of requested bandwidth from each ingress queue is three bytes (24 bits) and they are all stored in buffers. The buffers are realized in *N* RAMs (one for each ingress port), each RAM block is *3xMxC* bytes (three bytes for each destination-class flow), where *M* is the number of egress ports and *C* is the priority classes supported by switch.

Once all the new flow requests are available in the arbiter flow request memories arbitration is done in order to distribute port bandwidth fairly among all ingresses. The bandwidth allocation algorithm and time slot assignment algorithm are the same but they are now implemented in the Verilog HDL. When the new set of flow requests comes to the arbiter, this control information is stored in Flow Request Blocks inside the arbiter as shown in Figure 4.12. The Flow Request Blocks then invoke the FPA block for proportional bandwidth allocation on these requests. The FPA block reads the requested bandwidth stored in the memory by address indexing. At the same time it also computes the sums of all egress requests from Bandwidth Request Memories module and available egress bandwidth from switch flow memory (inside FPA module). The requested bandwidth is scaled in proportion to their requests and egress bandwidth available (Equation 4.2). The final functional block of the arbiter is the TSA block. The TSA block is activated by FPA block when the flow arbitration for new requests is completed. TSA block randomly reads the content of switch flow memory of FPA block (Goal) and assigns appropriate channel(s) (time slots) for the flow at corresponding ingress port and egress port. The detail TSA process is described in Section 4.3. The switch configuration (slot assignment) is then passed to switch core and port cards.

4.4.2.2 Advantages of Hardware Approach

The main purpose of migrating from a software approach to a hardware approach was to address the problem of arbitration timing limitations in software. The arbitration timing calculation for 32x32 switch for each of the functional block are given below:

- **Request Bandwidth Transfer Time**

SONET STS 12 header is used to carry the requested bandwidth to the arbiter. Each ingress queue flow request is represented by a 3 byte value. Considering a practical switch size of 32x32 with support for 3 traffic classes, we need 32x3x3 = 288 slots to carry all queue flow requests from ingress while one STS12 frame has 33x12 = 396 slots. The first STS12 frame carries flow request and the approximate time is 9/12 of 125 µs = 93.75 µs.

- **Worst Case Flow Arbitration Algorithm (FPA) Time**

Time for each request read from flow requests memories = 1 clock cycle.
Total request memory read addresses = 32x32x3 = 3072
Total time for reading requests from flow request memories = 3072 cycles.
Total time for calculating total egress requests = 0 cycles (pipelined with request read).
Total time for calculating total ingress requests = 31x32x3 = 2976 cycles.
Time for ingress/egress scaling = 0 cycles (pipelined with request read).
Total cycles for FPA= 3072 + (3072+2976) = 9120 cycles
For 100 MHz FPGA: Total time = (9120) cycles x 1s/(100 x10^6cycles) = 91.20 µs.

- **Worst Case Time Slot Algorithm Time**

Time for reading the FPA flow memory = 3072 cycles.
Time for finding the open time slots and sum per flow = $\log_2$ (512) = 9 cycles.
Time for calculating new ingress/egress opens = 0 cycles (pipelined with switch flow read).
For 100 MHz: Total time = (3072x9+3072) cycles x 1s/100x10^6 cycles = 307.20 µs.

4.5 Arbiter Implementation Summary

Both software and hardware approaches for the implementations of the arbiter were considered. The software approach for a 32x32 switch gives an arbitration and slot assignment time of 73.65 ms (approx. 589 STS12 frame time) whereas the arbitration and time slot assignment in hardware is 93.75 μs+91.20 μs+307. 20 μs = 492.15 μs; which is slightly less than four STS-12 frame time. So the hardware based approach is used, as it is required to allow the overall switch to react more quickly to arriving traffic bursts.

Chapter 5
Design Verification and Testing

In order to ensure the functional correctness of the hybrid switch, the performance of the switch is evaluated and compared to predefined expectations. In this chapter we describe the evaluation techniques and the results obtained from our testing procedures.

5.1 Introduction

For the purpose of testing and developing our hybrid switch in an FPGA, we employ various verification techniques to guarantee the functional correctness and eliminate logical errors. The verification techniques used in our hybrid switch are simulation based. A variety of test cases are fed to the device under test (DUT) for each block and simulated and tested. For each test case, the output is monitored and compared to the expected behavior. The simulation method, although incomplete in a sense that it inherently has very limited coverage for design verification, has the advantage that it can be realized directly by the developer within a common HDL environment and thus can be applied quickly and with limited manpower.

5.2 Simulation and Verification Procedure

A simulation environment usually consists of two basic components: a DUT and a testbench. The DUT is the design which is to be tested; the testbench is a unit responsible for driving DUT inputs and checking correct DUT behaviour. Simulation examines the design behavior under certain conditions and results are checked to ensure the correctness of the design. Our simulation verification process is shown in Figure 5.1. This is the modular decomposition of the DUT and the process of verification is to generate a testbench for each module and to work from the bottom towards the top, validating at each level before moving up.

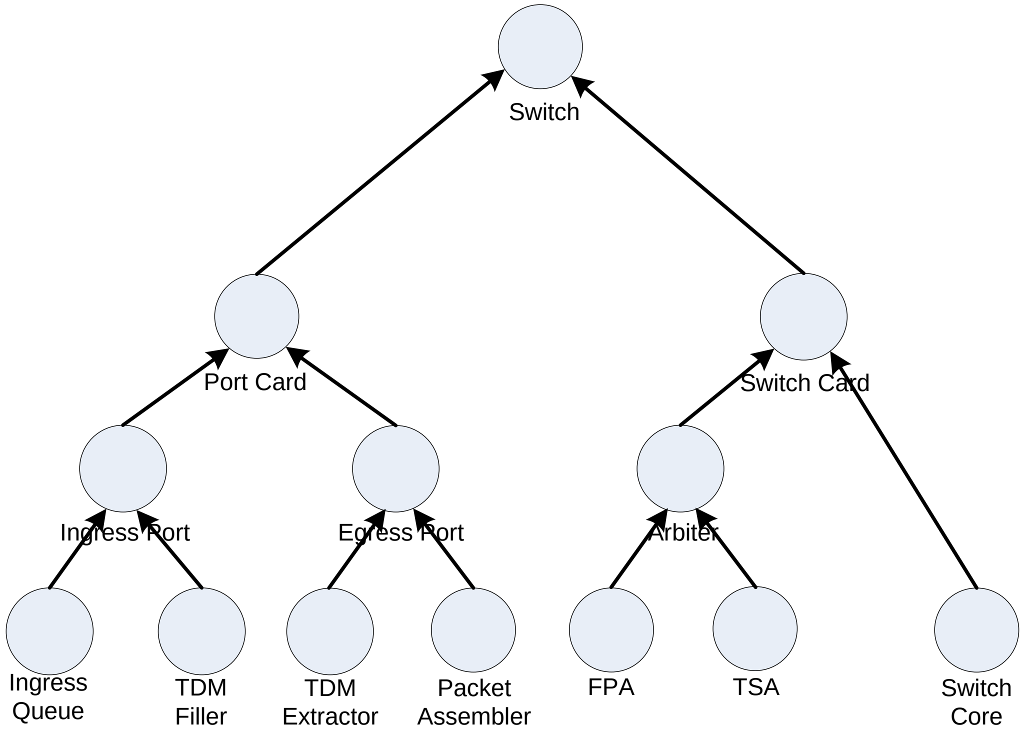


Figure 5.1 Simulation Process for Design Verification

The simulation was carried out for each module to determine if the individual units function correctly. Once an individual module passes the verification test, the modules that form a common unit, for example ingress queues and TDM filler for ingress port, TDM extractor and packet assembler for egress port and arbiter in Figure 5.1, were added and simulated. Again the ingress port and egress port units were added and simulated to verify the functional behavior of port card and arbiter unit and switch core unit were added and simulated to verity the functional behavior of switch card. The port card unit and switch card unit are added and simulated to test and verify the entire switch system.

5.3 Verification and Testing

Switching of TDM signals in our hybrid switch is static. It was assumed that the bandwidth allocation for TDM traffic was made first before doing any allocation for “best effort” IP traffic. This assumption makes the switching of TDM traffic relatively easy and there would not be any port contention for TDM traffic due to the availability of “all” bandwidth to TDM traffic. In this research, our main concern is to see how the switch behaves under different load conditions and adjusts to the current network condition when the system is bandwidth limited. Since we assume the TDM traffic is not bandwidth limited, the interface for TDM traffic was not build and the actual simulation and verification were carried out only on the IP packet traffic.

IP packets with different destinations and CoS were applied to the input ports and examined to see if the packets go to their corresponding queue memory and the depth indicator for that queue was incremented by the size of the packet. It was also verified that if enabling a particular flow releases the next packet belonging to the flow correctly and in the correct format.

The TDM filler is stimulated with predefined page 1 and page 2 channel assignment LUTs and checked if it enables the appropriate ingress queue at the ingress port. It was verified that page 1 and page 2 were swapped correctly at the end of each arbitration cycle. The TDM filler is also responsible for generating SONET STS-12 signals. A simulation was carried out to verify that the STS-12 signal generated was in the correct format and also to verify that the control information and the data carried by the communication link are correct.

The TDM Extractor was stimulated with predefined page 1 and page 2 channel extraction LUTs. It was verified that the data coming out from the switch core in the form of the stream of SONET STS-12 signal was extracted correctly by the packet assembler. It was also verified that the page 1 and page 2 memory for extraction swap at each arbitration cycle. Finally, the packet assembler module was monitored to see if the correct packet sequence was received, and that once the full packet was received, it was correctly sent to the FIFO memory at the egress port.

The arbiter module consists of a FPA unit and a TSA unit. To test the FPA unit, a random request matrix was applied and the bandwidth grant by the FPA was compared to the expected values. To test the TSA module, a random grant matrix was applied to check if the channel assignment was done correctly and without any conflict at any input or output ports. The number of slot assignments was compared with the number of bandwidth grants and the efficiency of TSA unit was evaluated for full load and partial load condition.

The switch core was loaded with two look up tables (page 1 and page 2) and verified that the input signals were switched to the correct output ports according to the values in the LUT. The switch core must be able to distinguish control signals and the data signals and it was also verified that the control signals were ignored by the switch core (control information is only useful to the arbiter) and only the data signals are switched according to the LUT's values.

On the next step of the design verification process the TDM filler and the ingress queue units were combined to form the ingress port module. Similarly, the TDM extractor and the packet assembler units were combined to form the egress port module, and the FPA and the TSA units were combined to form the arbiter module. Testing and verification were done on these larger modules. The IP load, and predefined channel assignment LUTs were applied to the ingress port module. Three signals: the packet coming out from the ingress queue, queue enable signal from TDM filler, and the STS-12 signals to the switch core were monitored to assure their correctness. The egress port module was tested by providing a random STS-12 signal stream carrying IP packets as an input. There were two LUTs to tell which data should go to which packet assembler (out of N assemblers) at what time. The outputs at N assemblers were monitored to see if the packets were received in the correct order. Similarly, the arbiter module was verified by simulating with random bandwidth request matrices. The bandwidth grant for each ingress/egress port were monitored and compared with predefined values. The time slot assignment was checked to verify there were no any ingress/egress conflicts for any flow grant.

The third verification step involves combining the ingress port module and the egress port module to form the port card module and combining arbiter module and the switch core module to form the switch card module. The port card module was verified by simulating it with the same ingress port and the egress port inputs and checking the output results and the switch card module was verified by simulating it with the same arbiter and switch core inputs and checking the output results.

On the final verification step the two modules: port card and the switch card modules were combined to form the entire switching system and simulated. The input to the switch was only the IP load, and the simulation was carried out with empty packet queues and empty LUTs. The system should generate its own LUT values at every arbitration cycles and the page 1 and page 2 LUT memory should be swapped automatically for read and write operation at the end of every arbitration cycle. At the end, the simulation results were checked to verify that

(1) the randomly generated packets were switched to the correct destination,

(2) the packet's priorities were fully respected, and

(3) channel assignment and extraction LUTs for read and write were swapped at the end of each arbitration cycle.

5.4 Summary of the Verification

The switch was tested with simulation-based techniques to ensure that the synthesized design, when manufactured, will perform the desired functionality. The verification testing was conducted on a modular basis and as one module passed the verification test it was combined with another "tested" module to make a bigger unit and verification was done on that bigger unit. The entire hybrid switching system was simulated and verified by combining each small modules and testing it.

The verification procedure reported in this chapter was modeled after the techniques used in industry. However, the inherent manpower limitations of a single student pursuing an M.Sc forced a reduction in the completeness of application of these verification techniques. Nevertheless, the verification carried out on our hybrid switch leaves us confident that the primary data and control paths all function properly, and the resulting switch is functional.

Chapter 6

Performance Results and Discussions

This chapter discusses the simulation experiments and the results obtained. The first section describes the load model used for the simulation. The second section describes the performance results of the simulation. Finally, the obtained results are analyzed and conclusions are drawn.

6.1 Introduction

The general purpose of the performance testing was to determine how some aspects of the switching system performs under a particular workload, which serves to validate and verify some quality attributes of the designed system. In order to carry out the performance testing, a load model was designed. This load model generates simplified IP packets. The switching system was evaluated for different IP load conditions. This process shows how different load affects the dynamics of flows in the switch.

6.2 Simplified IP Packet Structure

Most network data transmission technologies use IP packets to transmit data from a source device to destination. The IP packets have fixed header and payload format. Payload is the actual user data and packet header are carried in order to route actual data on a network from one node to another. A lot of information is carried in the actual packet header. However, in the simulation we are interested in the switch performance only and a lot of “real” IP header fields are not useful at all. So we decided to generate a very simple IP packet header whose structure is shown in Figure 6.3.

8b	8b
Packet Sequence Number	
Source	Destination
Length	
Class	Payload
Payload	

Figure 6.1 Structure of IP Packet Generated by Load Model

Packet Sequence Number: packet sequence number is a unique ID given to a packet by the sender. Packets in same class and going to the same destination port share the same sequence number incremented by one. This is 16 bit long which can tag up to 65536 packets. This sequence number is used to determine whether any packet losses have occurred during transmission.

Source: This field represents the endpoint address of the sender.

Destination: It represents the endpoint address of the intended receiver.

Length: This is the length of a packet, measured in bytes. This includes the header and the payload. This field allows the length of a packet to be up to 65,535 bytes. Such long datagrams are impractical for most hosts and networks. In our load model the packet size is limited to 1500 bytes, which represents the maximum transmission unit for Ethernet hosts.

Class: The Class is a one byte field on the packet header of the load model that provides an indication of the quality of service desired. This parameter is used to guide the selection of the actual service when transmitting a datagram through a particular network.

Payload: This is the actual data that the packet is delivering to the destination. In our typical IP load, this data includes TCP and IP header and TCP payload. For the simulation the payload is transmitted as a sequential integer number and verified at receiver to make sure no data was lost or added during transmission.

6.3 Load Model

Load model generates traffic loads for simulation purposes. A load model represents the expected concurrent number of users on the application and how much link capacity each user is using at a time and the traffic priority for each user traffic. The load model and its characteristics are explained below.

Since the performance testing was carried out for packet traffic only, the load applied to our switch consists entirely of IP packets. This IP load has two fixed characteristics: the distribution of IP packet sizes and the distribution of packets by class of service (CoS). The IP load has two variable characteristics: the overall intensity of the load, and the degree of hotspot concentration present in an otherwise uniform random load.

6.3.1Distribution of IP packets by Size

IP traffic is pre-dominated by small packets, with peaks at the common sizes of 44B, 256B, 576B and 1500B [19]. The small packets, 40-44 bytes in length, include TCP acknowledgement segments, TCP control segments and telnet packets carrying single characters. Many TCP implementations that do not implement Path Maximum Transmission Unit (MTU) Discovery use either 512 byte or 536 bytes as the default Maximum Segment Size (MSS) for nonlocal IP destinations, yielding a 552 byte or 576 byte packet size and ninety percent of the packets are 576 bytes or smaller [19]. A MTU size of 1500 is characteristic of Ethernet attached hosts. The distribution of packet sizes shows that nearly 50% of the packets 40-44 bytes length, almost 10% of the traffic peaks at 256 bytes, about 10% of traffic peak at 576 bytes and remaining 15% at 1500 bytes. In terms of bytes, 40-44 byte packets constitute a total of 7% by the byte volume and over half of the bytes are carried in packets of size 1500 bytes or larger.

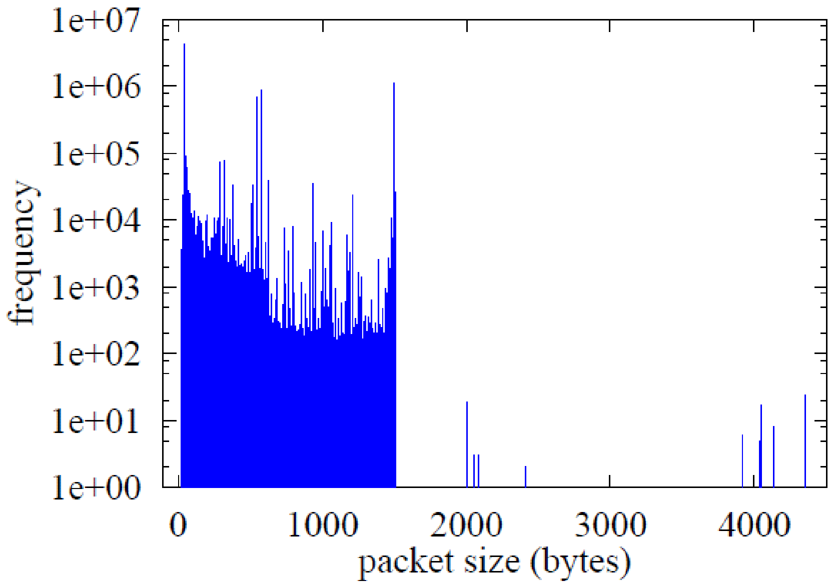


Figure 6.2 Internet Packet Distribution: Relative Frequency of Various Sizes (MCI Study[19], 1997)

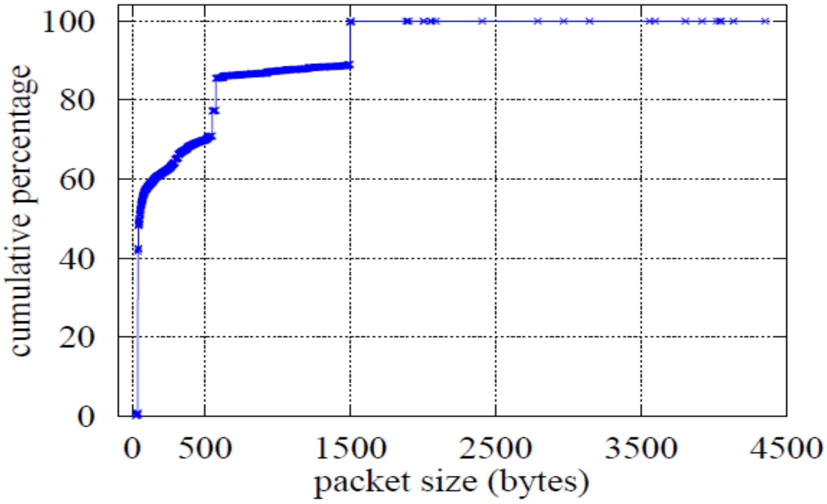


Figure 6.3 Internet Packet Distribution: Accumulated Distribution (MCI Study[19], 1997)

6.3.2 Distribution of IP packets by Class of Service (CoS)

The CoS feature for IP packets enables network administrators to provide differentiated types of service across the switching network. Differentiated service satisfies a range of requirements by supplying for each packet transmitted the particular kind of service specified for that packet by its CoS. The hybrid switch offers traffic management through CoS mechanisms enabling prioritization of customer applications. The CoS with three levels of priority for customer traffic according to the selected service type for each access; Traffic is modeled according to class using the following assumption:

Class 1: 10% customer traffic generated by the load model is class 1 traffic.
Class 2: 40% of the traffic generated by the load model is class 2 traffic.
Class 3: 50% customer traffic generated by the load model is class 3 traffic.

Each data Class of Service is allowed to use all the available IP bandwidth on the access up to 100% of the configured IP bandwidth, when not limited by higher priority traffic.

6.3.3 Distribution of IP Packets by Offered Load

Offered load is the measure of link bandwidth consumed by the ingress traffic. The load model consists of a parameter to control the IP load injected into the system. Varying the value of that parameter changes the offered load in the system. We begin by evaluating the effect of varying load on the performance of our hybrid switch to determine the offered load that maximizes throughput in circuit-switched hybrid networks, subject to QoS constraints for blocking probability. This problem is of interest in network design for sizing the service capabilities that can be provided, and thereby providing a measure of network capacity. We use a simulation technique, in which we increase or decrease the offered load and monitor the throughput graph that guides the search more directly toward the optimal solution, thereby resulting in faster and more-reliable convergence.

6.3.4 Distribution of IP Packets by Hotspot Port

Due to the bursty traffic, the load may exceed the capacity of any egress port. A port with heavy traffic is called a hot-spot port. It arises when one of the outputs of the network becomes very popular among the other ports and the bandwidth resources available at that location in the network are not enough to sustain the needs of the users. Hotspots are representative of the most common types of non-uniform traffic that is likely to occur and cause severe congestion. Performance degradation due to hotspot load is not restricted to hotspot traffic (i.e., traffic destined at the hotspot). In typical networks, hotspot and non-hotspot traffic compete for the same network resources, i.e. buffer space, link bandwidth and router ports. Therefore, hotspots may hinder the delivery of non-hotspot packets.

Under uniform load, all traffic coming to an ingress port is distributed equally among all the egress ports. Under hotspot load, a proportion of the traffic from each input goes to one output, the hotspot, while the rest of the traffic is assumed to be uniformly distributed over all the outputs.

6.4 Performance Matrices

The performance matrices for the simulation evaluation of the hybrid switch were:

6.4.1 Throughput

The hybrid switch has NxC incoming FIFO queues per port. At each time unit, new packets may arrive at the queues, each packet belonging to a specific input queue. A packet can only be stored in its input queue if there is enough space. Since the NxC queues have bounded capacity, which means the faster the packets arrive the greater the chances that packet loss will occur. The switch's throughput is the measure of the maximum load the switch can handle before queues start dropping packets.

It has been shown that if a suitable queueing policy and scheduling algorithm are used with IB-VOQ packet switch, then it is possible to achieve 100% throughput for all independent arrival processes [9]. However, they do not provide any strict delay guarantees and require a sophisticated arbiter which is based on computing a maximum weighted matching that requires a running time of O(N 2.5~3) [20]. Our hybrid switch is not designed to achieve 100% throughput due to the fact that arbitration problem of finding and assigning time slots is hard and we have reduced the algorithms to make them faster, smaller and simpler. With a full slot rearrangement scheme the switch may achieve 100%

throughput, however, achieving 100% throughput would make the arbitration process very slow which would have a negative impact on the queue performance (rapid growth) and make the switch unable to react to the current network condition. So instead of focusing on achieving 100% throughput we have focused on efficient implementation of the scheduling algorithm without any speedup and slot reassignment. Thus we expect reduced throughput utilization of our switch's data paths. To compensate, appropriate speed up will be built into any product based on these ideas. For instance, if we discover that our arbiter is limited to effectively allocating 70% of the datapath's bandwidth, then the entire switch will be speed up by 1.43 (1/0.7) to produce a design which will run effectively with a saturated (100%) traffic load.

6.4.2 Queue Depth Vs Arbitration Cycles

Ingress port queue depth is the indication of how fast the switch is able to make a connection between an ingress ports to an egress port for all requests. For our hybrid switch, the switching decision is made per class basis and the higher priority class should be served first before granting any bandwidth to the lower priority traffic. In our simulation result, the queue flow graph should be increasing for some initial arbitration cycles when the switch is open, indicating increase in the traffic at queue while switch is open, and the graph should go down as time passes. This indicates the switch is allocating bandwidth appropriately. In the long run, the queue depth graph for the first priority traffic should be near zero and the queue depth for the second priority traffic should be higher level than first but still near zero, and the flow graph for third priority should be on the highest level, well above other two, when saturating traffic onward.

6.4.3 Non-blocking and Uniform Loads

The hybrid switch is non blocking, which means any traffic going to one destination should not block the traffic going to the other destination(s). Under a uniform, non-hotspot load, the performance graph for all ports should look similar, indicating a distribution of bandwidth fairly among all ports. Under a hotspot load, the queue depth graph for the hotspot destination should be increasing even for the longer simulation period because it has exceeded the maximum throughput and all the traffic that contends for that destination port cannot be drained by the switch. The queue depth graph for other non-hotspot ports should look similar and should show systematically lower queue depths.

6.5 Simulations and Performance Results

6.5.1 Objectives

The main objective of the simulation is to evaluate the performance of the hybrid switch. The simulation results are used to verify if the following issues are addressed by the switch:

(1) provision of high performance and high throughput,

(2) respect for QoS by serving according to packet priority, and

(3) statistically non blocking for both uniform and hotspot traffic.

6.5.2 The Simulations

In order to examine the switch's performance, the RTL simulation was carried out for a switch of size 4x4. This simulation used the load model described in Section 6.1. The first part of the simulation experiment is carried out to determine the operating load of the switch under uniform load. The second part was used to study the switch performance at maximal throughput under uniform traffic, and the third part to study the switch performance for hotspot traffic while the switch is operating at its operating load.

The simulation run is parameterized with some input data. The following input data are required for the simulation experiments:

Offered load: the percentage of full load injected to the switch for a given experiment.

Port bandwidth distribution: the percentage of traffic going to different destinations. This input is used for simulation under hotspot loads.

Ingress buffered queue length: the size of the ingress queues which are used to store the delayed packets. Longer queues are required if the arbitration period is high, or the offered load is high, or the simulation time is long, or there is a hotspot port.

Length of simulation: the length of the simulation time.

6.5.3 Determining the Operating Load

Operating load represents the maximal offered non-hotspot load that the switch can handle without dropping packets. In order to determine the operating load, the rate of packet stream was varied from 33% to 90%. From 33% to 67%, the rate is increased by two steps of 17%. After 67% offered load, the simulation was carried out for 70% of the offered load and then by increment of 5% after that. Figure 6.4 shows the results of the simulation. The X-axis of the graph is the time in terms of arbitration cycles and the Y-axis is the ingress queue depth computed in the unit of byte. A maximal offered load at which the queue depth graphs seems to be saturated and traffic seems to be performing well is chosen as the operating load of the switch.

6.5.4 Results and Discussions

The results obtained from simulation are shown graphically. In the next sub-sections, first we present the results for queue depths for uniform traffic and different offered load conditions (Figure 6.4). These graphs show how the switch performs under different load factors, which tells about the maximum throughput (*M*) of the switch. The second graph (Figure 6.5) shows for a load of *M*, how the queue depth of the logical egress port for each class changes with respect to the arbitration cycle. The third graph (Figure 6.6) shows the queue depths for a hotspot port and the other non hotspot ports which prove the uniform distribution of port’s bandwidth among all ports. On the fourth graph (Figure 6.7) we show queue depth graph of different CoS traffic.

6.5.4.1 Variation of Queue Depth with Offered Load for Uniform Traffic

In order to determine the throughput, the simulation was carried out for different load factors. The resulting graph is shown in Figure 6.4.

The queue depth of the ingress VOQ varies with respect to the load applied to the switch and the simulation time. As the load of the network is increased, there are more packets to be injected. As long as the new packets are allowed through the crossbar by the arbiter they are not delayed at the injection point. But, as the number of packets in the network increases it becomes harder and harder for new packets to find channels at the output link. So the average delay of packets starts increasing. If the packet injection is too high, the switch may not be able to handle all the incoming traffic which eventually leads to packet drop.

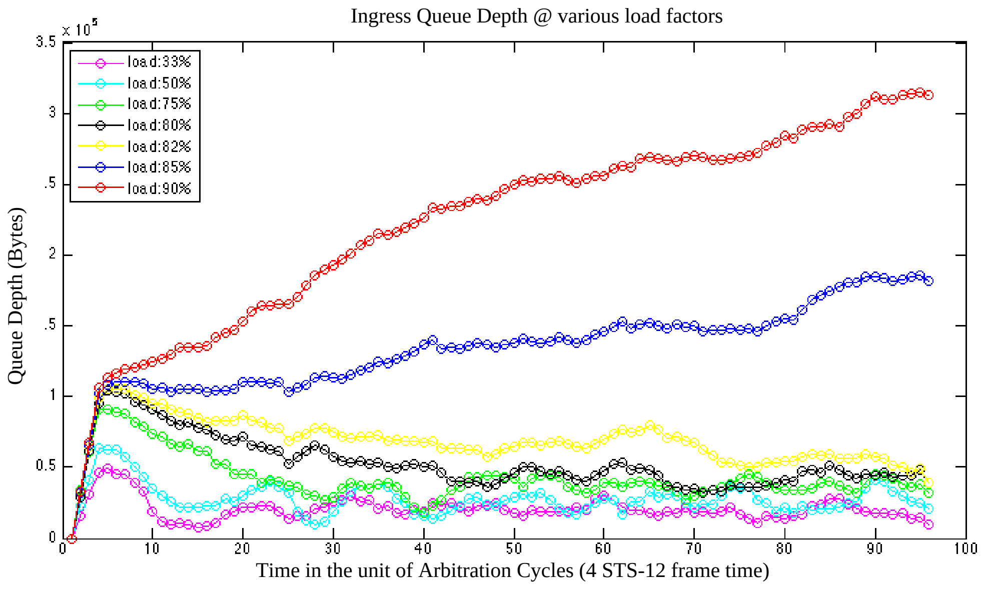


Figure 6.4 Ingress Queue Length for Various Load Factors

Figure 6.4 shows the queue depth is minimum for low load factor and higher as the load factor increases. The switch seems to be able to handle all the incoming traffic when the offered load is around 80%. When the load is at 85% switch is unable to cope with the incoming traffic and the backlog traffic increases continually, which is indicated by the linear increase in the queue depth. Thinking the load factor of 82% might be a good compromise between good performance (80%) and bad performance (85%), simulation was carried out for 82% of offered load. The graph for 82% shows a very good response in terms of the overall queue depth; however, it seems to suffer from a spike of traffic at arbitration cycle 60. Although it seems to recover from that spike, it takes about 5 arbitration cycles for the recovery (unless the traffic spike was for more than one cycle). With some safety margin, the maximum throughput of the switch is considered to be 80% and used for other hotspot and non-hotspot uniform traffic simulation.

6.5.4.2 Variation of Queue Depth with Packet Classes for Uniform Traffic

The hybrid switch must be able to process all the backlog traffic at the ingress ports fairly while respecting the customer QoS requirements. The following graph (Figure 6.5) shows the performance of the switch for different traffic classes when the switch is running at the maximum allowed load (80% load factor).

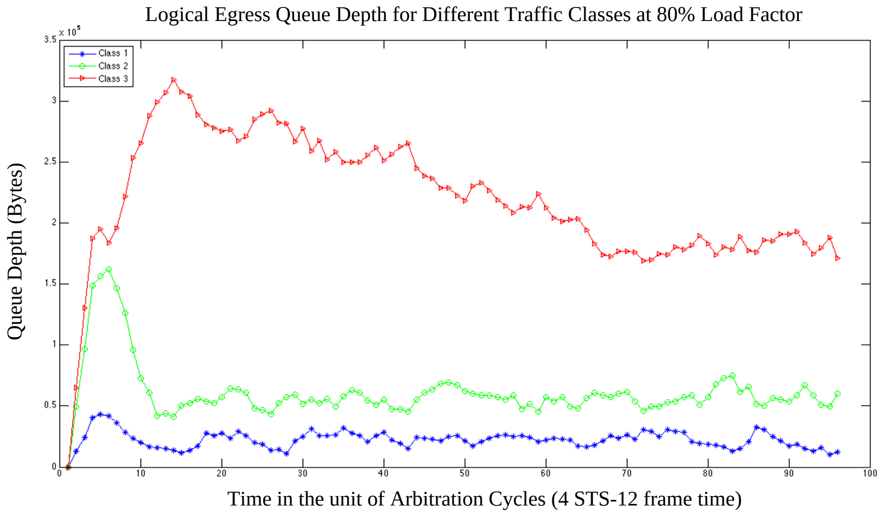


Figure 6.5 Egress Queue Depth for Various Load Classes of Traffic at Operating Load

Our hybrid switch gives more bandwidth access to the higher priority traffic. From this simulation we show that the bandwidth is actually allocated according to traffic priority. As shown in Figure 6.5, the higher priority traffic (class 1 and class 2) perform very well and their backlog traffic is much lower than lower priority traffic (class 3). This can be explained in the following way. As more and more packets are injected into the system, the increasing routing conflict disturbs the natural packet flow and tries to restrict original forward motion of the traffic because of the conflicts over resources. To resolve these conflicts, arbitration of the port and bandwidth is carried out by the switch. The arbiter's channel distribution for forwarding packets from ingress to egress port is based on the priority. As the higher priority traffic (class 1 and class 2) get all the bandwidth resources before class 3 traffic, it is easy to find forwarding channels for these packets and hence their backlog traffic, represented by queue depth, is smaller.

6.5.4.3 Variation of Queue Depth with Hotspot Port

This part of the simulation compares the switch performance from the viewpoint of queue depth for one hotspot egress port in the presence of other non-hotspot traffic. The traffic injected for the hotspot port is greater than M. The offered load to the switch is 80% of the full load. The goal was to identify the links which are the most heavily used (network hotspot) and see how the location and intensity of hotspots relates to the overall performance result seen in the previous simulation and see if it maintains the strictly non-blocking characteristic. For hotspot simulations, 30% of the traffic at each ingress port is directed to one particular egress port and 25% of the traffic is directed to each of the remaining egress ports. In this way the hotspot gets 4x30%=120% of the offered load while the other ports get 100% of the offered load. In this simulation, each ingress port sends traffic at 105% of 80%= 84% of the full load and the hotspot port gets load of 120% of 80%=96% of full load and the other port get load of 100% of 80% =80% of the full load.

Figure 6.6 shows the performance of the hybrid switch under this hotspot load. The graph shows that the overall queue depth for the hotspot destination is increasing while the other non hotspot destination queues are performing very well. The hotspot backlog is increasing because the packet injection ration of the load model for that particular port is way higher than the egress port can handle. The queue size is increasing, which indicates the egress port is unable to cope with traffic incoming rate. This will eventually cause packet loss for traffic going to that particular destination.

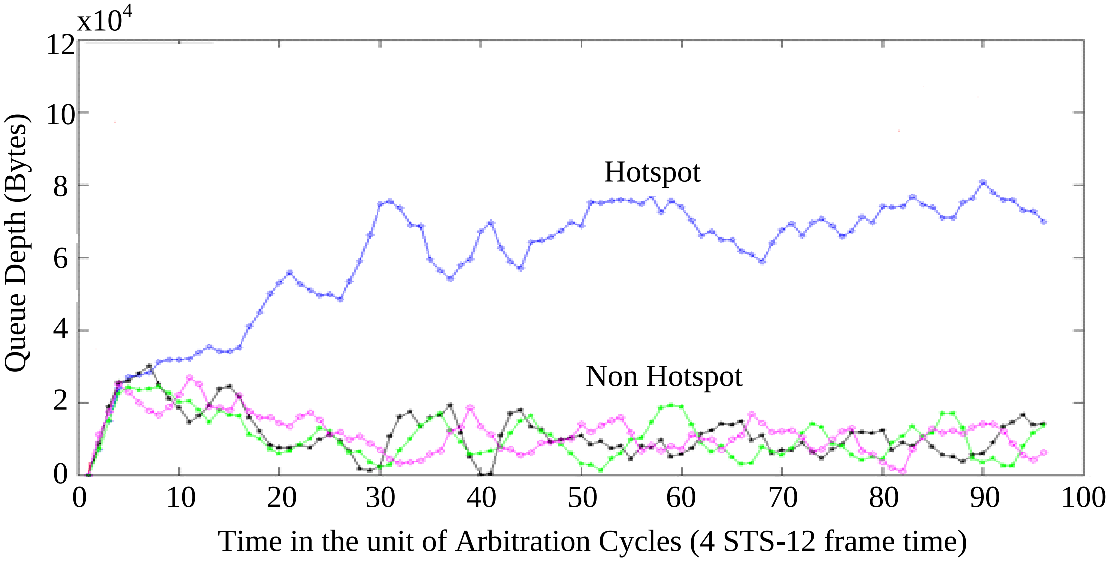


Figure 6.6 Egress Queue Depths for Various Load Classes of Traffic at Operating Load

The important result from this graph records the non-blocking nature of the switch. Figure 6.6 also shows that even though the queue backlog for hotspot traffic is increasing, the backlog for other destinations is stable and all the flows are behaving in a similar fashion in terms of queue depth. This shows that even though there are many packets requesting bandwidth to the hotspot port, the switch is non-blocking in nature, so traffic going to the non-hotspot destinations proceeds normally, regardless of the hotspot traffic.

6.5.4.4 Variation of Queue Depth with Hotspot Classes

In this part of the hotspot load model simulation the switch performance is evaluated to see if the priority order is respected for hotspot traffic as in the uniform traffic model. Figure 6.7 shows the performance result for simulation of different priorities of hotspot traffic and their performance in terms of the queue depth.

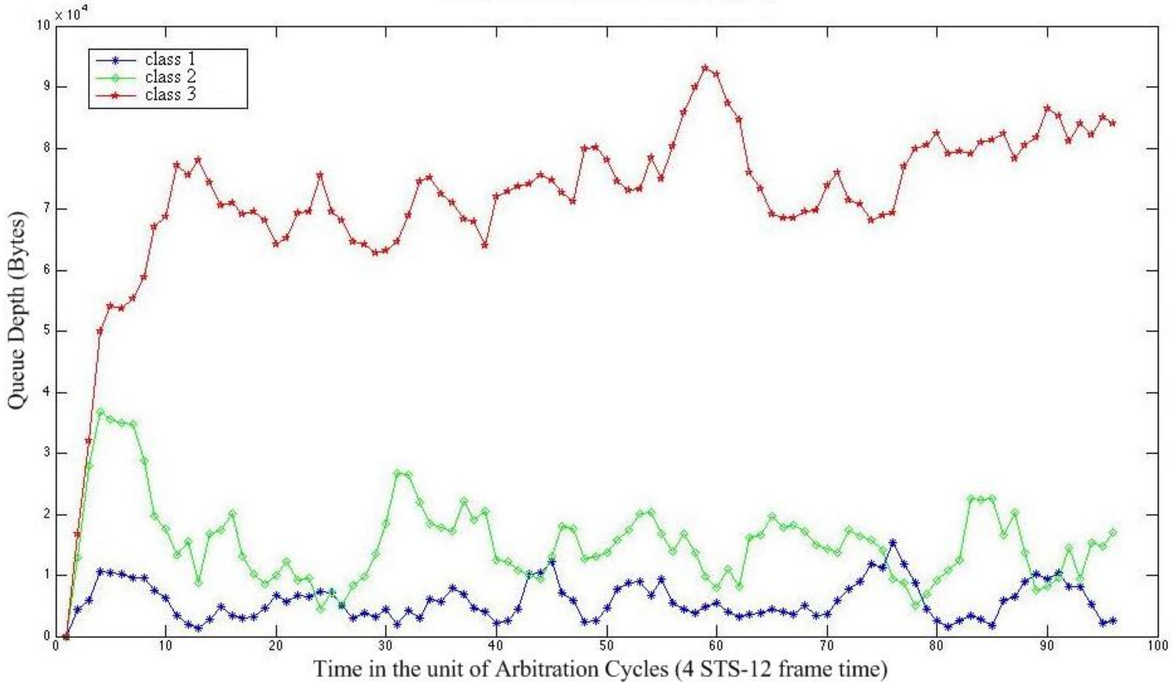


Figure 6.7 Egress Queue Length for Various Classes of Hotspot Traffic

The above graph shows that the overall queue depth for the hotspot destination increases with time (arbitration cycle) and that this increase comes from class 3 traffic. This was because the incoming packets are queued according to their destination and class, and as more and more packets come to the ingress port, the lower priority traffic has more and more conflicts that restricts their forward motion, and hence remains in queue as backlog packets for a long time. The other higher priority traffic has more access to the channel's bandwidth and is performing well without any service deterioration. The graph for the hotspot destination classes is similar to the one in the case uniform load model case.

6.5.5 Summary of the Results

This chapter described the simulation experiments analyzed by the obtained results. The performance metrices of the hybrid switch in order to evaluate the performance of the switch were described. The objectives of the simulation and the load model for the simulation were described. The simulation results were presented in graphical form and were analyzed in detail. The first simulation result showed that the maximal operating load of the hybrid switch is 80%. The second simulation result showed the bandwidth allocation in hybrid switch is strictly in the priority order. The third simulation showed under the hotspot destination port, the hybrid switch is non-blocking for non-hotspot traffic. The fourth and final simulation result showed higher priority traffic is not blocked to the hotspot destination.

Chapter 7

FPGA Mapping: Logic Utilization, Delay and Power Estimation

The purpose of this chapter is to provide a hardware implementation cost analysis for our hybrid switch. The basic elements of the hybrid switch are implemented using HDL-based design. The cost of the system implementation on FPGA is calculated in terms of logic utilization, maximum clock speed, and the power consumption.

7.1 Introduction

A fast estimation of the resources is an important basic principle in order to be able to generate optimal digital circuits that can be mapped into FPGAs or Application Specific Integrated Circuits (ASICs). In the scientific field several methods exist to determine the utilization of individual IC resources such as area, size, and power consumption. In this section we present our approach of estimating the cost and area of hybrid switch for an FPGA. The estimation is based on the simulation of RTL code.

7.2 Field-Programmable Gate Arrays (FPGA)

A typical FPGA device consists of a pre-fabricated array of configurable logic blocks (CLBs) surrounded by configurable routing. Each logic block consists of resources which can be configured to define logics, registers, mathematical functions and even Random Access Memory (RAM). A periphery of configurable pads (I/O ports) provides connection to other electronic devices. The function of all of these configurable resources can be defined at any time during the operation of the device to form a large logic circuit. Configurable logic and routing can be formed together to ensure the exact function of a digital processing algorithm. Parallel and pipelined data flows are possible, providing an excellent resource for execution of a signal processing algorithm.

In case of hybrid switch's integrated circuits we focused on the following resources of interest:

7.2.1 Logic Utilization

FPGA architectures is constructed from a sea of basic logic units, where each unit consists of a four-input look-up table (LUT), programmable register, and any associated specialized circuits, such as a carry chain, cascade logic, primitive logic gates and multiplexers. Even when a FPGA contains additional dedicated circuits such as multipliers, the bulk of a typical design's logic functions are still implemented by these basic units [21].

In the case of FPGAs, logic resource is measured in terms of number of logic blocks and embedded components (like multipliers, memories, shift registers, etc.). Routing, powering and the clock network are excluded, because they are pre-fabricated on the FPGA board. Altera uses the "Logic Element" (LE) methodology to measure the logic capacity of a FPGA design. This is a generic basic unit that can be used to fairly measure the size of a design across different FPGA architectures.

7.2.2 Propagation Delay

Propagation delay is the time required for a digital signal to travel from the input(s) of a component to its output. Propagation delay is important because it has a direct effect on the speed at which a digital device can operate. The frequency f measured in Hertz (which means cycles per second) of an oscillator used to time or synchronize the operations of a circuitry. The higher the clock frequency, the faster the operation of the circuit is. The period of the clock (Tperiod) is the time taken to complete one cycle and is the inverse of the frequency f: Tperiod = 1/f . The maximum clock frequency (F_{max}) of the circuitry is the measure of maximum frequency that can be applied to the circuit without any timing violation. This depends on the components' propagation delay and on their routing delay: the slowest component and the wiring delays limit the maximum frequency. We evaluate the propagation delay in terms of F_{max}.

7.2.3 Power Consumption

Power consumption is in general the energy over time (P = E/t) that is supplied to a system to maintain its operation. In the case of integrated circuits there are two main components of power consumption: dynamic and static consumption. Static power is the minimum power required to keep the device 'powered-up' with the clock inputs not switching and the I/Os drawing minimal power. Dynamic power is the power consumed when both the I/Os and logic cells are switching. Dynamic power dissipation accounts for over 95 percent of power consumed and is predicted by three factors supply voltage (Vdd), physical capacitance being switched (C) and switching frequency (f) [22].

$$\text{Power consumed (P)} = 1/2CV_{dd}^{2}f_{eff} \qquad (7.1)$$

This means, that power consumed is proportional to the capacitance (C), operating voltage (V_{dd}) and effective frequency (f_{eff}). Reducing any one of these variables reduces power dissipation.

7.3 FPGA Cost Estimation Methodology

A methodology that embodies the estimation of area and power in this thesis involves the use of RTL code simulation tool in the IC design flow as shown in Figure 7.1. The design flow begins with HDL coding, test bench generation, and simulation. The code coverage (area) in terms of the logic elements (LEs) utilized is determined before optimizing the code. If code is too big to fit in FPGA, iteration through test bench generation and simulation is necessary. Once acceptable code coverage is achieved power estimation and optimization are performed.

FPGA vendors provide tools to estimates the design area, maximum frequency the design can run (F_{max}), and the board power supply. These tools also help designers optimize device's area, performance and power consumption. Altera tools generate area estimates based on the RTL code and delay and power estimates based on the parameters such as design resource utilization, routing utilization, clock frequencies, device, I/O loading, temperature, and silicon process. Each of these parameters can affect static power, dynamic power, or both.

The flow Summary section of the Quartus II compilation results was used to determine the resource utilization and delay (in terms of F_{max}), and the Quartus II PowerPlay Power Analyzer was used for the power estimation. The PowerPlay power analyzer is integrated in the Quartus II software and generates design power estimates based on Quartus II software place-and-route information and toggle rate data from a variety of sources. The power analyzer performs power analysis based on the exact resources, logic functions, and routing paths used in the target design. It can derive toggle rates from user entry, statistical circuit analysis techniques, RTL simulation, and gate-level simulation. The power analyzer can optionally filter out pulses, or glitches, from simulation data when those glitches are too fast to toggle the logic and routing of an actual FPGA [23].

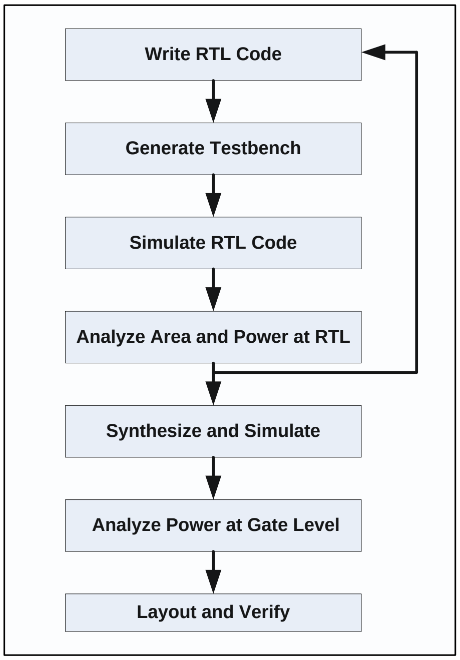


Figure 7.1 RTL Area and Power Analysis in IC Design Flow

7.4 Synthesis Results and Analysis

The synthesis for 4x4 switch fabric was carried out. The device chosen for synthesis was Altera's Stratic III FPGA. The Table 7.1 shows the result of the synthesis of Verilog HDL code.

Table 7.1 FPGA Resource Utilization Table for 4x4 Hybrid Switch

Implementation	Module	Logic Elements (ALUTs)	Fmax (MHz)	Power(mW)
Ingress Port	Queue	12351	150.40	689
	TDM Filler	351	263.78	628
Egress Port	TDM Extractor	150	413.56	629
	Request Matrix Generator	1683	125.85	634
Arbiter	FPA	4770	78.99	642
	TSA	6690	69.65	640
Switch Core	SwitchCore	702	219.39	632
Total		**26697**		**4494**

The resource utilization for the 4x4 hybrid switch fabric is 26697 logic elements. This represents 23.5% of the Stratix III EP3SL150 device (target board) from Altera. Table 7.1 shows the resource utilization for each component of the Ingress Port, Egress Port, Arbiter, and Switch Core blocks. Note that the actual logic utilization generally does not match the sum of the individual sizes due to glue logic and optimization. The power consumed by the switch fabric is about 4.5W and the F_{max} for the digital circuit is about 70 MHz. However, the logic utilization reported by the synthesis tool should only be treated as a rough estimate because of advanced fitting algorithms, such as register packing, and other factors such as megafunctions, and Quartus II netlist optimization techniques were not considered.

The 4×4 hybrid switch can be potentially scaled up to enhance the network capacity in optical communication systems where non-blocking large-scale optical cross-connect switching is required. For the purpose of modeling costs of the switch architecture, we approximate the resources utilized when the switch scales up. The following table shows the scaling factors of different components of the switch when the switch scales up:

Table 7.2 Scale-up Cost for Hybrid Switch

Parts	**Scaling Factor**
Ingress/Egress Port	*N*
Arbiter	*NxR+C(T+F)*
Switch Core	N^2
Memories	$(NxC)^2$

Where,

N= number of ports

C= number of traffic classes

R= Request Matrix Generator Block

T= TSA Block

F = FPA Block

The number of logic elements increases linearly for any increase in the size of our hybrid switch. This is because every port is independent of the other and a separate logic is needed in every port in order to differentiate traffic, maintain queue, and regulate packet flows. The arbiter block changes with N and C. We need separate Request Matrix Generator block because all the ingress ports are synchronized and the bandwidth request from each ingress flow occurs at the same time. So the Request Matrix Generator is scaled by N. The FPA and TSA are done per class basis, and thus they are scaled-up by C. The switch core is scaled up by N^2 because now it needs a lookup table to set N^2 crossbar points. However, the crossbar setup in switch core is implemented using multiplexer whose logic cost is relatively low. The exponential scaling factor on crossbar does not have the same overall impact on the switch. Ignoring the memory cost, the following calculation shows logic cost for 16x16 switch fabric with three classes of traffic:

Ingress Port LE's = (12351+351) x 16/4 = 50808

Arbiter LE’s	= (1683x16/4+4770+6690)	= 18192
Switch Core LE’s	= 702 x $(16/4)^2$	= 11232
Total LE’s		**= 80232**

From the preliminary analysis, the 16x16 switch can be implemented in our target board with 70.6% logic utilization.

7.5 Summary of FPGA Mapping

The logic utilization and speed for our hybrid switch are reasonable for hardware implementation. The estimated logic cost for 16x16 switch architecture in Stratix III EP3SL150 device is 80232 ALUTs, and the architecture can be clocked at 70 MHz frequency. However, with some logic and timing optimizations, the required logic elements can be decreased and the overall timing can be improved. The power consumption estimate of 4.5 W for our switch is high relative to most of the FPGA designs experienced. However, switch datapaths are notorious for high power consumption because most signals and most flops change value on each clock tick. Only further work can determine the accuracy of our estimate.

Chapter 8

Conclusions and Future work

8.1 Conclusions

In this thesis, the current internet trends at LAN and MAN were introduced. A new concept for data switching named “hybrid switching” and its implementation were proposed. The switch is able to support both, TDM based voice traffic and Internet based packet traffic on a converged platform. This switch offers simple and low cost hardware and a lower level of management because of its capability to support multiple protocols in one platform by using a rack-mounted chassis approach in which a wide variety of interface and function cards can be inserted to perform the specific roles. The logic circuit for the switch was implemented in Verilog HDL and the simulation was done in ModelSim.

We began this thesis with a “big picture” of the internet and the network equipments in LAN and MAN environments. Based on the motivation described in Chapter 1, we introduce the architectural overview of the hybrid switch in Chapter 2. This chapter also discussed the problem of the multiple queue scheduling and maintaining QoS commitment to customers. To provide QoS and the ability of switch traffic without HoL blocking, the IP packet traffic coming to the ingress port is differentiated into three classes and virtual output queue is maintained for each traffic class.

A major part of the architecture was the design of the arbiter. For our hybrid switch, we need an arbiter to make a scheduling decision for a number of ingress lines connecting to the egress lines. This is called the traffic scheduling problem. The arbiter of our hybrid switch gives higher priority to TDM voice traffic and thus the bandwidth allocation for TDM flows are done first, followed by the packet traffic. All the TDM flows get guaranteed bandwidth while the packet traffic gets bandwidth per flow basis, based on the priority and the weight (amount of bandwidth requested) of the request.

The first step of packet traffic scheduling is the generation and communication of bandwidth requests from the ingress ports to the arbiter. The bandwidth requests are generated at ingress ports using the current queue depth and the recent traffic flow as shown in Equation 4.1. These requests are communicated to the arbiter by SONET overheads. The proportional arbitration scheme (or FPA) used by the arbiter allocates bandwidth to these requests fairly and on the strict priority basis; i.e. bandwidth requested by any ingress flow is weighted against the total requests and the bandwidth is granted accordingly, and the traffic with highest priority is served first and the residual bandwidth is allocated to lower order traffic.

The FPA has the simplest implementation and provides differentiated services. The bandwidth allocated by this block is then assigned VT1.5 channels. This process of VT1.5 TDM channel assignment for each flow is called Time Slot Algorithm (TSA). The time slot assignment involves reading the bandwidth allocated by FPA, calculating the free channels at both ends, and assigning channels without any port conflict or channel over-allocation. The entire process is described in detail in Chapter 4.

Based on the arbitration scheme, we had two options for the implementation of the arbiter. The first option was the software approach, programmed in C or assembly language and the second option was hardware based, programmed in Verilog HDL. The preliminary analysis of both approaches showed that the software approach would be too slow to respond to sudden changes in the traffic (traffic bursts) and would result in a switch that would not be useful as a real world implementation. The hardware solution, however, was found to react more quickly to the arriving traffic burst and was more suitable for practical implementation.

To verify the correctness of our hybrid switch, a modular based verification and testing procedure was followed. The procedure is shown in Figure 5.1. As described in Chapter 5, the verification work starts from the bottom and proceeds to the top, validating at each level before moving up. The obtained simulation result in Chapter 6 showed that all the circuits functioned properly as expected. From the performance results, we can draw the following conclusions about the switch:

- The maximum offered load that switch can handle for the uniform traffic is 80%.
- The switch guarantees QoS requirement by offering differentiated service by traffic type.
- The switch is completely non-blocking in the presence of hotspots.
- Hotspot traffic does not affect the switch's ability to provide differentiated service.

8.2 Thesis Contributions

The major contributions of this research were:

- The design of probably the first switch that can support both TDM and packet flows over a single backplane.
- Definition of a data path to carry TDM and packet flows over a single channel.
- Developed a simple arbitration scheme. The time slot assignment problem is solved by using a novel binary add algorithm.
- Verification and performance testing were done in order to make sure the hybrid switch meets QoS objectives and has reasonable performance.

8.3 Future Work

There are numerous things that remain to be investigated for the hybrid switch. The following summarizes the work needed to be done in order to build a more practical hybrid switch.

8.3.1 Prototype Fabrication in FPGA

The initial goal of this thesis was the design and implementation of the hybrid switch. We were able to design and see the implementation result by simulation. Our effort was successful: the hybrid switch has met the design specification and performance goals. The next step would be advancing our work in rapid prototyping, in particular with regard to FPGA. Results obtained from functioning machines are inherently more credible than simulation studies, and are more likely to be reproduced elsewhere.

8.3.2 Simulation for TDM and Packet Traffic

Although the bandwidth allocation for TDM traffic would be static, the simulation result would prove this static bandwidth allocation will work with the dynamics of packet traffic. The load modeling for TDM traffic and the simulation for both traffics is relegated to future work.

8.3.3 Switch with Speed-up

The simulation result shows that the maximum throughput of the switch for uniform random load is 80%. In order to maximize this throughput and align the switch speed with the line speed we need speed-up in our switch. In order to obtain a practical switch for OC-12, we need to have a speed-up factor of 1/0.80 = 1.25 in our switch. One of the ways to get the required speed-up is to use STS-16 signaling in the switch. While SONET STS-16 frame does not exist in practice, this signaling can still be used in the internal communication to get our required speed-up.

8.3.4 Broadband Switching by Scaling up Granularity to STS-1

With increasing demand for higher transmission speed of digital networks, the switching system needs to meet the demands for path routing of broadband networks. Our future work would be scaling the switch to broadband network by moving from VT1.5 channels to STS-1 channels. Moving into STS-1granularity, however, may require ingress flows of smaller capacities to be aggregated into STS-1 frame before entering the switch core, even if the full capacity of this STS-1 is not utilized. This may increase inefficiencies of the resource usage. In order to avoid this inefficiency, we need an improved algorithm that can overcome it while avoiding the unmanageable cost in the arbitration

8.3.5 Search for an Improved Arbitration Scheme

Though we have come up with a useful flow arbitration algorithm, the arbitration policy, for example, request generation, channel distribution, and time slot assignment were developed by testing and evaluation method. The arbiter does not produce optimal results for flow request as we discussed in Chapter 4, and also fails to implement the rearrangement of the time slots, if required, to make the full utilization of the link's bandwidth. The arbitration may work well, but it is not perfect, and, by no means, the only solution. It would be wise to continue to look for improved arbitration schemes.

References

[1] K. Y. Yun, "A terabit multiservice switch," IEEE Micro, vol. 21, pp. 58-70, January 2001.

[2] Nokia Siemens Networks, "Ethernet and TDM carrier-grade services over one single network," SURPASS Multi-Service Optical Networks, July 2007. [Online]. Available: http://www.nokiasiemensnetworks.com/NR/rdonlyres/AB0CCC4C-4E0D-4EE1-B668-A9EFCBA8CBFA/0/SURPASSMultiServiceOpticalNetworks_V5.pdf. [Accessed July 23, 2008].

[3] C. McCrosky, K. Iniewski and D. Minoli, Network Infrastructure and Architecture Designing High-Availability Networks. New York: John wiley & sons, 2008.

[4] S. Shepard, SONET/SDH Demystified. New York: Mc Graw Hills Inc., 2004.

[5] D. Pan and Y. Yang, "FIFO-based multicast scheduling algorithm for virtual output queued packet switches," IEEE Trans. Computer, vol. 54, pp. 1283-97, October 2005.

[6] S. Keshav and R. Sharma, "Issues and trends in router design," IEEE Communications Magazine, vol. 36, pp. 144-51, May 1998.

[7] I. Elhanany and D. Sadot, "Analysis of non-uniform cell destination distribution in virtual output queueing systems," IEEE Communications Letters, vol. 6, pp. 367-369, September 2002.

[8] Y. Tamir and G. L. Frazier, "High-Performance Multi Queue Buffers for VLSI Communication Switches," 15th Annual International Symposium on Computer Architecture, pp. 343-354, 30 May-2 June, 1988.

[9] N. McKeown, A. Mekkittikul, V. Anantharam and J. Walrand, "Achieving 100% throughput in an input-queued switch," IEEE Trans. Communications, vol. 47, pp. 1260-7, August 1999.

[10] J. G. Dai and B. Prabhakar, "Throughput of data switches with and without speedup," 19th Annual Joint Conference of the IEEE Computer and Communications Societies - IEEE INFOCOM2000, vol. 2, pp. 556-564, March 2000.

[11] D. Shah, "Maximal matching scheduling is good enough," IEEE Global Telecommunications Conference GLOBECOM, vol.6, pp. 3009-3013, December 2003.

[12] E. Leonardi, M. Mellia, M. A. Marsan and F. Neri, "Stability of maximal size matching scheduling in input-queued cell switches," IEEE International Conference on Communications, vol. 3, pp. 1758-1763, June 2000.

[13] E. Leonardi, M. Mellia, F. Neri and M. Ajmone Marsan, "Bounds on average delays and queue size averages and variances in input-queued cell-based switches," 20th Annual Joint Conference of the IEEE Computer and Communications Societies, vol. 2, pp. 1095-1103, April 2001.

[14] B. Vandalore, R. Jain, R. Goyal and S. Fahmy, "Design and analysis of queue control functions for explicit rate switch schemes," Proceedings 7th International Conference on Computer Communications and Networks , pp. 780-786, October 1998.

[15] N. Parlante, "Binary Trees," October 2000. [online]. Available: http://cslibrary.stanford.edu/110/BinaryTrees.html [Accessed: September 25, 2008].

[16] D. D. Gajski and F. Vahid, "Specification and design of embedded hardware-software systems," IEEE Design & Test of Computers, vol. 12, pp. 53-67, Spring 1995.

[17] Altera Corporation, "Nios II processor: The world's most versatile embedded processor," May 2008. [online]. Available: http://www.altera.com/products/ip/processors/nios2/ni2-index.html [Accessed: June 5, 2008].

[18] Altera Corporation, "Nios II Software Developer's Handbook," May 2008. [online]. Available: http://www.altera.com/literature/hb/nios2/n2sw_nii5v2.pdf [Accessed: June 6, 2008].

[19] K. Thompson, G. J. Miller and R. Wilder, "Wide-area Internet traffic patterns and characteristics," IEEE Network, vol. 11, pp. 10-23, November 1997.

[20] S. Mneimneh and Kai-Yeung Siu, "On achieving throughput in an input-queued switch," IEEE/ACM Transactions on Networking, vol. 11, pp. 858-67, October 2003.

[21] Altera Corporation, "An analytical review of FPGA logic efficiency in stratix, virtex-II & virtex-II pro devices" Altera White Page, May 2003.

[22] S. Deng. "Estimating IC power consumption at the RT level" EE Times Asia, October 1999.

[23] Altera Corporation. "Stratix II vs. virtex-4 power comparison & estimation accuracy," Altera White Paper, August 2005.