SWITCHING ELEMENT DESIGN FOR BISDN

by

Graeme B. Mund
B.A.Sc.Eng., University of Saskatchewan, 1982

A Thesis Submitted in Partial Fulfillment
of the Requirements for the Degree of

MASTER OF APPLIED SCIENCE

in the Department of
Electrical and Computer Engineering

We accept this thesis as conforming
to the required standard

[redacted]

Dr. F. El-Guibaly, Supervisor

[redacted]

Dr. W. D. Little, Departmental Member

[redacted]

Dr. H. H. L. Kwok, Departmental Member

[redacted]

Dr. M. Serra, Outside Member

[redacted]

Dr. G. C. Shoja, External Examiner

Supervisor: Dr. F. El-Guibaly

ABSTRACT

A general overview of interconnection networks is given. The delta network is chosen for the switch fabric of a Broadband ISDN ATM switch. Architectural options for the switching elements of a delta network are identified. Formulas are developed which analyze the throughput and delay performance of delta networks composed of switching elements based on these architectures. The formulas are more general than those found in the literature. Simulations are performed to validate the analysis.

The optimum switching element architecture is chosen. A design is given for a 25 MHz CMOS 2×2 switching element capable of handling proposed Broadband ISDN data rates of 155.520 Mbits/sec. Datapaths throughout the switching element are nine bits wide, including a parity bit. With virtual cut-through enabled, a packet may exit a switching element as early as three clock cycles after entering the switching element.

Examiners:

[redacted]

Dr. F. El-Guibaly, Supervisor

[redacted]

Dr. W. D. Little, Departmental Member

[redacted]

Dr. H. H. L. Kwok, Departmental Member

[redacted]

Dr. M. Serra, Outside Member

[redacted]

Dr. G. C. Shoja, External Examiner

Table of Contents

Abstract ii

Table of Contents iii

List of Tables vi

List of Figures vii

Acknowledgments xiii

Dedication xiv

List of Symbols xv

1 Introduction 1

1.1 Introduction 1

1.2 Contributions in this Thesis 3

1.3 Outline of the Thesis 3

2 Interconnection Networks 4

2.1 Introduction 4

2.2 Static Network Topologies 5

2.3 Dynamic Network Topologies 10

2.3.1 Crossbar Switch 13

2.3.2 Single-stage interconnection networks 13

2.3.3 Multistage interconnection networks 15

2.4 Switching Methodology 29

2.5 Centralized and Distributed Control Strategies 31
2.6 Synchronous and Asynchronous Operation Modes 32
2.7 Summary 33

3 Switching Element Architecture **35**
3.1 Introduction 35
3.2 Selection of Network Topology 35
3.3 Architecture Options 38
3.3.1 Switch size and buffer size 38
3.3.2 Buffer placement 38
3.3.3 Dilation 43
3.3.4 Buffer selection 43
3.4 Summary 44

4 Delta Network Analysis and Simulation **46**
4.1 Introduction 46
4.2 Analysis 46
4.2.1 Buffers before crossbar 49
4.2.2 Buffers inside crossbar 57
4.2.3 Solving the state equations iteratively 62
4.3 Simulation 64
4.4 Analysis and Simulation Results 66
4.4.1 Discussion of results 67
4.5 Further Work 92
4.6 Summary 92

5 Switching Element Design **95**
5.1 Introduction 95
5.2 The ATM Switch Model 97
5.3 Routing Tag Specifications 99
5.4 Switching Element Specifications 102
5.5 Design Considerations 104
5.5.1 Clocks 104
5.5.2 SE-to-SE protocol 104

5.5.3 Design for testability 105
5.5.4 Fault detection 108
5.6 Functions and Blocks Required 109
5.6.1 Input port server 110
5.6.2 Buffer memory 115
5.6.3 Output port server 122
5.7 Summary 128

6 Conclusions **129**
6.1 Summary of Results 129
6.2 Further Work 131

Bibliography **132**

List of Tables

2.1 Comparison of $N \times N$ network costs. 30

List of Figures

2.1 Examples of two-dimensional static networks. 6
2.2 Mesh-connected two-dimensional arrays. 7
2.3 Chordal Ring network with $n = 8$ and $w = 3$. 8
2.4 Examples of m-ary n-cubes. For clarity, not all links are shown. . . 9
2.5 Recursive definition of hypercube. 11
2.6 A Cube-Connected Cycles network for 32 nodes. 12
2.7 Some legitimate states for a 2×2 switching element. 12
2.8 An $N \times M$ crossbar. 13
2.9 The Shuffle-Exchange network. 14
2.10 Thompson's non-blocking GCN. 17
2.11 The generalized 3-stage Clos network ($m \geq 2n - 1$). 18
2.12 The Beneš network. 20
2.13 Data Manipulator for $N = 8$. 22
2.14 8×8 banyan network. 24
2.15 8×8 Generalized Cube network, showing name representation and path from input 2 to output 4. 25
2.16 The Baseline network. 26
2.17 Hybrid synchronization scheme. 34

3.1 Examples of switching fabrics found in the literature. 37
3.2 Switching element SE1 (buffers before the switching multiplexors). . 38
3.3 Blocking for networks based on SE1. 39
3.4 Buffer and link arrangement of switching element SE2A. 40

3.5 Blocking for networks based on SE2A. 41
3.6 Buffer and link arrangement of switching element SE2B. 41
3.7 Blocking for networks based on SE2B. 42
3.8 An 8×8 delta network. 44

4.1 Conceptual view of the network used in the analysis. 48
4.2 Model of SE with buffers before the crossbar. 50
4.3 State transition diagram for an SE buffer. 55
4.4 Model of SE with buffers inside the crossbar. 58
4.5 Detail of a buffer group for model of SE with buffers inside the crossbar. 59
4.6 Total packet arrivals versus number of outputs for 95% and 99% confidence levels. 67
4.7 Throughput versus Offered Load: 2×2 SE with buffers before the crossbar. 71
4.8 Normalized Delay versus Offered Load: 2×2 SE with buffers before the crossbar. 71
4.9 Throughput versus Offered Load: 2×2 SE with buffers inside the crossbar and no look-ahead routing. 72
4.10 Normalized Delay versus Offered Load: 2×2 SE with buffers inside the crossbar and no look-ahead routing. 72
4.11 Throughput versus Offered Load: 2×2 SE with buffers inside the crossbar and 1-stage look-ahead routing. 73
4.12 Normalized Delay versus Offered Load: 2×2 SE with buffers inside the crossbar and 1-stage look-ahead routing. 73
4.13 Throughput versus Offered Load: 8×8 network based on 2×2 SE's with buffers before the crossbar. 74
4.14 Normalized Delay versus Offered Load: 8×8 network based on 2×2 SE's with buffers before the crossbar. 74
4.15 Throughput versus Offered Load: 8×8 network based on 2×2 SE's with buffers inside the crossbar and no look-ahead routing. 75
4.16 Normalized Delay versus Offered Load: 8×8 network based on 2×2 SE's with buffers inside the crossbar and no look-ahead routing. . . 75

4.17 Throughput versus Offered Load: 8×8 network based on 2×2 SE's with buffers inside the crossbar and 1-stage look-ahead routing. . . 76
4.18 Normalized Delay versus Offered Load: 8×8 network based on 2×2 SE's with buffers inside the crossbar and 1-stage look-ahead routing. 76
4.19 Throughput versus Number of Stages: network based on 2×2 SE's with buffers before the crossbar, offered load 1.0. 77
4.20 Normalized Delay versus Number of Stages: network based on 2×2 SE's with buffers before the crossbar, offered load 1.0. 77
4.21 Throughput versus Number of Stages: network based on 2×2 SE's with buffers inside the crossbar, no look-ahead routing, offered load 1.0. 78
4.22 Normalized Delay versus Number of Stages: network based on 2×2 SE's with buffers inside the crossbar, no look-ahead routing, offered load 1.0. 78
4.23 Throughput versus Number of Stages: network based on 2×2 SE's with buffers inside the crossbar, 1-stage look-ahead routing, offered load 1.0. 79
4.24 Normalized Delay versus Number of Stages: network based on 2×2 SE's with buffers inside the crossbar, 1-stage look-ahead routing, offered load 1.0. 79
4.25 Throughput versus Number of Stages: network based on 2×2 SE's with buffers before the crossbar, offered load 0.8. 80
4.26 Normalized Delay versus Number of Stages: network based on 2×2 SE's with buffers before the crossbar, offered load 0.8. 80
4.27 Throughput versus Number of Stages: network based on 2×2 SE's with buffers inside the crossbar, no look-ahead routing, offered load 0.8. 81
4.28 Normalized Delay versus Number of Stages: network based on 2×2 SE's with buffers inside the crossbar, no look-ahead routing, offered load 0.8. 81
4.29 Throughput versus Number of Stages: network based on 2×2 SE's with buffers inside the crossbar, 1-stage look-ahead routing, offered load 0.8. 82

4.30 Normalized Delay versus Number of Stages: network based on 2×2 SE's with buffers inside the crossbar, 1-stage look-ahead routing, offered load 0.8. 82
4.31 Throughput versus Offered Load for dilated and non-dilated networks (simulated): 8×8 network based on SE2A. 83
4.32 Normalized Delay versus Offered Load for dilated and non-dilated networks (simulated): 8×8 network based on SE2A. 83
4.33 Throughput versus Offered Load for dilated and non-dilated networks (simulated): 8×8 network based on SE2B. 84
4.34 Normalized Delay versus Offered Load for dilated and non-dilated networks (simulated): 8×8 network based on SE2B. 84
4.35 Packet delay distribution for various arbitration schemes (simulated): 8×8 network based on SE2A, 32 slots/SE, offered load 1.0. 85
4.36 Packet delay distribution for various arbitration schemes (simulated): 8×8 network based on SE2A, 32 slots/SE, offered load 0.8. 85
4.37 Packet delay distribution for various arbitration schemes (simulated): 8×8 network based on SE2B, 32 slots/SE, offered load 1.0. 86
4.38 Packet delay distribution for various arbitration schemes (simulated): 8×8 network based on SE2B, 32 slots/SE, offered load 0.8. 86
4.39 Packet delay distribution with and without virtual cut-through (simulated): 8×8 network based on SE2A, 32 slots/SE, offered load 1.0. 87
4.40 Packet delay distribution with and without virtual cut-through (simulated): 8×8 network based on SE2A, 32 slots/SE, offered load 0.8. 87
4.41 Packet delay distribution with and without virtual cut-through (simulated): 8×8 network based on SE2A, 16 slots/SE, offered load 1.0. 88
4.42 Packet delay distribution with and without virtual cut-through (simulated): 8×8 network based on SE2A, 16 slots/SE, offered load 0.8. 88

4.43 Packet delay distribution based on action taken for blocked requests (simulated): 8×8 network based on SE2A, 32 slots/SE, offered load 1.0. 89
4.44 Packet delay distribution based on action taken for blocked requests (simulated): 8×8 network based on SE2A, 32 slots/SE, offered load 0.8. 89
4.45 Throughput versus Number of Stages for global and local arbitration communication methods: network based on SE2A, offered load 1.0. 90
4.46 Normalized Delay versus Number of Stages for global and local arbitration communication methods: network based on SE2A, offered load 1.0. 90
4.47 Throughput versus Number of Stages for global and local arbitration communication methods: network based on SE2B, offered load 1.0. 91
4.48 Normalized Delay versus Number of Stages for global and local arbitration communication methods: network based on SE2B, offered load 1.0. 91

5.1 Example of BISDN architecture. 96
5.2 Example of local exchange node architecture. 96
5.3 ATM switch architecture. 97
5.4 8×8 ADN constructed by adding 2 stages to an 8×8 delta network. 98
5.5 SE-to-SE handshaking: initiating a packet transfer. 105
5.6 SE-to-SE handshaking: completing a packet transfer. 106
5.7 Top-level block diagram for SE. 109
5.8 Block diagram of input port server. 111
5.9 Block diagram detailing the IPS datapath. 113
5.10 State diagram for the input port controller. 114
5.11 Block diagram of buffer memory. 116
5.12 Block diagram of FIFO. 117
5.13 Timing diagram for the write operation. 118
5.14 Timing diagram for the read operation. 118
5.15 Algorithm for managing the FIFO pointers during write operations. 119
5.16 Algorithm for managing the FIFO pointers during read operations. 120

5.17 Block diagram of FIFO controller. 121
5.18 State diagram for the RPTR controller. 121
5.19 State diagram for the FPTR controller. 122
5.20 Block diagram of output port server. 123
5.21 State diagram for the arbiter. 125
5.22 State diagram for the output port controller. 127

Acknowledgments

I thank my supervisor, Dr. Fayez El-Guibaly, for his guidance, financial support, and for not giving up on me when progress was slow.

I acknowledge the Alberta Microelectronic Centre for the use of computing equipment and computer-aided design tools.

My family, friends, and work colleagues expressed interest and encouragement and for this I thank each one of you.

Donna Dressler-Mund, my wife, was my most vital support in completing this thesis. Even when the going was tough, she never lost faith in me. Her encouragement and self-sacrifice provided me with the incentive and means to complete this project. For this, I am eternally grateful.

To my wife, Donna
and my son, Michael

and

To a dear friend, Shannon

List of Symbols

ACK	Acknowledgement
ADM	Augmented Data Manipulator
ADN	Augmented Delta Network
AN	Access Node
ATD	Asynchronous Time Division
ATM	Asynchronous Transfer Mode
BE	Buffer Empty (eg., BE0, BE1)
BEC	Buffer Empty Clear
BES	Buffer Empty Set
BF	Buffer Full (eg., BF0, BF1)
BFC	Buffer Full Clear
BFS	Buffer Full Set
BISDN	Broadband Integrated Services Digital Network
BIST	Built-In Self Test
CCC	Cube-Connected Cycles
CCITT	International Telegraph and Telephone Consultative Committee
CLR	Clear
CMOS	Complementary MOS
CNT	Count
CP	Customer Premises Node
CRC	Cyclic Redundancy Check
DFT	Design for Testability
DPM	Dual Port Memory
DRN	Dynamic Redundancy Network
ERR	Error
ESC	Extra Stage Cube
FC	FIFO Controller
FCLR	FPTR Clear
FERR	FPTR Error
FFT	Fast Fourier Transform

FIFO	First In, First Out
FINC	FPTR Increment
FM	FIFO Memory (eg., FM0, FM1, FM2, FM3)
FPS	Fast Packet Switching
FPTR	Front Pointer
FSM	Finite-State Machine
GCN	Generalized Connection Network
HOL	Head-of-Line
IADM	Inverse Augmented Data Manipulator
IPC	Input Port Controller
IPS	Input Port Server
ISDN	Integrated Services Digital Network
ITC	Input Trunk Controller
JTAG	Joint Test Action Group
LEN	Local Exchange Node
MIN	Multistage Interconnection Network
MOS	Metal Oxide Semiconductor
NAK	Negative Acknowledgement
NFER	Next FPTR Equals RPTR
NREF	Next RPTR Equals FPTR
NMOS	N-channel MOS
OPC	Output Port Controller
OPS	Output Port Server
OTC	Output Trunk Controller
PERR	Parity Error
PM2I	Plus-Minus 2^i
PMOS	P-channel MOS
POK	Parity Okay
PR	Packet Ready (eg., PR0, PR1)
PTM	Packet Transfer Mode
RCLR	RPTR Clear
RE	Read Enable (eg., RE0, RE1)
REQ	Request
RERR	RPTR Error
RINC	RPTR Increment
RMN	Remote Multiplexer Node
RPTR	Rear Pointer
RSFF	Reset-Set Flip-Flop
SAF	Stuck-at-fault
SE	Switching Element

SF	Switch Fabric
SIMD	Single Instruction stream—Multiple Data stream
SONET	Synchronous Optical Network
STD	Synchronous Time Division
TAP	Test Access Port
TC	Trunk Controller
TDI	Test Data Input
TDO	Test Data Output
TEN	Transit Exchange Node
VCT	Virtual Cut-through
VLSI	Very Large Scale Integration
WE	Write Enable (eg., WE0, WE1)
XFR	Transfer

Chapter 1

Introduction

1.1 Introduction

Historically, the evolution of telecommunications has seen the development of different networks to provide subscribers with various communications services. Thus, the subscriber must interface to several networks to meet video, voice, and data requirements. Furthermore, for two subscribers to communicate with each other, they must have access to the same network, or some form of internetwork gateway must be available between the particular networks for which they have access. The desire, then, is to provide the subscriber with integrated access to all services over a single connection to the network. This is the objective of developing an Integrated Services Digital Network (ISDN).

The CCITT I-Series Recommendations specify a Basic User-Network Interface of 192 kbits/sec (I.430) and a Primary Rate User-Network Interface of 1,544 (or 2,048) kbits/sec (I.431) [1]. For Basic Access, this bandwidth is provided as two 64-kbits/sec B channels and one 16-kbits/sec D channel. For Primary Rate Access, 23 (or 30) 64-kbits/sec B channels and one 64-kbits/sec D channel are provided. These channels, based on a circuit-switching approach at 64 kbits/sec per channel, can provide good voice and text data service. However, the circuit-switched approach results in an inflexible network, since channel rates must be predefined. With changing customer requirements, future services will require a wide range of average bit rates, peak bit rates, and call holding times, which cannot be optimized by the 64-kbits/sec circuit-switched ISDN [2].

The future Broadband ISDN (BISDN) must be flexible so that it can adapt to meet the unexpected future service requirements. Current research for a BISDN switching and transmission methodology focuses on methods which the CCITT has categorized as Packet Transfer Mode (PTM) [3]. Asynchronous time division (ATD) and fast packet switching (FPS) are both variants of PTM. ATD is also referred to as asynchronous transfer mode (ATM).

ATD evolved from STD (synchronous time division). In STD, time slots are assigned on a per call basis and are characterized by their position within a frame. In ATD, time slots are assigned on a dynamic basis and characterized by a label designating the logical connection. Packet size in ATD is typically small and fixed. ATD was originally proposed to transport video primarily, as well as data and voice. According to [3], the CCITT has opted for the ATD technique with fixed length packets (called cells). However, many issues still remain to be resolved by the CCITT.

FPS evolved from packet switching. In conventional packet-switched networks, link protocols are implemented by software. This imposes delay and performance problems, making them inadequate for BISDN. In FPS, link protocols are less complex, since links are assumed to be very reliable and error correction can be done end-to-end if necessary. This allows all protocol processing to be done in hardware, thereby obtaining high speeds. Time stamping and distinction between different services is still performed inside the network. Packet size in FPS is typically large and variable. FPS was originally proposed to transport high-speed data primarily.

Enormous amounts of research have been performed under both the ATD concept and the FPS concept. Although the CCITT appears to favor the ATD approach, whether ATD or FPS gains final acceptance, and with what options, is still questionable. Thus, due to the lack of standards, many BISDN experiments have been implemented with trunk controllers (TC's) which convert any desired network transmission protocol (ATD or FPS) to a unique protocol for the particular switch fabric. This allows experiments to be performed on the switch fabric and on the network transmission protocols independently. Such an approach is adopted in this thesis to effect the design of a 2×2 switching element for a specific BISDN switch fabric.

1.2 Contributions in this Thesis

A set of equations is developed for analyzing the performance of delta networks. The equations are more general than those found in the literature. Those in the literature are for analyzing networks composed of rectangular $a \times a$ switching elements (SE's) with buffers before the crossbar. The equations presented here extend the analysis to arbitrary $a \times b$ SE's, with buffers either before or inside the crossbar. The equations also allow for the analysis of a proposed look-ahead routing scheme.

The high-level design for a 2×2 SE with buffers inside the crossbar is given. This SE may be used in the switching fabric of a BISDN node.

1.3 Outline of the Thesis

Before beginning the design of an SE for the switching fabric of a BISDN node, an interconnection network for the switching fabric must be selected. Chapter 2 presents a brief introduction to interconnection networks. Presented are network topologies, switching methodologies, control strategies, and operation modes. This will provide the reader with the necessary background on interconnection networks for subsequent chapters.

In Chapter 3, a network topology is selected and SE architectural parameters are identified.

In Chapter 4, a set of equations is developed which allows investigating the effects of varying several of the SE architectural parameters identified in Chapter 3. Simulation results are included to support the equations, as well as to determine the effects of varying certain SE parameters not modeled by the equations.

In Chapter 5, a specific SE architecture is selected, based on the results of the analysis and simulations performed in Chapter 4. A high-level design for the SE is given.

Finally, Chapter 6 provides some concluding remarks for the thesis.

Chapter 2

Interconnection Networks

2.1 Introduction

This chapter presents a brief introduction to interconnection networks. More detailed information is found in many good surveys in the literature [4]–[9].

Interconnection networks made their debut in telephone switching from the need to be able to connect any given subscriber to any other subscriber. Some of the earliest theoretical work done was by Clos [10] and Beneš [11]. With the advent of concurrent processing as a means of increasing processing speed, interconnection networks were designed to link processors and memories together in various arrangements according to the type of parallelism desired.

Over the past few decades, many differing network topologies have been presented, usually with the goal of reducing the amount of delay through the network and/or the number of switching elements in the network.

The study of interconnection networks can be broken into the following categories:

1. Network Topology
2. Switching Methodology
3. Control Strategy
4. Operation Mode

2.2 Static Network Topologies

The *topology* of a network is the interconnection pattern used to connect the input lines to the output lines. The topology of a network may be either static or dynamic.

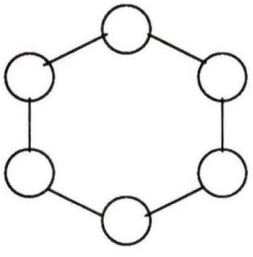
In static networks, links between processors cannot be reconfigured to interconnect the processors in a different arrangement. Static networks have been classified according to the number of dimensions required for layout [6]. A one-dimensional static network is simply the linear array of processors. Two-dimensional static network architectures include the ring and the star. These configurations have been used for local area networks and distributed computing (ring: [12]–[15], star: [16], [17]). Another useful two-dimensional architecture is the tree structure which has the advantage of the logarithmic-time delay property for broadcasting and searching [18]. Figure 2.1 shows examples of the ring, star, and tree structures.

The ring, star, and tree structures all have non-uniform layout characteristics. As nodes are added to a ring, the ring diameter increases. For layout, nodes must be packed in a non-circular fashion to conserve chip area. As nodes are added to a star, the center node requires a greater number of I/O ports. In the tree, the length of the communication path between sibling nodes increases from the leaf nodes towards the root node.

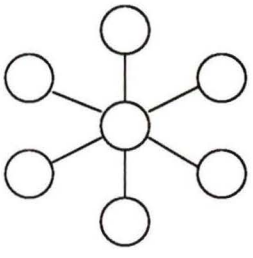
In designing two-dimensional array structures which are uniform in layout, mesh-connected structures are considered. The only mesh-connected structures which yield a simple and regular layout are the equilateral triangular array, the square array, and the hexagonal array shown in Figure 2.2 [18].

The Completely Connected network [19] is a network in which each node has a link to every other node of the network. It shows the greatest extreme of connectivity, with the advantage that a message can reach its destination by traversing a single link. Also, if certain nodes or links fail, alternate paths may be used to reach the destination in more than a single hop. The drawback is that each node must support N-1 links to other nodes, which is not feasible even for modest N.

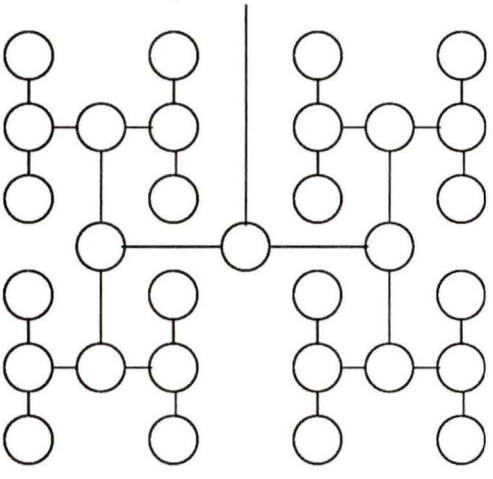
The Chordal Ring [20] is a three-dimensional network based on the ring structure. Each node of the ring has an additional link, called a chord, to another node across the network based on an odd chord length w. Assuming an even number



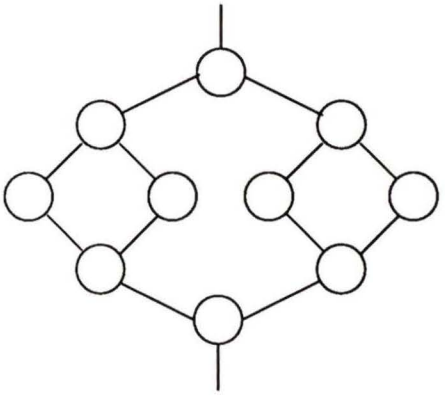
a) Ring structured network



b) Star structured network

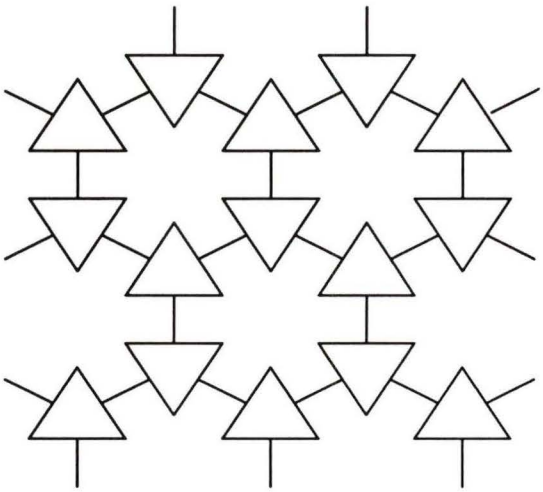


c) Binary Tree structured network

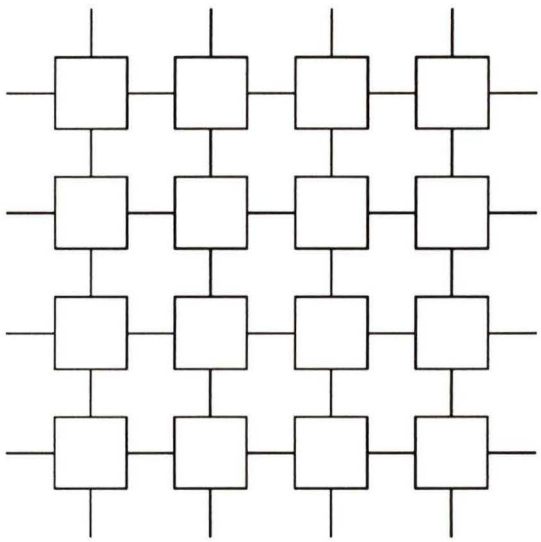


d) Double Tree structured network

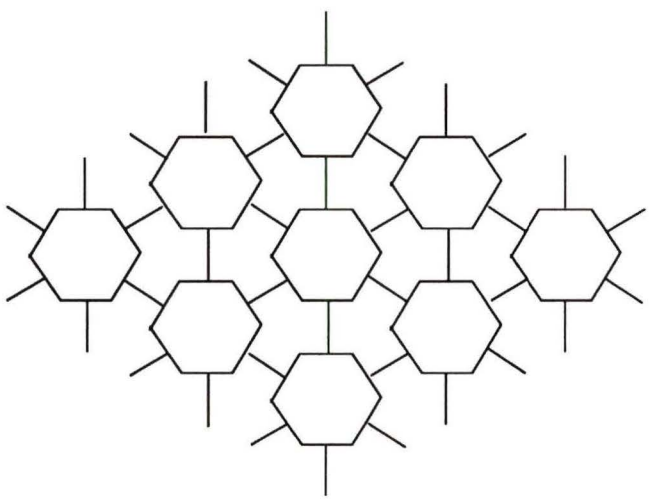
Figure 2.1: Examples of two-dimensional static networks.



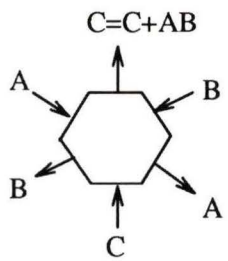
a) Equilateral Triangular array



b) Square array



c) Hexagonal array



d) Example of data flow through each element of the hexagonal array.

Figure 2.2: Mesh-connected two-dimensional arrays.

of nodes indexed 0,1,2,...,$n-1$ around the ring, each odd-numbered node i is connected to node $(i+w)\text{mod}(n)$, while each even-numbered node j is connected to node $(j-w)\text{mod}(n)$ [20]. Although the number of links to be supported by each node is only three, adding nodes to the network is difficult. Not only must the ring's diameter increase, but existing links must be rearranged. Figure 2.3 shows a Chordal Ring network with $n = 8$ and $w = 3$.

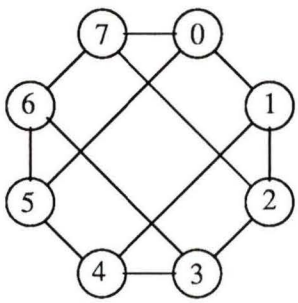


Figure 2.3: Chordal Ring network with $n = 8$ and $w = 3$.

In general, $N = m^n$ processing elements may be interconnected in a regular n-dimensional 'cube' with m processors per edge of the cube, referred to as an m-ary n-cube. A benefit of this structure is that neighboring nodes have addresses which differ in only one dimension. The routing tag of a message need only be searched until a bit is encountered which is different than the corresponding bit of the local node address. This immediately determines a dimension in which the message can move. A message can reach its destination in at most $(m-1)n$ hops. Figure 2.4 shows a few m-ary n-cubes for different values of m and n.

The *hypercube* structure is a subset of the m-ary n-cube structures with $m = 2$ (i.e., 2-ary or binary n-cube). Thus, the single parameter n, termed the *hypercube dimension*, characterizes the hypercube. An n-dimensional hypercube contains 2^n nodes, each node has n links to neighboring nodes, and a message from a node can reach any other node in at most n hops. The structure of the hypercube makes message forwarding simple. A message arriving at a node may be forwarded on any link for which the corresponding bit in the message's destination address differs from the local node address.

A zero-dimensional hypercube is a single node (by definition). A linear array of two nodes, a near-neighbor mesh of four nodes, and a binary 3-cube are examples of one-, two-, and three-dimensional hypercubes respectively. An $(n+1)$-

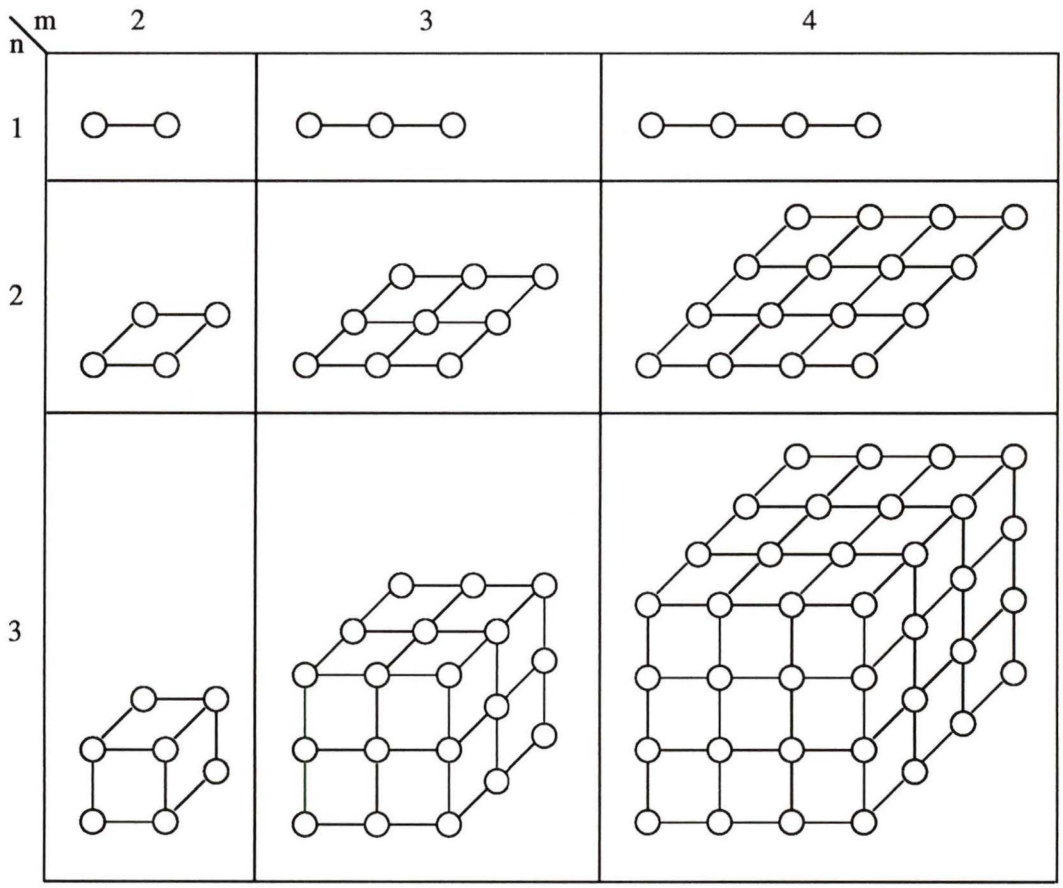


Figure 2.4: Examples of m-ary n-cubes. For clarity, not all links are shown.

dimensional hypercube can be created by constructing two n-dimensional hypercubes and adding a link from each node in the first hypercube to its 'twin' in the second [21]. This is depicted in Figure 2.5.

A drawback of the m-ary n-cubes is that the number of links supported by a processor increases with n, making them not readily usable for VLSI design. Preparata and Vuillemin [22] introduced the Cube-Connected Cycles (CCC) interconnection pattern as an alternative to the cube-connected network. The CCC reduces the connections per processor to three. The CCC can emulate both the cube-connected network and the shuffle-exchange network with no significant degradation of performance. Although the CCC has the same capability and bounded fan-out of modules as the shuffle-exchange interconnection, it has a layout area $O(N^2/log_2^2 N)$, which is smaller than the best known shuffle-exchange layout $O(N^2/log_2^{3/2} N)$[22].

A CCC with $N = 2^n$ nodes consists of 2^{n-r} cycles of 2^r nodes per cycle interconnected as an $(n-r)$-cube, where r is the smallest integer such that $r + 2^r \geq n$. For example, Figure 2.6 shows a CCC for $N = 32 = 2^5$ nodes. The smallest integer value for r is 2, resulting in 8 cycles of 4 nodes per cycle interconnected as a 3-cube.

2.3 Dynamic Network Topologies

Dynamic networks are usually composed of one or more stages of switching elements connected to one another by links. In general, a switching element may be considered as the smallest entity in the network which may be controlled to provide a certain permutation of its inputs at its outputs. It is typically a small, single-stage 2×2 crossbar, as shown in Figure 2.7. Such a switching element is a *two-function switching element* if only *straight* and *exchange* permutations (or states) are allowed. A *four-function switching element* also provides *lower broadcast* and *upper broadcast*. A *full communication switching element* provides straight, exchange, and same-side permutations. This can be used to allow a connection between two terminals on the same side of the network. *Paths* (or connections) in a network consist of an alternating sequence of links and switching elements. The paths can be reconfigured by changing the state of the network's switching elements.

Dynamic networks may be grouped into three classes, based on the stages of

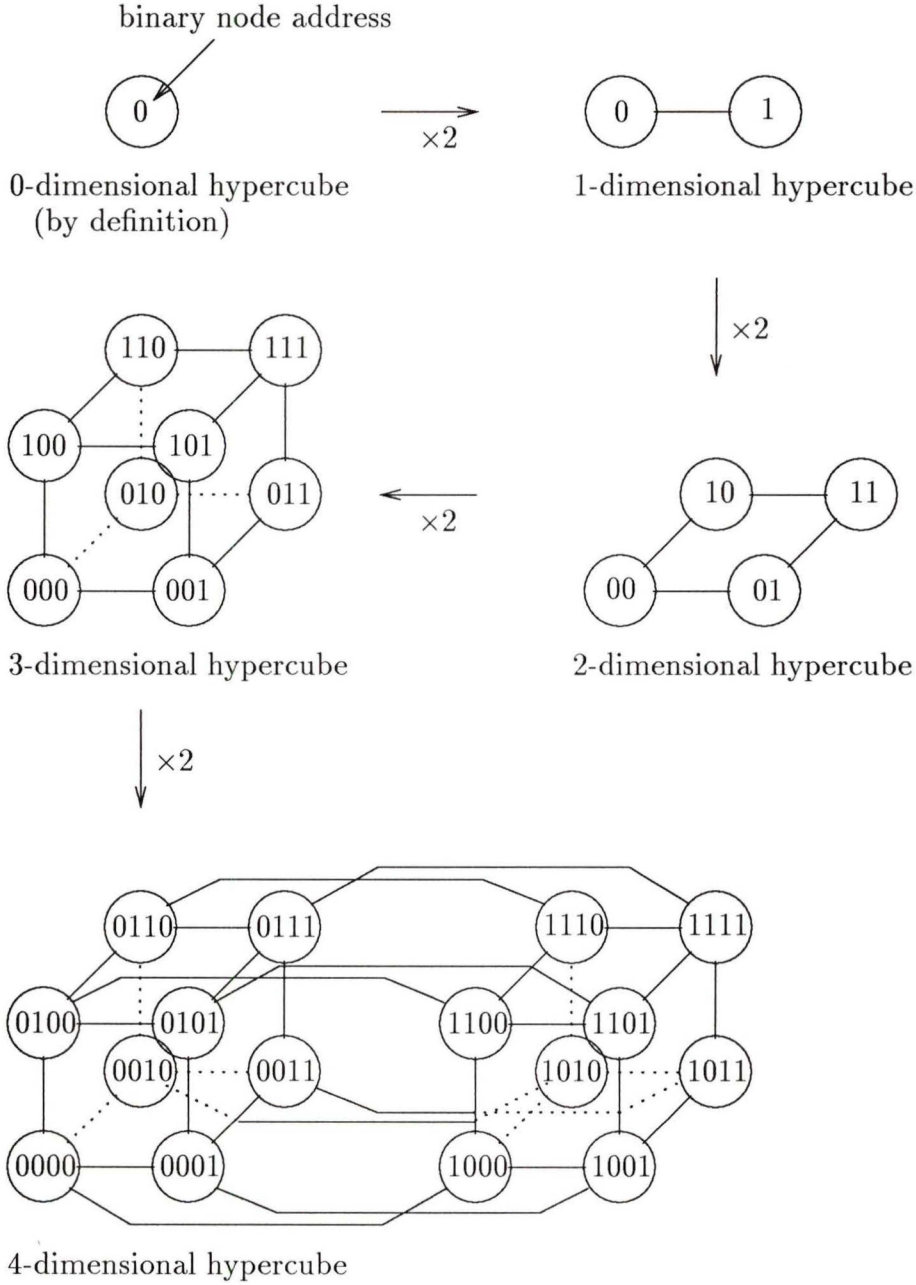


Figure 2.5: Recursive definition of hypercube.

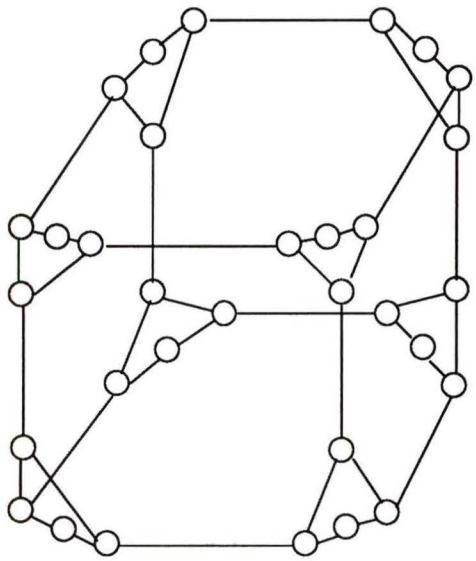


Figure 2.6: A Cube-Connected Cycles network for 32 nodes.

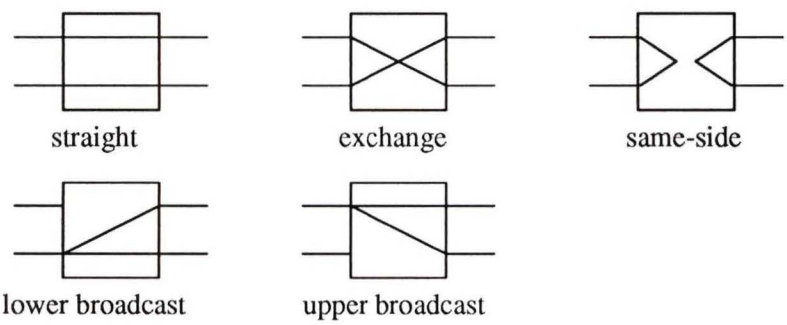


Figure 2.7: Some legitimate states for a 2×2 switching element.

switching elements. They are crossbar, single-stage, and multistage [6].

2.3.1 Crossbar Switch

A Crossbar Switch is essentially an $N \times M$ switch array (Figure 2.8) capable of connecting any input to a free output (irrespective of existing connections). An advantage of the crossbar is that there is a single switch delay on the path from any input to any output (independent of N and M). However, the major disadvantage of the crossbar is its layout cost which is $O(NM)$, making it infeasible for large systems. References to the use of crossbars may be found in [6] and [7].

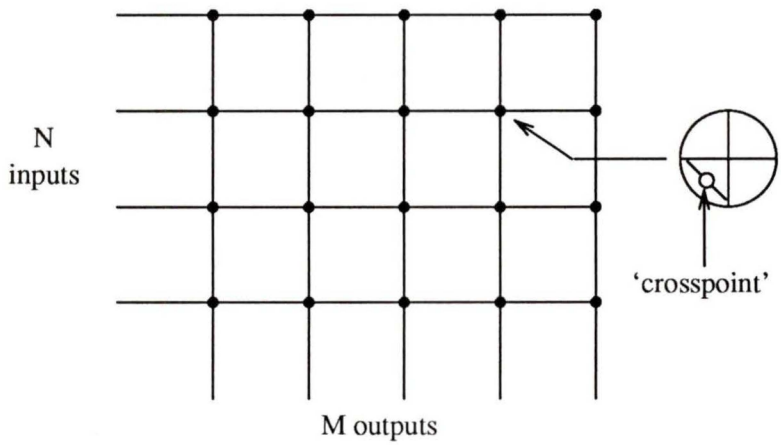


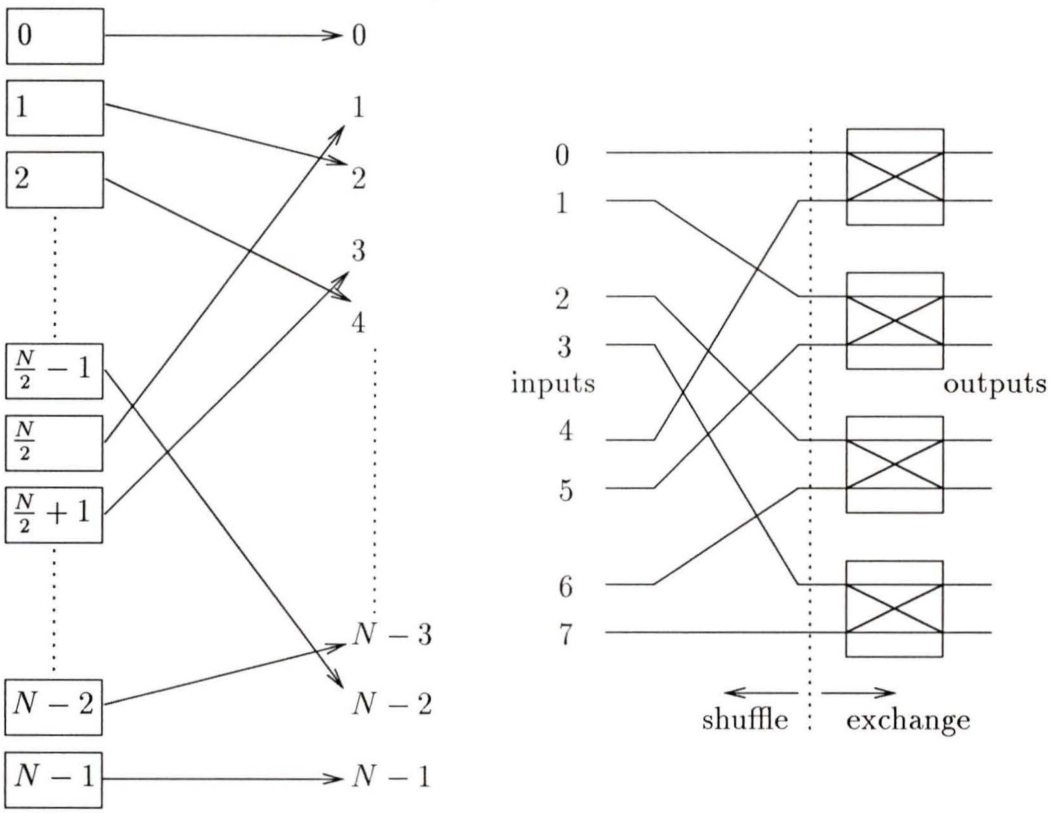
Figure 2.8: An $N \times M$ crossbar.

2.3.2 Single-stage interconnection networks

A useful single-stage network is the shuffle-exchange. The *perfect shuffle* interconnection pattern was introduced in 1971 by Stone [23]. Its name is derived by drawing an analogy to the perfect shuffle of a deck of cards. The inputs (the cards in the analogy) are divided into two groups (piles) and combined (shuffled) so as to alternate inputs of one group with inputs from the other group. This is shown pictorially in Figure 2.9(a).

The perfect shuffle may be expressed in terms of the binary representation of the n indices of each of the N inputs ($n = \log_2 N$):

$$shuffle(b_{n-1}b_{n-2}\ldots b_1 b_0) = b_{n-2}b_{n-3}\ldots b_1 b_0 b_{n-1}$$



a) A Perfect Shuffle on N inputs.

b) An 8×8 Shuffle-Exchange.

Figure 2.9: The Shuffle-Exchange network.

From this definition, it is evident that inputs 0 and N-1 do not change their position in the shuffle. Therefore, when used as an interconnection network, the perfect shuffle is usually followed by an exchange connection, as shown in Figure 2.9(b). The exchange may be defined as follows:

$$exchange(b_{n-1}b_{n-2}\ldots b_1b_0) = b_{n-1}b_{n-2}\ldots b_1\overline{b_0}$$

The exchange connection is simply a stage of 2×2 switching elements. Any arbitrary permutation of the N inputs is possible in $3(log_2N) - 1$ passes through the shuffle-exchange network [24].

2.3.3 Multistage interconnection networks

Multistage interconnection networks (MIN's) consist of several stages of switching elements and can usually connect any input terminal to any output terminal. In discussing MIN's, N refers to the number of input ports. If the MIN has the same number of outputs as inputs, then it is an $N \times N$ MIN, otherwise it is an $N \times M$ MIN. A *Permutation Network* is a MIN capable of performing all N! one-to-one mappings of its N inputs to its N outputs. A Generalized Connection Network (GCN) is a MIN capable of performing one-to-many mappings of inputs onto outputs, which is much more powerful than the one-to-one mappings of inputs onto outputs performed by other permutation networks. Several GCNs have been presented in the literature [25]–[28]. MIN's not capable of all N! permutations are either single-path networks (exactly one path between a given input and output) or multiple-path networks (more than one path from a given input to output).

If simultaneous connections of more than one input/output pair results in conflicts in the use of the network's internal links, then the network is a *blocking network*, otherwise, it is a *non-blocking network* [11]. If additional requests for connection can be satisfied without disturbing existing connections and irrespective of which state the history of connections and disconnections has left the network in, then the network is said to be *non-blocking in the strict sense.* If blocking states exist, but these blocking states can be avoided by following some rule in establishing new connections, then the network is said to be *non-blocking in the wide sense.*

A *rearrangeable non-blocking network* can satisfy all requests for connections by rearranging its existing connections, thereby circumventing blocking. This should not be confused with a network which is non-blocking in the wide sense. A network which is non-blocking in the wide sense does not rearrange existing connections—rather, new connections are established such that blocking of future requests will not occur.

Permutation networks may be either non-blocking or rearrangeable. Single-path networks are blocking and multiple-path networks may be blocking or rearrangeable.

A MIN may be *one-sided* or *two-sided.* One-sided networks have input and output ports on the same side, whereas two-sided networks have input ports on one side and output ports on another side. Generally, one-sided MIN's have an internal structure based on some two-sided network. Examples of one-sided MIN's given in [6] include one-sided cellular, one-sided baseline, and one-sided Clos.

Non-blocking MIN's

Thompson's $N \times N$ GCN [28] consists of two major sections—an $N \times N$ Generalizer and an $N \times N$ Connection Network. The $N \times N$ Connection Network is a switching network with N inputs and N outputs capable of passing any of the N! one-to-one permutations of inputs onto outputs. Its structure resembles that of the Beizer network [29]. The $N \times N$ Generalizer provides a particular number of copies of each input somewhere among its outputs. The $N \times N$ Generalizer can be realized by an $N \times N$ Hyperconcentrator (Figure 2.10(a)) followed by an $N \times N$ Infrageneralizer (Figure 2.10(b)), creating a topology identical to the $N \times N$ Connection Network used. The Hyperconcentrator selects the inputs to be propagated, grouping them beginning with its first output. The Infrageneralizer takes these selected inputs and provides the required number of copies of each. To do this, it uses four-function switching elements (all other switching elements of this network are the two-function type). The final Connection Network attaches the copies to the correct outputs.

A reduction of contact pairs can be made between the Hyperconcentrator and Infrageneralizer by eliminating a stage of switching elements which are identical in both. Also, a further reduction of contact pairs can be made by eliminating a

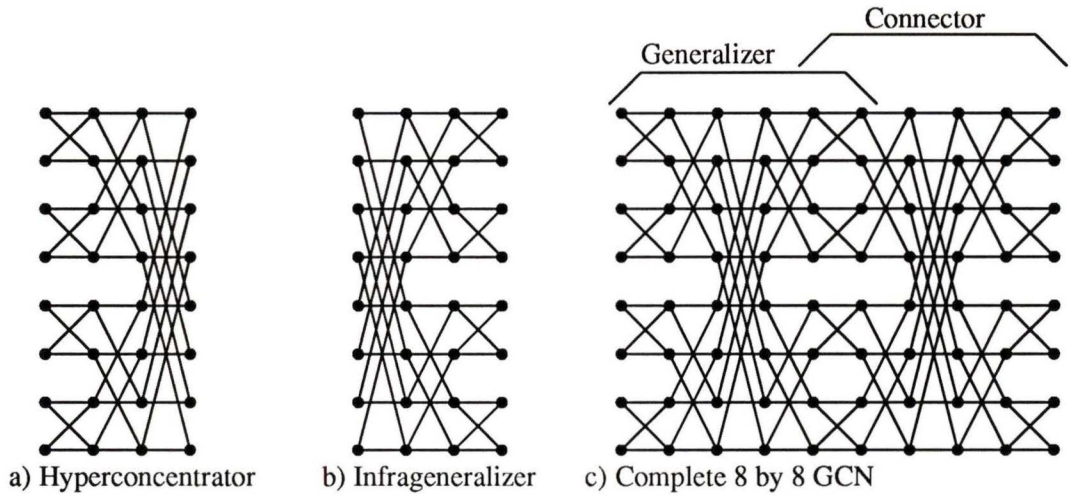


Figure 2.10: Thompson's non-blocking GCN.

stage of switching elements between the Generalizer and the Connection Network, finally arriving at the GCN shown in Figure 2.10(c).

The Clos network [10] is a non-blocking (in the strict sense) $N \times N$ Permutation Network (i.e., capable of performing one-to-one connections only). The optimal number of stages for the Clos network depends on N. For $N < 24$, the crossbar has fewer crosspoints (switching elements) than any Clos topology. For $24 \leq N < 161$, a 3-stage Clos network has the minimum number of crosspoints. If $N = n \times r$, n and r integers, then r input $n \times m$ crossbars, m intermediary $r \times r$ crossbars, and r output $m \times n$ crossbars, may be used to construct a 3-stage Clos network, where $m \geq 2n - 1$ (see Figure 2.11). The optimal value for n depends on N, as shown in [10]. For cases where N is not evenly divisible by an integer, two sizes of input crossbars, two sizes of intermediary crossbars, and two sizes of output crossbars are required.

Rearrangeable MIN's

Beneš in 1965 presented a construction for a rearrangeable MIN built of stages of identical square switches symmetrically arranged around a center stage [11]. This appears to be a rediscovery of the 1962 work of Beizer [29], but since Beneš developed several theorems pertaining to the network, it is commonly referred to as the Beneš Network.

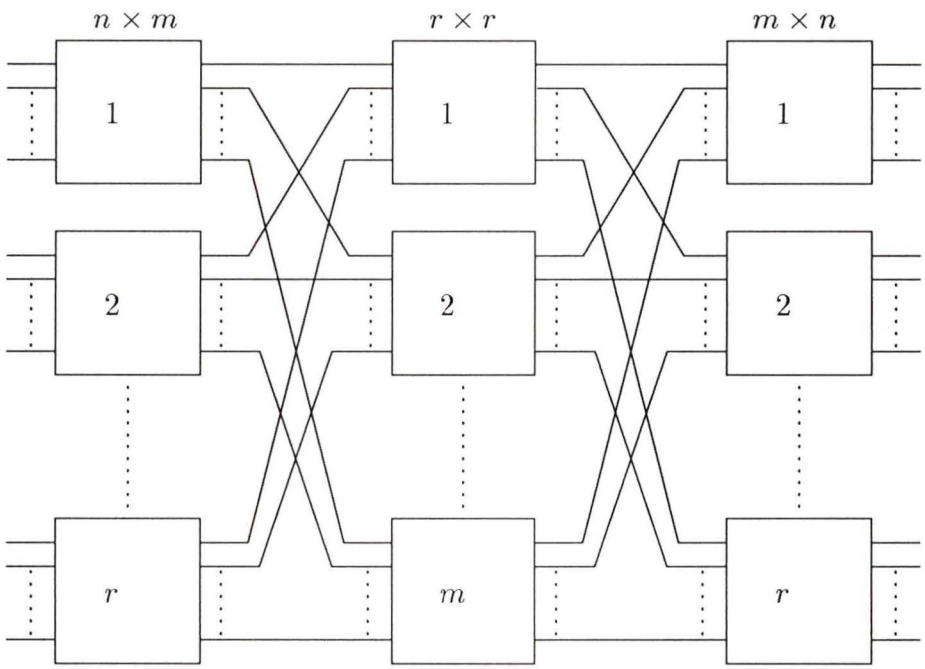


Figure 2.11: The generalized 3-stage Clos network ($m \geq 2n - 1$).

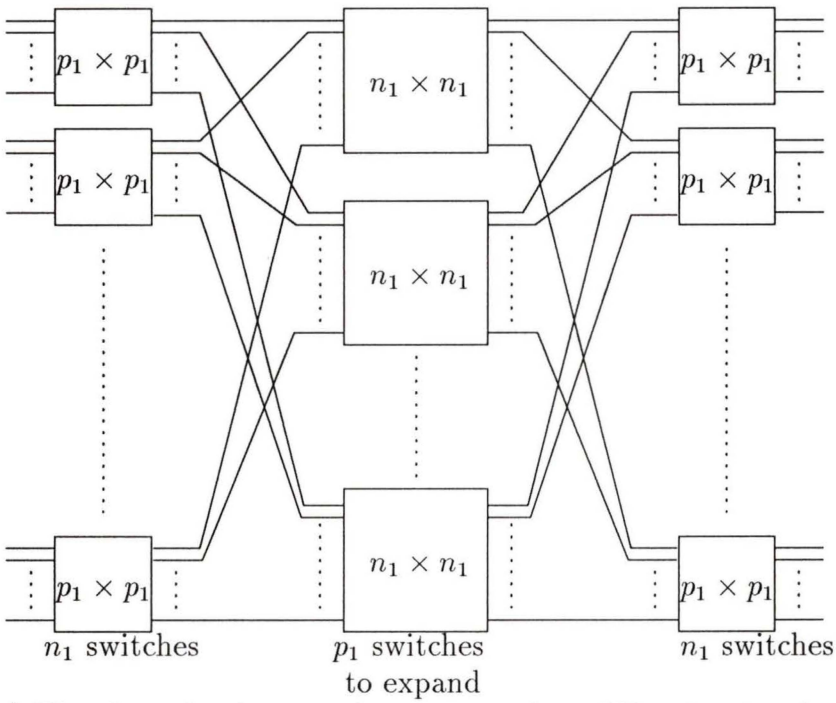
In general, Beneš found that the optimal (minimal cost) network should have as many stages as possible and switches that are as small as possible. The largest switches should be in the middle stage. The number of stages needed for an $N \times N$ network is $2x - 1$ where x is the number of prime divisors of N. To construct the $N \times N$ network, the following steps could be performed.

1. Factor $N = p_1 p_2 \ldots p_k \ldots p_x$, $k = 1, 2, \ldots$, where p_k is the k-th of x primes of N arranged in ascending order.
2. Set $k = 1$
3. Construct three stages of switches, the first and last stages having n_k $p_k \times p_k$ switches and the middle stage having p_k $n_k \times n_k$ switches.
4. Connect the first stage outputs to the middle stage inputs as follows:
   - Connect the p_k outputs of the first switch of the first stage, one to each of the middle p_k switches (first input).
   - Connect the p_k outputs of the second switch of the first stage, one to each of the middle p_k switches (second input).
   - Repeat this for each of the switches of the first stage.
5. Connect the last stage inputs to the middle stage outputs such that the connections "mirror" those from the first stage to the middle stage.
6. Repeat steps 3–5 recursively for each of the p_k middle switches, until all primes have been exhausted.

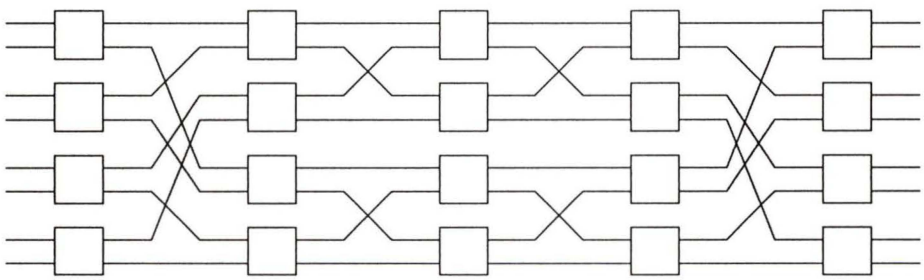
The first iteration of this recursive construction is shown in Figure 2.12.

Beneš also presented equations for determining the minimum cost in crosspoints of the network. The minimum cost for a network with $N = 2^n$ inputs is $4N(log_2 N - 1)$ (not $4N(log_2 N - 2)$, as Beneš stated) with the middle stage composed of 4×4 switches. If the entire network is to be composed of 2×2 switches, then the cost will be slightly higher at $2N(2log_2 N - 1)$ (i.e., $2N$ more crosspoints).

Joel in 1968 replaced all 2×2 crossbars in the Beneš network with a two-state or binary (β) element [30]. The two states allowed are straight and exchange. Joel replaced the 4×4 crossbar switches of the middle stage with a 3-stage 4×4 Beneš



a) First iteration in recursive construction of Beneš network.



b) 8×8 Beneš network based on 2×2 switching elements.

Figure 2.12: The Beneš network.

equivalent network composed of 6 β elements. This represented great savings on components since the 4×4 crossbar required 16 electromagnets, while the Beneš 3-stage equivalent required only 6 electromagnets. With respect to VLSI, a 4×4 crossbar requires 16 control lines, whereas the 6 β elements require just 6 control lines in total. On the other hand, the crosspoints are also a critical issue since they correspond to transmission gates or three-state buffers. A 4×4 crossbar has 16 crosspoints whereas the 6 β elements have a total of 24 crosspoints, making the 4×4 crossbar more favorable based on crosspoints. The area taken by the two methods in an actual VLSI implementation would be the deciding factor.

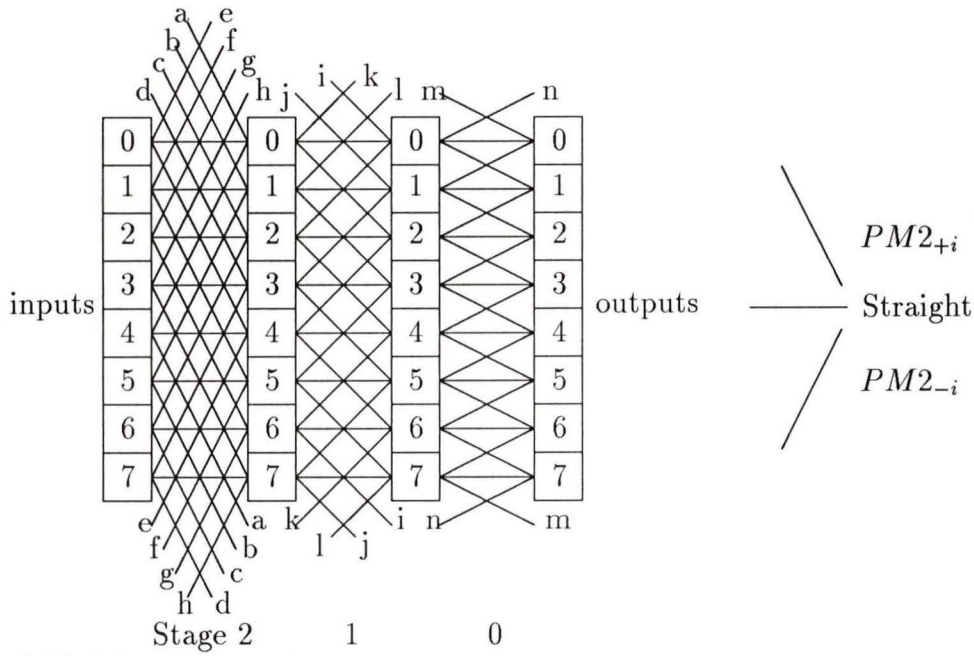
Waksman in 1968 modified the Beneš network by removing the top right output switch from the main network and from every $N/2$ subnetwork in the recursive construction [31]. This resulted in $4(Nlog_2N - N + 1)$ crosspoints, $2N - 4$ fewer than Joel's, but still being 4 more than Beneš' with a 4×4 crossbar middle stage.

Multiple-path blocking MIN's

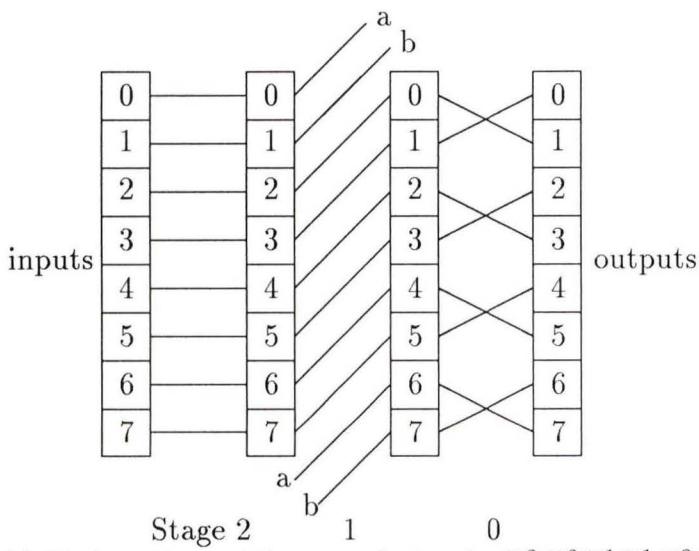
A type of multiple-path blocking MIN is the Plus-Minus 2^i (PM2I) network [32]. The name comes from the mapping of source addresses to destinations, since it can add or subtract 2^i from the address in stage i. The *data manipulator* network [33] is a PM2I network for N processors. It consists of $n = log_2N$ stages, each stage consisting of N cells (or switching elements) and the associated links to the next stage (Figure 2.13(a)). The stages are ordered from $n - 1$ on the left to 0 on the right. An additional column of output nodes follows the last stage. The cells have three inputs and three outputs and are capable of selecting one input to be routed to one, two, or all three outputs. The interconnection functions for each cell j of stage i are:

1. $PM2_{-i}(j) = (j - 2^i) \mod N$
2. identity (straight across)
3. $PM2_{+i}(j) = (j + 2^i) \mod N$

$PM2_{+(n-1)}(j) = PM2_{-(n-1)}(j)$, so for stage $(n - 1)$, there is actually only 2 different outputs from each cell.



a) Link interconnection arrangement.



b) Link setting with control signals $H_1^2 H_2^2 U_1^1 U_2^1 D_1^0 U_2^0$.

Figure 2.13: Data Manipulator for $N = 8$.

Each cell of stage i has three control signals which select *up* (u^i), *down* (d_i), and *horizontal* (h^i) output lines. Cells whose i-th bit is 0 have these control signals grouped together as U_1^i, D_1^i, and H_1^i. Similarly, cells whose i-th bit is 1 have these control signals grouped together as U_2^i, D_2^i, and H_2^i. Figure 2.13(b) shows how the signals H_1^2, H_2^2, U_1^1, U_2^1, D_1^0, and U_2^0 would set the network.

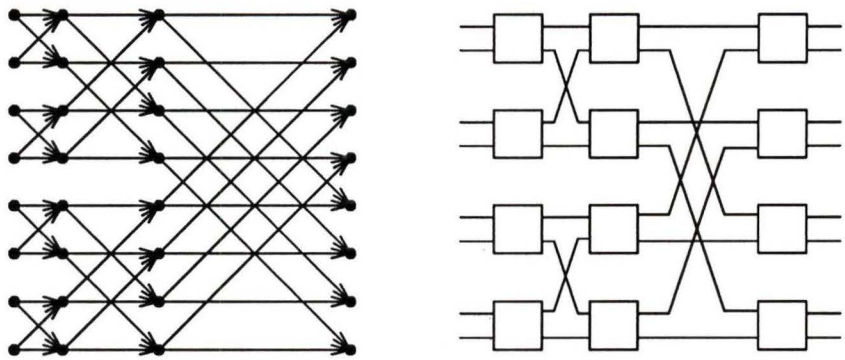
The *augmented data manipulator* (ADM), presented by Siegel and Smith [34], is a data manipulator with individual cell control, allowing greater control over permutations and, therefore, greater flexibility. The *inverse augmented data manipulator* (IADM) is an ADM in which the stages are traversed in the reverse order.

The *Gamma network*, proposed by Parker and Raghavendra [35], is topologically equivalent to the IADM from which it was adapted. It uses 3×3 crossbar switching elements whereas the IADM could only accept a single input and direct it to one of three outputs.

Single-path blocking MIN's

A popular class of single-path blocking MIN's is the *banyan network class.* Goke and Lipovski [36] define the banyan network in terms of its graphical representation. In the banyan structure, there is one and only one path from any given input to any given output. Figure 2.14 shows an interconnection graph which represents a banyan structure and the equivalent representation by switching elements. An L-level banyan is a banyan with its vertices arranged in $(L + 1)$ levels, numbered apexward from 0 to L, so that arcs of the graph can only exist between vertices in adjacent levels. A *regular banyan* is defined to be an L-level banyan with a constant number F (the *Fanout*) of arcs incident into each vertex and a constant number S (the *Spread*) of arcs emanating out from each vertex. F and S determine the size of the switching elements ($F \times S$) in a regular banyan network and, when $F = S$, create a *rectangular banyan* with the same number of vertices (and, therefore, switching elements) in each level.

An SW banyan is a regular banyan created by recursively expanding a crossbar structure. Details on this may be found in [36]. The banyan network of Figure 2.14 is a rectangular SW banyan with $S = F = 2$ (i.e., 2×2 switching elements). Many



a) Interconnection graph. b) Switch representation.

Figure 2.14: 8×8 banyan network.

blocking networks have been developed which are topologically equivalent to the rectangular SW banyan with $S = F = 2$. Two in particular were presented as a basis for comparison of networks in this class. They are Siegel and Smith's generalized cube network [34] and Wu and Feng's baseline network [37].

The $N \times N$ *generalized cube* (Figure 2.15) is defined to consist of $n = log_2 N$ stages, each stage consisting of N interconnecting lines and $N/2$ 2×2 *interchange boxes* (or switching elements). Data first passes through stage $n-1$, then $n-2$, etc., and finally the last stage 0. The outputs of a given interchange box are given the same numbering as its inputs. At stage i, $0 \leq i < n$, input lines differing only in their i-th bit position are paired together as inputs to an interchange box. The interchange boxes may be of the two-function variety or the four-function variety. Also, several control structures are defined. *Individual stage control* uses a single signal to set the state of all switches in a given stage to the same state. *Individual box control* allows each interchange box to be set to a desired state independently. *Partial stage control* uses $i+1$ control signals to control stage i, $0 \leq i < n$.

Siegel and Smith [34] showed that the generalized cube network is functionally equivalent to the STARAN flip network [38], the omega network [39], and the indirect binary n-cube [40]. They also defined an Augmented Data Manipulator (ADM) to be a Data Manipulator [33] with individual cell control and showed that the generalized cube with four-function boxes and individual box control performs a subset of the ADM functions.

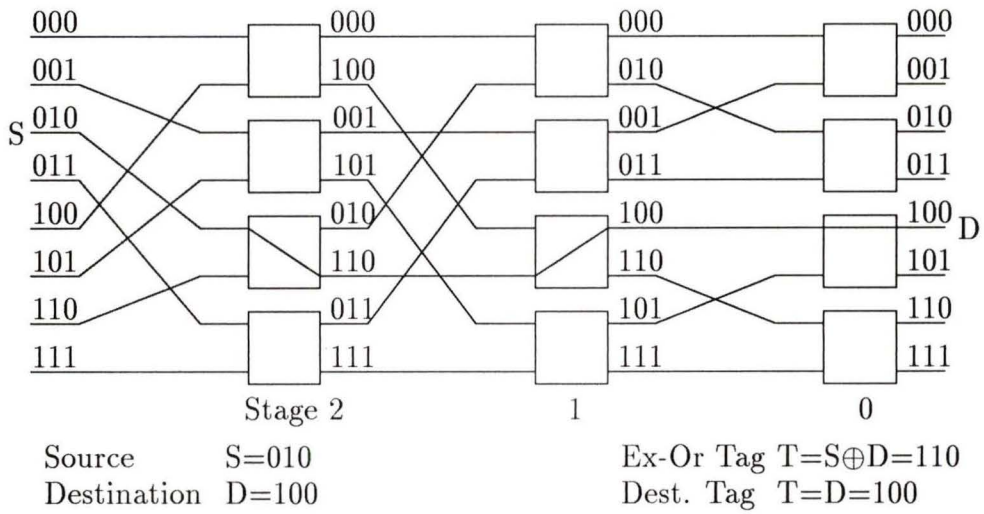


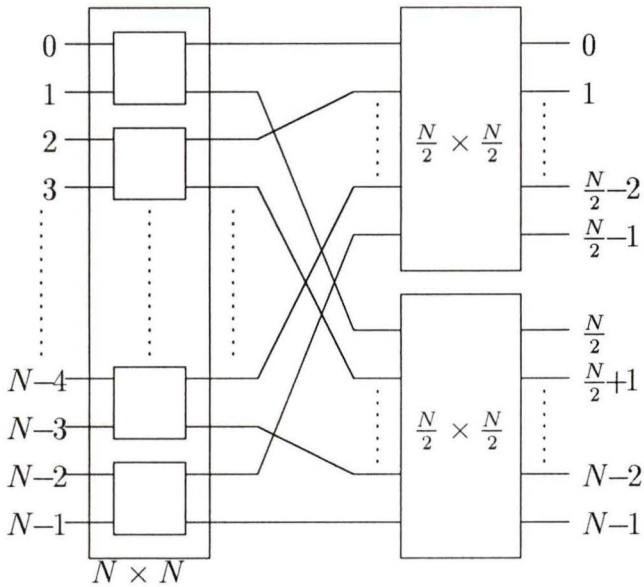
Figure 2.15: 8×8 Generalized Cube network, showing name representation and path from input 2 to output 4.

The $N \times N$ *baseline*, like the generalized cube, consists of $n = log_2 N$ stages of $N/2$ 2×2 switching elements. The switching element type may be two-function, four-function, or full communication. Stages are numbered from 0 on the left to $n - 1$ on the right (i.e., opposite to the generalized cube network). The levels of links are numbered from 0 on the left to n on the right.

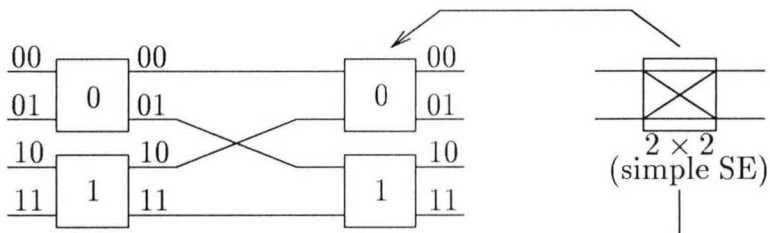
The baseline may be generated in a recursive manner. Figure 2.16(a) shows the first iteration of this process. The first stage is an $N \times N$ block and the second stage is two $N/2 \times N/2$ subblocks. After interconnecting these two stages with links as shown, the process is repeated recursively on the $N/2$ subblocks until the $N/2$ subblock is a 2×2 switching element. A *reverse baseline* is a baseline network in which the links are traversed in the opposite direction. Figures 2.16(b) and 2.16(c) show a 4×4 baseline and an 8×8 baseline respectively.

Wu and Feng [37] showed that the baseline network is topologically equivalent to the SW banyan network ($S = F = 2$) [36], the flip network [38], the omega network [39], the indirect binary n-cube [40], the modified data manipulator [33], and the reverse baseline.

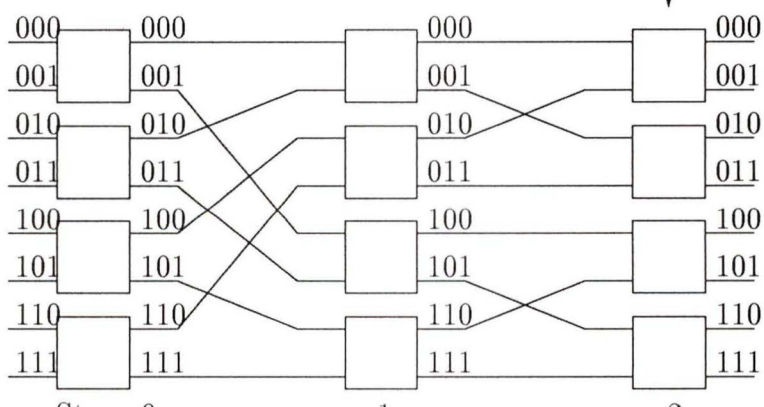
Another class of single-path blocking networks is the *delta network class* pro-



a) Recursive construction of the baseline network.



b) 4 × 4 baseline network.



c) 8 × 8 baseline network, showing name representation.

Figure 2.16: The Baseline network.

posed by Patel in 1981 [41]. It is approximately as general as the SW banyan, and all rectangular SW banyans may be considered a subset of the delta class. The network is created by superimposing trees of crossbar switches, for which a large number of link patterns is possible. Thus, for the same size network, many different topologies are possible and each may result in a different set of permutation capabilities. However, all delta networks of a given size give identical performance in terms of probability of acceptance or blocking for random access.

Patel describes formally a method for constructing delta networks with regular link patterns. He defines a q-shuffle of qr objects, q and r positive integers, as

$$S_{q*r}(i) = \begin{cases} qi \bmod (qr-1) & 0 \leq i < qr-1 \\ i & i = qr-1 \end{cases}$$

where i is the index to be shuffled. An $a^n \times b^n$ delta network consists of n stages of $a \times b$ crossbar modules, with adjacent stages interconnected by the a-shuffle. The desired output of a switching element in stage j is controlled by the j-th base-b digit taken from the destination address.

Fault tolerant MIN topologies

A MIN is i-fault tolerant if it can tolerate any set of i faults and still be able to connect any input to any output. Several approaches to incorporating fault tolerance into multistage networks have been proposed. One approach is to add extra stages of switches as in the extra stage cube [42], augmented delta network [43], and extra-stage gamma [44].

The Extra Stage Cube (ESC) is a network constructed by adding an extra switching stage at the input of the generalized cube, as well as hardware to allow the optional bypass of the extra stage or the output stage [42]. This creates two paths between any input/output pair, providing 1-fault tolerance to any switch or link failure (the hardware used to bypass the extra stage and last stage must be fault free).

A delta network may be augmented by adding extra stages resulting in multiple disjoint paths between a given input/output pair [43]. This network, referred to as an Augmented Delta Network (ADN), has no fault tolerance for the input stage switches or the output stage switches. Adding an extra stage of N/b $b \times b$ crossbar

switches results in b paths between any input and output. This provides $(b-1)$-fault tolerance to any link or interior switch failures [45].

Other methods of incorporating fault tolerance into multistage networks include adding paths as in the multipath Omega network [46] or adding links as in the F-network [47], dynamic redundancy network [48], and enhanced IADM [49].

Switches in the F-network [47] are 4×4 selectors, meaning they will route from only one input to one or more (broadcast) output(s) at a given time. Two of the four links from an SE to its successor are identical to those found in the generalized cube structure. At any SE (except those in the last stage), there are always two possible alternatives for correctly routing a message to its destination.

The dynamic redundancy network (DRN) may be derived from a graph representation of the generalized cube network. A *row* may be defined to be all nodes having the same address plus the links interconnecting these nodes. S rows are added to the N rows initially present. The original generalized cube network has two links from each switch j at stage i to stage $i-1$. The DRN has three links, one to switch $(j-2^i) \bmod (N+S)$, one to switch j, and one to switch $(j+2^i) \bmod (N+S)$ (resembling the Data Manipulator interconnection pattern). The S additional rows provide redundancy for devices connected to the network. As long as N adjacent rows function, an $(N+S) \times (N+S)$ DRN can provide all generalized cube interconnections.

In the IADM, there is always at least two paths between each source and destination (assuming a source does not send to itself). However, there is no way of dynamically avoiding a faulty straight link. For this reason, McMillen and Siegel [49] proposed the enhanced IADM. They defined two forms for the enhanced IADM:

1. Redundant Straight Link—This makes the network 1-link-fault tolerant, but still leaves the problem of no node-fault tolerance.

2. Adding Half-Links—These are additional links added to each of stages 1 through $n-1$. They connect a switch m in stage i to switches $((m+2^{i-1}) \bmod N)$ and $((m-2^{i-1}) \bmod N)$ in stage $i+1$. Using half-links and a single-stage look-ahead technique to avoid faulty links and/or nodes, the enhanced IADM becomes dynamically 2-node-fault tolerant (which implies it is also

2-link-fault tolerant).

Other techniques to incorporate fault tolerance into MIN's include:

- increasing switch size and adding links (Augmented C-Network [50], Kappa Network [51]),
- replicating a MIN and forward cross-linking (Merged Delta Network [50]) or lateral cross-linking (Augmented Bidelta Network [52], H-Network [53]),
- replicating a MIN and adding an extra stage (INDRA [54]).

Adams, Agrawal, and Siegel [45] compare the various fault tolerant MIN topologies.

MIN comparisons

One factor used for comparing MIN's is the complexity of their implementation. This is termed the *network cost* and is usually based on an estimate of the layout area. For VLSI implementations, the area could be dominated by the devices or by the wiring. The equivalent number of 'crosspoints' in the network may be used to estimate the number of devices. To estimate area based on the wiring, the sum of the number of links between stages may be used. Another factor for comparing MIN's is the *network delay*, which may be estimated by the number of stages of switches from any input to any output. The cost and delay are usually given in order notation. Table 2.1 compares the cost in terms of crosspoints of several networks discussed above.

2.4 Switching Methodology

Switching methodology refers to the way in which information is routed through the network. It may be circuit switched, message switched, packet switched, or some hybrid form [56].

In *circuit switching*, a physical path is established between source and destination before transfer of data. This path is maintained for the duration of the 'call', guaranteeing a certain bandwidth to the communicators. If communication is intermittent, the unused bandwidth is wasted.

Table 2.1: Comparison of $N \times N$ network costs.

Network	Type	Switching element size	Number of crosspoints	Cost $O()$
Crossbar		$N \times N$	N^2	N^2
Ofman [26]	GCN	2×2	$20Nlog_2N - 2N$	$Nlog_2N$
Thompson [28]	GCN	3×3	$7.6Nlog_2N$	$Nlog_2N$
Thompson [28]	GCN	2×2	$8Nlog_2N - 6N$	$Nlog_2N$
Clos, 3-stage [10]	NB	$N^{1/2} \times (2N^{1/2} - 1)$	$6N^{3/2} - 3N$	$N^{3/2}$
Pippenger [55]	NB	5×5	$16Nlog_5^2N$	$Nlog_5^2N$
Beizer [29]	RE	2×2	$4Nlog_2N - 2N$	$Nlog_2N$
Beneš [11]	RE	$2 \times 2^*$	$4Nlog_2N - 4N$	$Nlog_2N$
Joel [30]	RE	2×2	$4Nlog_2N - 2N$	$Nlog_2N$
Pippenger [55]	RE	3×3	$6Nlog_3N$	$Nlog_3N$
Waksman [31]	RE	2×2	$4Nlog_2N - 4N + 4$	$Nlog_2N$
baseline [37]	BL	2×2	$2Nlog_2N$	$Nlog_2N$
gen. cube [34]	BL	2×2	$2Nlog_2N$	$Nlog_2N$

GCN = Generalized Connection Network
NB = Non-blocking
RE = Rearrangeable
BL = Blocking
*Beneš [11] has 4×4 middle stage.

In *message switching*, a physical path from source to destination is not established in advance. When a sender has data to send, it is sent into the network as a single (perhaps long) message. Each node in the network must store received messages, check for errors, and forward messages to the next appropriate network nodes. For this reason, this method is sometimes referred to as *store-and-forward.* The requirement to store the entire message, which could be quite long, usually necessitates using a disk drive at each node.

In *packet switching*, a physical path from source to destination is not established in advance. When a sender has data to send, it is broken into packets which have a predefined maximum size. Each node in the network stores received packets, checks for errors, and forwards the packet to the next appropriate node. Since each packet is a separate entity, any given packet in a multipacket message can be forwarded before the next packet has been fully received, thereby reducing delay and increasing throughput. Also, because packets have a known upper limit on size, the network's nodes can be designed to queue packets in local memory, removing the requirement for a disk drive as in message switching.

Virtual cut-through [57] has properties which resemble both circuit switching and packet switching. As in packet switching, no path is established in advance. The packet header is examined to determine the appropriate outgoing link. If the link is available, then forwarding of the packet begins immediately, before the entire packet has arrived. If the link is not available, the packet is stored and forwarded in the usual way. Thus, if the network is not congested, it is possible for the beginning of a packet to arrive at the destination while the tail of the packet has not left the source. In this case, the physical path from source to destination resembles that of the circuit switched network. This method precludes the use of error checking at the node level, since forwarding the packet begins before the entire packet has been received.

2.5 Centralized and Distributed Control Strategies

A *centralized control scheme* is one in which the switching elements are controlled by a common control unit. It has been used extensively in telephone crossbars

and SIMD computers [4]. The Beneš Network [11] is an example of a well known network which uses centralized control. Pease [40] uses a two-level centralized control scheme for the indirect binary n-cube microprocessor array. A circuit switching network requires a path to be established before data is sent, making centralized control a favorable approach.

A *distributed control scheme* is one in which the switching elements each make their own decisions regarding the path to use for sending data. This has been done in Multistage Interconnection Networks using routing tags which are computed by the source and placed in the header of the packet. Two methods using routing tags are the *exclusive-or routing-tag scheme* and the *destination-tag scheme.* In the exclusive-or routing-tag scheme, the routing tag T is the bitwise exclusive-or of the source and destination network addresses. An interchange box in stage i examines the i-th bit of T (t_i) and sets itself to straight if $t_i = 0$ or exchange if $t_i = 1$. In the destination-tag scheme, the destination address D is the tag. An interchange box in stage i examines the i-th bit of D (d_i) and uses the upper output if $d_i = 0$ or the lower output if $d_i = 1$ (Figure 2.15) [32].

2.6 Synchronous and Asynchronous Operation Modes

Under synchronous (or clocked) operation, all nodes in the network use a central global clock. It is a method by which all cells can know when to sample their inputs and when to hold their outputs constant.

Under asynchronous (or self-timed) operation, there is no global clock—all nodes operate independently, synchronizing their communication locally by some form of protocol.

Factors which generally favor clocked designs are their relative simplicity and lower hardware costs. However, as systems become larger, problems with clock skew become greater, which tends to make asynchronous design a better choice for large designs.

Wann and Franklin [58] investigated the pros and cons of both asynchronous and synchronous design for interconnection networks. For clocked schemes, the design of the control logic is simple while clock distribution is a problem (to avoid

clock skew). For asynchronous schemes, the design of the control logic is difficult while no clock distribution problems arise since there is no global clock. Wann and Franklin develop equations for determining path delay and clock skew. They present an example network and show graphically the trade-off between data rate and clock skew for asynchronous and synchronous implementations.

Fisher and Kung [59] describe a hybrid method of synchronization which employs both clocked and self-timed principles. The design is partitioned into smaller areas, each with a local clock distribution node. All clock distribution nodes use a handshaking protocol to synchronize to each other, and then distribute clock signals to their local domain. This is shown in Figure 2.17.

2.7 Summary

A general overview of interconnection networks has been given. Several types of static and dynamic network architectures were shown. Other considerations available to the network designer include method of switching data (packet, circuit, or other), centralized or distributed control, and synchronous or asynchronous operation.

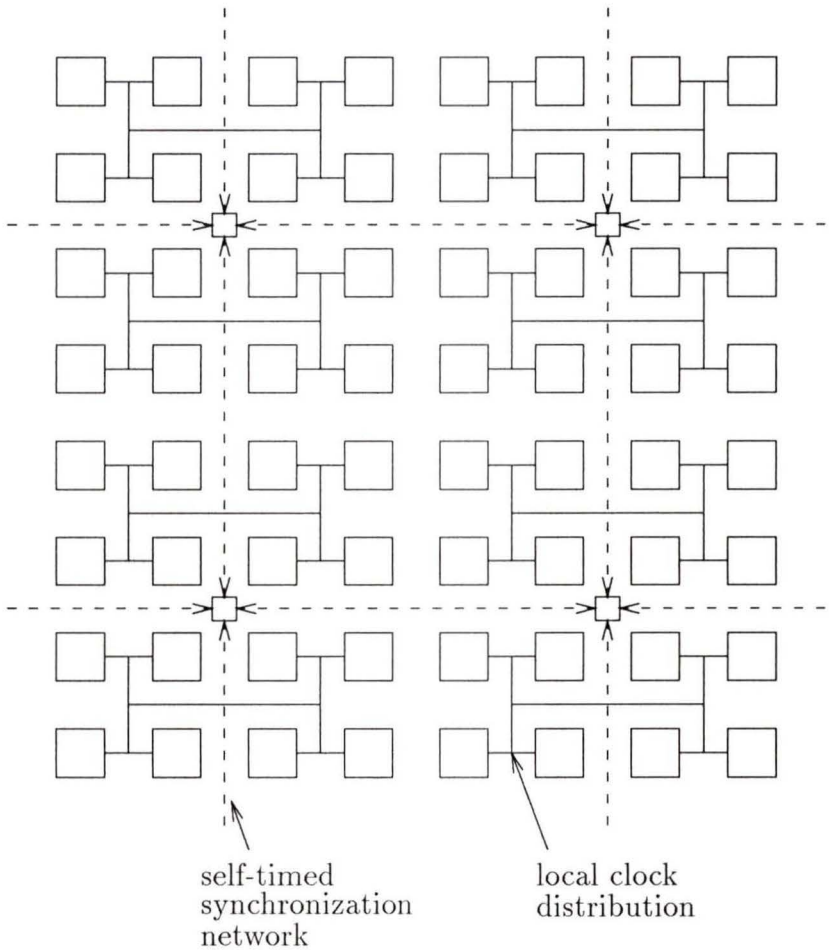


Figure 2.17: Hybrid synchronization scheme.

Chapter 3

Switching Element Architecture

3.1 Introduction

In this chapter, architectural options for a buffered switching element (SE) will be identified. Analysis and simulations performed in a subsequent chapter will show the merits in these various design options.

3.2 Selection of Network Topology

In general, the static network topologies (Section 2.2) are not amenable to both VLSI design and an expandable switch fabric. Thus, only dynamic network topologies (Section 2.3) are considered here. For initial considerations, it is assumed that a VLSI chip implementation will be pin-limited to less than four input ports by four output ports. This assumption stems from the desire to have datapaths which are several bits wide in order to increase the bandwidth of the SE.

The crossbar network is infeasible due to its high system complexity. Assuming 2×2 SE's, an $N \times N$ crossbar would require $N/2 \times N/2 = N^2/4$ SE's. A single stage shuffle-exchange network would require only $N/2$ SE's, however, trunk controllers (TC's) would be required to receive and forward messages not destined for them, thereby infringing on the total available TC bandwidth. Multistage Interconnection Networks (MIN's) have lower complexity than the crossbar, while allowing the delivery of a message to its intended destination in one pass through the network.

In telecommunications network design, some degree of blocking is deemed an acceptable tradeoff against the higher complexity of non-blocking and rearrangeable MIN's [60]. Furthermore, blocking networks allow the use of a distributed control strategy (i.e., routing tags) and a virtual cut-through switching methodology.

Several authors have proposed the use of MIN's for a BISDN switch fabric [2], [61]–[66]. These MIN's are typically of the banyan [36] or delta [41] class. An $N \times N$ delta network consists of $a \times a$ SE's arranged in $\log_a N$ stages of N/a identical SE's per stage. Typically, a is 2, 3, or 4 to keep chip input and output requirements within acceptable ranges.

Delta networks are referred to as *self-routing*. A packet to be sent through the MIN is prefixed by a routing tag. At each stage of the MIN, a single bit of this routing tag determines which of the two SE outputs the packet is to be routed to. Once the routing decision has been made, the routing tag is rotated to bring the next bit of the tag into position for the next stage.

Delta networks are internally blocking. This means that a packet may require the use of a link which is already in use for another connection. Several approaches have been taken to avoid this problem, or at least reduce its effects (see Figure 3.1):

1. By presenting the MIN with requests for unique destinations, and having these requests arranged in increasing (or decreasing) order of destination address, the MIN will not block [67], [68]. In this process, arbitration is performed on all requesting packets to select a set of packets requesting unique destinations. The successful requests are ordered and sent through the MIN. The unsuccessful requests must be held in an input queue until the next arbitration phase.

2. Buffers may be added to each of the SE's within the MIN [61], [62]. Packets are buffered until the desired SE output is available. Thus, packets can advance by a stage at a time, rather than trying to complete a connection across the entire MIN.

3. The routing MIN may be preceded by a distribution network which serves to distribute the packets more evenly across the inputs of the routing MIN [64]–[66]. This has the effect of distributing the load more evenly on the internal

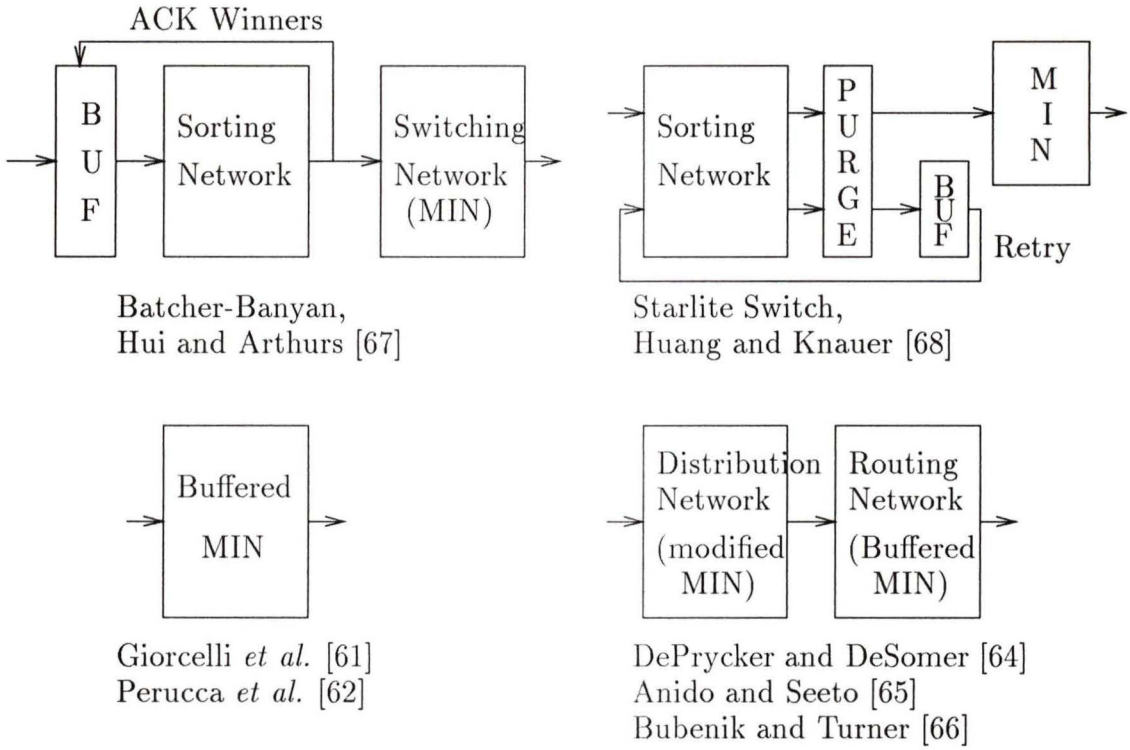


Figure 3.1: Examples of switching fabrics found in the literature.

links of the routing MIN. The distribution network is also implemented as a self-routing MIN. The routing MIN may also be buffered.

It is interesting to note that, although the major objective is to improve MIN bandwidth by the addition of buffers, the architecture used for the buffered SE's within the MIN is typically the poorest for throughput and delay performance. This is generally due to architectural simplicity being the main design criterion.

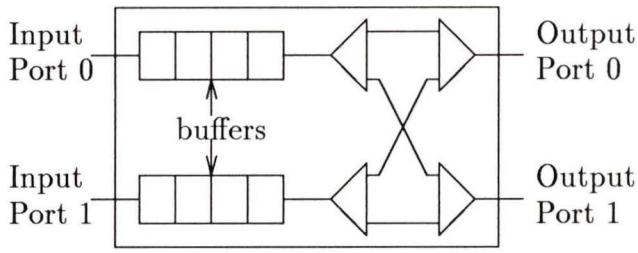


Figure 3.2: Switching element SE1 (buffers before the switching multiplexors).

3.3 Architecture Options

3.3.1 Switch size and buffer size

Switch size refers to the number of inputs and outputs that an SE has. An $a^n \times b^n$ delta network consists of n stages of $a \times b$ SE's. For rectangular networks (i.e., number of inputs equal to number of outputs), $a = b$. Typically, switch size ranges from 2×2 to 4×4.

Buffers may be added to an SE to allow packets to be stored until the desired output is available. Buffer size refers to the number of *slots* in a buffer of an SE. A slot is essentially a block of memory large enough to hold a single packet. Since the buffers can represent a large portion of the hardware layout area, networks with similar total buffer size are compared.

Figure 3.2 shows a 2×2 SE with size 4 buffers.

3.3.2 Buffer placement

Buffers may be added to the SE before the switching multiplexors, as shown in Figure 3.2. This type of SE will be referred to as SE1.

In buffered delta networks, several types of internal blocking can be identified. One is *switch blocking*, which is also present in the unbuffered network. This occurs where more than one input to a multiplexer requires the multiplexer's output. One input will be granted the use of the output while all others will be blocked. In the unbuffered network, packets at the blocked inputs are lost. The buffered network

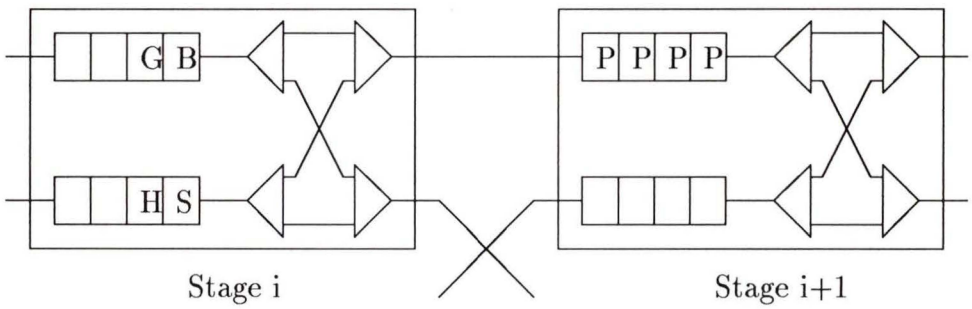


Figure 3.3: Blocking for networks based on SE1 (see text for explanation).

prevents the loss of the packet by buffering. However, it is the buffers which lead to a second form of blocking, known as *buffer blocking*. A path through the switching multiplexers may be available (i.e., no switch blocking), but the destination buffer in the successor SE is full, preventing the packet from advancing.

Due to buffer blocking, a third type of blocking may occur, known as *Head of the Line* (HOL) blocking. Consider the case where a packet at the front of a buffer (i.e., the *head of the line* packet) is blocked due to insufficient buffer space in the next stage (buffer blocked). The successor packet to this blocked HOL packet is prevented from proceeding to the next stage, even though there may be space available for it in the next stage. Thus, the term HOL blocking.

Figure 3.3 identifies the three types of blocking between two SE's with buffers before the switching multiplexors. The buffer in stage $i+1$ is full. Assume packets B and S in stage i require the upper output of the switch and that packet B wins the contention. Then packet B is buffer blocked while packet S is switch blocked. The packets G and H are HOL blocked.

HOL blocking can be reduced by buffering packets according to the desired output of the current SE. In this case, it does not seem appropriate to buffer packets before the internal switching multiplexors of the SE, since reducing HOL blocking requires examining the routing tag. Instead, the tag is examined and the packet is then buffered in a buffer corresponding to the desired output of the current SE. Arbitration is performed for all such buffers for a given output port. Therefore, the buffers are placed between the internal input and output multiplexors of the

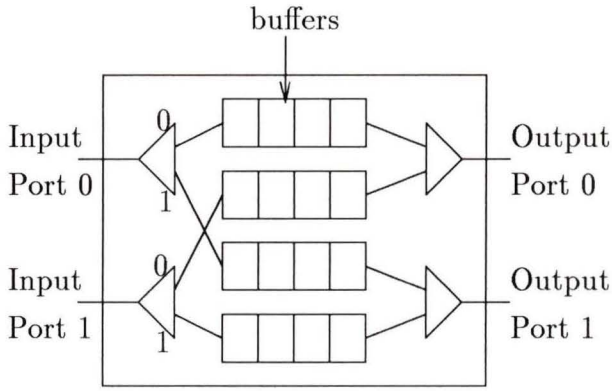


Figure 3.4: Buffer and link arrangement of switching element SE2A. The numbers in the input multiplexors indicate routing information of the packet header.

SE as shown in Figure 3.4. This SE will be referred to as SE2A.

Figure 3.5 shows the three types of blocking between two SE's with buffers between the switching multiplexors. The uppermost buffer in stage $i+1$ is full. If packets B and S both request the full buffer, then one will be buffer blocked, while the other will be switch blocked. On the other hand, if one of these packets requests the non-full buffer (where packet A is located), then it will be able to proceed. Packet H is HOL blocked if packet B is blocked in any way.

It is possible to extend this technique further. While examining the tag to determine routing within the current SE, a look-ahead examination can be performed in parallel to determine which output will be required in the successor SE. Packets can then be buffered in independent buffers based on the output port required on the current SE and the successor SE. This routing information is shown by the numbers at the input multiplexors in Figure 3.6. This SE will be referred to as SE2B.

Figure 3.7 shows the three types of blocking between two SE's which use this scheme. As before, assume packet B is destined for the full buffer (P's) in stage $i+1$ and is, therefore, buffer blocked. Packet C is destined for either the full buffer or the buffer with packet A and, therefore, may be able to proceed. Packets D and E require the buffers where packets J and K are located and can, therefore, proceed.

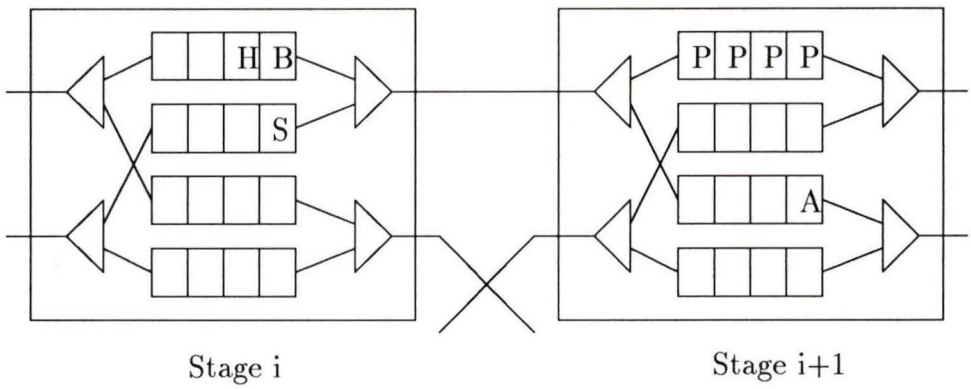


Figure 3.5: Blocking for networks based on SE2A (see text for explanation).

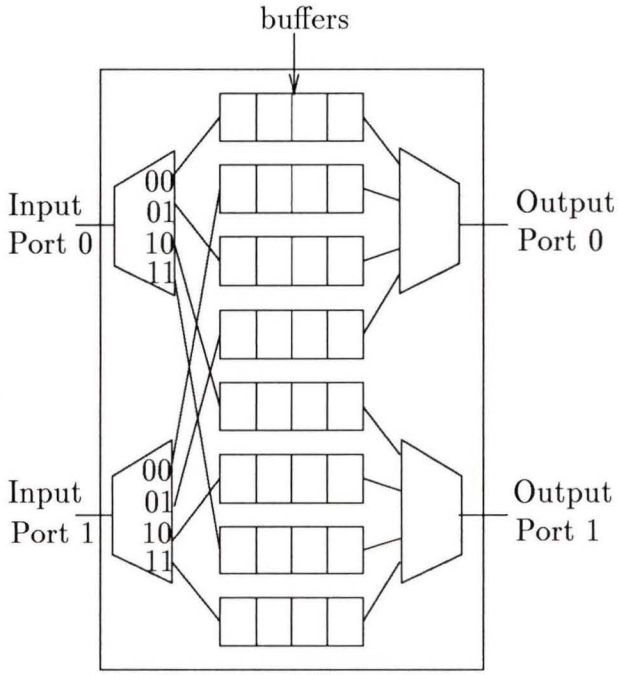


Figure 3.6: Buffer and link arrangement of switching element SE2B. The numbers in the input multiplexors indicate routing information of the packet header.

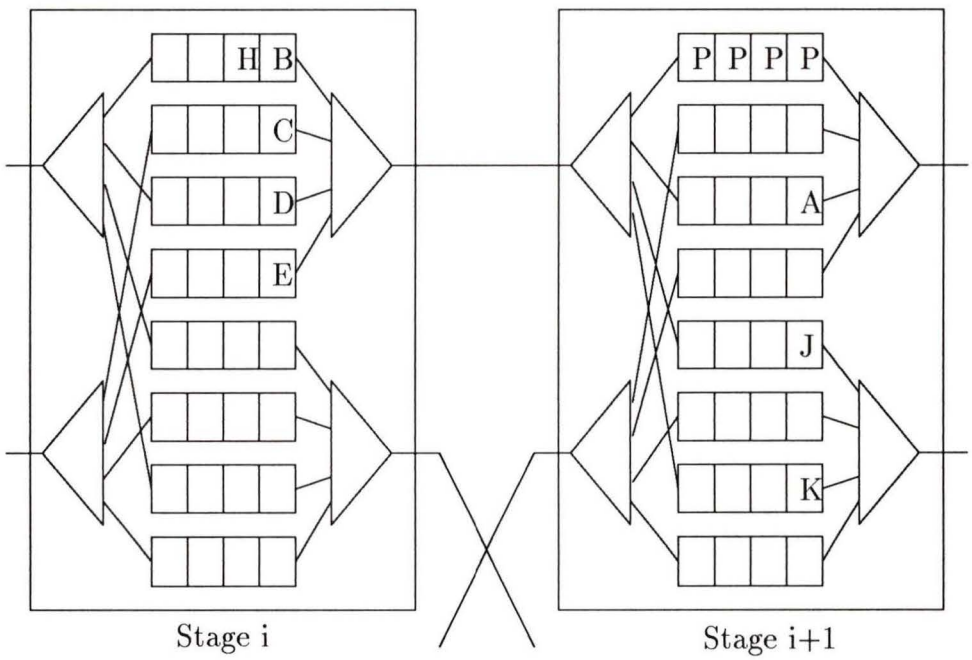


Figure 3.7: Blocking for networks based on SE2B (see text for explanation).

As mentioned above, in SE2B, two bits of the routing tag are examined (rather than one bit as is customary for SE2A). Let the stages of an $N \times N$ delta network be numbered from 0 to $(n-1)$, where $n = \log_2 N$. Let the routing tag be represented by n bits, $b_0 b_1 \ldots b_{n-1}$. In stage i of the network, $0 \leq i < n-1$, bit b_i determines whether the upper output port ($b_i = 0$) or lower output port ($b_i = 1$) of the SE should be used. Bit b_{i+1} distinguishes packets which require the upper output port on the successor SE from those which require the lower output port on the successor SE. There are four buffers associated with each input port for saving these four types of packets. It should be noted that, in the case of a 2-dilated network, the two parallel links constituting an input port cannot send a packet to the same buffer simultaneously. Also, for $i = n-1$ (i.e., the last stage), only the one remaining bit of the tag is used, since no successor SE exists.

By comparing Figures 3.3, 3.5, and 3.7, it is evident that SE2A reduces HOL blocking by 50% with respect to SE1, while SE2B reduces it a further 50%, for a total reduction of 75%.

3.3.3 Dilation

A network in which each link is replaced by d parallel links is said to be d-dilated [69]. This has the effect of increasing the available bandwidth between SE's. However, the SE's ability to use the additional bandwidth is dependent on its internal architecture. A d-dilated network is $(d-1)$-fault tolerant to link failures (but not switch failures). Figure 3.8 shows a 2-dilated delta network.

3.3.4 Buffer selection

In SE1, the HOL packets may be destined for the same output port. Similarly, in SE2A and SE2B, a group of buffers have packets destined for the same output port. Thus, the SE's require an arbitration scheme to select one of the packets to be forwarded. Kumar and Jump [70] simulated different schemes for arbitrating among contending buffers in non-dilated networks composed of 2×2 SE's (types SE1 and SE2A). The different arbitration schemes considered were:

1. *random selection* - One of the packets contending for an output port is selected at random.

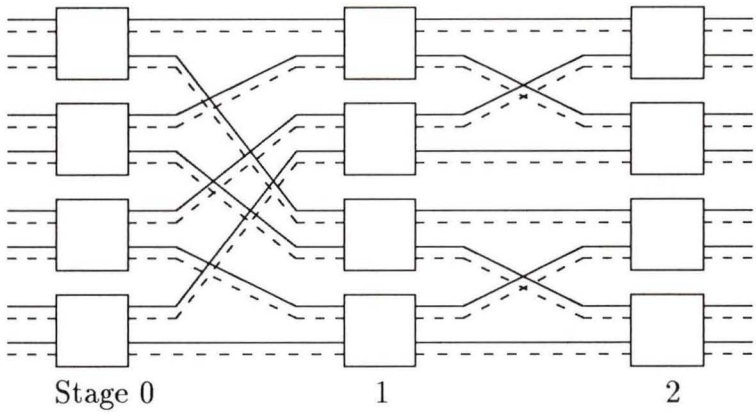


Figure 3.8: An 8×8 delta network. Dashed lines are links added to make a 2-dilated network, as explained in the text.

2. *rotating priority* - All buffers have a permanent cyclic order. Buffers are considered in this order, beginning with the highest-priority buffer. The buffer next to the highest-priority buffer becomes the highest-priority buffer for the next arbitration.

3. *selection based on buffer occupancy* - Buffers are considered to have a priority equal to the number of packets occupying them. A packet is offered from the highest priority buffer. In the event of a tie, one of the buffers is chosen at random.

Kumar and Jump found negligible difference in performance among the different arbitration schemes, so the easiest method to implement was recommended. Simulations or analysis would be required to determine if this holds true for SE2B.

For d-dilated networks, the particular arbitration scheme could be applied up to d times (once for each available link) during any given stage cycle.

3.4 Summary

In this chapter, it was shown that several authors have used rectangular delta networks for BISDN switching fabrics. Such networks have SE's with input, output, and area requirements suitable for VLSI implementation.

Architectural options for the design of an SE were presented. The options include switch size, buffer size, buffer placement, dilation, and buffer selection.

Larger switch size reduces the number of stages in the network, thereby reducing delay through the network. However, larger switch size translates into higher pincounts for the individual SE's.

Larger buffer size improves throughput at the expense of longer delays through the switch. It also increases the chip size in a VLSI implementation.

Buffers placed between the switching multiplexors of the SE give better throughput and delay performance than buffers placed before the switching multiplexors. However, the control logic is more complicated for buffers between the switching multiplexors, which may explain why such an approach is not typically used.

Dilation can provide better throughput and delay performance at the expense of higher pincounts on the individual SE's.

Buffer selection may affect the overall delay of packets through the network.

The above parameters affect network performance.

Chapter 4

Delta Network Analysis and Simulation

4.1 Introduction

Patel [41] introduced unbuffered delta networks and analyzed their performance using a probabilistic model. Since then, several papers have been written which describe analyses for the delta network. These papers generally considered networks composed of 2×2 SE's, and were for unbuffered networks [69], [71], [72] or buffered networks with buffers before the crossbar (or switching multiplexors) [72]–[79]. Yoon *et al.* [80] extended the analysis to $a \times a$ SE's with multiple buffers before the crossbar.

In this chapter, buffered delta networks consisting of $a \times b$ SE's will be analyzed using a probabalistic model. A generalized set of equations will be developed for analyzing networks with buffers before the crossbar and buffers inside the crossbar (between the switching multiplexors of the SE).

4.2 Analysis

Two sets of equations will be presented. The first set is for networks based on SE's with buffers before the crossbar and is actually a generalization of [80] (generalized to consider non-rectangular delta networks). The second set of equations is for networks based on SE's with buffers inside the crossbar and represents new work

in the analysis of such networks.

Figure 4.1 shows the conceptual view of the network used in this analysis. The analysis employs the following set of assumptions. These assumptions are consistent with those made in [41], [78], [80].

1. Packets arrive at each source node with equal probability, i.e., all network inputs have the same packet arrival rate.

2. All packets have a fixed size.

3. Packets are directed uniformly over all network outputs.

4. Packets arriving at an SE are buffered. Each buffer operates in a FIFO fashion. A buffer accepts packets from a single input and produces packets at a single output. The packet at the output of the buffer is called the *Head-of-Line* (HOL) packet.

5. When a routing conflict occurs at an SE, a packet is selected at random to move forward.

6. The network operates synchronously at a rate of τ, called the *stage cycle.* The stage cycle consists of two phases.

   (a) τ_1 (arbitration phase): During this phase, HOL packets in an SE compete for a given output port. Then, based on availability of buffer space in the successor SE, the selected packet is notified as to whether or not it may move forward.

   (b) τ_2 (transfer phase): Packets selected to move forward during τ_1 do so.

7. The arbitration phase may be based on either a global communication between all SE's of the network, or on local communication only between SE's in adjacent stages. For the case of global communication, a packet selected to move forward may do so if the destination SE has room for the packet, or if it will have room at the end of the current stage cycle (by forwarding one of its packets). For the case of local communication only, a packet selected to move forward may do so only if the destination SE *currently* has room for the packet. In the latter case, a successor SE with a full buffer will not know

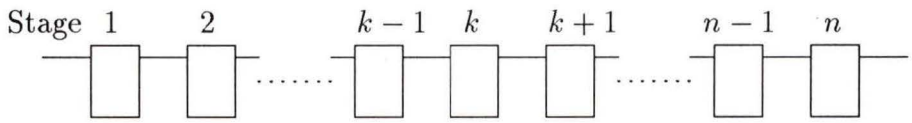


Figure 4.1: Conceptual view of the network used in the analysis.

until the end of the arbitration phase as to whether or not it will be able to forward one of its packets.

For buffered networks, the performance measures of interest are throughput and delay. The *throughput* of a network is the average number of packets passed by the network per stage cycle. The *delay* is the average number of stage cycles taken for a packet to pass through a network.

The following variables will be used in this analysis. For brevity, $\overline{z}$ will be used to represent $(1 - z)$ for any variable z. For example, $\overline{p}_0(k,t) = [1 - p_0(k,t)]$.

n = number of switching stages in the network.

a = number of inputs to an SE.

b = number of outputs from an SE.

β = number of slots in a buffer. A slot can hold one packet.

Q_{IN} = offered load = input traffic intensity for each network input, relative to the maximum rate of one packet per stage cycle per network input.

$p_j(k,t)$ = probability that there is exactly j packets in a buffer of an SE at stage k at the beginning of the t-th stage cycle.

$p_{ij}(k,t)$ = transition probability from the state of having i packets in the buffer to j packets in the buffer of an SE at stage k at the end of the t-th stage cycle.

$q(k,t)$ = probability that a packet is ready to come to a given buffer of an SE at stage k during the t-th stage cycle.

$v(k,t)$ = probability that a given buffer of an SE at stage k will be able to accept a packet during the t-th stage cycle.

$r(k,t)$ = probability that a packet in a buffer of an SE at stage k is able to move forward during the t-th stage cycle, given that there is a packet in that buffer.

$r'(k,t)$ = probability that a packet in a buffer of an SE at stage k is able to pass that SE to the desired output port during the t-th stage cycle by winning the contention among competing packets, given that there is a packet in that buffer.

$s(k,t) = \overline{p}_0(k,t)r(k,t)$ = throughput of a buffer of an SE at stage k during the t-th stage cycle.

In steady state, $p_j(k,t)$, $q(k,t)$, $r(k,t)$, and $s(k,t)$ converge to $p_j(k)$, $q(k)$, $r(k)$, and $s(k)$, respectively. Additionally, at steady state, the following variables are defined.

$\lambda(k)$ = packets input per stage cycle per port of an SE at stage k.

$\mu(k)$ = packets output per stage cycle per port of an SE at stage k.

$B(k)$ = average number of packets in a buffer in stage k.

S_N = network throughput = ratio of network aggregate output traffic to network aggregate input traffic.

$d(k)$ = average delay experienced by a packet passing through stage k.

D = normalized delay = average delay per stage for the network.

4.2.1 Buffers before crossbar

In this section, equations are developed for analyzing the SE with buffers before the crossbar. Figure 4.2 shows the model for this SE.

There are b^j ways that j buffered *Head of the Line* (HOL) packets in an SE can map to the b outputs of the SE. Now suppose that a certain output is not requested. Then j packets request the $(b-1)$ remaining outputs in $(b-1)^j$ ways

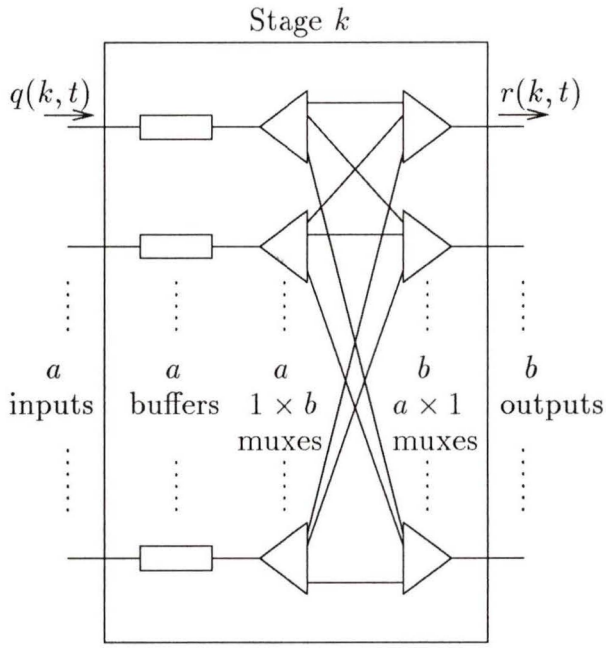


Figure 4.2: Model of SE with buffers before the crossbar.

($b \geq 2$). Thus, the number of ways the particular output is requested is $b^j - (b-1)^j$. Therefore, the probability that a particular output is requested is

$$\frac{b^j - (b-1)^j}{b^j} = 1 - (\frac{b-1}{b})^j = 1 - (1 - \frac{1}{b})^j, \quad b \geq 2. \tag{4.1}$$

Each input to an $a \times b$ SE goes directly to a buffer. Thus, there are a buffers. The probability that an $a \times b$ SE at stage $k-1$ has a total of j HOL packets in its a buffers at the beginning of the t-th stage cycle is given by

$$\binom{a}{j} \overline{p}_0(k-1,t)^j p_0(k-1,t)^{a-j}, \quad a \geq 1. \tag{4.2}$$

The probability that a packet is ready to come to an SE in stage k is the sum over all possible numbers of buffered HOL packets of the product of probabilities (4.1) and (4.2).

$$\begin{aligned}
q(k,t) &= \sum_{j=0}^{a} \binom{a}{j} \overline{p}_0(k-1,t)^j p_0(k-1,t)^{a-j}[1 - (1 - \frac{1}{b})^j] \qquad (4.3)\\
&= \sum_{j=0}^{a} \binom{a}{j} \overline{p}_0(k-1,t)^j p_0(k-1,t)^{a-j} - \\
&\quad \sum_{j=0}^{a} \binom{a}{j} \overline{p}_0(k-1,t)^j p_0(k-1,t)^{a-j}(1 - \frac{1}{b})^j \\
&= 1 - p_0(k-1,t)^a \sum_{j=0}^{a} \binom{a}{j} \left(\frac{\overline{p}_0(k-1,t)}{p_0(k-1,t)}\right)^j \left(1 - \frac{1}{b}\right)^j \\
&= 1 - p_0(k-1,t)^a \sum_{j=0}^{a} \binom{a}{j} \left[\frac{\overline{p}_0(k-1,t)}{p_0(k-1,t)} \left(1 - \frac{1}{b}\right)\right]^j \\
&= 1 - \left[1 - \frac{\overline{p}_0(k-1,t)}{b}\right]^a, \quad \begin{cases} a \geq 1 \\ b \geq 2 \\ 2 \leq k \leq n+1 \end{cases} \qquad (4.4)
\end{aligned}$$

where use has been made of the fact that $\sum_{j=0}^{a} \binom{a}{j} x^j = (1+x)^a$ (Binomial Series). At the input of the network, $q(1,t)$ is given by Q_{IN}, the input traffic intensity.

Special consideration must be given to the case $b = 1$, since the above analysis imposed the restriction $b \geq 2$ to avoid 0^0. For $b = 1$, the probability that the output is requested is 0 when $j = 0$ and 1 otherwise. Thus, (4.3) becomes

$$\begin{aligned} q(k,t) &= \sum_{j=1}^{a} \binom{a}{j} \overline{p}_0(k-1,t)^j p_0(k-1,t)^{a-j} \\ &= \sum_{j=0}^{a} \binom{a}{j} \overline{p}_0(k-1,t)^j p_0(k-1,t)^{a-j} - \\ &\quad \binom{a}{0} \overline{p}_0(k-1,t)^0 p_0(k-1,t)^{a-0} \\ &= 1 - p_0(k-1,t)^a \ . \end{aligned} \tag{4.5}$$

Substituting $b = 1$ into (4.4) yields the same result as (4.5). Thus, (4.4) holds for $b \geq 1$ and can be given as in (4.6).

$$q(k,t) = 1 - \left[1 - \frac{\overline{p}_0(k-1,t)}{b}\right]^a, \quad \begin{cases} a \geq 1 \\ b \geq 1 \\ 2 \leq k \leq n+1 \end{cases} \tag{4.6}$$

Next, consider $r(k,t)$, the probability that a packet in a buffer of an SE at stage k is able to move forward during the t-th stage cycle, given that there is a packet in that buffer. This may be viewed as the product of the probability that the packet is selected to move forward at the given SE ($r'(k,t)$) with the probability that it will be accepted by the successor SE ($v(k,t)$), for stages $1 \leq k \leq n$.

$$r(k,t) = r'(k,t)v(k+1,t) \tag{4.7}$$

$$v(k,t) = \overline{p}_\beta(k,t) + f_0 p_\beta(k,t) r(k,t) \tag{4.8}$$

$$v(n+1,t) \doteq 1 \tag{4.9}$$

where f_0 indicates the type of communication during the arbitration phase: $f_0 = 0$ for local, $f_0 = 1$ for global.

The probability $r'(k,t)$ is based on the following:

- The probability of the given packet selecting a particular output is $1/b$.

- The probability $a-1$ remaining buffers contain j HOL packets is

$$\binom{a-1}{j} \overline{p}_0(k,t)^j p_0(k,t)^{a-1-j} .$$

- Of the j HOL packets, the probability that i of them are contending for the same output as the given packet is

$$\binom{j}{i} \left(\frac{1}{b}\right)^i \left(1-\frac{1}{b}\right)^{j-i} .$$

- The probability the packet under consideration is chosen is $\left(\frac{1}{i+1}\right)$.

- The number of ways of choosing one of b output ports is $\binom{b}{1}$.

Therefore, $r'(k,t)$ is given by

$$\begin{aligned} r'(k,t) &= \frac{1}{b} \times \sum_{j=0}^{a-1} \binom{a-1}{j} \overline{p}_0(k,t)^j p_0(k,t)^{a-1-j} \times \\ &\quad \sum_{i=0}^{j} \binom{j}{i} \left(\frac{1}{b}\right)^i \left(1-\frac{1}{b}\right)^{j-i} \times \frac{1}{i+1} \times \binom{b}{1} \\ &= \sum_{j=0}^{a-1} \binom{a-1}{j} \overline{p}_0(k,t)^j p_0(k,t)^{a-1-j} \times \\ &\quad \sum_{i=0}^{j} \binom{j}{i} \left(\frac{1}{b}\right)^i \left(1-\frac{1}{b}\right)^{j-i} \frac{1}{i+1} . \end{aligned}$$

Since $\binom{j}{i} \frac{1}{i+1} = \frac{1}{j+1} \binom{j+1}{i+1}$ and $\binom{a-1}{j} \frac{1}{j+1} = \frac{1}{a} \binom{a}{j+1}$, the above equation may be rewritten as

$$\begin{aligned} r'(k,t) &= \frac{1}{a} \sum_{j=0}^{a-1} \binom{a}{j+1} \overline{p}_0(k,t)^j p_0(k,t)^{a-1-j} \times \\ &\quad \sum_{i=0}^{j} \binom{j+1}{i+1} \left(\frac{1}{b}\right)^i \left(1-\frac{1}{b}\right)^{j-i} \end{aligned} \tag{4.10}$$

$$= \frac{1}{a}\sum_{j=0}^{a-1}\binom{a}{j+1}\overline{p}_0(k,t)^j p_0(k,t)^{a-1-j} \times \tag{4.11}$$

$$\left(\frac{b-1}{b}\right)^j \left[\sum_{i=0}^{j}\binom{j+1}{i+1}\left(\frac{1}{b-1}\right)^i\right]$$

$$= \frac{1}{a}\sum_{j=0}^{a-1}\binom{a}{j+1}\overline{p}_0(k,t)^j p_0(k,t)^{a-1-j} \times \tag{4.12}$$

$$\left(\frac{b-1}{b}\right)^j (b-1)\left[\sum_{i=0}^{j+1}\binom{j+1}{i}\left(\frac{1}{b-1}\right)^i - 1\right]$$

$$= \frac{1}{a}\sum_{j=0}^{a-1}\binom{a}{j+1}\overline{p}_0(k,t)^j p_0(k,t)^{a-1-j} \times \tag{4.13}$$

$$\left(\frac{b-1}{b}\right)^j (b-1)\left[\left(1+\frac{1}{b-1}\right)^{j+1} - 1\right]$$

$$= \frac{b}{a}\sum_{j=0}^{a-1}\binom{a}{j+1}\overline{p}_0(k,t)^j p_0(k,t)^{a-1-j}\left[1-\left(1-\frac{1}{b}\right)^{j+1}\right], \tag{4.14}$$

where use has been made of the fact that $\sum_{j=0}^{j+1}\binom{j+1}{i}x^j = (1+x)^{j+1}$ (Binomial Series).

After further manipulation, (4.14) yields

$$r'(k,t) = \frac{b}{a\overline{p}_0(k,t)}\left[1-\left(1-\frac{\overline{p}_0(k,t)}{b}\right)^a\right] \tag{4.15}$$

$$= \frac{b}{a}\frac{q(k+1,t)}{\overline{p}_0(k,t)}, \quad \begin{cases} a \geq 1 \\ b \geq 1 \\ 1 \leq k \leq n \end{cases} \tag{4.16}$$

Note that $r'(k,t) = 1$ as $\overline{p}_0(k,t) \to 0^+$.

Substituting (4.16) into (4.7) yields (4.17).

$$r(k,t) = \frac{b}{a}\frac{q(k+1,t)}{\overline{p}_0(k,t)}v(k+1,t), \quad a \geq 1,\ b \geq 1,\ 1 \leq k \leq n. \tag{4.17}$$

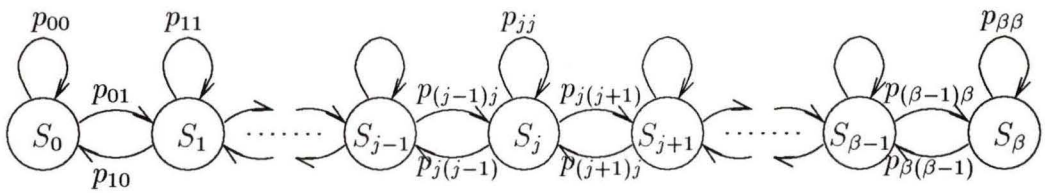


Figure 4.3: State transition diagram for an SE buffer.

Next, consider the state transition probabilities. Figure 4.3 shows the state transition diagram for an SE's buffer [80]. State S_j represents the buffer state of having j packets. The transition probability p_{ij} represents the probability of moving from state S_i to state S_j. Since a buffer has one input and one output, only state changes to adjacent states or back to the same state are possible during a stage cycle.

The state transition probabilities may be defined as follows ($1 \leq k \leq n$):

$$p_{00}(k,t) = \overline{q}(k,t) \tag{4.18}$$

$$p_{01}(k,t) = q(k,t) \tag{4.19}$$

$$p_{jj}(k,t) = \overline{q}(k,t)\overline{r}(k,t) + q(k,t)r(k,t), \quad 1 \leq j \leq \beta - 1 \tag{4.20}$$

$$p_{(j-1)j}(k,t) = q(k,t)\overline{r}(k,t), \quad 2 \leq j \leq \beta \tag{4.21}$$

$$p_{(j+1)j}(k,t) = \overline{q}(k,t)r(k,t), \quad 0 \leq j \leq \beta - 2 \tag{4.22}$$

$$p_{\beta(\beta-1)}(k,t) = (1 - f_0 q(k,t))r(k,t) \tag{4.23}$$

$$p_{\beta\beta}(k,t) = \overline{r}(k,t) + f_0 q(k,t)r(k,t) \tag{4.24}$$

where $f_0 = 0$ for local communication, $f_0 = 1$ for global communication. For $\beta = 1$, only $p_{00}(k,t)$, $p_{01}(k,t)$, $p_{(j+1)j}(k,t)$, and $p_{\beta\beta}(k,t)$ apply.

The state probabilities are given by:

$$p_0(k,t+1) = p_0(k,t)p_{00}(k,t) + p_1(k,t)p_{10}(k,t) \tag{4.25}$$

$$p_j(k,t+1) = p_{j-1}(k,t)p_{(j-1)j}(k,t) + p_j(k,t)p_{jj}(k,t) +$$

$$p_{j+1}(k,t)p_{(j+1)j}(k,t), \quad 1 \leq j \leq \beta - 1 \tag{4.26}$$

$$p_\beta(k,t+1) = p_{\beta-1}(k,t)p_{(\beta-1)\beta}(k,t) + p_\beta(k,t)p_{\beta\beta}(k,t) \tag{4.27}$$

For $\beta = 1$, only $p_0(k,t+1)$ and $p_\beta(k,t+1)$ apply.

By setting $b = a$ and $f_0 = 1$, the above results agree with those in [80] for rectangular delta networks composed of $a \times a$ SE's.

The above equations may be solved iteratively to reach steady-state values. At steady-state, the following results may be determined.

$$\lambda(k) = q(k)v(k) \tag{4.28}$$

$$s(k) = \overline{p}_0(k)r(k) \tag{4.29}$$

$$\mu(k) = \frac{a}{b}s(k) \tag{4.30}$$

$$\begin{aligned} S_N &= \frac{(\text{number of network outputs}) \times \mu(n)}{(\text{number of network inputs}) \times Q_{IN}} \\ &= \frac{b^n \mu(n)}{a^n Q_{IN}} \end{aligned} \tag{4.31}$$

At any stage k, the average number of packets in a buffer is

$$B(k) = \sum_{j=1}^{\beta} j p_j(k) \, . \tag{4.32}$$

The delay experienced by a packet as it moves through a buffer is $1/s(k)$ per slot. Therefore, the average delay experienced by a packet passing through a buffer in stage k is

$$d(k) = \frac{B(k)}{s(k)} \, . \tag{4.33}$$

The average delay per stage for the network is

$$D = \frac{1}{n}\sum_{k=1}^{n} d(k) \, . \tag{4.34}$$

The equations presented in this section will subsequently be summarized in Section 4.2.3.

4.2.2 Buffers inside crossbar

For SE's with buffers inside the switching multiplexors, it is possible to use a "lookahead" routing scheme. Normally, in a non-lookahead scheme, packets are placed into a buffer corresponding to a particular output of the SE. With m-stage lookahead, routing tag bits which will be used at m subsequent stages are previewed at the SE. Packets are then grouped into b^m buffers according to the path required through the m subsequent stages. With this scheme, packets which are blocked at later stages do not prevent other packets in the SE from proceeding.

The following variables are now defined, in addition to those previously defined.

$q'(k,t)$ = probability that a packet is ready to come to an input port of an SE at stage k during the t-th stage cycle.

m = number of stages of "lookahead" routing performed by the SE, with $m = 0$ meaning no lookahead routing.

Figure 4.4 shows the context of the labels used in this analysis. To aid with the visualization of this model, a *buffer group* is defined as a group of buffers in an SE which are connected to the same output multiplexor of that SE. Therefore, there are b buffer groups. Figure 4.5 shows the detail for a buffer group. Within a buffer group, there are a buffer blocks. A *buffer block* is defined as a set of buffers in a buffer group which are connected to the same input multiplexor of the SE. The number of buffers in a buffer block is dependent on the number of stages of lookahead, m. For one stage of lookahead, a buffer block requires b buffers. For two stages of lookahead, each buffer block requires $b \times b$ buffers. In general, for m stages of lookahead, each buffer block requires b^m buffers. Therefore, a buffer group has ab^m buffers, giving a total of ab^{m+1} for the SE. Input multiplexors are $1 \times b^{m+1}$ and output multiplexors are $ab^m \times 1$. The number of routing tag bits used to route at an SE will be $(m+1)\log_2 b$.

A packet is ready to come to an input port of an SE at stage k any time there is at least one packet in any of the associated ab^m buffers in stage $(k-1)$.

$$\begin{aligned} q'(k,t) &= 1 - \text{Prob[all } ab^m \text{ buffers of } (k-1)\text{-th stage empty]} \\ &= 1 - p_0(k-1,t)^{ab^m} \qquad (4.35) \\ q'(1,t) &= Q_{IN} = \text{input traffic intensity} \end{aligned}$$

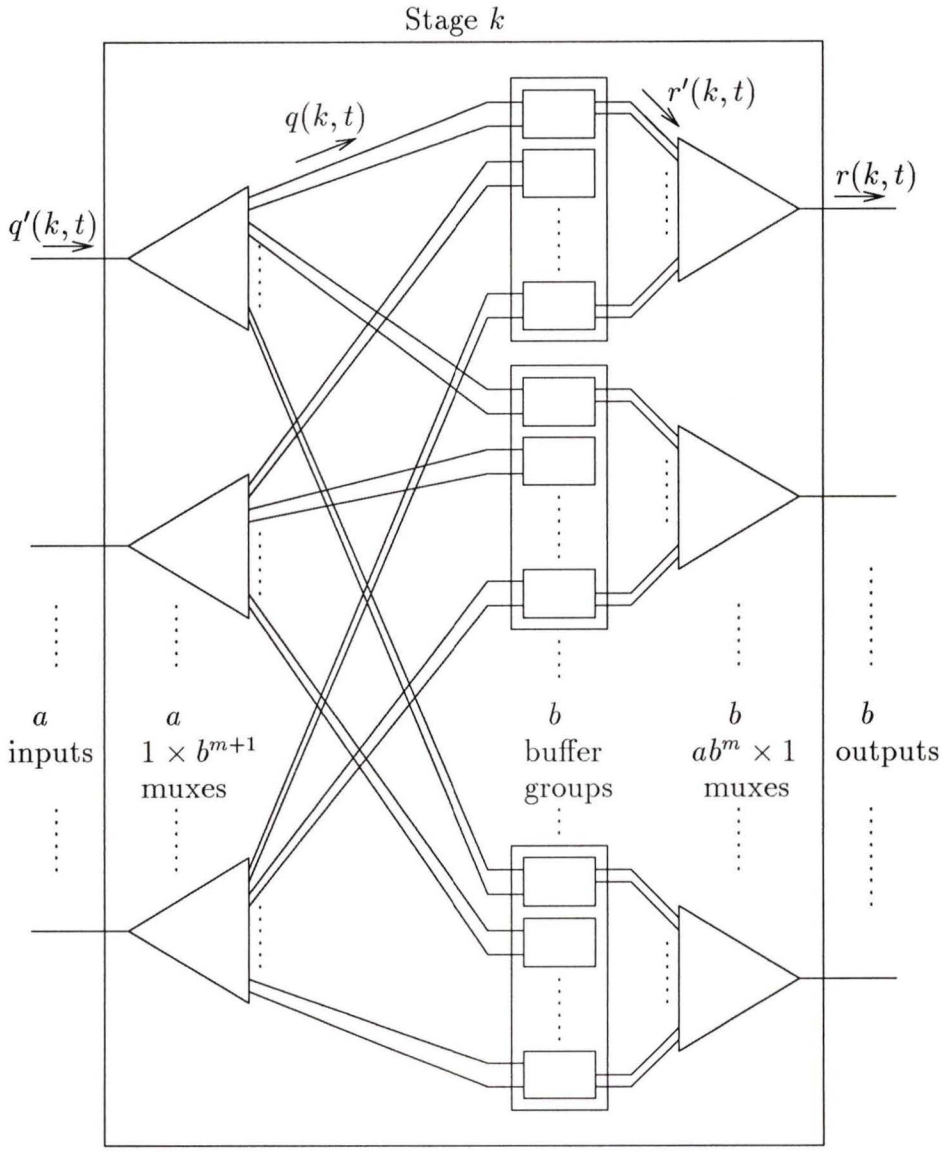


Figure 4.4: Model of SE with buffers inside the crossbar.

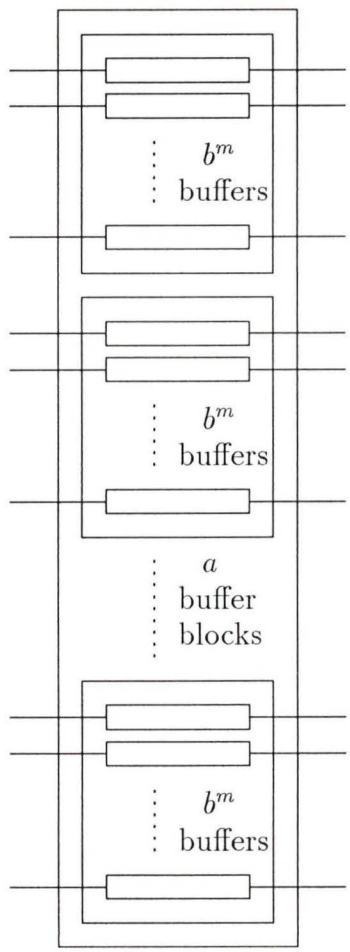


Figure 4.5: Detail of a buffer group for model of SE with buffers inside the crossbar.

The probability $q(k,t)$ that a packet is ready to come to a given buffer of an SE at stage k during the t-th stage cycle then is simply

$$\begin{aligned} q(k,t) &= \frac{1}{b^{m+1}} q'(k,t) \\ &= b^{-(m+1)}[1 - p_0(k-1,t)^{ab^m}] , \qquad (4.36) \\ q(1,t) &= b^{-(m+1)} Q_{IN} . \qquad (4.37) \end{aligned}$$

The probability $r(k,t)$, in terms of $r'(k,t)$ and $v(k+1,t)$, matches that for the SE with buffers before the crossbar ($1 \leq k \leq n$).

$$\begin{aligned} r(k,t) &= r'(k,t)v(k+1,t) \qquad (4.38) \\ v(k,t) &= \overline{p}_\beta(k,t) + f_0 p_\beta(k,t) r(k,t) \qquad (4.39) \\ v(n+1,t) &\doteq 1 \qquad (4.40) \end{aligned}$$

where f_0 indicates the type of communication during the arbitration phase: $f_0 = 0$ for local, $f_0 = 1$ for global.

There are ab^m buffers which may contend for the same output of the SE. Thus, the following is noted:

- The probability $ab^m - 1$ remaining buffers contain j HOL packets is

$$\binom{ab^m - 1}{j} \overline{p}_0(k,t)^j p_0(k,t)^{ab^m - 1 - j} .$$

- The probability the packet under consideration is chosen is $1/(j+1)$.

Therefore,

$$\begin{aligned} r'(k,t) &= \sum_{j=0}^{ab^m - 1} \binom{ab^m - 1}{j} \overline{p}_0(k,t)^j p_0(k,t)^{ab^m - 1 - j} \frac{1}{j+1} \\ &= \sum_{j=0}^{ab^m - 1} \binom{ab^m - 1}{j} \frac{1}{j+1} \overline{p}_0(k,t)^j p_0(k,t)^{ab^m - 1 - j} . \qquad (4.41) \end{aligned}$$

Since $\binom{ab^m - 1}{j} \frac{1}{j+1} = \frac{1}{ab^m} \binom{ab^m}{j+1}$, (4.41) may be written as

$$r'(k,t) = \frac{1}{ab^m} \sum_{j=0}^{ab^m - 1} \binom{ab^m}{j+1} \overline{p}_0(k,t)^j p_0(k,t)^{ab^m - 1 - j} . \qquad (4.42)$$

After some manipulation, (4.42) yields

$$r'(k,t) = \frac{1}{ab^m}\frac{[1-p_0(k,t)^{ab^m}]}{\overline{p}_0(k,t)} . \tag{4.43}$$

Note that $\lim_{\overline{p}_0(k,t)\to 0^+} r'(k,t) = 1$.

Substituting (4.36) into (4.43) yields (4.44).

$$r'(k,t) = \frac{b}{a}\frac{q(k+1,t)}{\overline{p}_0(k,t)}, \quad \begin{cases} a \geq 1 \\ b \geq 1 \\ 1 \leq k \leq n \end{cases} \tag{4.44}$$

Substituting (4.44) into (4.38) yields (4.45).

$$r(k,t) = \frac{b}{a}\frac{q(k+1,t)}{\overline{p}_0(k,t)}v(k+1,t), \quad \begin{cases} a \geq 1 \\ b \geq 1 \\ 1 \leq k \leq n \end{cases} \tag{4.45}$$

The state transition diagram for this type of SE matches that for the SE with buffers before the crossbar (see Figure 4.3). Furthermore, all the state transition probabilities and state probabilities are identical to those for the SE with buffers before the crossbar, i.e., equations (4.18)–(4.24) and (4.25)–(4.27).

The above equations may be solved iteratively to reach steady-state values. At steady-state, the following results may be determined.

$$\lambda(k) = b^{m+1}q(k)v(k) \tag{4.46}$$

$$s(k) = \overline{p}_0(k)r(k) \tag{4.47}$$

$$\mu(k) = ab^m s(k) \tag{4.48}$$

$$\begin{aligned} S_N &= \frac{(\text{number of network outputs}) \times \mu(n)}{(\text{number of network inputs}) \times Q_{IN}} \\ &= \frac{b^n \mu(n)}{a^n Q_{IN}} \end{aligned} \tag{4.49}$$

$$B(k) = \sum_{j=1}^{\beta} j p_j(k) \tag{4.50}$$

$$d(k) = \frac{B(k)}{s(k)} \tag{4.51}$$

$$D = \frac{1}{n}\sum_{k=1}^{n} d(k) \tag{4.52}$$

The equations presented in this section will subsequently be summarized in Section 4.2.3.

4.2.3 Solving the state equations iteratively

In Section 4.2.1, the state equations for a delta network using switching elements with buffers before the crossbar were derived. Then, in Section 4.2.2, the state equations for a delta network using switching elements with buffers inside the crossbar were derived. By introducing three new variables here, the two sets of equations can be merged into a single set of equations.

f_1 = "Input Splitting", which is the fraction of the traffic arriving at a buffer with respect to the traffic arriving at the input to the SE.

f_2 = "Output Splitting", which is the fraction of the traffic leaving a buffer destined for a particular output of the SE.

f_3 = "Contention Factor", which is the number of buffers which can contend for a given output of the SE.

These variables assume the following values:

	SE1	SE2
f_1	1	$b^{-(m+1)}$
f_2	b^{-1}	1
f_3	a	ab^m

SE1 refers to SE's with buffers before the crossbar, while SE2 refers to SE's with buffers inside the crossbar. Additionally, f_0 will be used to indicate the communication scheme available during arbitration: $f_0 = 0$ for local, $f_0 = 1$ for global.

Following is the complete set of equations which can be used to analyze networks based on either type of SE. Unless stated otherwise, $a \geq 1$, $b \geq 1$, $m \geq 0$, and $1 \leq k \leq n$.

$$q(k+1,t) = f_1\left[1-(1-f_2\overline{p}_0(k,t))^{f_3}\right] \tag{4.53}$$

$$q(1,t) = f_1 Q_{IN} \tag{4.54}$$

$$v(k,t) = \overline{p}_\beta(k,t) + f_0 p_\beta(k,t) r(k,t) \tag{4.55}$$

$$v(n+1,t) \doteq 1 \tag{4.56}$$

$$r(k,t) = \frac{b}{a}\frac{q(k+1,t)}{\overline{p}_0(k,t)} v(k+1,t) \tag{4.57}$$

$$p_{00}(k,t) = \overline{q}(k,t) \tag{4.58}$$

$$p_{01}(k,t) = q(k,t) \tag{4.59}$$

$$p_{jj}(k,t) = \overline{q}(k,t)\overline{r}(k,t) + q(k,t)r(k,t), \quad 1 \le j \le \beta-1 \tag{4.60}$$

$$p_{(j-1)j}(k,t) = q(k,t)\overline{r}(k,t), \quad 2 \le j \le \beta \tag{4.61}$$

$$p_{(j+1)j}(k,t) = \overline{q}(k,t) r(k,t), \quad 0 \le j \le \beta-2 \tag{4.62}$$

$$p_{\beta(\beta-1)}(k,t) = (1-f_0 q(k,t)) r(k,t) \tag{4.63}$$

$$p_{\beta\beta}(k,t) = \overline{r}(k,t) + f_0 q(k,t) r(k,t) \tag{4.64}$$

$$p_0(k,t+1) = p_0(k,t)p_{00}(k,t) + p_1(k,t)p_{10}(k,t) \tag{4.65}$$

$$\begin{aligned} p_j(k,t+1) = {} & p_{j-1}(k,t)p_{(j-1)j}(k,t) + p_j(k,t)p_{jj}(k,t) + \\ & p_{j+1}(k,t)p_{(j+1)j}(k,t), \quad 1 \le j \le \beta-1 \end{aligned} \tag{4.66}$$

$$p_\beta(k,t+1) = p_{\beta-1}(k,t)p_{(\beta-1)\beta}(k,t) + p_\beta(k,t)p_{\beta\beta}(k,t) \tag{4.67}$$

$$\lambda(k) = \frac{1}{f_1} q(k) v(k) \tag{4.68}$$

$$s(k) = \overline{p}_0(k) r(k) \tag{4.69}$$

$$\mu(k) = f_2 f_3 s(k) \tag{4.70}$$

$$S_N = \frac{b^n \mu(n)}{a^n Q_{IN}} \tag{4.71}$$

$$B(k) = \sum_{j=1}^{\beta} j p_j(k) \tag{4.72}$$

$$d(k) = \frac{B(k)}{s(k)} \tag{4.73}$$

$$D = \frac{1}{n}\sum_{k=1}^{n} d(k) \tag{4.74}$$

A program was written to solve equations (4.53) through (4.74), given the following inputs:

SE = a flag to indicate type of SE (SE1 or SE2), which in turn was used to set f_1, f_2, and f_3.

f_0 = type of communication during arbitration phase (0 for local, 1 for global).

n = number of switching stages in the network.

a = number of inputs to an SE.

b = number of outputs from an SE.

β = number of slots in a buffer.

Q_{IN} = input traffic intensity.

m = number of stages of lookahead (SE2 only).

The results are given in Section 4.4.

4.3 Simulation

In Section 4.2, two types of SE's were analyzed: SE1 with buffers before the crossbar and SE2 with buffers inside the crossbar. These two types of SE's have been modeled and simulated with the discrete-time simulator *Extend* on the Macintosh. For future work, it is recommended that an event-driven simulator be used due to the length of time taken by simulations with the discrete-time simulator. Only 2×2 SE's were modeled. In the case of SE2, parallel links were also modeled.

The simulation is based on 2 clock ticks per stage cycle. During the first phase of the clock, SE's forward a byte to their successors. In the second phase of the clock, SE's acknowledge receipt of bytes received from their predecessors in the previous clock phase. Packets of any size can be used in the simulation. The first byte of a packet is the header byte. An SE receiving a header byte decides whether

or not to accept the packet and provides the appropriate acknowledgement to the sender. An SE receiving a Negative Acknowledgement (NAK) will resubmit the request in the next stage cycle. An SE receiving a Positive Acknowledgement (ACK) will continue to forward subsequent bytes of the packet until the entire packet has been moved into the successor SE.

At the input to the network, Input Trunk Controllers (ITC's) modeled the arrival of packets with destination addresses uniformly distributed over all network outputs. Part of the information sent with the packet was the time of its arrival to the network. At the output of the network, Output Trunk Controllers (OTC's) received packets and logged data such as number of packets received and delays of packets received.

Before collecting data from a network under simulation, the network is simulated for a period of time to reach "steady state". The network starts with no packets in it. As the simulation progresses, the average number of packets in the network reaches a steady state value. The time required to reach steady state depends on the total number of slots in the network and the type of SE's used.

Once steady state is reached, data collection begins. The simulation continues until the number of packets arriving at any destination is within an acceptable margin of the mean for that destination. To determine the amount of time this takes, it is assumed that the destination addresses are uniformly distributed over all possible destination addresses. Essentially, this is a generalized Bernoulli process in which each trial (packet arrival) can result in k outcomes (destinations) $E_1, E_2, \ldots, E_k$ with probabilities $p_1, p_2, \ldots, p_k$ respectively. The probability distribution of the random variables $X_1, X_2, \ldots, X_k$, representing the number of occurrences for $E_1, E_2, \ldots, E_k$ in n independent trials (packet arrivals) is the multinomial distribution given by (4.75) [81], with $\sum_{i=1}^{k} x_i = n$ and $\sum_{i=1}^{k} p_i = 1$.

$$f(x_1, x_2, \ldots, x_k; p_1, p_2, \ldots, p_k, n) = \frac{n!}{x_1! x_2! \cdots x_k!} p_1^{x_1} p_2^{x_2} \cdots p_k^{x_k} \tag{4.75}$$

The mean and variance are given by (4.76) and (4.77), respectively [82].

$$\mu_i = np_i \tag{4.76}$$

$$\sigma_i^2 = np_i(1 - p_i) \tag{4.77}$$

In this case, $p_1 = p_2 = \ldots = p_k = 1/k$. Therefore, $\mu_1 = \mu_2 = \ldots = \mu_k = n/k$ and $\sigma_1^2 = \sigma_2^2 = \ldots = \sigma_k^2 = n(k-1)/k^2$.

With $(1-\alpha)100\%$ confidence, μ_i is contained in the interval $x_i \pm z_{\alpha/2}\sigma_i$, where $z_{\alpha/2}$ is the z-value leaving an area of $\alpha/2$ to the right (or equivalently, a z-value including an area of $1-\alpha/2$ to the left) on the normal curve. Thus, with $(1-\alpha)100\%$ confidence, the greatest relative error e of x_i estimating μ_i is given by (4.78).

$$e = \frac{z_{\alpha/2}\sigma_i}{\mu_i} = z_{\alpha/2}\sqrt{\frac{k-1}{n}} \tag{4.78}$$

Therefore, for a given $(1-\alpha)100\%$ confidence that the error will not exceed $(e)100\%$, the number of packets which must arrive at the network outputs is given by (4.79).

$$n = \left(\frac{z_{\alpha/2}}{e}\right)^2 (k-1) \tag{4.79}$$

Figure 4.6 plots the total packet arrivals versus number of outputs for confidence levels of 95% and 99%. Simulations were run long enough to provide a 99% confidence level that the number of packets arriving at any destination was within 3% of the expected mean. A simulation of an 8×8 network for a single input traffic rate required about an hour on the facilities used. The same simulation for a 64×64 network required about twelve hours. Solving the equations of Section 4.2.3 on comparable computer facilities required only seconds.

The simulation results from this work are given in Section 4.4.

4.4 Analysis and Simulation Results

In analyzing results for the two types of SE's, networks with the same a, b, and n are compared. Furthermore, SE's with the same amount of buffer space must be compared, since buffer space consumes a large portion of the chip area in a hardware implementation. Where possible, graphs in this section have maintained

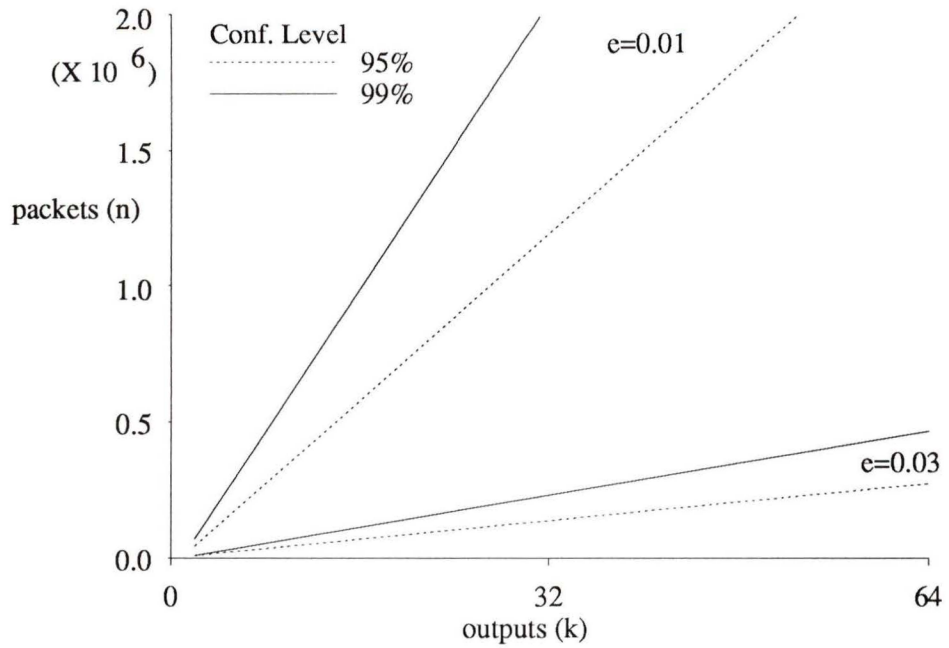


Figure 4.6: Total packet arrivals versus number of outputs for 95% and 99% confidence levels.

consistency in line and point types to facilitate comparisons. The arbitration communication method is assumed to be local, unless indicated otherwise.

See Section 4.2 for definitions of *offered load* (Q_{IN}), *network throughput* (S_N), and *normalized delay* (D). For ideal throughput performance, the network can pass any offered load from inputs to outputs (i.e., $S_N = Q_{IN}$). For ideal normalized delay performance, each stage would give the minimum possible delay of 1 stage cycle (i.e., $D = 1$).

The analysis and simulation results are summarized in Section 4.6. Section 4.4.1 provides a detailed discussion of the results.

4.4.1 Discussion of results

Figures 4.7–4.12 show performance results at the switching element level, while Figures 4.13–4.18 show performance results for three stage networks. In each case, both the analysis and simulation results are given. In the discussion that follows, SE1 refers to SE's with buffers before the crossbar, SE2A refers to SE's with buffers inside the crossbar and no look-ahead routing, and SE2B refers to SE's with buffers

inside the crossbar and 1-stage look-ahead routing.

From Figures 4.7 and 4.9, it is evident that maximum throughput for SE1 is 0.75, while that of SE2A approaches unity. Also, from Figures 4.8 and 4.10, it is seen that adding buffers to SE1 increases normalized delay to a much greater extent than adding buffers to SE2A. Thus, the overall performance of SE2A is much better than that of SE1, a conclusion previously reached by Kumar and Jump [70] based strictly on simulation results.

Figures 4.11 and 4.12 show the performance of SE2B. For 8 and 16 slots/SE, SE2A gives slightly better throughput than SE2B. For greater than 16 slots/SE, the throughput is about equal. Below an offered load of 0.9, SE2A and SE2B give comparable normalized delay performance. Above an offered load of 0.9, SE2B gives better normalized delay performance.

From Figures 4.13–4.18, the same conclusions on the relative performance of the various SE's at the network level can be drawn. An exception is that the normalized delay performance of SE2A with 8 slots/SE is slightly better than that of SE2B. Also, it should be noted that, although there is good agreement between simulation and analysis at lower offered loads, some disagreement appears for the higher offered loads. Several authors have attributed this disparity to the fact that, in the analysis, blocked requests are not resubmitted, as they are in the simulation [83]–[85]. Since the simulation resubmits blocked packets, the request rate between SE's of the network is somewhat higher than that estimated by the analysis. Thus, the analysis tends to overestimate the throughput and delay performance of the network as the offered load approaches 1.0.

Previously (compare Figure 4.10 to 4.12 and Figure 4.16 to 4.18), it appeared that SE2B provided better normalized delay performance than SE2A above approximately 0.8 offered load. Below that, there appeared to be little difference in the normalized delay performance of the two SE's. To confirm this for varying network sizes, throughput and normalized delay were plotted against network size for offered loads of 1.0 and 0.8.

Figures 4.19–4.24 show the network performance with respect to the size of the network (in stages) for an offered load of 1.0. As mentioned previously, differences between simulation and analysis are attributed to the difference in the handling of blocked requests.

Comparing Figures 4.19–4.22 shows that, in general, the performance of SE1 is worse than SE2A for any network size. The only exception is when SE2A has single-slot buffers, in which case SE1 can provide slightly better throughput. From Figures 4.21 and 4.23, as found before, SE2A gives slightly better throughput performance than SE2B for 8 and 16 slots/SE. For both SE2A and SE2B, throughput approaches offered load for more than 16 slots/SE. Figures 4.22 and 4.24 show, as found above, that normalized delay is better with SE2B than SE2A for an offered load of 1.0.

Figures 4.25–4.30 show the network performance with respect to the size of the network (in stages) for an offered load of 0.8. Figures 4.25 and 4.26 for SE1 at an offered load of 0.8 show practically identical results to those of SE1 at an offered load of 1.0 (Figures 4.19 and 4.20) due to the network being saturated in both cases.

Comparing Figures 4.27 and 4.29, it is found that SE2A provides better throughput than SE2B for 8 and 16 slots/SE. Over 16 slots/SE, the performance is equivalent. From Figures 4.28 and 4.30, SE2A is found to provide marginally better normalized delay performance than SE2B for all network sizes. Thus, if the network can be designed such that the offered load does not exceed 0.8, then the simpler design of SE2A may be used. If the offered load may exceed 0.8, then the design of SE2B could be used to achieve better normalized delay performance.

Figures 4.31 and 4.32 show a performance comparison between dilated and non-dilated 8×8 networks based on SE2A. Figures 4.33 and 4.34 show a performance comparison between dilated and non-dilated 8×8 networks based on SE2B. In both cases, performance is somewhat improved by the dilation of the network. However, this small improvement may not justify the more complex hardware required by the dilated network (more pins per SE and more complex arbitration logic).

Simulations were performed on 8×8 dilated and non-dilated networks (based on SE2A and SE2B switches) for the random, rotating, and buffer occupancy arbitration schemes. In addition, simulations were performed for the proposed arbitration scheme for d-dilated networks (Section 3.3.4).

Figures 4.35 and 4.36 show the packet delay distribution for various arbitration schemes for non-dilated 8×8 networks based on SE2A at offered loads of 1.0 and 0.8, respectively. It can be seen that the random, rotating, and buffer occupancy arbitration schemes perform about the same for non-dilated networks,

an observation also made by Kumar and Jump [70].

Figures 4.37 and 4.38 show the packet delay distribution for various arbitration schemes for non-dilated 8×8 networks based on SE2B at offered loads of 1.0 and 0.8, respectively. At an offered load of 0.8, all arbitration schemes simulated performed alike. However, at an offered load of 1.0, the random and rotating arbitration schemes show similar performance, but the buffer occupancy arbitration scheme shows somewhat poorer performance. The degradation for the buffer occupancy arbitration scheme seems to happen only at an offered load of 1.0. No explanation has been found for this anomaly.

Figures 4.39–4.42 show the packet delay distribution for operating with and without virtual cut-through for non-dilated 8×8 networks based on SE2A (with 16 or 32 slots/SE) at offered loads of 1.0 and 0.8. Virtual cut-through does offer some delay performance improvements at an offered load of 0.8, but very little at an offered load of 1.0.

When an SE is blocked from sending a given packet, it continues trying to send the same packet until it meets with success. Alternatively, the SE could be designed to continue trying different packets until one is accepted. Figures 4.43 and 4.44 show the packet delay distribution depending on the action taken for blocked requests for non-dilated 8×8 networks based on SE2A at offered loads of 1.0 and 0.8, respectively. As can be seen, there is really no advantage to trying different packets in response to a blocked request.

Figures 4.45–4.48 compare the local and global arbitration communication methods for SE2A and SE2B. As expected, the global scheme gives better performance, particularly for the lower numbers of slots/SE. For higher numbers of slots/SE, the throughput performance is the same and the delay performance only marginally better for the global scheme. Thus, for higher slots/SE, the method which is simplest to implement should be used. Generally, this is the local communication method, since SE's need communicate only with their nearest neighbors.

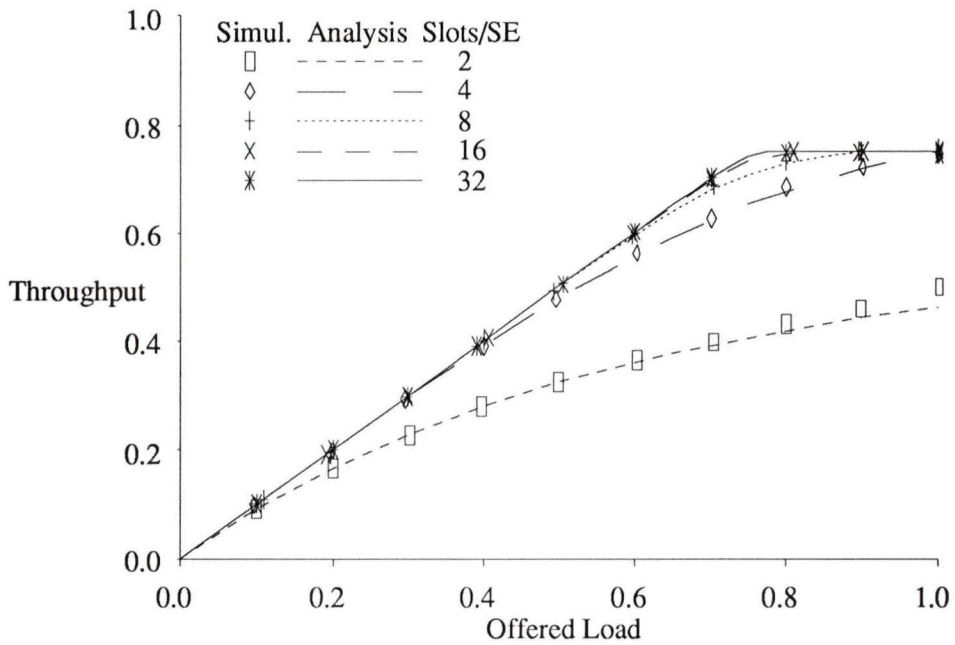


Figure 4.7: Throughput versus Offered Load: 2×2 SE with buffers before the crossbar.

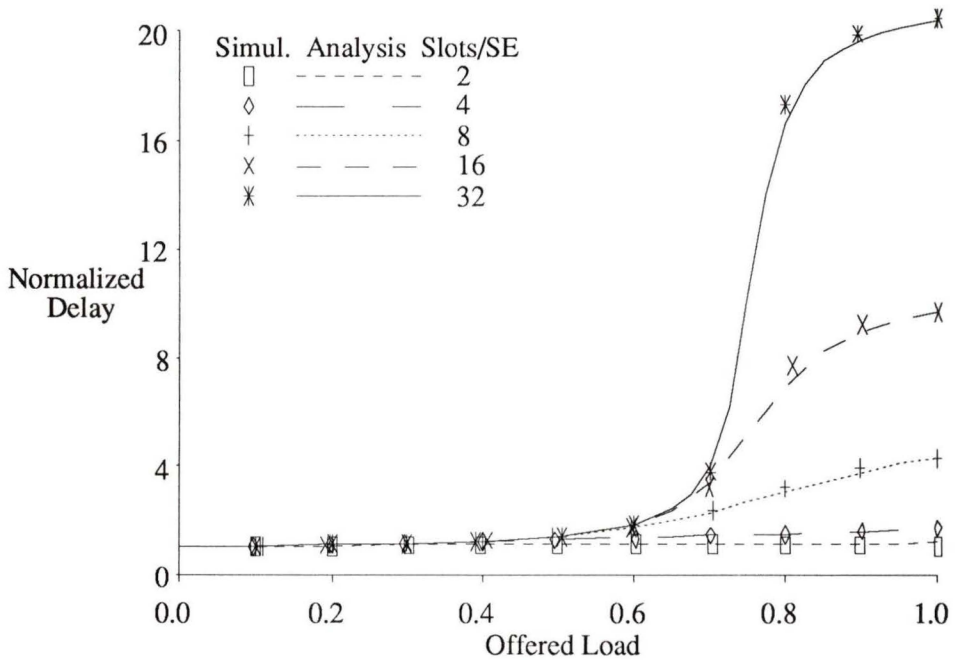


Figure 4.8: Normalized Delay versus Offered Load: 2×2 SE with buffers before the crossbar.

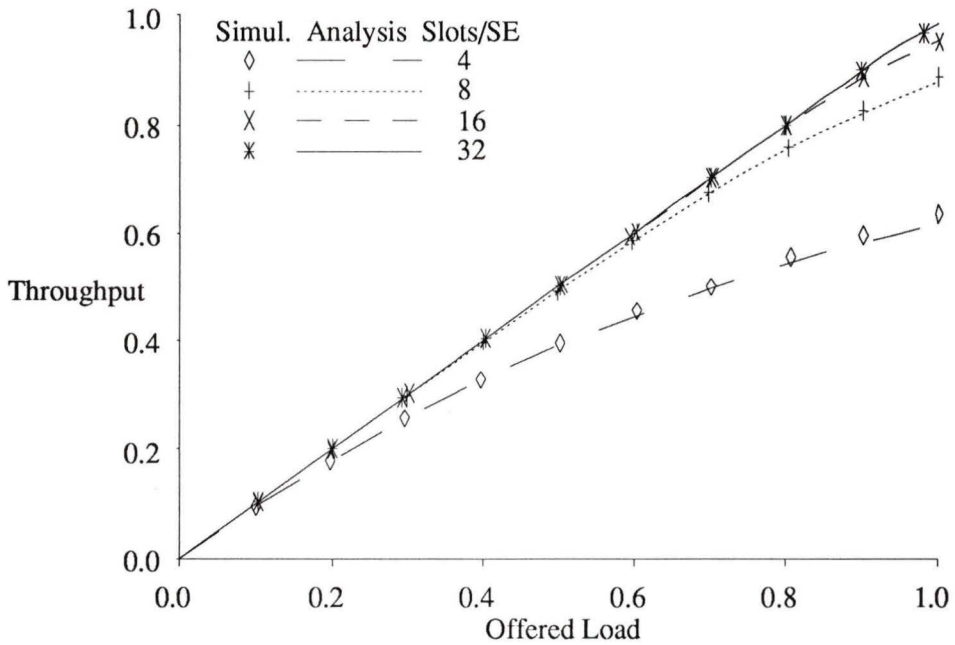


Figure 4.9: Throughput versus Offered Load: 2×2 SE with buffers inside the crossbar and no look-ahead routing.

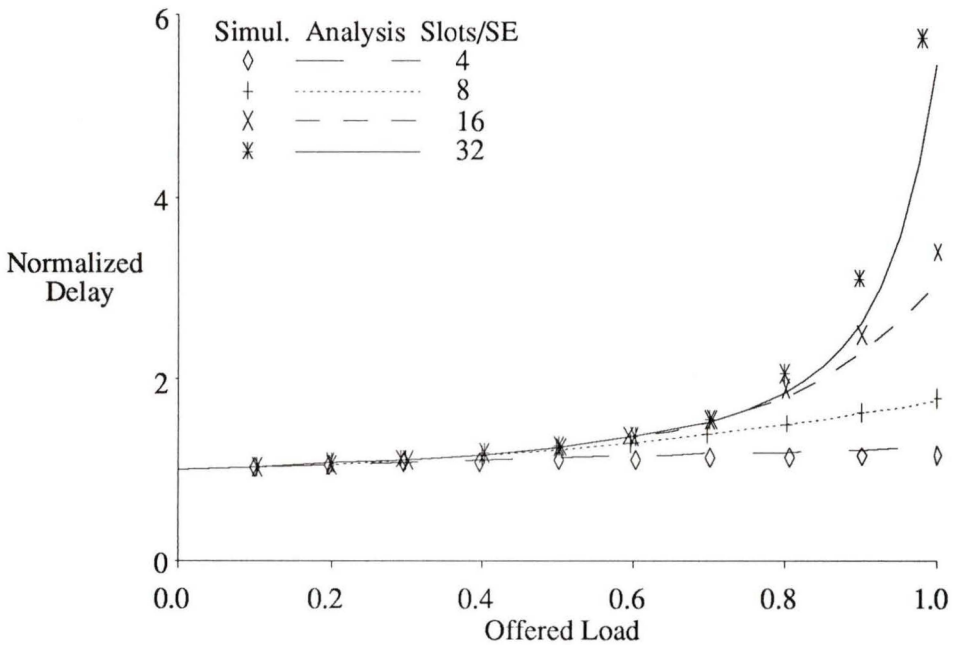


Figure 4.10: Normalized Delay versus Offered Load: 2×2 SE with buffers inside the crossbar and no look-ahead routing.

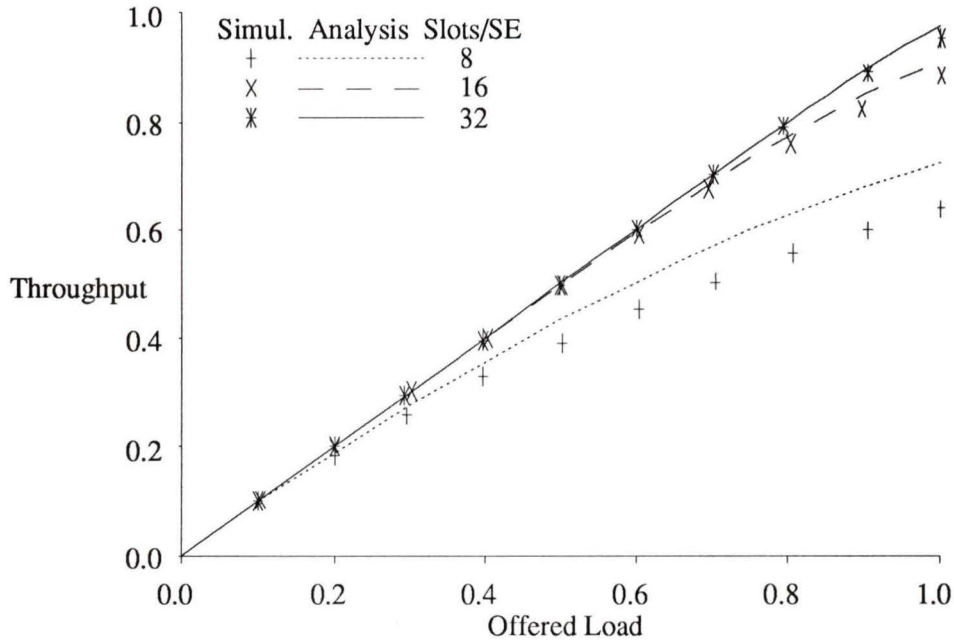


Figure 4.11: Throughput versus Offered Load: 2×2 SE with buffers inside the crossbar and 1-stage look-ahead routing.

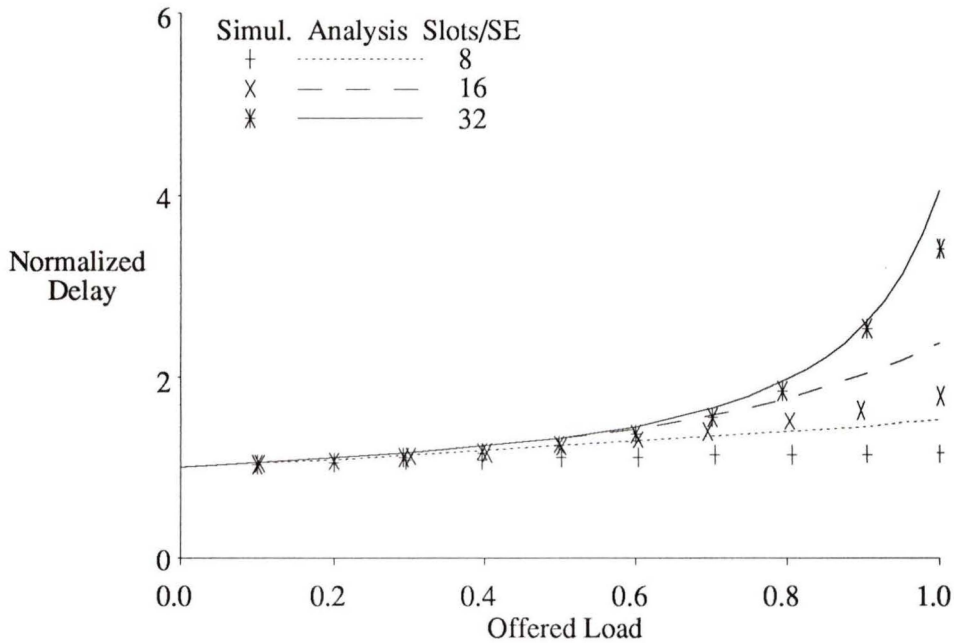


Figure 4.12: Normalized Delay versus Offered Load: 2×2 SE with buffers inside the crossbar and 1-stage look-ahead routing.

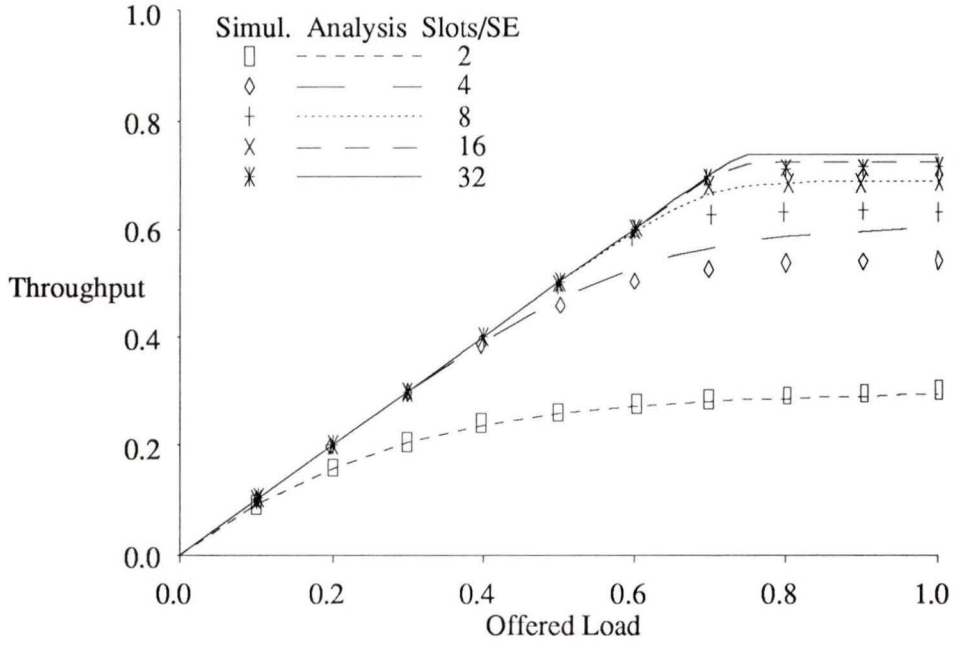


Figure 4.13: Throughput versus Offered Load: 8×8 network based on 2×2 SE's with buffers before the crossbar.

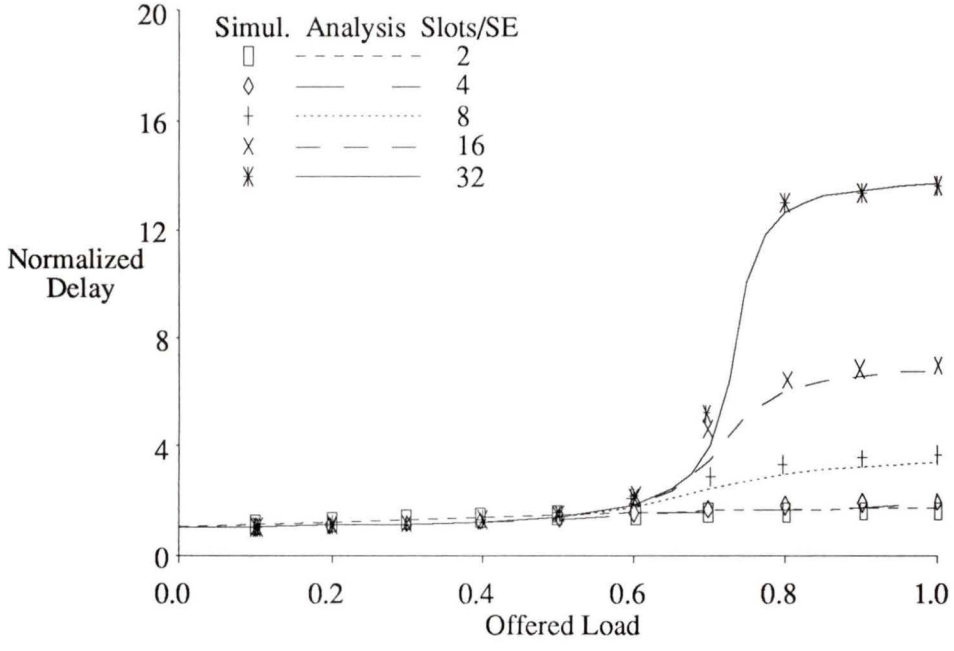


Figure 4.14: Normalized Delay versus Offered Load: 8×8 network based on 2×2 SE's with buffers before the crossbar.

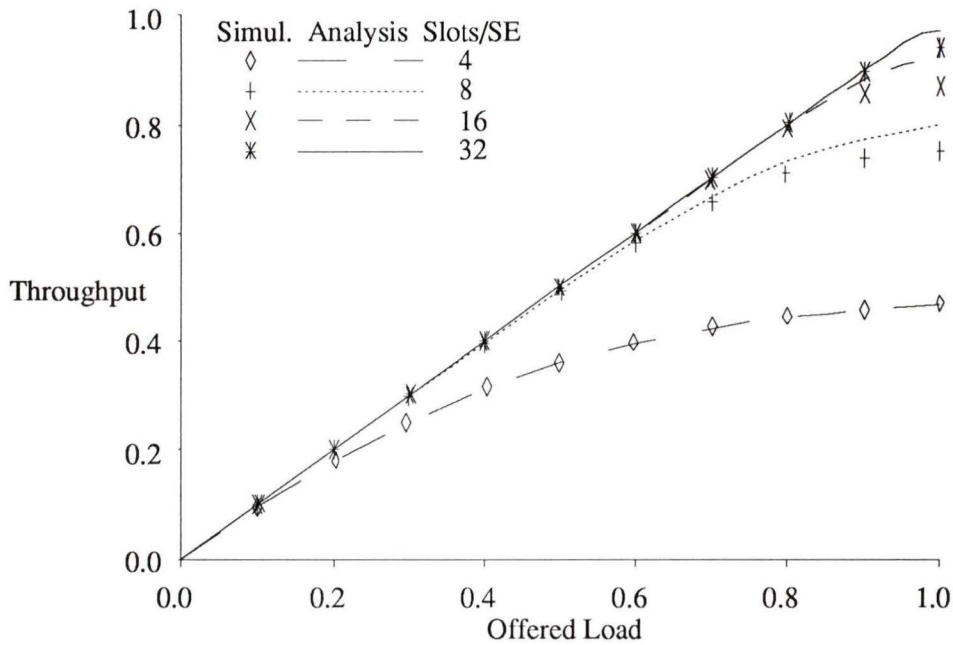


Figure 4.15: Throughput versus Offered Load: 8×8 network based on 2×2 SE's with buffers inside the crossbar and no look-ahead routing.

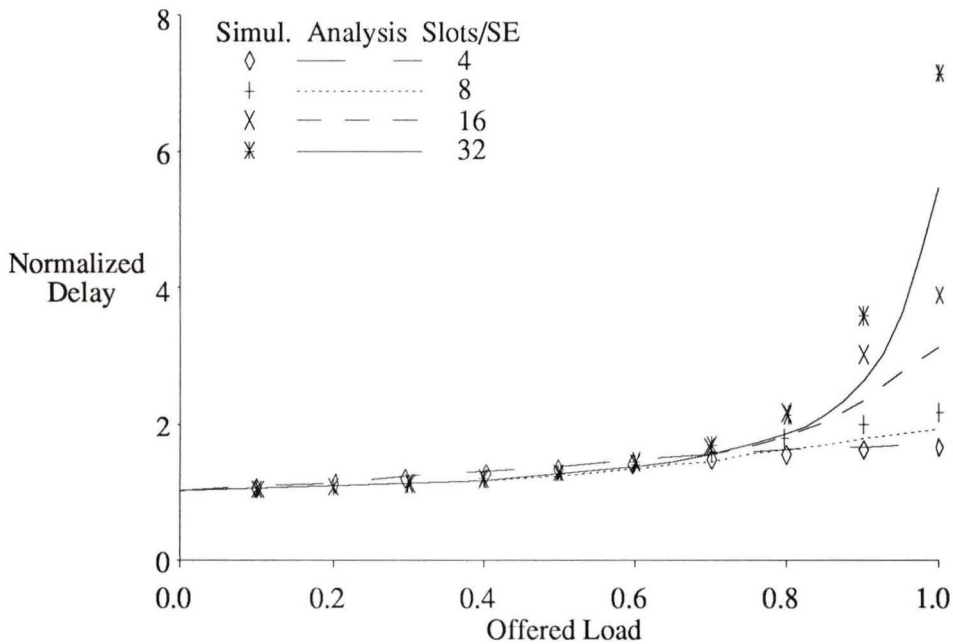


Figure 4.16: Normalized Delay versus Offered Load: 8×8 network based on 2×2 SE's with buffers inside the crossbar and no look-ahead routing.

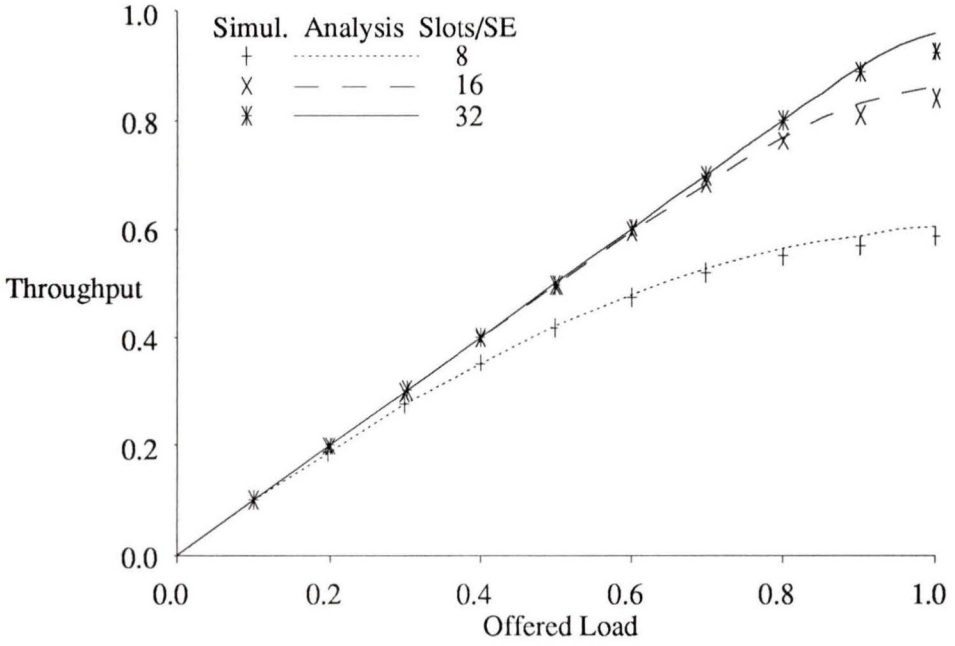


Figure 4.17: Throughput versus Offered Load: 8×8 network based on 2×2 SE's with buffers inside the crossbar and 1-stage look-ahead routing.

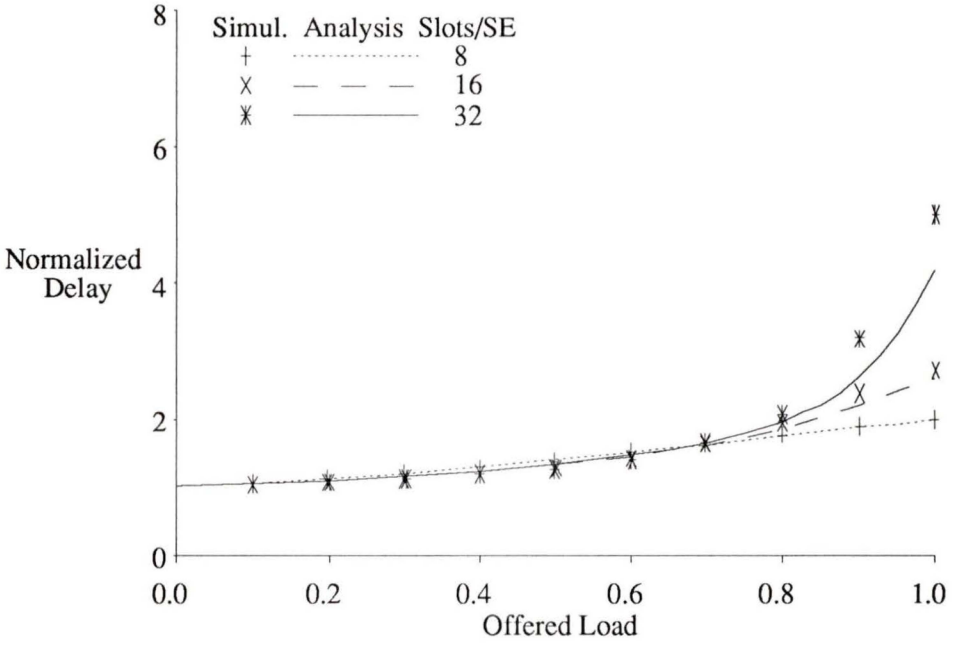


Figure 4.18: Normalized Delay versus Offered Load: 8×8 network based on 2×2 SE's with buffers inside the crossbar and 1-stage look-ahead routing.

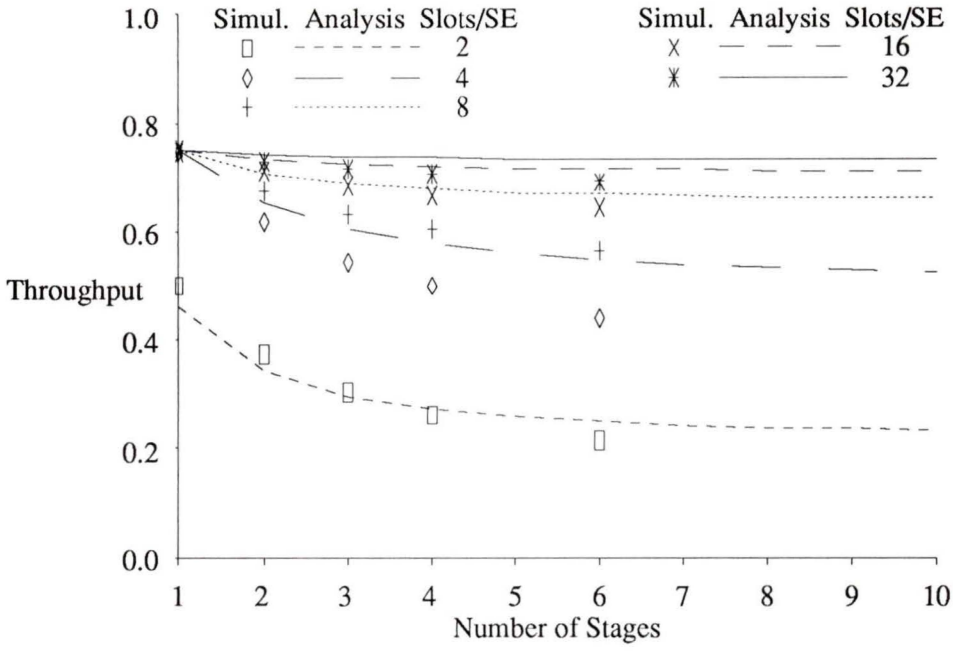


Figure 4.19: Throughput versus Number of Stages: network based on 2×2 SE's with buffers before the crossbar, offered load 1.0.

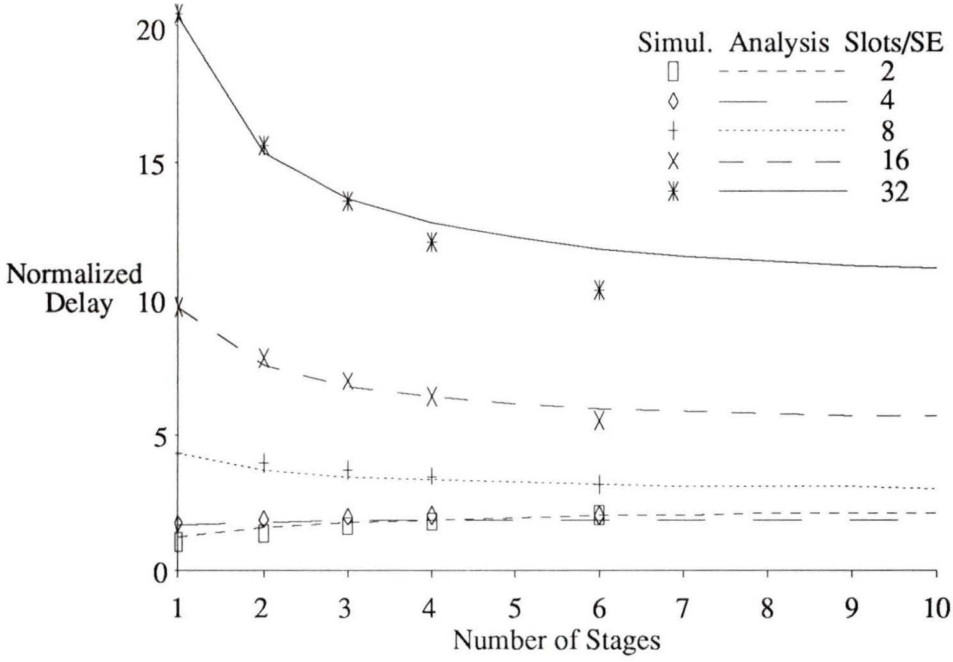


Figure 4.20: Normalized Delay versus Number of Stages: network based on 2×2 SE's with buffers before the crossbar, offered load 1.0.

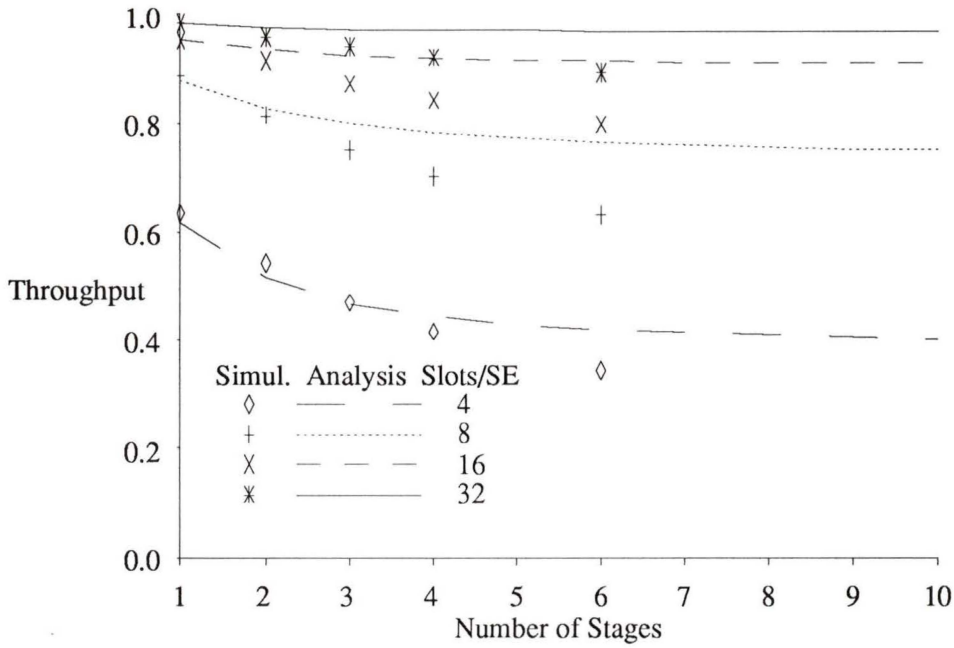


Figure 4.21: Throughput versus Number of Stages: network based on 2×2 SE's with buffers inside the crossbar, no look-ahead routing, offered load 1.0.

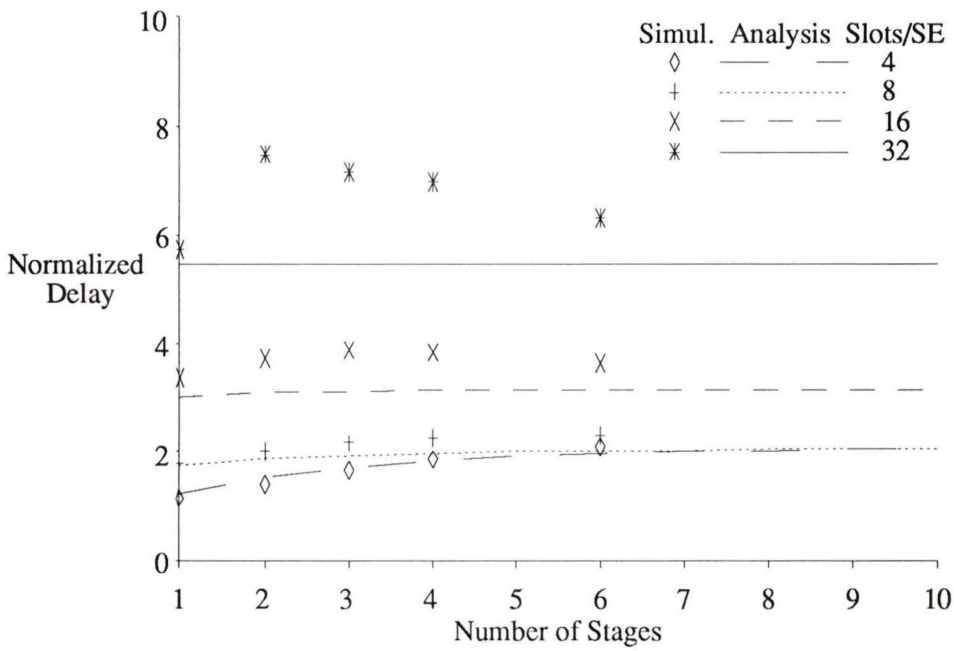


Figure 4.22: Normalized Delay versus Number of Stages: network based on 2×2 SE's with buffers inside the crossbar, no look-ahead routing, offered load 1.0.

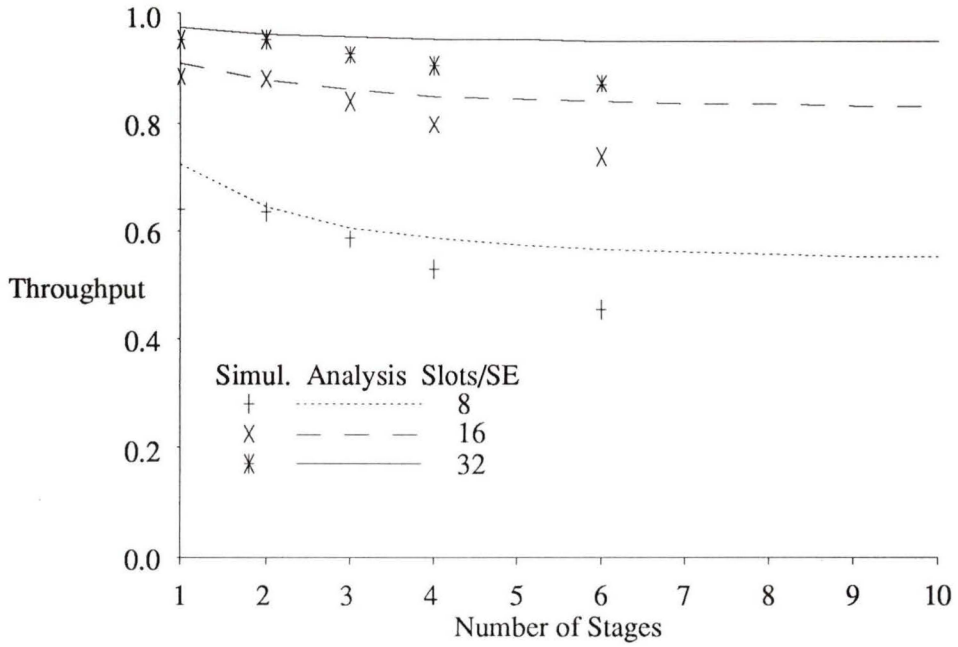


Figure 4.23: Throughput versus Number of Stages: network based on 2×2 SE's with buffers inside the crossbar, 1-stage look-ahead routing, offered load 1.0.

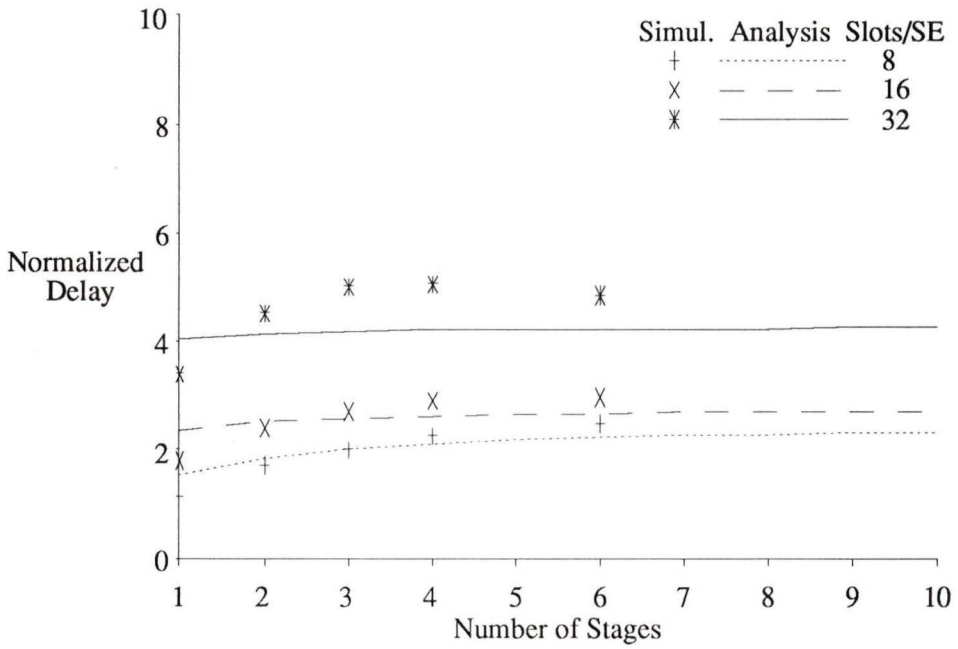


Figure 4.24: Normalized Delay versus Number of Stages: network based on 2×2 SE's with buffers inside the crossbar, 1-stage look-ahead routing, offered load 1.0.

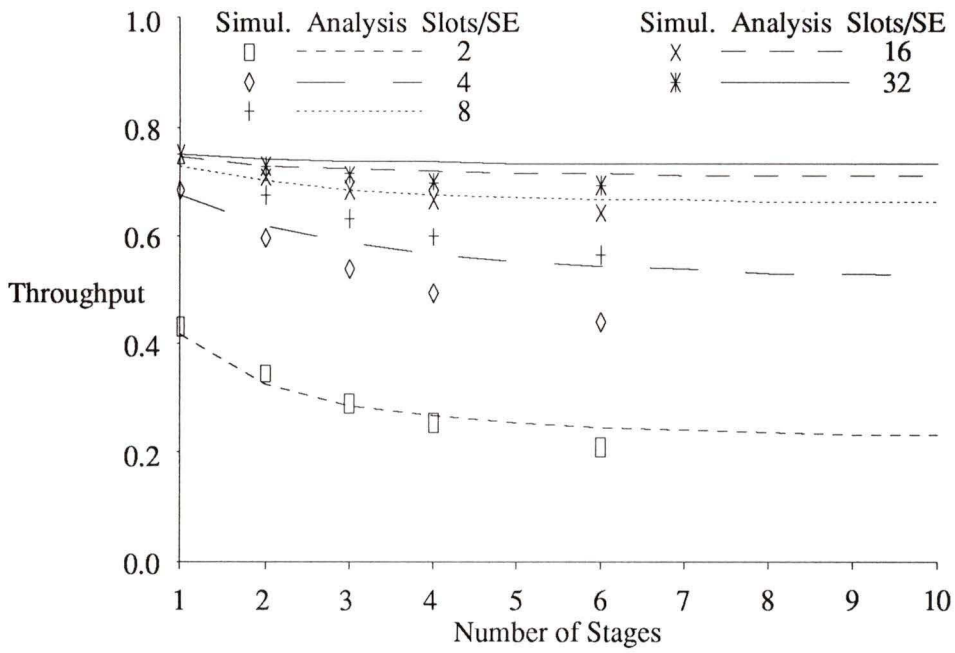


Figure 4.25: Throughput versus Number of Stages: network based on 2×2 SE's with buffers before the crossbar, offered load 0.8.

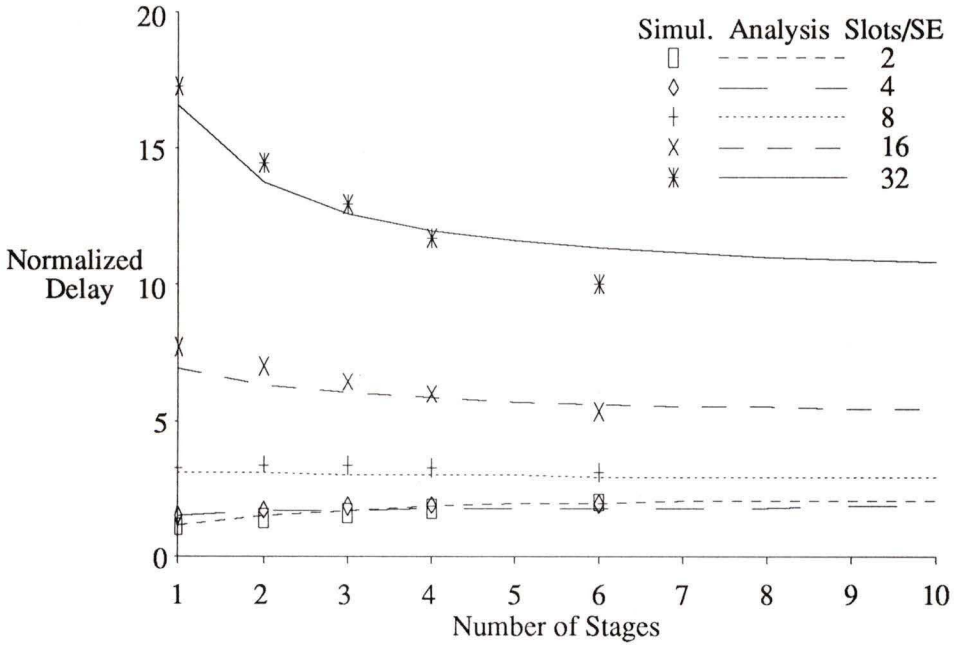


Figure 4.26: Normalized Delay versus Number of Stages: network based on 2×2 SE's with buffers before the crossbar, offered load 0.8.

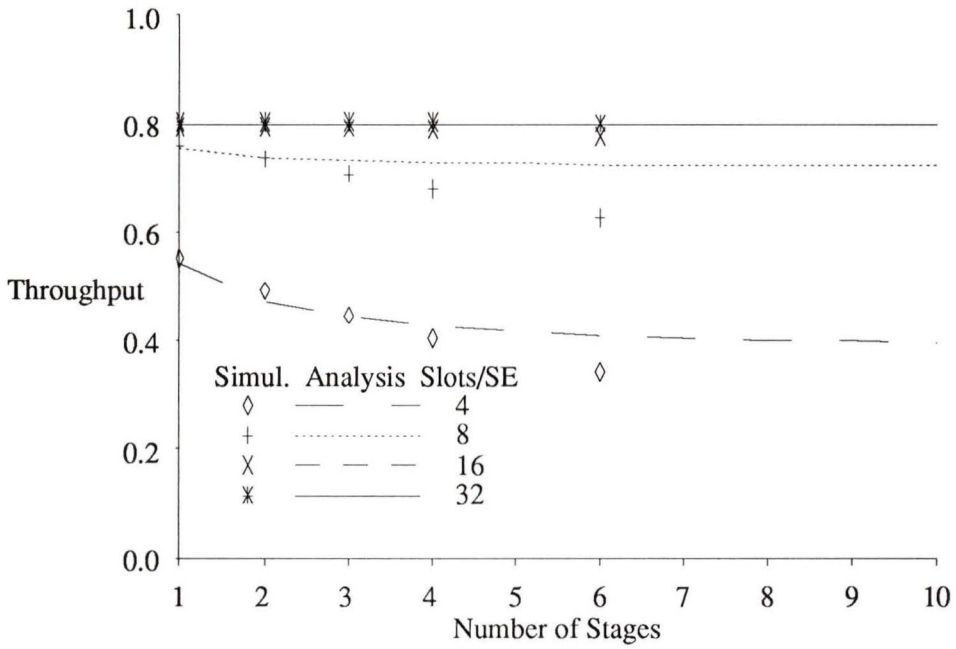


Figure 4.27: Throughput versus Number of Stages: network based on 2×2 SE's with buffers inside the crossbar, no look-ahead routing, offered load 0.8.

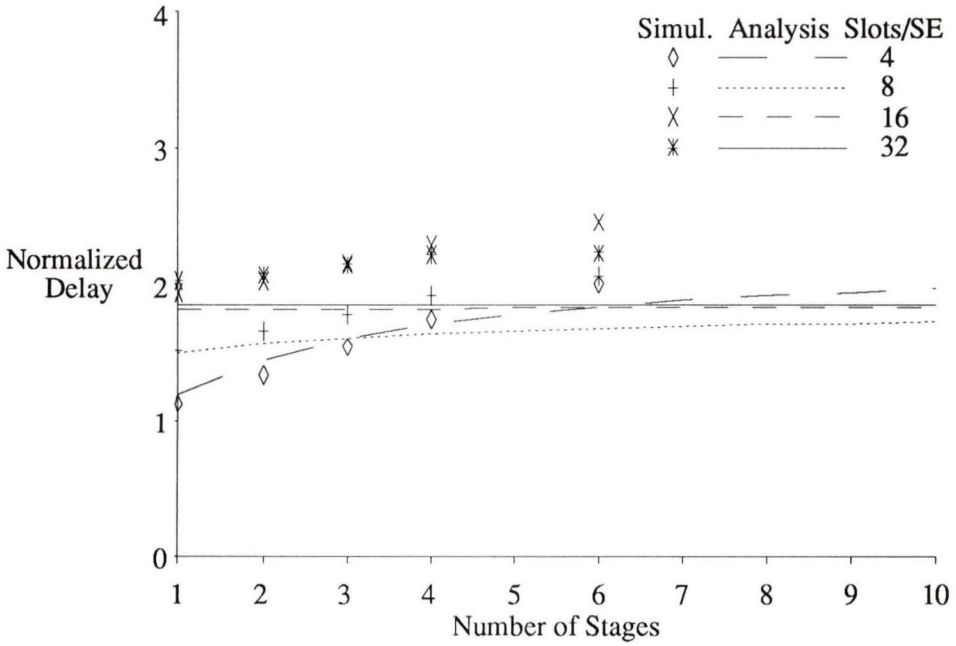


Figure 4.28: Normalized Delay versus Number of Stages: network based on 2×2 SE's with buffers inside the crossbar, no look-ahead routing, offered load 0.8.

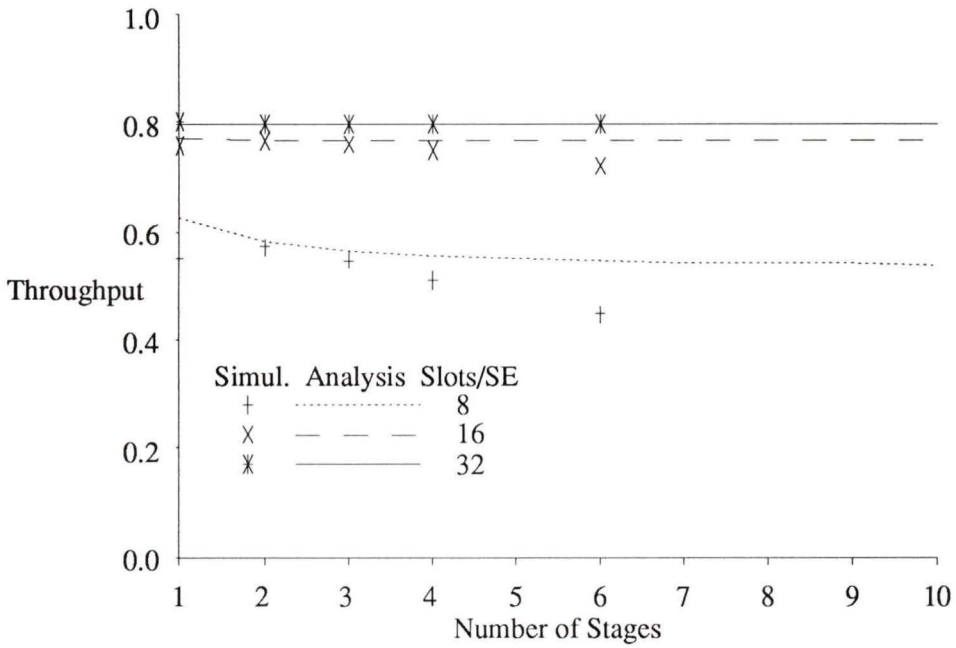


Figure 4.29: Throughput versus Number of Stages: network based on 2×2 SE's with buffers inside the crossbar, 1-stage look-ahead routing, offered load 0.8.

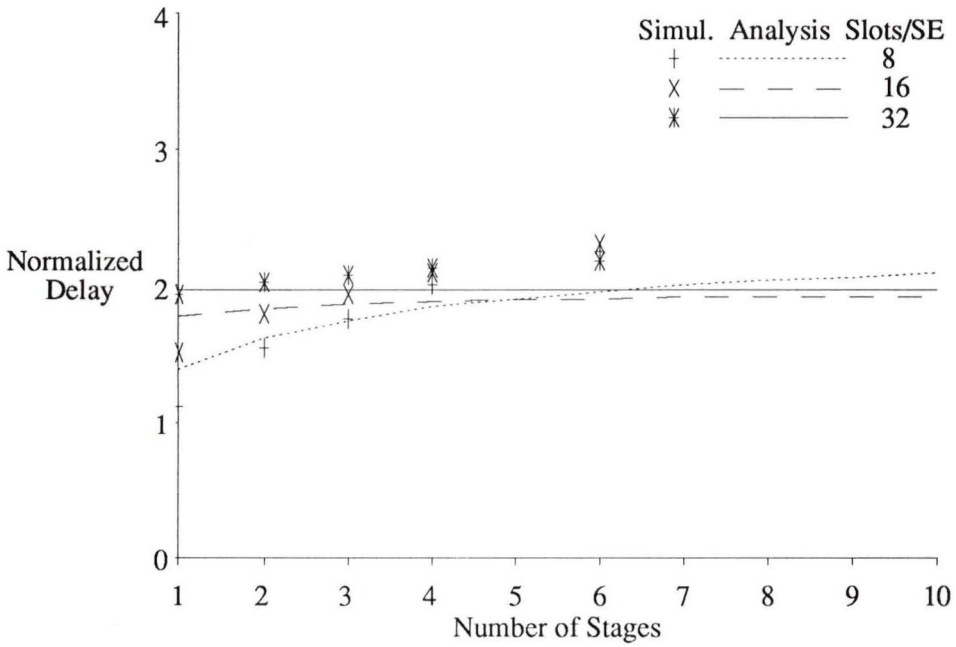


Figure 4.30: Normalized Delay versus Number of Stages: network based on 2×2 SE's with buffers inside the crossbar, 1-stage look-ahead routing, offered load 0.8.

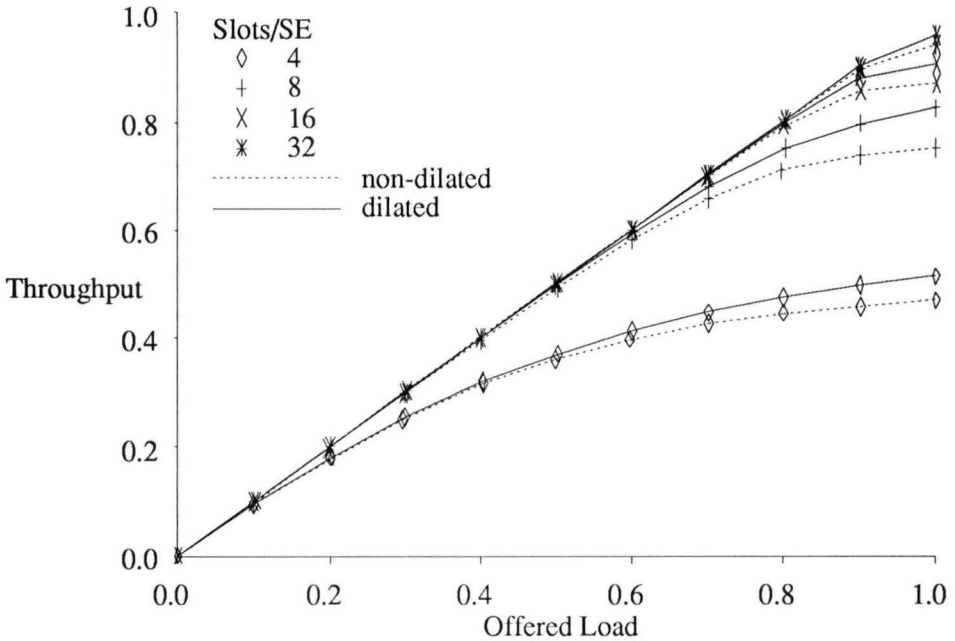


Figure 4.31: Throughput versus Offered Load for dilated and non-dilated networks (simulated): 8×8 network based on SE2A.

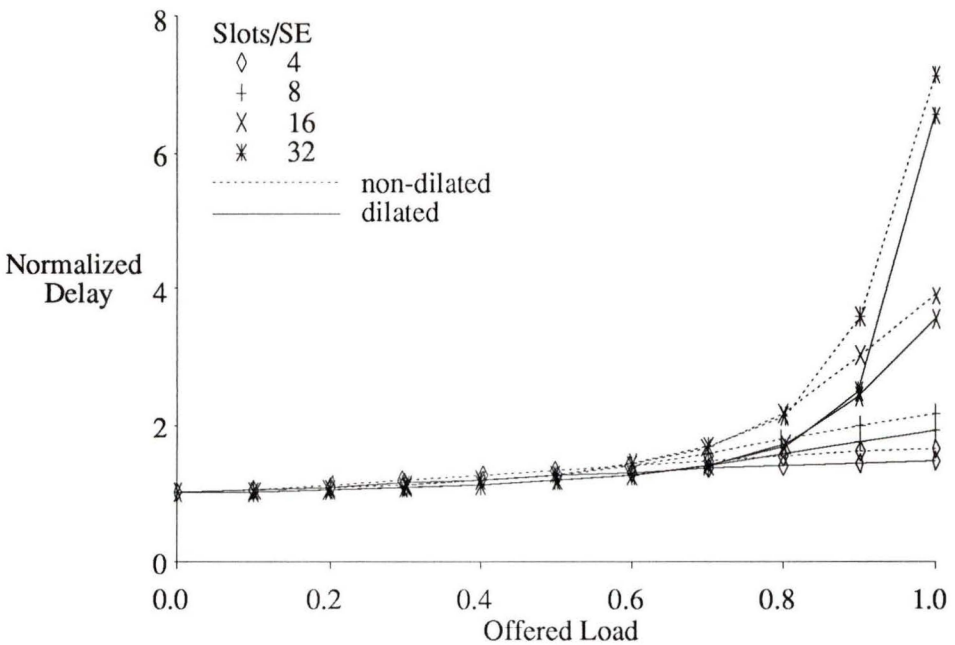


Figure 4.32: Normalized Delay versus Offered Load for dilated and non-dilated networks (simulated): 8×8 network based on SE2A.

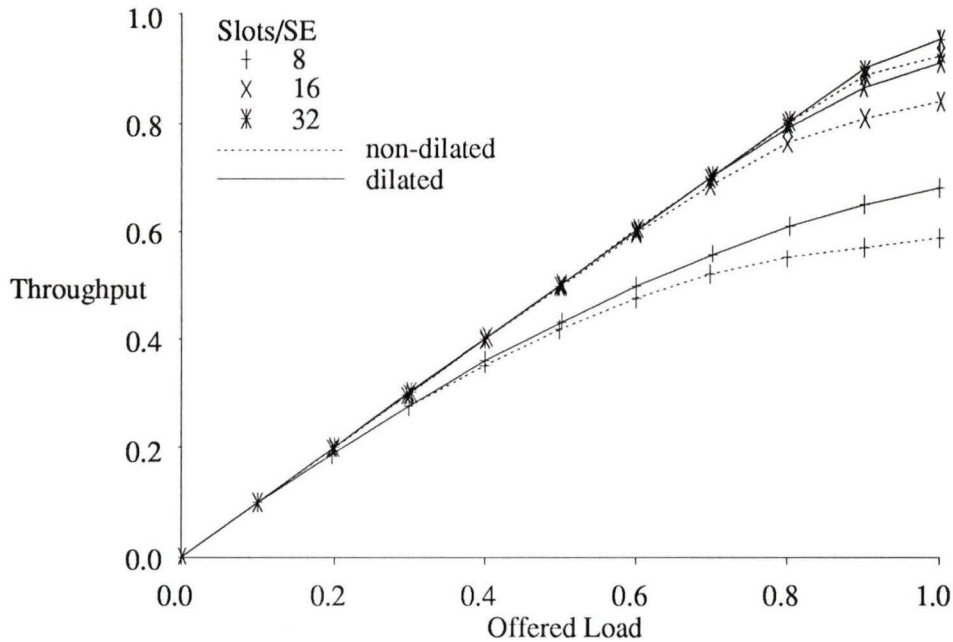


Figure 4.33: Throughput versus Offered Load for dilated and non-dilated networks (simulated): 8×8 network based on SE2B.

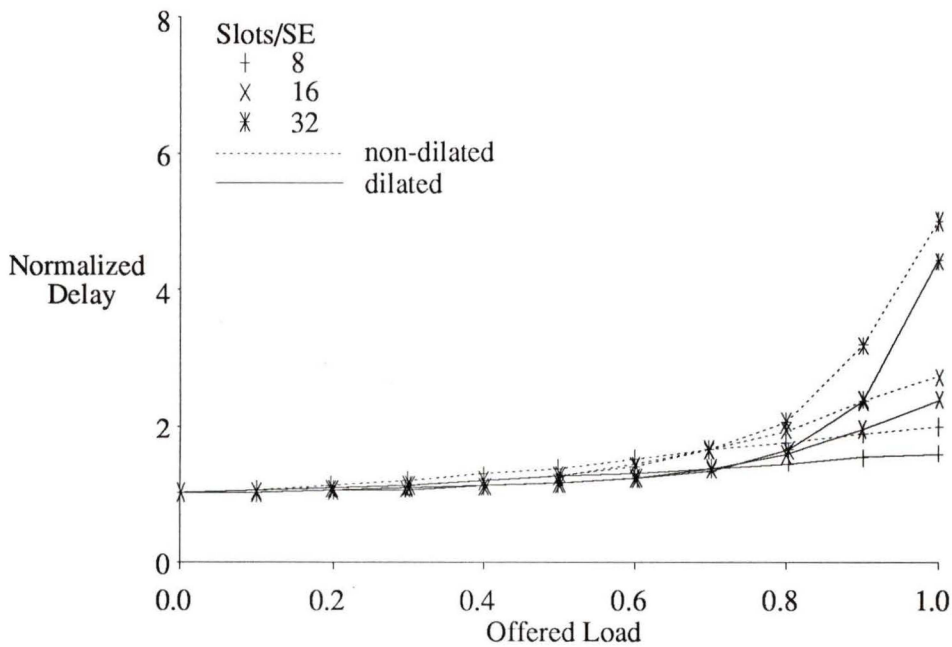


Figure 4.34: Normalized Delay versus Offered Load for dilated and non-dilated networks (simulated): 8×8 network based on SE2B.

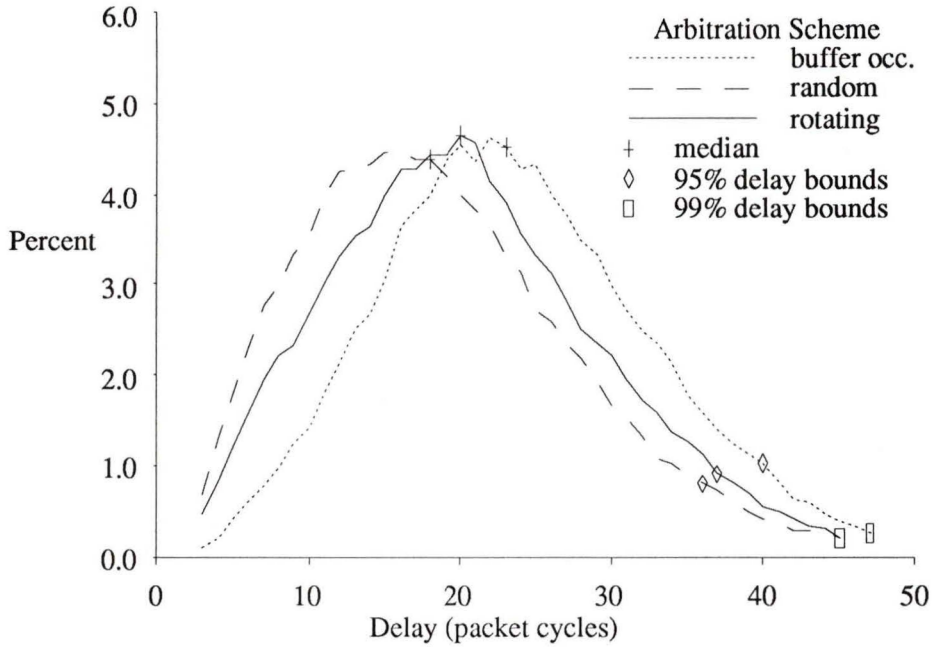


Figure 4.35: Packet delay distribution for various arbitration schemes (simulated): 8×8 network based on SE2A, 32 slots/SE, offered load 1.0.

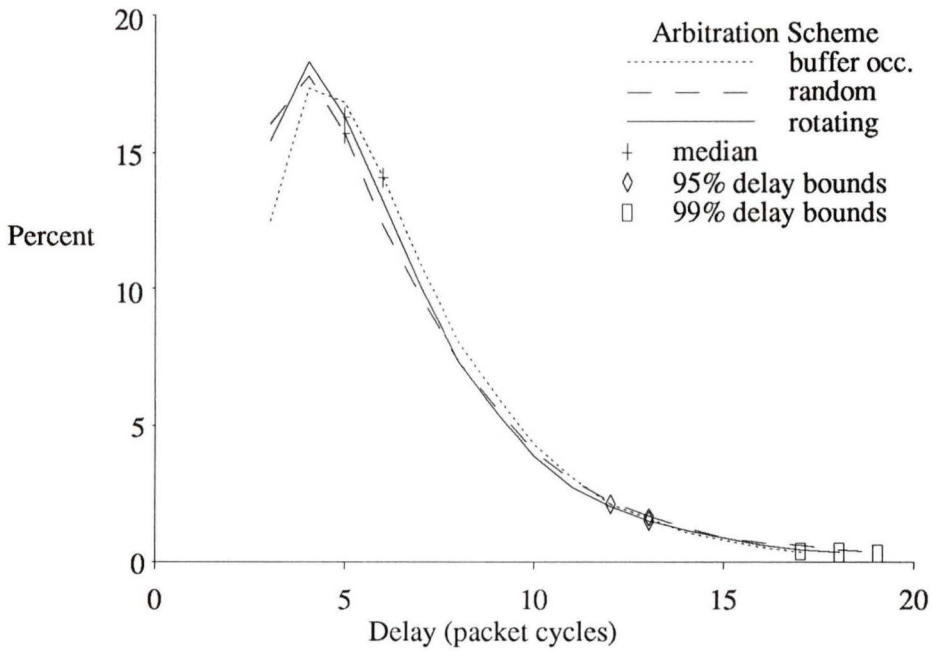


Figure 4.36: Packet delay distribution for various arbitration schemes (simulated): 8×8 network based on SE2A, 32 slots/SE, offered load 0.8.

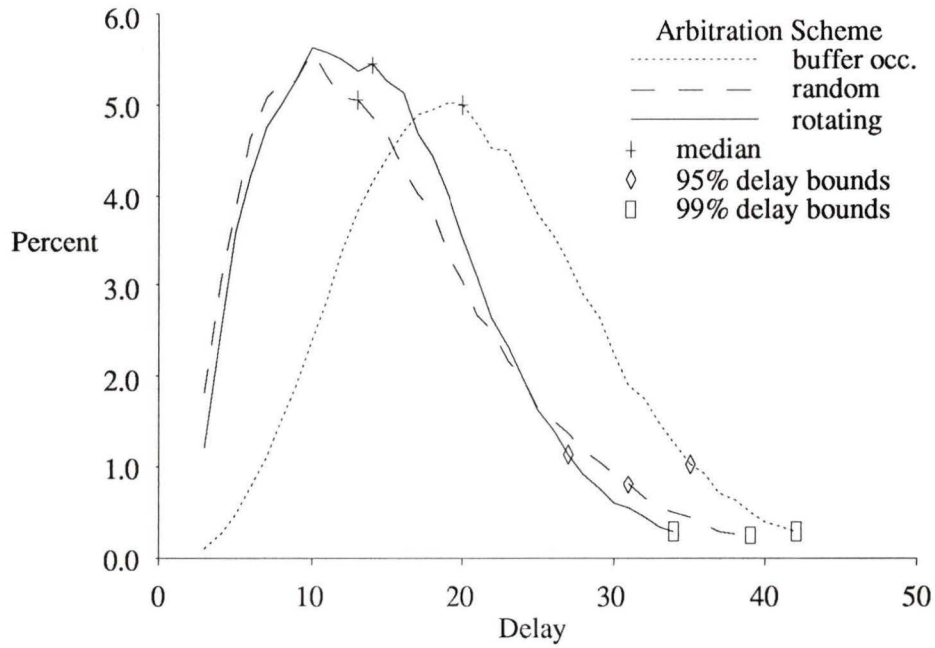


Figure 4.37: Packet delay distribution for various arbitration schemes (simulated): 8×8 network based on SE2B, 32 slots/SE, offered load 1.0.

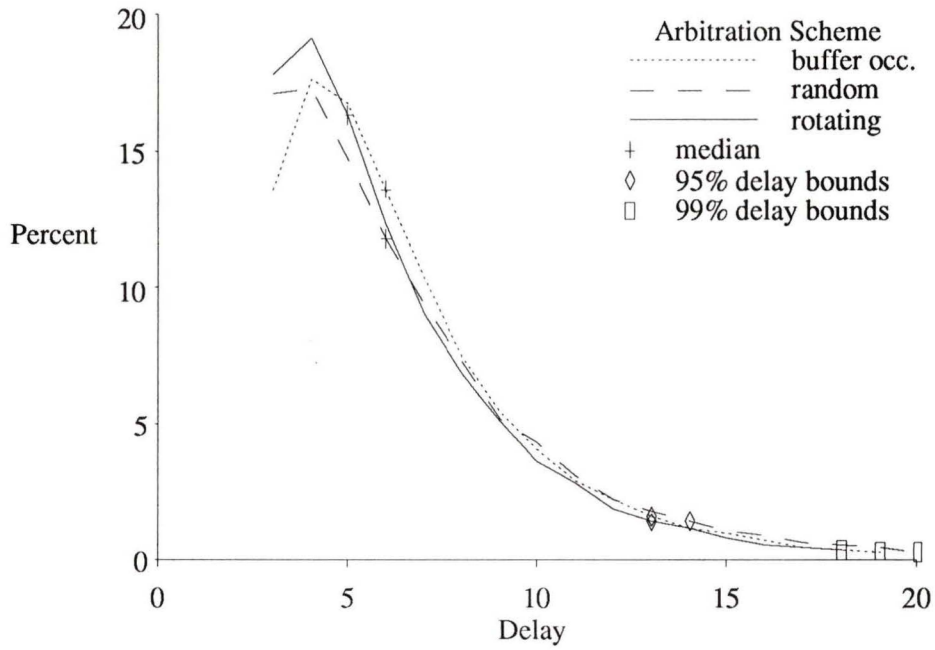


Figure 4.38: Packet delay distribution for various arbitration schemes (simulated): 8×8 network based on SE2B, 32 slots/SE, offered load 0.8.

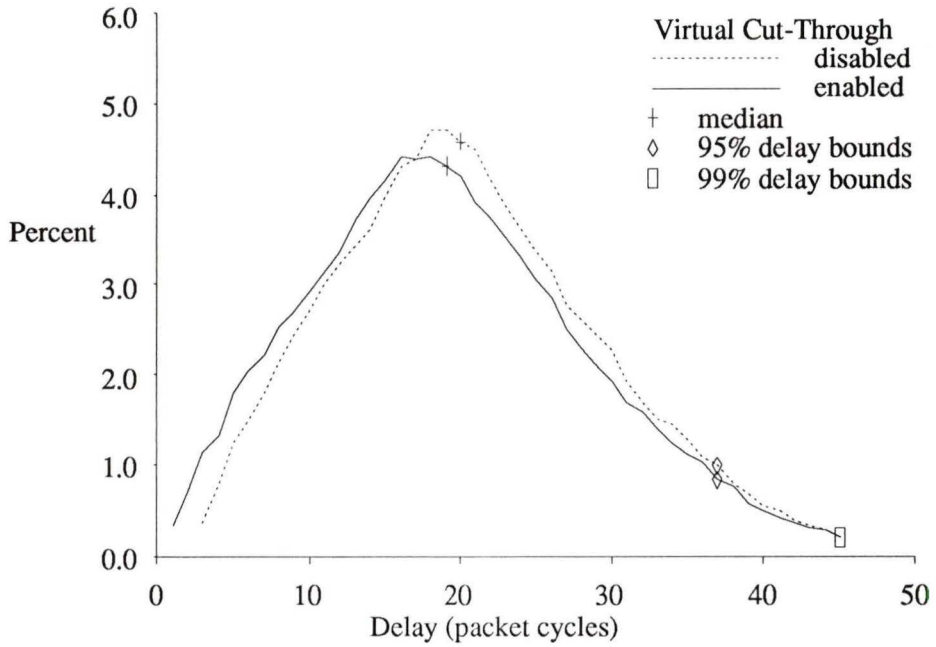


Figure 4.39: Packet delay distribution with and without virtual cut-through (simulated): 8×8 network based on SE2A, 32 slots/SE, offered load 1.0.

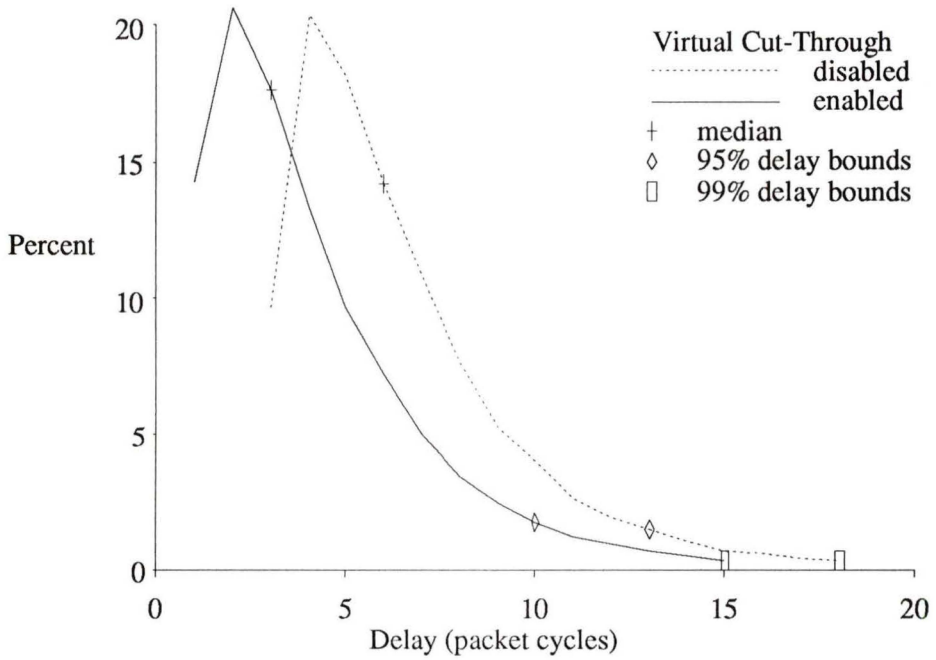


Figure 4.40: Packet delay distribution with and without virtual cut-through (simulated): 8×8 network based on SE2A, 32 slots/SE, offered load 0.8.

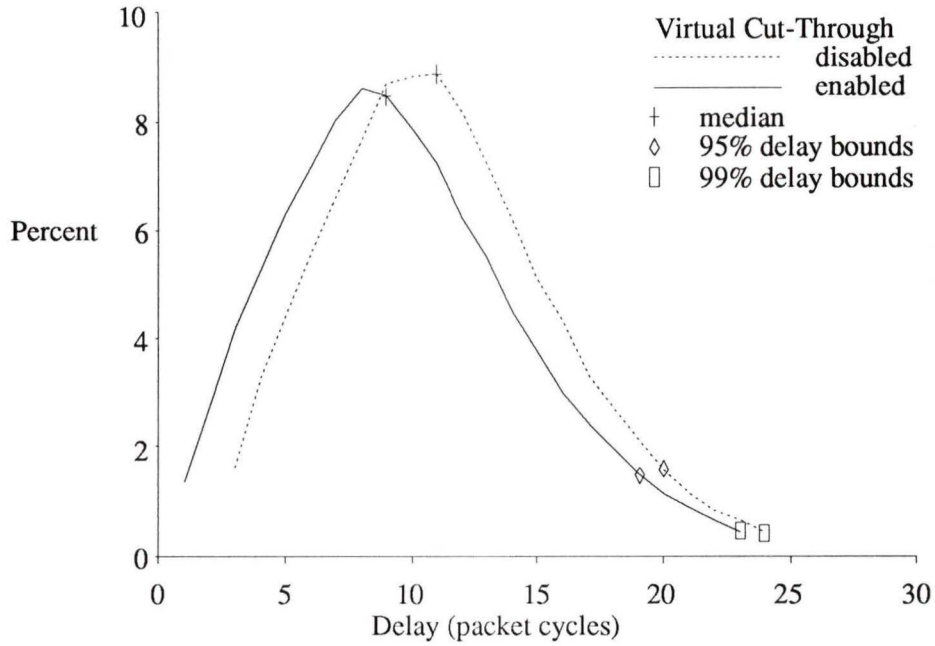


Figure 4.41: Packet delay distribution with and without virtual cut-through (simulated): 8×8 network based on SE2A, 16 slots/SE, offered load 1.0.

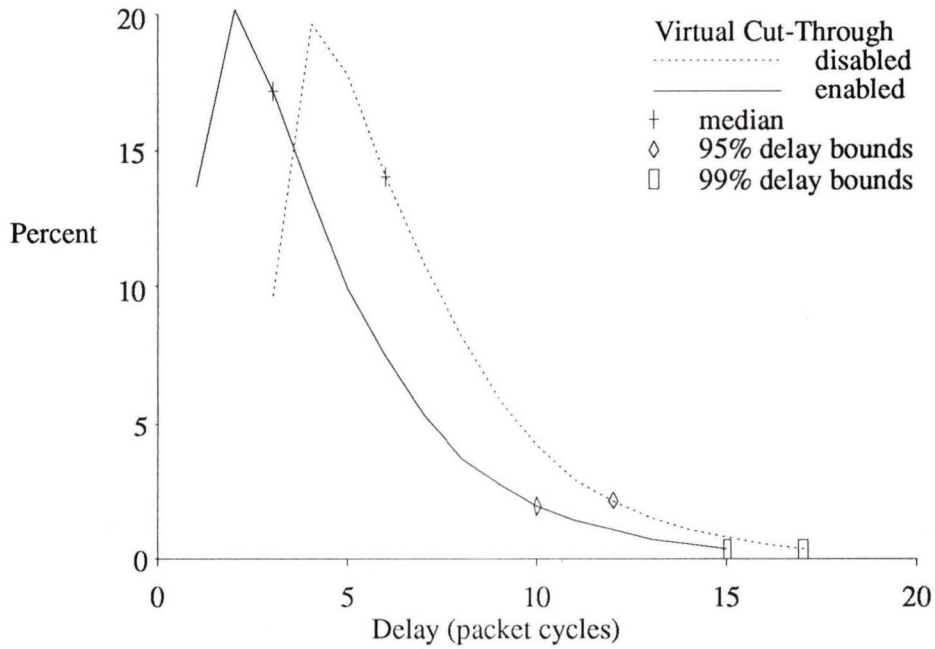


Figure 4.42: Packet delay distribution with and without virtual cut-through (simulated): 8×8 network based on SE2A, 16 slots/SE, offered load 0.8.

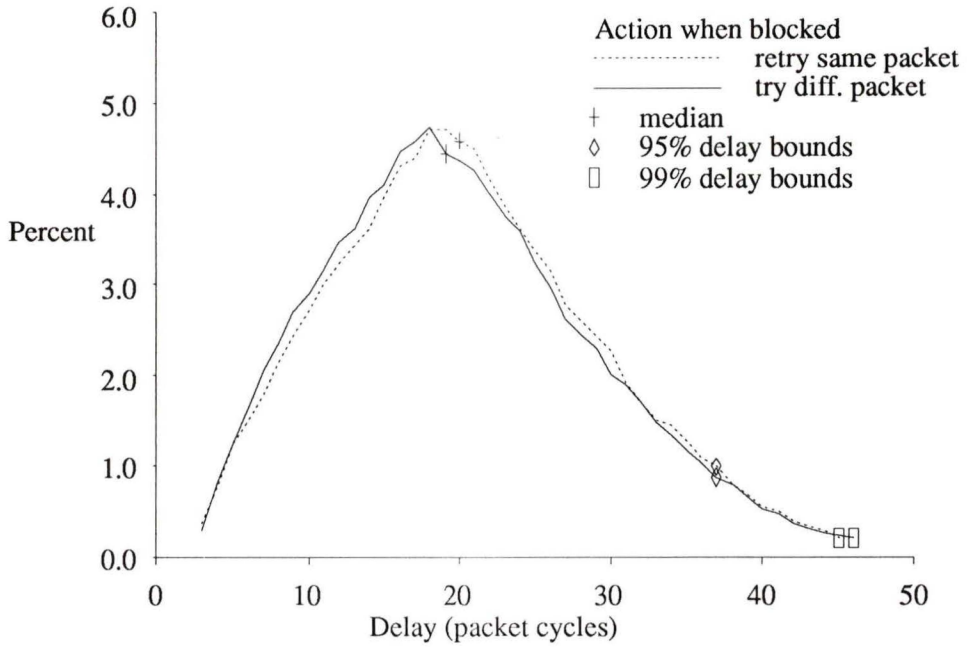


Figure 4.43: Packet delay distribution based on action taken for blocked requests (simulated): 8×8 network based on SE2A, 32 slots/SE, offered load 1.0.

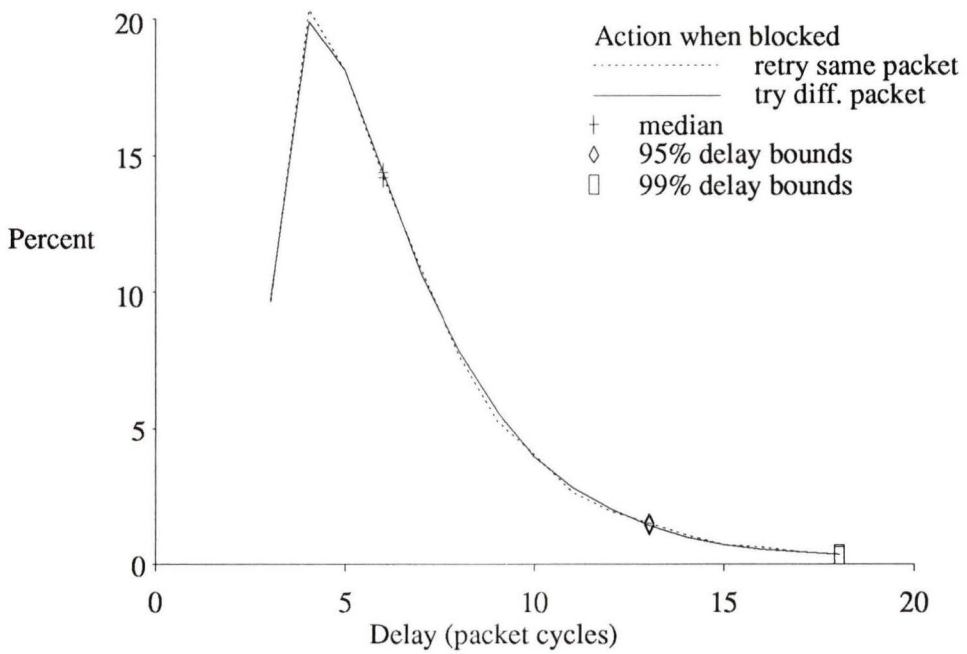


Figure 4.44: Packet delay distribution based on action taken for blocked requests (simulated): 8×8 network based on SE2A, 32 slots/SE, offered load 0.8.

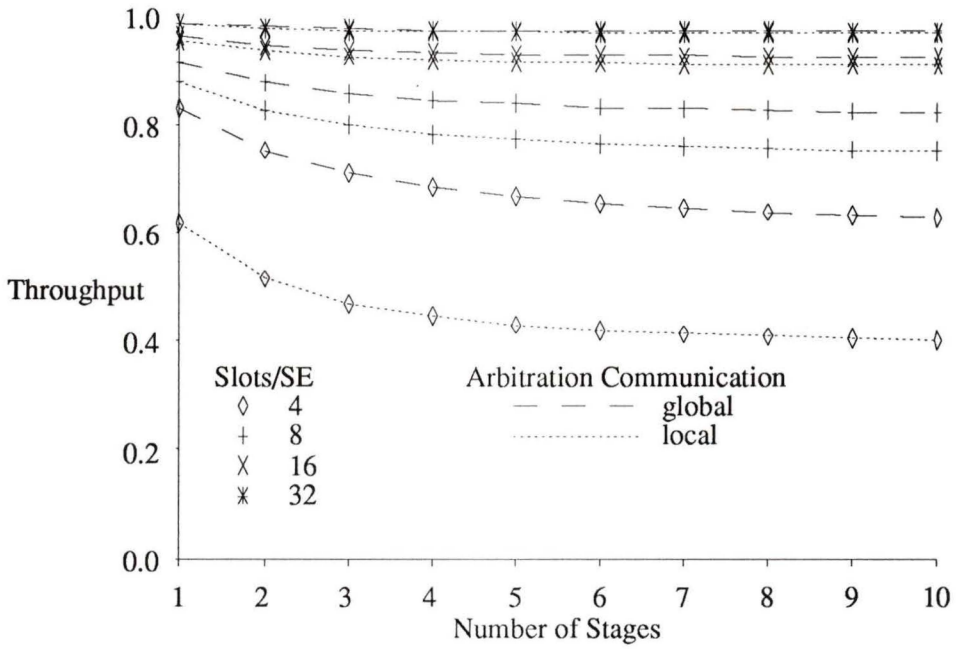


Figure 4.45: Throughput versus Number of Stages for global and local arbitration communication methods: network based on SE2A, offered load 1.0.

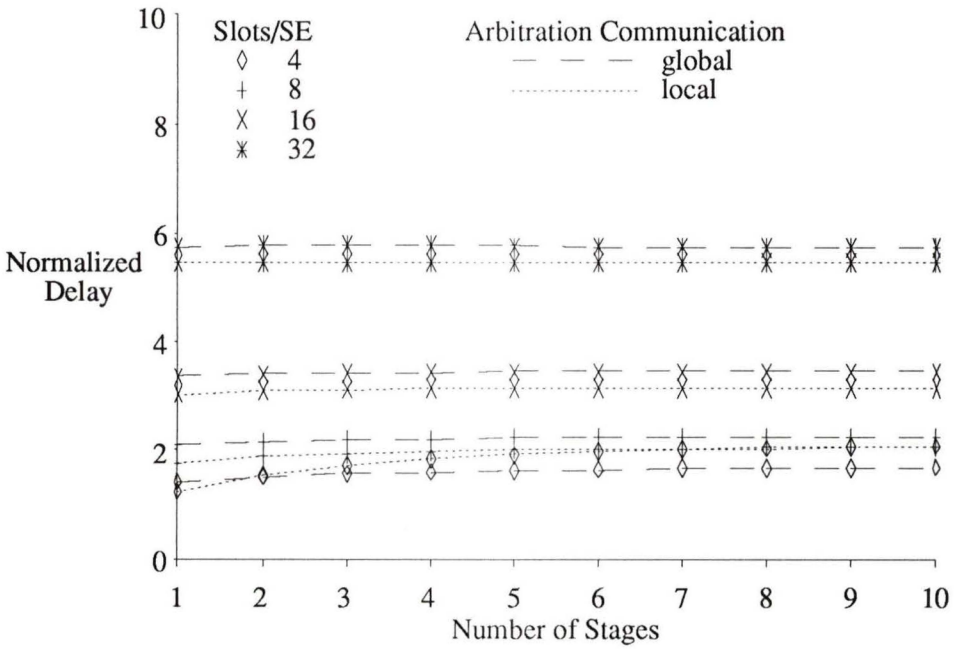


Figure 4.46: Normalized Delay versus Number of Stages for global and local arbitration communication methods: network based on SE2A, offered load 1.0.

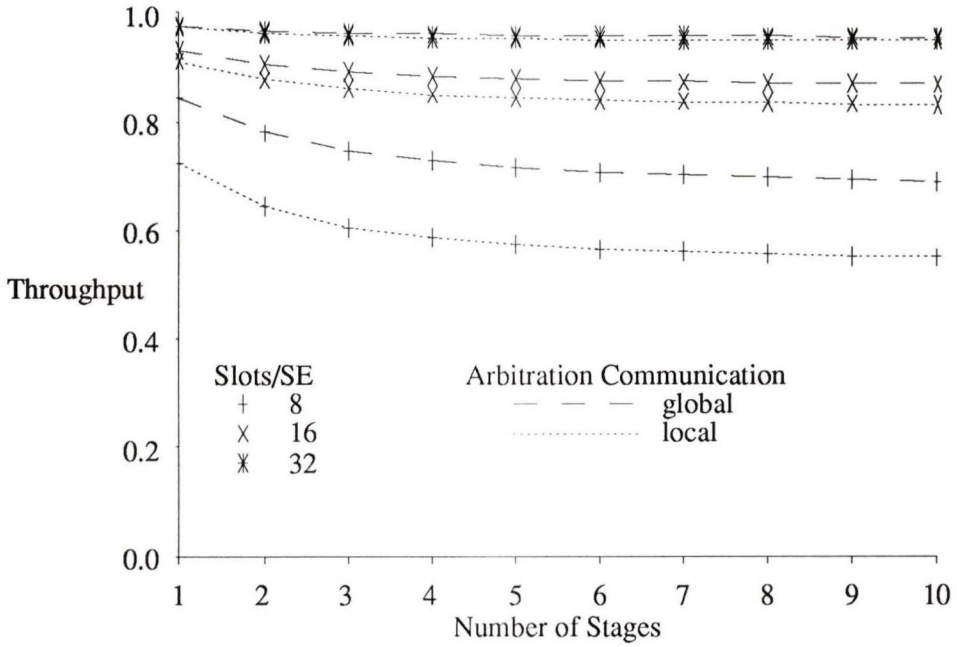


Figure 4.47: Throughput versus Number of Stages for global and local arbitration communication methods: network based on SE2B, offered load 1.0.

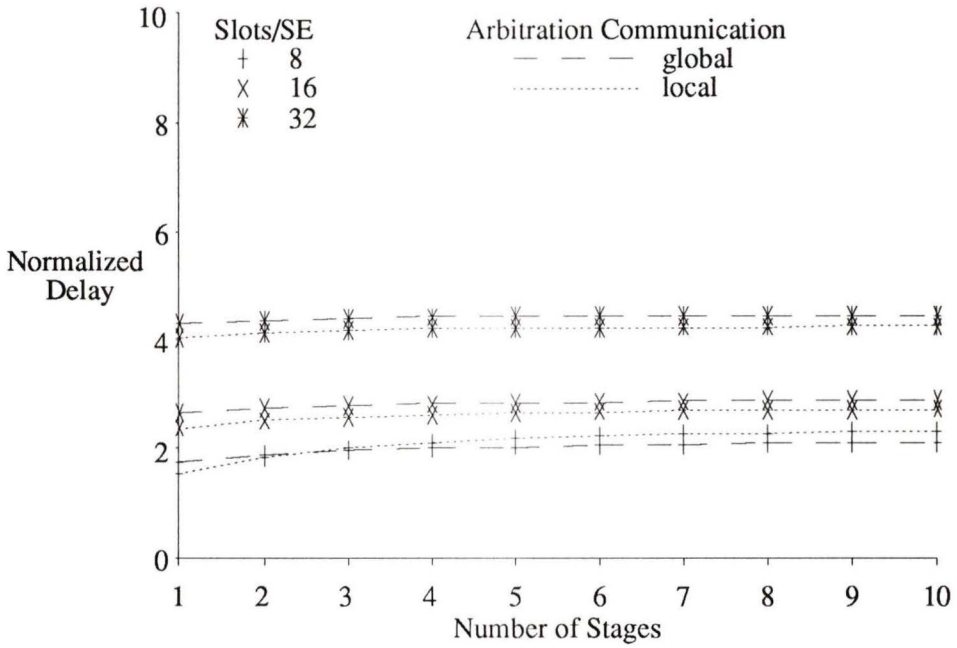


Figure 4.48: Normalized Delay versus Number of Stages for global and local arbitration communication methods: network based on SE2B, offered load 1.0.

4.5 Further Work

The equations developed in this chapter for the analysis of delta networks could be refined in several ways:

1. The equations tend to overestimate the throughput of a network since blocked requests are assumed to be dropped. The equations could be modified to account somewhat for the effect of resubmitting blocked requests. Several papers have addressed this issue by using an *adjusted request rate* [83]–[85].

2. The analysis given in this chapter assumed only *random* arbitration for contention resolution. There are several other possibilities, the two most probable candidates being *rotating* (round robin) arbitration and *buffer occupancy* arbitration. Examples for the rotating arbitration scheme are given in [86].

3. The analysis given in this chapter considered only single links from each port of the SE. The equations could be further developed to consider dilated networks. A d-dilated network is a network in which each link has been replaced by d parallel links [69].

4. Virtual cut-through could be considered. This would require refinement of the equations such that time is relative to a *bus cycle*, the time required to transfer a single byte of a packet between two SE's of the network. A variable would then be introduced giving the number of bytes in a packet.

4.6 Summary

In this chapter, a generalized set of equations was developed for analyzing buffered delta networks. The equations allow analysis based on network size (number of stages), switch size (arbitrary $a \times b$ SE's), buffer size (1 or more slots per buffer), buffer placement (before or inside the crossbar), number of stages of look-ahead routing performed, and arbitration communication method (local or global). For a given input traffic intensity, the equations estimate network throughput and delay performance.

Simulations were performed to confirm the validity of the equations. Deviations between the analysis and the simulations are attributed to the way blocked requests

are handled by each. To simplify the analysis, blocked requests are assumed to be dropped. In a real SE, as modeled by the simulation, blocked requests are resubmitted in the following stage cycle.

SE's with buffers inside the crossbar perform better than those with buffers before the crossbar. Thus, the case of buffers before the crossbar is excluded as an architectural option.

For SE's with buffers inside the crossbar, it was found that 1-stage look-ahead routing can improve the delay performance above an offered load of about 0.8. If it is known that the offered load for a network will not exceed 0.8, then the simpler architecture of an SE without look-ahead routing should be used.

For SE's with buffers inside the crossbar and an offered load greater than 0.8, larger buffers can improve the throughput performance slightly at the expense of greater delays. If the offered load does not exceed 0.8, then the extra buffers offer no performance gains.

Simulations showed minor improvements in throughput and delay performance when dilation was used for high offered loads. However, the added complexity for dilated SE's (more pins per SE and more complex arbitration logic) does not justify the small gains in performance. Furthermore, for greater than 8 slots/SE, there was no performance gain from dilation for offered loads less than 0.8.

It was also shown (by simulation) that the various arbitration schemes performed about the same. An exception was the buffer occupancy arbitration scheme when used with 1-stage look-ahead routing. The buffer occupancy arbitration scheme degenerated in terms of delay performance for an offered load near 1.0. This did not happen when look-ahead routing was not used. Thus, without look-ahead routing, the simplest arbitration scheme to implement should be used. When look-ahead routing is used, either the offered load must be kept below about 0.8, or the buffer occupancy arbitration scheme should not be used.

From simulation, it was found that virtual cut-through provides reasonable improvements in the delay performance for offered loads less than 0.8. As the offered load approaches 1.0, the performance gains from virtual cut-through diminish. Thus, virtual cut-through should be used in the SE design if possible. Also, to improve the flexibility of the network, provisions should be made to disable virtual cut-through.

Simulations showed that there is no advantage to trying different packets in

response to a blocked request. Thus, it is recommended that the selected packet continue to be resubmitted until successfully transmitted rather than go to the bottom of the arbitration priority list.

Comparisons of the performance of the local and global arbitration communication methods revealed very little difference for higher buffer sizes (greater than 8 slots/SE). Thus, the simplest to implement is recommended.

Chapter 5

Switching Element Design

5.1 Introduction

This chapter presents a proposed design for a 2×2 SE to be used in a buffered delta network.

Although a few BISDN architectural models have been used in the literature, they are generally similar. For purposes of this design, the BISDN architecture proposed by the T1S1 Technical Sub-Committee has been used as a reference [87]. Figure 5.1 shows an example of that network architecture. The architecture is hierarchical. Remote Multiplexer Nodes (RMN's) concentrate data from Customer Premises (CP's) Nodes and feed it to Access Nodes (AN's). Several AN's are concentrated into a single Local Exchange Node (LEN) and LEN's are further concentrated into Transit Exchange Nodes (TEN's). Finally, TEN's are connected to other TEN's to achieve wide area network coverage.

Asynchronous Transfer Mode (ATM) is the proposed transfer mode solution for the network. ATM is a packet-oriented transfer mode using an asynchronous time division multiplexing technique. ATM packets, referred to as *cells*, are small and fixed in size. The models for the AN, the LEN, and the TEN all contain an ATM switch. The SE design proposed in this chapter is targetted at these ATM switches. Figure 5.2 shows the model of the LEN as given in [87]. For details on the BISDN architecture or nodal architectures, refer to [87].

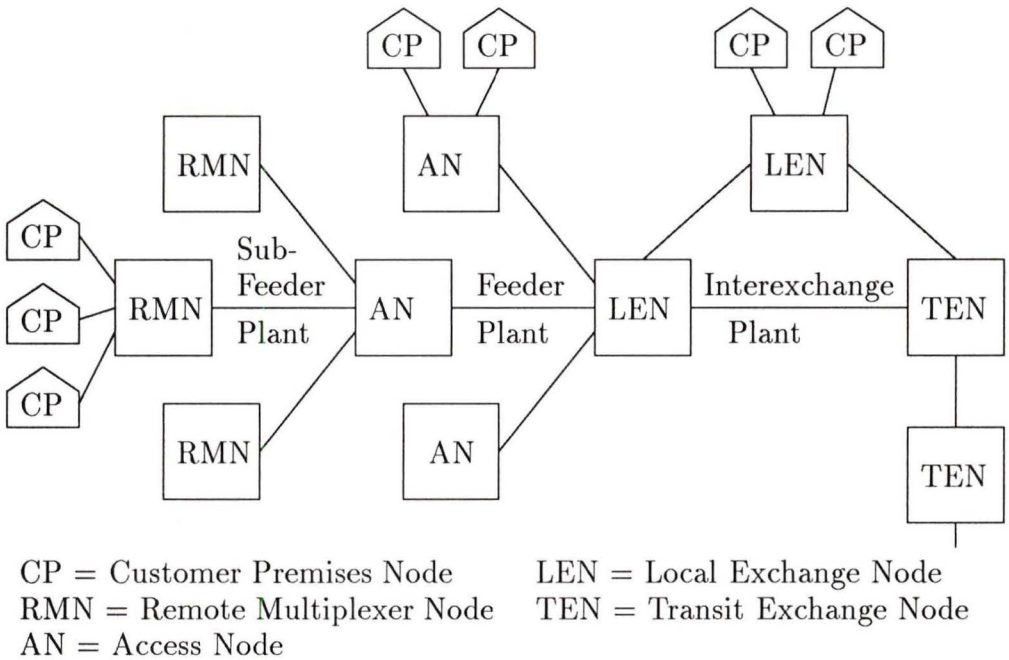


Figure 5.1: Example of BISDN architecture [87].

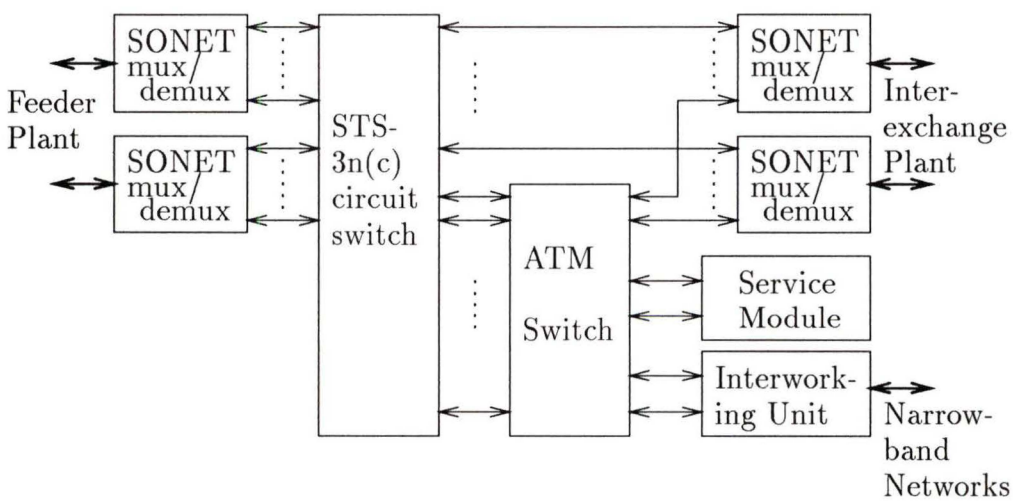


Figure 5.2: Example of local exchange node architecture [87].

5.2 The ATM Switch Model

The ATM switch model used here (Figure 5.3) is similar to those used in [61]–[63]. The switch fabric (SF) used is a multistage interconnection network (MIN). Trunk controllers (TC's) interface the external links to the SF and are responsible for:

1. Protocol conversion between the network format (in this case, ATM) and the SF format.
2. Performing error detection and correction.
3. Performing virtual channel translation.
4. Prefixing the necessary routing tag to the packet to be routed through the SF.

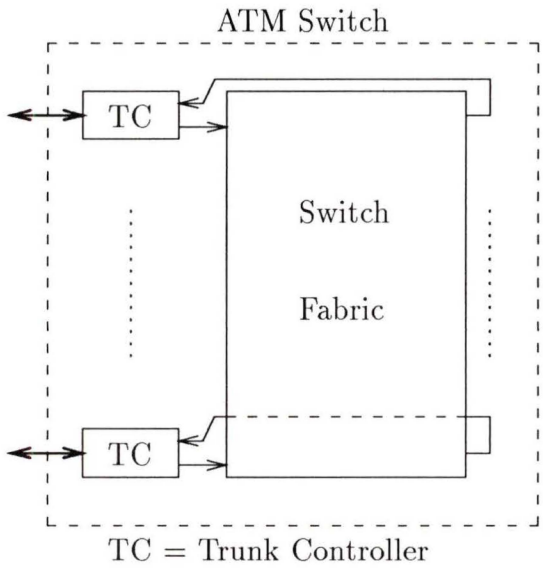


Figure 5.3: ATM switch architecture.

Some degree of network fault tolerance is desirable. Several fault tolerant MIN topologies are available (see [45] for a survey).

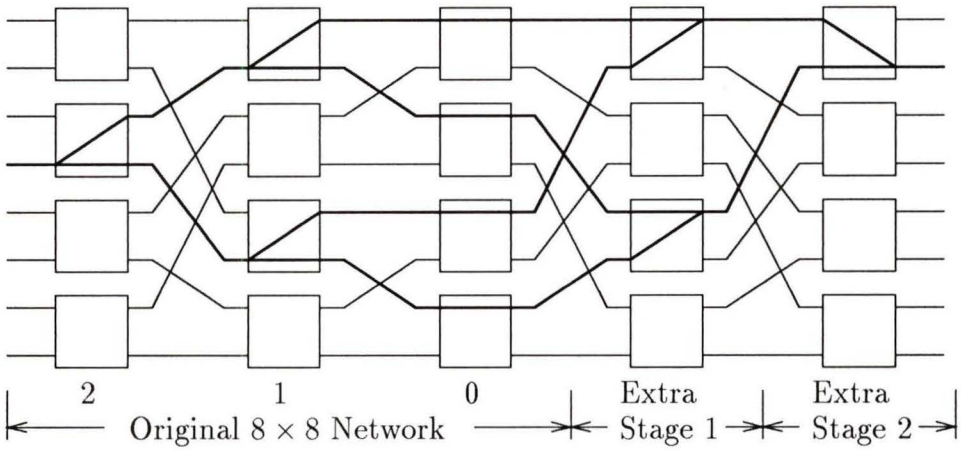


Figure 5.4: 8×8 ADN constructed by adding 2 stages to an 8×8 delta network.

Fault tolerant MIN's offer either static or dynamic rerouting. Under *static rerouting*, the SE's do not route around a faulty SE. A different routing tag must be used to take advantage of extra paths in the network. With *dynamic rerouting*, each SE has a local alternate route for forwarding a packet and having it reach the same destination. By selecting the appropriate path, the SE can avoid a faulty path. The advantage here is that the fault need not be known externally to the MIN, i.e., by the TC's which select the routing tags. A concern, though, is that packets could arrive at the destination TC's out of sequence. This would require resequencing of packets, which would increase the delay through the network. Thus, ideally, some method of avoiding the need to resequence packets in dynamically reroutable MIN's should be found if these MIN's are to be used.

As in [43], the Augmented Delta Network (ADN) will be adopted here as the MIN topology. The ADN is a rectangular delta network with additional stages for fault tolerance. Static rerouting is performed by the TC's in the event of a fault in the MIN. Figure 5.4 shows an 8×8 ADN constructed by adding 2 stages to an 8×8 delta network. The possible paths from one input to one output are shown.

The ADN has at least single-fault-tolerance to SE and link failures in any stage, except the first and last stages which are zero-fault-tolerant to SE failures. A 2-dilated ADN has additional fault tolerance to link failures, but still no fault

tolerance in SE failures in the first or last stages.

Complete system single-fault-tolerance may be achieved by duplicating the system, i.e., having more than one switching plane. This could be done by providing a duplicate plane of TC's and SF. The duplicate could be operated as a hot stand-by, ready to take over in the event of a fault in the operating plane. Off-line in-circuit testing could be performed on the non-operating plane. Alternatively, the two planes could operate in a load sharing fashion, with each one ready to assume the additional load of the other in the event of a failure.

5.3 Routing Tag Specifications

Each packet sent through the SF is prefixed by a header. A header consists of a routing tag identifying the destination, as well as other packet-related control information for the destination TC (priority, testing, etc.).

At each SE, one bit of the routing tag identifies which output port the packet is to be routed to at that particular stage of the SF ("0" for upper output port, "1" for lower output port). Thus, the routing tag requires one bit for each stage of the network. A 14-bit routing tag would be required for a 14-stage, 4096×4096 ADN (i.e., a 12-stage delta network with two extra stages for fault tolerance).

To remove the necessity of the SE knowing which bit to use (which would require that the SE have knowledge of which stage it is in), the least significant bit of the routing tag will always be used. When the SE is through using this bit, it will rotate the routing tag to position the appropriate bit to be used by the successor SE. At the same time, it will replace the most significant bit of the tag by a "0" if the packet arrived on the upper input port or a "1" if it arrived on the lower input port. At the destination, the routing tag will contain the source address. This information may be useful to the TC's.

For reasons discussed in Section 5.4, datapaths throughout the system will be byte-wide. For purposes of the SE design, it is assumed that a two-byte header will meet the requirements. All bits will be treated as though they are routing tag bits. Thus, upon reaching the destination, any bits not used during the routing process (including control information bits) will be located in the least significant bit locations.

An error in the routing tag will result in the packet being routed to the wrong destination, effectively doubling the error rate compared to just losing the packet. Thus, it is important to ensure that packets with routing tags in error are either dropped entirely or have their routing tags corrected before forwarding.

Error correction requires additional logic over that required for error detection. This translates to additional delays through the SE. It is preferable to use error detection only and to discard erroneous packets. An edge protocol can perform error recovery where required [60].

To determine the method of error detection to use, one must consider the types of errors expected along the byte-wide datapath, as well as the effect of the error on the entire header.

Intrachip A single stuck-at-fault (SAF) of a datapath bit will cause a single-bit-error per byte. Similarly, an open datapath line can be modeled by a single SAF error. Such errors would be detectable by an error detection circuit capable of detecting a single-bit-error per header byte.

It is also possible for two datapath lines to be shorted to each other. In this case, when the bits are complementary, the net result will be a voltage somewhere between the power rails (the actual value depends on NMOS and PMOS resistances of the associated drivers). Typically, the PMOS and NMOS devices are ratioed such that their resistances are approximately equal (so that they can charge and discharge capacitive loads in roughly equal times). In such cases, two shorted lines would result in approximately 1/2 V_{DD} appearing on the bus lines. Since both receiving buffers could erroneously read these lines, there is potential for a double-bit error per header byte.

By ratioing the PMOS and NMOS devices of the output drivers, the voltage level during a short can be adjusted. For example, if the PMOS and NMOS device widths and lengths are made equal, the bus line voltage during a short will be roughly 1/3 V_{DD} (mobility of a PMOS device is about half that of an equal-sized NMOS device). Thus, receiving buffers will interpret the shorted lines as both being low. Since only one line is truly low, this results in a single bit error for the byte.

It is not expected that burst errors would originate within a chip.

Interchip As for the intrachip case, a single-bit-error per byte can occur due to a bus line between SE's having a single SAF, being open, or being shorted to another bus line.

It is not expected that burst errors would originate between chips on the same card. Typically, systems are designed such that sources of such errors are not present. One scheme used is to include a ground plane on the circuit board. Additionally, ground lines can be placed between bus lines to further reduce the possibility of crosstalk.

Intercard Single SAF, open datapath bits, and shorted datapath bits would have the same affect as the interchip case.

The potential for burst errors between cards depends on the interconnection scheme used. If the connections are made by cables, for example, burst errors could be generated by moving the cables or by crosstalk from other cables. To reduce the likelihood of this, shielded cables could be used. Alternatively, if a backplane containing a ground plane is used, such errors would not be expected.

In summary, single-bit-error detection on each byte along the datapath will detect the majority of expected errors. For the few instances of burst errors anticipated, an edge protocol handled by the TC's will provide acceptable performance.

The fact that the routing tag consists of two bytes transmitted in succession over the same data bus means that a single SAF can result in a double-bit-error in the routing tag. Furthermore, if the parity is handled by appending a parity byte to the routing tag, then the total bit-error rate would be tripled. In effect, detecting single-bit-errors per byte necessitates detecting triple-bit-errors in a three byte header.

Alternatively, each byte of the header could consist of data and the corresponding parity check bits. Then, a single-bit-error parity scheme would simply require one extra bit to indicate odd (or even) parity. A double-bit-error parity scheme would require additional bits. For example, the (15,11) Hamming Code may be shortened to a (12,8) code in which there are four check bits for every eight data bits. However, this would mean that data buses would have to be twelve bits wide. For a 2×2 SE, this would necessitate an additional sixteen pins, as opposed to just four for the single-bit-error parity scheme.

In light of the above, a simple single-bit-error (per byte) parity scheme will be used. The data bus will be extended from eight bits to nine bits by adding a single parity bit, thus allowing the detection of all odd numbers of bit errors in data bytes. This approach has been adopted for several parallel bus architectures, including IEEE 896, MULTIBUS II, VERSABUS, NUBUS, and DEC's SBI [88]. This scheme will allow the hardware to be simplified and to operate more quickly than if a complex error detection scheme had been implemented.

Double-bit-errors and burst errors affecting headers will cause packets to be routed incorrectly. An edge protocol administered by the TC's can be used to detect these errors. This will mean the addition of checksum bytes to the payload of the packets to implement, for example, a Cyclic Redundancy Check (CRC) scheme. A 16-bit CRC (for example, CRC-16, $x^{16} + x^{15} + x^2 + 1$) will detect all errors with an odd number of bits, all double errors, all 16-bit or shorter error bursts, 99.997% of 17-bit error bursts, and 99.998% of 18-bit and longer error bursts [56]. Thus, two bytes will be allocated in the packet for an optional CRC-16.

5.4 Switching Element Specifications

The telecommunications equipment must remain available during power failures. Thus, there will be occasion to operate from emergency backup battery supplies. For this reason, the target technology for the SE is CMOS, due to its low DC power requirements.

The rates targetted for BISDN vary substantially. They generally fit in three ranges:

- $<$ 100 Mbits/sec ([61], [62], [89])
- 100–250 Mbits/sec ([66], [67], [87], [90])
- $>$ 250 Mbits/sec ([3], [91], [92])

The model used in [87] is based on the Synchronous Optical Network (SONET) STS-3c rate of 155.520 Mbits/sec. This is the rate that will be used here to determine the speed requirements for the SF.

The STS-3c SONET structure consists of 90 bytes overhead and 2340 bytes payload which is transferred in 125 microseconds. In this case, the payload is ATM cells, which are 53 bytes each (5 header and 48 data bytes). Thus, the SONET payload will carry 44 ATM cells.

Each ATM cell will be prefixed by a 2-byte header before entering the SF. Optionally, a 2-byte CRC-16 may be appended to the packet by the TC's. If the CRC-16 is not required, then the TC will append two null bytes. Thus, for the SF, packets will consist of 53 ATM cell bytes, 2 header bytes, and 2 CRC bytes, for a total of 57 bytes. With 57 bytes per packet, the maximum data rate at each input to the SF will be 20.064 Mbytes/sec (57 bytes/packet $\times$ 44 packets in 125 μs). If the restriction of 80% loading of the SF is used (to reduce delays), then each input of the SF must carry 25.08 Mbytes/sec. To attain this data rate with CMOS, it will be necessary to use at least a byte-wide datapath throughout the SF. Wider datapaths could be used to further reduce clock rates at the expense of higher pincounts on the SE's. In this design, a 9-bit datapath will be used—8-bit data plus 1-bit parity (see Section 5.3). This approach was used by Bubenik and Turner [66], [90].

It was found in Chapter 4 that the SE architecture can be simplified in several ways without performance degradation if the offered load will not exceed 0.8. Thus, the network will be designed to take advantage of these architectural simplifications.

In summary, the general architectural specifications for the SE are:

- Buffers inside the switch (i.e., between the switching multiplexers of the SE) with 4 slots/buffer.
- No look-ahead routing.
- Single links (i.e., non-dilated).
- Rotating arbitration scheme with local communication only.
- Virtual cut-through routing, with enable/disable control.
- 9-bit datapaths (8-bit data, 1-bit parity).

- Synchronous CMOS design with minimum clock rate of 25.08 MHz.
- Incorporate design for testability.

5.5 Design Considerations

5.5.1 Clocks

In this design, there will be the requirement for several interdependent finite-state machines (FSM's). The FSM's will use flip-flops to store the state. A single clock, ϕ, will be routed throughout the design, with adjacent FSM's clocked on different edges of the clock. The two edges of the clock will be referred to as ϕ_{lh} and ϕ_{hl} for the low-to-high and high-to-low transitions, respectively.

Clock skew will have the effect of moving the edges of the clock. Proper operation can be maintained provided that the data at the inputs to the flip-flops are still valid (or have become valid, depending on the direction of skew) within one setup time of the clock edge. The low and high durations of the clock must be adjusted to accommodate the worst-case skew.

There will be no gated clocks in this design since this can result in spurious clocking of the flip-flops. Flip-flops will always be clocked on either ϕ_{lh} or ϕ_{hl}. To maintain the present state, the gating signal will multiplex the present flip-flop output back to the flip-flop input.

5.5.2 SE-to-SE protocol

The SE-to-SE protocol involves a 9-bit datapath (8 data bits plus 1 parity bit) and a 2-line control path. The control lines are REQ (Request) and ACK (Acknowledge). Data out of an SE becomes valid on ϕ_{hl} and remains valid until the following ϕ_{hl}. Data into an SE is latched on ϕ_{lh}.

Figure 5.5 shows the handshaking sequence for initiating a packet transfer. A transmitting SE places the first byte of the header (H1) on the databus to the receiving SE and sets REQ true on ϕ_{hl}. If the receiving SE is able to accept the packet, it will set ACK true on ϕ_{lh}, the earliest possible response being on the first ϕ_{lh} following the setting of REQ to true. The transmitting SE will continue to place the first byte of the header on the databus until ACK becomes true. The

example in Figure 5.5 shows the receiving SE unable to accept the packet on the first ϕ_{lh}, but then having room to accept the packet on the second ϕ_{lh}. Once ACK has been set true by the receiving SE, the transmitting SE must place the subsequent bytes of the packet (the remaining header byte and all data bytes) onto the databus on consecutive ϕ_{hl}'s.

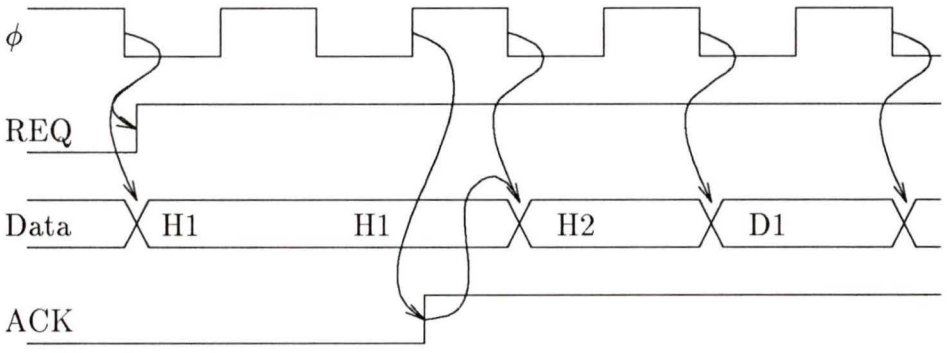


Figure 5.5: SE-to-SE handshaking: initiating a packet transfer.

Figure 5.6 shows the handshaking sequence for completing a packet transfer. As the last byte is placed onto the databus, the transmitting SE also sets REQ false to indicate to the receiver that this is the last byte. The receiver must then set ACK false on the following ϕ_{lh} to indicate to the transmitter that it has seen REQ return to false. The transmitter will check that ACK is returned to the false state before proceeding.

5.5.3 Design for testability

The cost of testing the final product is mainly proportional to the time required to perform the testing. Thus, the design will incorporate features to simplify testing of the final product. This is referred to as *design for testability* (DFT).

Probably the most important concept used in DFT is that of increasing the observability and controllability of the design. *Observability* refers to the ability to determine the states of internal signals from the circuit's outputs. *Controllability* refers the ability to control internal signals from the circuit's inputs. A typical method for providing this is through the use of scan paths.

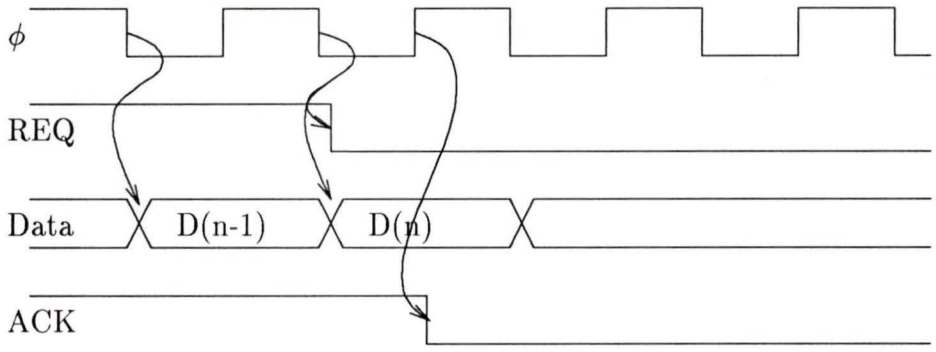


Figure 5.6: SE-to-SE handshaking: completing a packet transfer.

In scan path design, the circuit is designed such that there are two modes of operation: normal and test. In the test mode, all flip-flops are connected as a shift register forming a *scan path.* A test pattern can then be shifted into the shift register (controllability) as the current state of the flip-flops is shifted out (observability). The input and output of the scan path are made available at external circuit pins (usually multiplexed on pins which have a circuit function in normal mode). For further reading and references, refer to Chapter 2 in [93].

All flip-flops in a design could be placed into a single scan path. However, this may result in a rather long test pattern to be shifted in for each test, causing the testing time to increase. Thus, the flip-flops can be grouped into smaller scan paths to reduce the length of the pattern to be scanned into any given scan path. Each scan path can be made available at external circuit pins, as before, by multiplexing the test signals with normal-mode signals. An additional benefit of several scan paths is that certain scan paths may not need to be reloaded for every test, further shortening the test time.

For this design, all FSM flip-flops clocked on ϕ_{lh} (ϕ_{lh}-FSM's) will be placed in one or more scan paths. The number of scan paths used would have to be determined during the low-level circuit design. Similarly, all FSM flip-flops clocked on ϕ_{hl} (ϕ_{hl}-FSM's) will be placed in one or more scan paths which are exclusive of the scan paths used for the ϕ_{lh}-FSM's. The motivation for separating the ϕ_{lh} and ϕ_{hl}-FSM's into disjoint scan paths stems from the fact that the outputs of

ϕ_{lh}-FSM's are inputs to ϕ_{hl}-FSM's and vice versa. The separation will allow the inputs of the ϕ_{hl}-FSM's to be fixed at the beginning of a set of tests (by setting the outputs of the ϕ_{lh}-FSM's). The response of the ϕ_{hl}-FSM's to the fixed inputs could be checked for each ϕ_{hl}-FSM state before changing the inputs. A similar method would then be used to test the ϕ_{lh}-FSM responses.

Not all FSM outputs will be latched (i.e., Mealy FSM's are used). Such signals should have a scan path flip-flop added to provide observability. Furthermore, the signal should be multiplexed with the output of the added flip-flop to provide controllability for that signal.

Counters in the design can be incorporated into a scan path, provided they are short in length. There are two 2-bit counters in this design which could be tested this way. Longer counters would require an impractical number of test patterns for exhaustive testing in a scan path. Such counters could be broken into smaller counters for scan path testing or signature analysis of the count outputs. An alternate method for this design would take advantage of the fact that there are four 6-bit counters. In test mode, the counters would be cycled through the complete counting range while their outputs are compared to each other. Any difference in the outputs would generate an error signal for that count, indicating a faulty counter. An exhaustive test would require only 64 clock pulses.

The buffer memory in this design requires special considerations with respect to DFT. Normally, the buffer memory is embedded within the core of the SE and is not directly accessible from the SE's external pins. For testability, it is recommended that a test mode be included with the capability to multiplex the address and data buses to external pins. This would allow test equipment to apply the test vectors directly to the memory. Alternatively, a built-in self test (BIST) for the memory could be provided. This would greatly simplify the external test vector requirements, as well as allow the circuit to perform in-circuit memory tests. However, since a reasonable memory test consists of non-trivial sequences of read and write operations, it is expected that BIST for the memory would increase the silicon area requirements significantly. For examples of memory tests, refer to Chapter 1 in [93].

The DFT measures discussed above are applicable to testing individual chips out-of-circuit (for example, in a chip tester). Once the chip has been incorporated into a system, additional DFT techniques are required for in-circuit tests.

A method of chip testing which has in-circuit test capabilities is the IEEE Standard 1149.1, developed by the Joint Test Action Group (JTAG) and often referred to as the *boundary-scan technique*. With this method, each chip has a Test Access Port (TAP) consisting of four (or optionally, five) pins. The TAP includes a Test Data Input (TDI) pin and a Test Data Output (TDO) pin for shifting in/out test patterns serially. The internal scan paths can be configured to be accessed by the TDI and TDO. At the board level, the TDO of one chip is connected to the TDI of the next chip. The Trunk Controllers (TC's) could use the TAP to test the chips in-circuit. The standard also provides for a BIST command, which could be used to run the memory tests and any other BIST's included in the design. Reference [94] provides a tutorial-style reference for the IEEE Standard 1149.1.

Finally, schemes have been presented in the literature for performing on-line testing of MIN's. Mourad and Özden [95] presented a method for on-line detecting and diagnosing of faults in rectangular SW-banyan networks. Since the ADN is topologically equivalent to the rectangular SW-banyan [37], these methods can be applied to the ADN. These tests are aimed at testing the datapaths, as well as the routing and conflict resolution capabilities of the SE's. An appropriate SE model, differing from that used in [95], would have to be used to develop such tests for the SE's used here. Although this method can be used to easily check the integrity of the datapaths and routing logic, it would be difficult to use it to test such features as parity-error handling, buffer full status, or buffer empty status. Furthermore, it would not be possible to test the fault-detection error handling of the FSM's. Thus, a compromise may be to use such tests while on-line to check for datapath and routing problems only. While off-line, the TAP could be used for more thorough testing. By having a duplicate SF, one plane can maintain normal operation while the other is taken off-line for these tests.

5.5.4 Fault detection

The fact that several FSM's are interconnected in the SE can be used to provide a level of fault detection. Each FSM provides an error signal which is set TRUE when any of its inputs are inappropriate in the context of the other inputs or the current state of that FSM. Every FSM which has a special error state will also provide a transition path back to a legal state with the hope that processing can

resume. This will be especially useful when the fault is of a transient nature.

All error signals could be latched into an error-holding register which is only cleared by a system reset or under special commands while in test mode. If any bit of the register is set, a single error output signal could be set. This could be used to identify SE's which have detected errors or to switch to an alternate switching plane. In addition, the contents of the register could be made available as part of a scan path. Before the system is reset, it would be switched to an off-line test mode and the contents of all registers in the scan path shifted out. This would allow analysis of the types of errors detected by the SE.

The SE will also detect parity errors on received bytes. These errors could be counted in a register and made available on the scan path. A large number of errors at any given SE would suggest datapath problems between that SE and its predecessor.

5.6 Functions and Blocks Required

The top-level block diagram for the SE is shown in Figure 5.7.

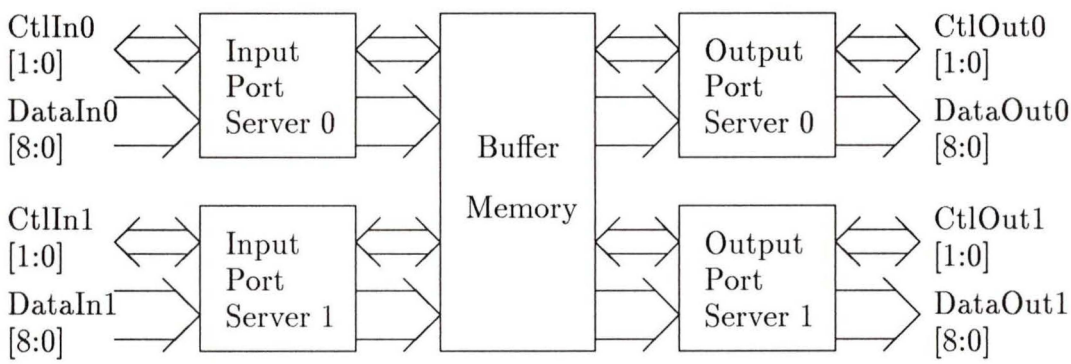


Figure 5.7: Top-level block diagram for SE.

There are three types of modules at the top level of the hierarchy:

- Input Port Servers (2 instances) interface to and provide handshaking with the predecessor SE's, controlling the acceptance of packets and routing them to the appropriate FIFO's within the SE.

- Buffer Memory contains FIFO's for storage of arriving packets.
- Output Port Servers (2 instances) select packets from the buffer memory and transfer them to the successor SE's.

In the presentation of the design, modules sensitive to the low-to-high transition of ϕ will have the label ϕ_{lh} at the clock input. Similarly, modules sensitive to the high-to-low transition of ϕ will have the label ϕ_{hl} at the clock input.

5.6.1 Input port server

Figure 5.8 shows the block diagram for an input port server (IPS). It consists of an input port controller (IPC), a write address counter, and datapath logic.

The IPC is responsible for:

- Handshaking with the predecessor SE (REQ, ACK).
- Monitoring parity for all received bytes. Datapaths throughout the SE are nine bits wide to accommodate eight data bits and one parity bit. For headers which have parity errors, the packet will be discarded. For the data segment of the packet, parity errors will be logged and the parity corrected. The parity error logging may aid in locating the source of a fault.
- Controlling the datapath to cause rotation of the routing tag by one bit while maintaining correct parity.
- Accepting packets and routing to an appropriate FIFO, provided that the FIFO is not full.
- Providing write enable signals to the buffer memory.
- Controlling the write address counter so as to provide byte-within-slot addressing to the buffer memory for packet transfers.

The write address counter is held at zero when CLR is true and counts by one when CLR is false. It provides an indication that the last byte of a slot is about to be addressed. In effect, if there are 57 bytes to transfer, then the allowed addresses are 0 through 56 and the counter provides the LAST signal when the count is 55. This is used to verify that the correct number of bytes were transferred.

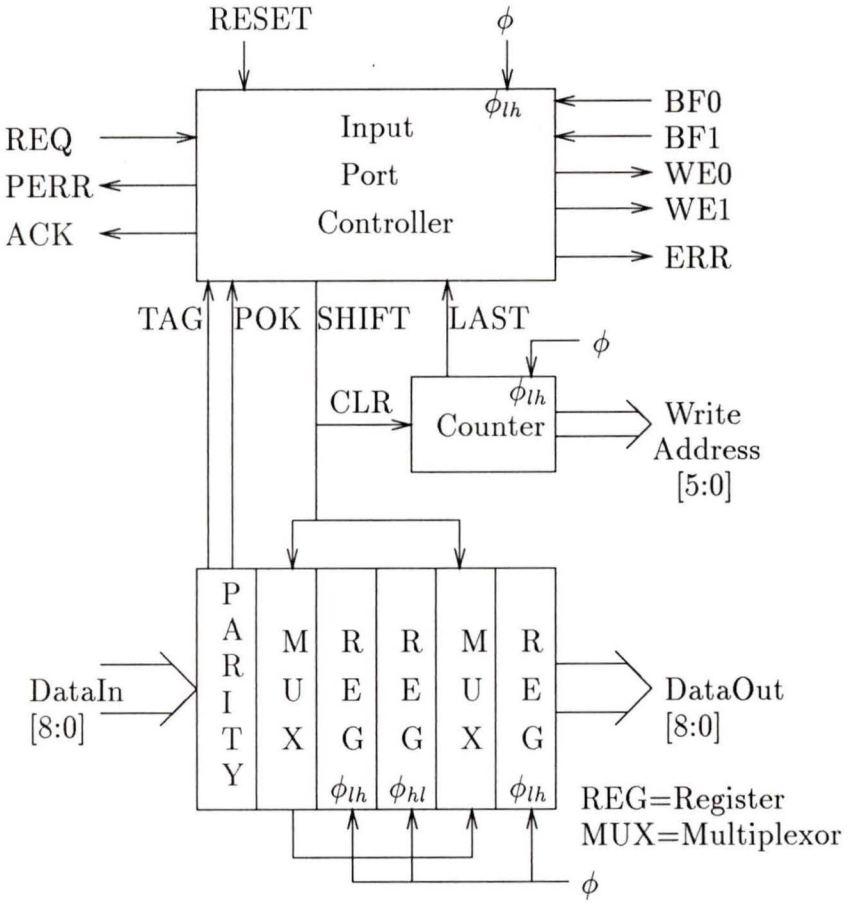


Figure 5.8: Block diagram of input port server.

The IPS datapath is detailed in Figure 5.9. Odd parity is generated for the arriving data and compared with the arriving parity bit. If the parities are the same, the parity okay (POK) signal is true.

While processing the header bytes, the SHIFT signal selects the bit-shifted version of the header bytes. The arriving port address (0 or 1) replaces bit b_7 of the header byte in stage 1. In stage 2, b_7 is replaced by b_0 of stage 1. Thus, after two ϕ_{lh} clock edges, the outputs of the two stages will hold a shifted version of the complete two-byte header with the arriving port address indicated in bit b_{15} (b_7 of the second byte). The adjust parity blocks maintain proper parity during the shift operations.

For the data portion of the packet, the SHIFT signal selects the straight-through path. The parity generator continues to correct bad parity and provide the POK signal.

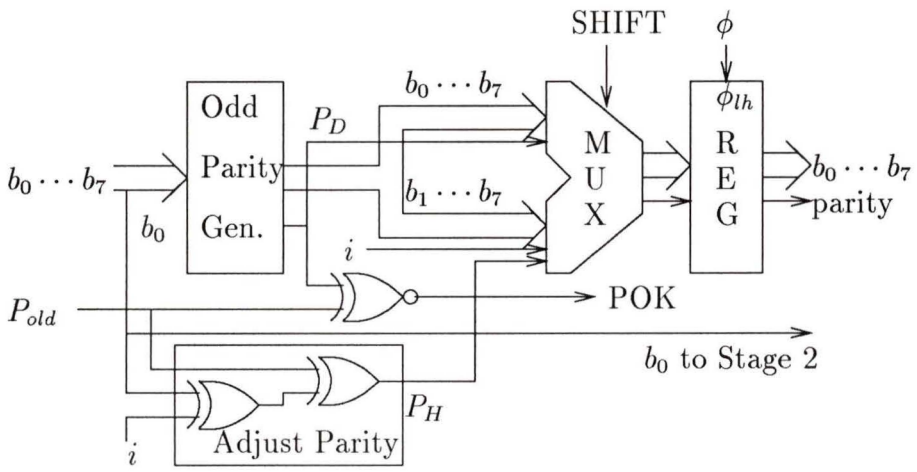
Figure 5.10 shows a state diagram for the input port controller (Mealy-type FSM). The outputs, with the exception of SHIFT, must be latched in order to provide stable signals for the FSM's controlled by the IPC. SHIFT must be valid at the datapath multiplexors prior to the FSM's clock edge, so it is not latched. However, a scan path flip-flop would be provided to allow observability and controllability of the SHIFT signal.

From any state, a RESET will cause the next state to be INIT, which initializes the output signals. When the RESET condition is removed, the next state will be the IDLE state.

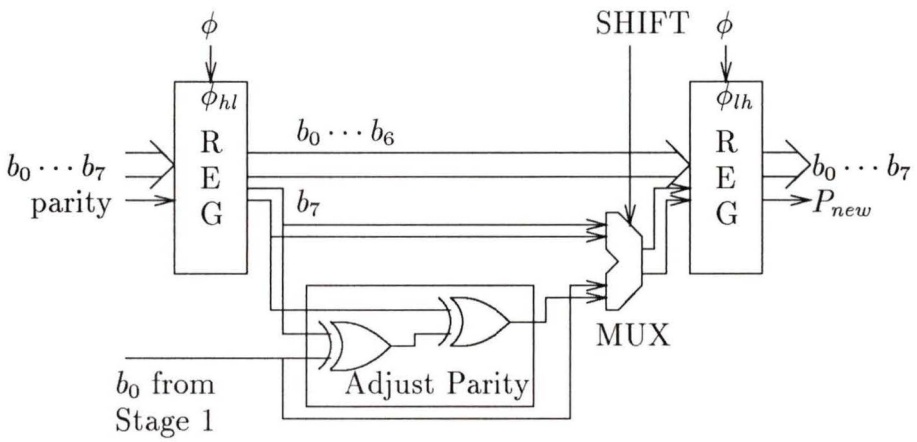
In the IDLE state, the IPC waits for a REQ from the predecessor SE. When REQ is true, the first header byte is available. If the parity is incorrect ($\overline{\text{POK}}$), the SE accepts the packet (ACK) but does not store it. This is done by giving a Parity Error (PERR) signal and moving to state DROP. A return to IDLE occurs when REQ becomes false.

If the parity is correct for the first header byte, then acceptance depends on there being room in the required FIFO. The TAG bit ($x = 0$ or $x = 1$) identifies the required FIFO. If the FIFO has room (as indicated by the appropriate buffer full signal), then the packet is to be accepted and stored. Thus, the next state is BYTEx.

At state BYTEx, the second header byte is arriving. If the parity is incorrect,



Stage 1

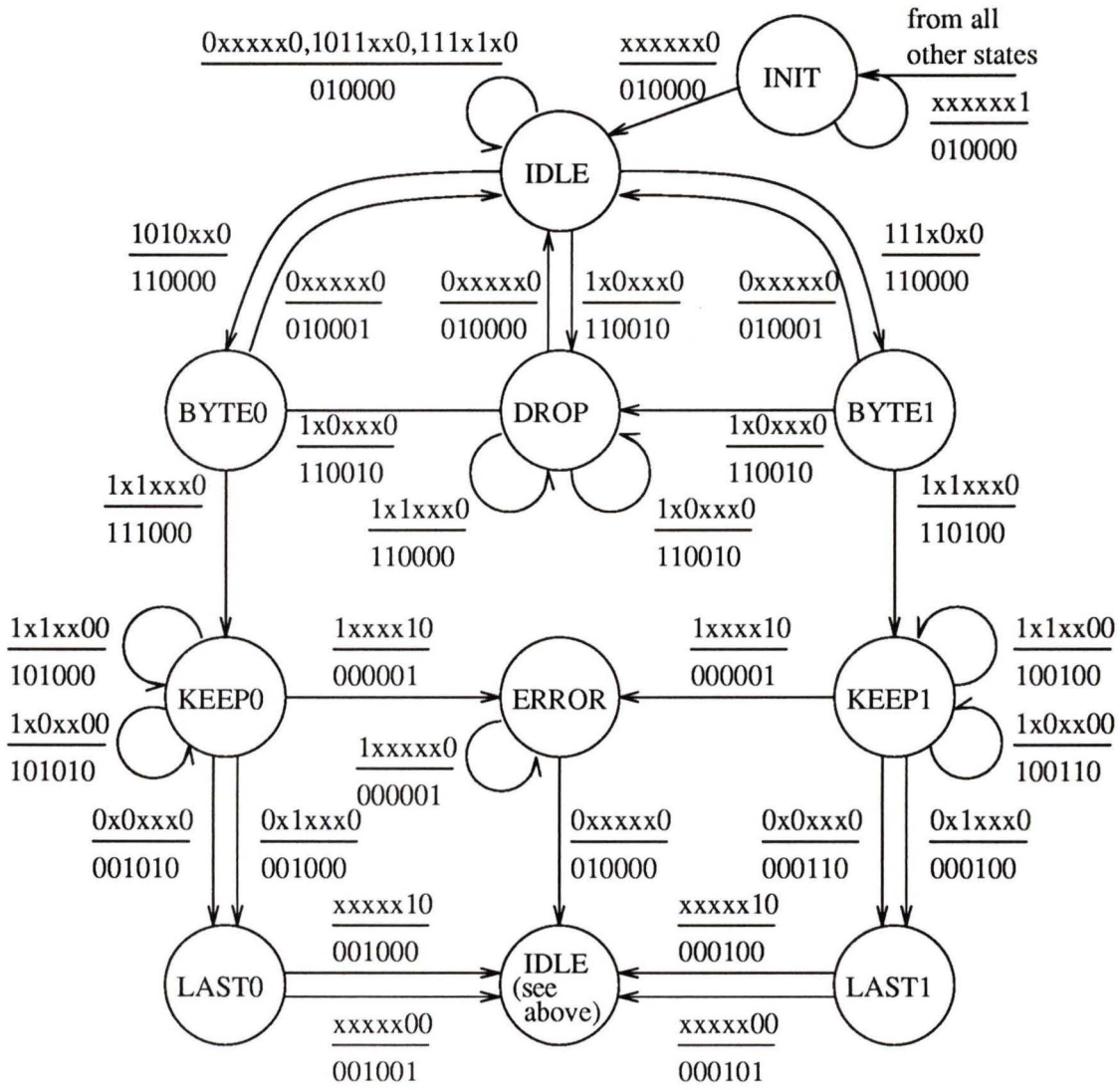


Stage 2

i = port number
P_D = parity for data bytes
P_H = parity for header bytes

REG=Register
MUX=Multiplexor

Figure 5.9: Block diagram detailing the IPS datapath.



Inputs: REQ,TAG,POK,BF0,BF1,LAST,RESET

Outputs: ACK,SHIFT,WE0,WE1,PERR,ERR

Figure 5.10: State diagram for the input port controller.

then the packet must be dropped. PERR is given and the next state is DROP. If the parity is correct, then the packet is to be stored. The appropriate write enable signal, WEx, is given and the next state is KEEPx. For the remainder of the packet, data bytes with parity errors cause PERR to be given.

As the packet is being stored, LAST is monitored. This signal should not become true while REQ is still true, since this would indicate overflow of the slot within the FIFO. If this happens, the next state becomes ERROR. In the ERROR state, the IPC stops storing the packet, sets an error flag (ERR), and simply waits for REQ to become false before returning to the IDLE state. This error may indicate a fault exists in the predecessor SE's output stage or within this SE's input stage.

Provided that the LAST signal has remained false while in state KEEPx, a change of REQ to false indicates the last byte of the packet is on the bus from the predecessor SE. This is acknowledged by returning ACK to false and changing states to LASTx. From LASTx, the last byte is stored in the FIFO. The LAST signal should now be true here. If it is not true, then an underflow of the slot in the FIFO has occurred and the ERR flag is set. The next state is IDLE.

5.6.2 Buffer memory

The buffer memory, shown in Figure 5.11, consists of four instances of a FIFO memory (FM) module. IPS0 routes packets to either FM1 or FM3 (IPS1 to either FM2 or FM4) based on a single bit of the routing tag. OPS0 will select packets from FM1 and FM2 (OPS1 from FM3 and FM4). The write enable signals (WE0, WE1) allow each IPS to write to the specific FM's. Similarly, the read enable signals (RE0, RE1) allow each OPS to select the individual FM's.

Figure 5.12 shows the block diagram for an FM. It consists of a FIFO controller (FC) and a dual port memory (DPM).

The FC performs the following functions:

- Manage front and rear queue pointers (FPTR, RPTR).
- Use the virtual cut-through flag (VCT) to determine when to signal to the OPS that a packet is ready (PR) to be transmitted.

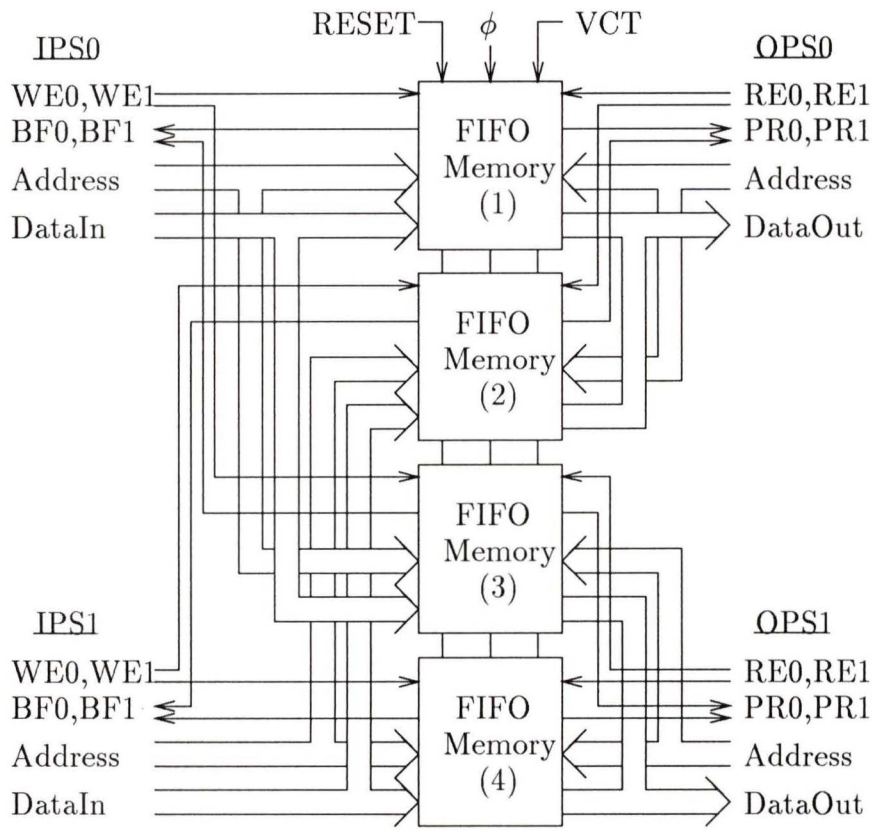


Figure 5.11: Block diagram of buffer memory.

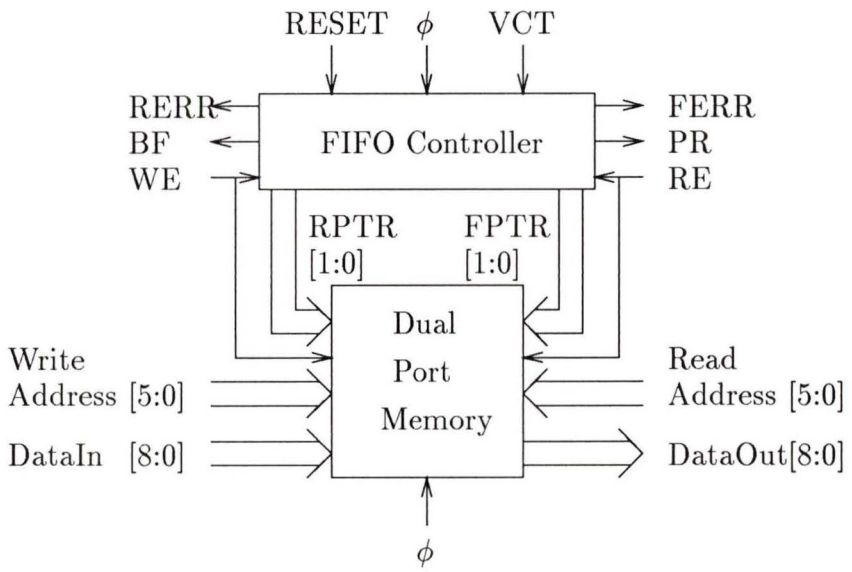


Figure 5.12: Block diagram of FIFO.

- Provide buffer full (BF) status to the IPS.

The FC will expect WE to be active for the duration that the IPS is storing a packet. When the WE signal returns to the inactive state, the rear pointer will be updated. The same is true for the RE signal from the OPS, except that the front pointer is updated instead. Figures 5.13 and 5.14 show the timing relationships for a 2-byte transfer (a 2-byte transfer is used for illustrative purposes only—in the actual design, it would be a 57-byte transfer). For the write operation, ϕ_{hl} is used to latch the data. For the read operation, data are valid on ϕ_{hl}. The DPM, which is 256 by 9 bits in size, has separate address and data buses for the read and write operations. Thus, read and write operations may occur simultaneously. It is not necessary that the DPM be able to read and write the same memory location simultaneously, since this condition cannot occur.

Initially, the front and rear FIFO pointers (FPTR and RPTR) are set to zero, buffer empty (BE) is set to true, and buffer full (BF) is set to false. The FPTR points to the next slot which is to be read from the FIFO. The RPTR points to the next available slot for writing to the FIFO. The algorithms for managing the

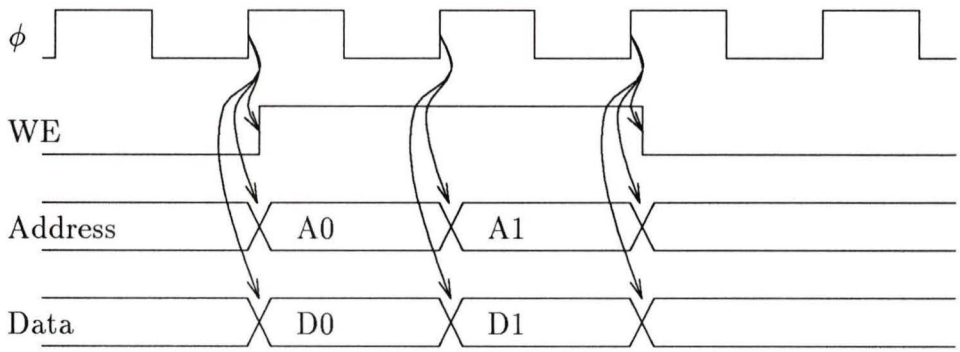


Figure 5.13: Timing diagram for the write operation.

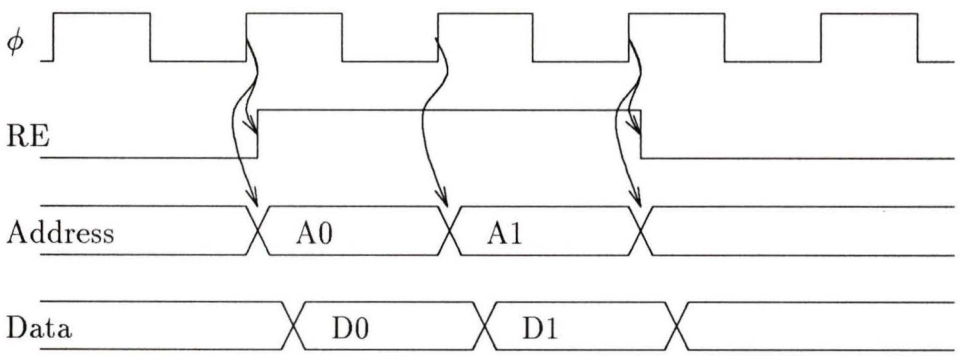


Figure 5.14: Timing diagram for the read operation.

```
if WE then
    if BF then
        {overflow error}
    else
        NEXT_RPTR ← (CURRENT_RPTR + 1) mod NUM_SLOTS
        if VCT then
            BE ← false {inform OPS immediately that packet is available}
        fi
        if NEXT_RPTR = CURRENT_FPTR then
            BF ← true
        fi
        while WE do
            {wait for end of write operation}
        od
        CURRENT_RPTR ← NEXT_RPTR {update RPTR}
        BE ← false {in case not done above}
    fi
fi
```

Figure 5.15: Algorithm for managing the FIFO pointers during write operations.

FIFO pointers are given in Figures 5.15 and 5.16.

It should be noted that if an attempt is made to simultaneously set and reset either the BF or BE flags, the reset operation overrides the set operation in order to achieve correct operation.

The FC would be implemented as shown in Figure 5.17. There are two FSM's: one to implement the write algorithm and one to implement the read algorithm. The FSMs' outputs control the counters which represent the values for the two FIFO pointers, and also control the BF and BE flags which are held by two synchronous RS flip-flops (RSFF's). These RSFF's can be implemented using D-type flip-flops with the reset and clear signals controlling the D-input. Finally, the FSM's provide indication of error conditions (FERR, RERR) which can be used to identify a fault within the SE. RERR indicates an attempt was made to write to a full buffer. FERR indicates an attempt was made to read from an empty buffer.

Since there are four slots, the pointer registers are implemented with 2-bit

```
if RE then
    if BE then
        {underflow error}
    else
        NEXT_FPTR ← (CURRENT_FPTR + 1) mod NUM_SLOTS
        while RE do
            {wait for end of read operation}
        od
        if NEXT_FPTR = CURRENT_RPTR then
            BE ← true
        fi
        CURRENT_FPTR ← NEXT_FPTR {update FPTR}
        BF ← false
    fi
fi
```

Figure 5.16: Algorithm for managing the FIFO pointers during read operations.

counters. The counters change state only on ϕ_{hl}. The clear signals (FCLR, RCLR) are used to set the count to zero. The increment signals (FINC, RINC) are used to advance the count by one. The compare module simply provides indication of when the next RPTR equals the current FPTR (NREF) and when the next FPTR equals the current RPTR (NFER). The FSM's use this information to set the BE and BF flags.

The state diagrams for the Mealy FSM's are shown in Figures 5.18 and 5.19.

The RPTR and FPTR FSM's are similar in flow. A RESET will cause the next state to be INIT. When RESET becomes false and WE (RE) is false, the next state is IDLE. Upon reception of WE (RE), the next state is WRITE (READ) with the necessary outputs to implement the write (read) algorithms given previously. From the WRITE (READ) state, a return to false of the WE (RE) signal will cause the next state to be IDLE. Refer to the state diagrams for details of the output signal values for all state transitions.

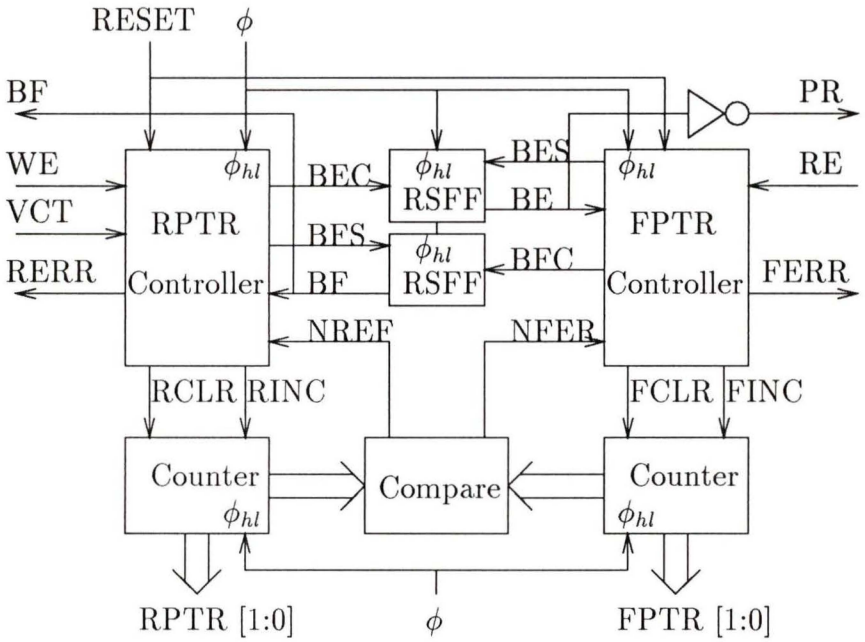


Figure 5.17: Block diagram of FIFO controller.

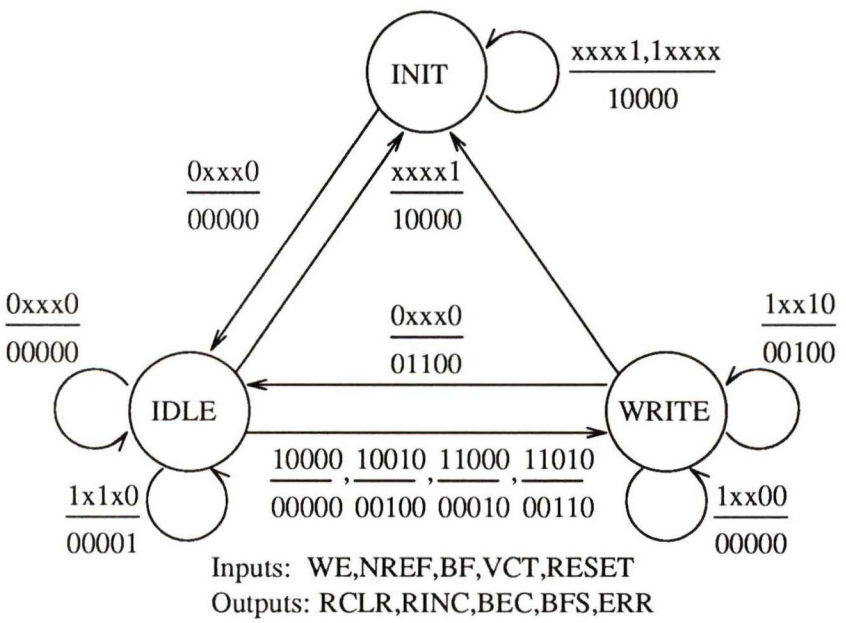


Figure 5.18: State diagram for the RPTR controller.

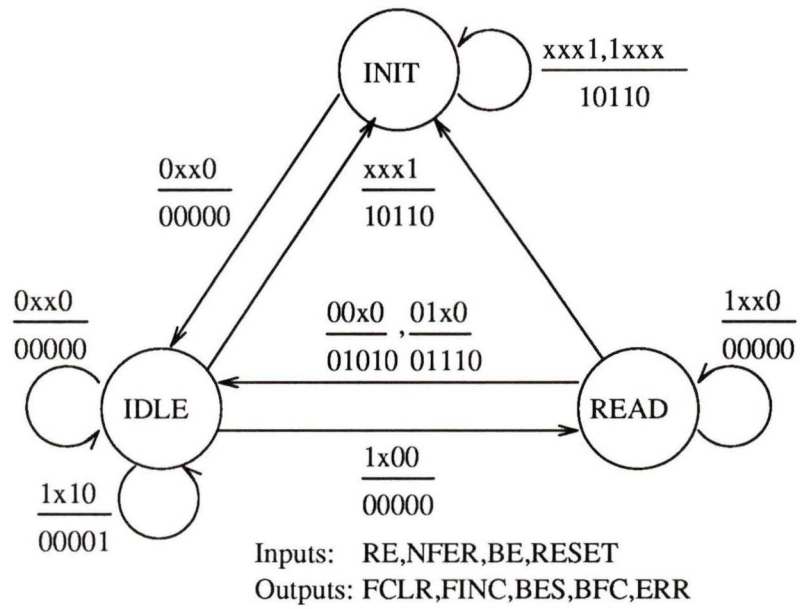


Figure 5.19: State diagram for the FPTR controller.

5.6.3 Output port server

Figure 5.20 shows the block diagram for an output port server (OPS). It consists of an arbiter, an output port controller (OPC), a read address counter, and datapath logic.

The arbiter is responsible for:

- Recognizing when there are packets to send (PR0, PR1) and arbitrating among competing requests.
- Providing read enable signals (RE0, RE1) to the buffer memory, as well as a START signal to the OPC.
- Recognizing a DONE signal from the OPC, indicating that the selected packet was transferred.

The OPC is responsible for:

- Controlling the read address counter so as to provide byte-within-slot addressing to the buffer memory for packet transfers.

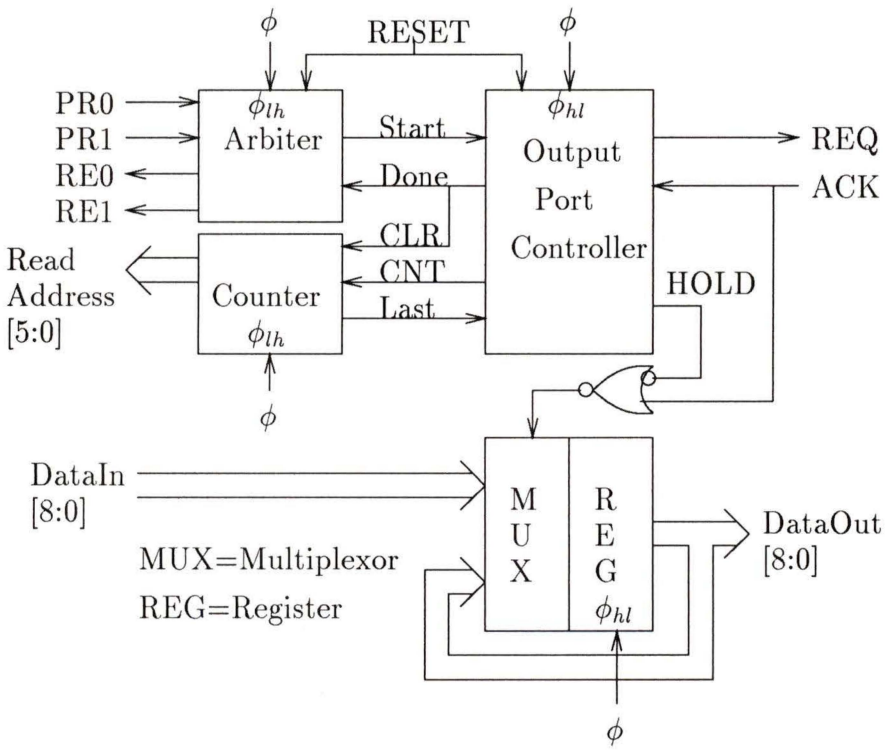


Figure 5.20: Block diagram of output port server.

- Starting protocol upon START signal from arbiter and providing a DONE signal to the arbiter to indicate completion of transfer.
- Handshaking with the successor SE (REQ, ACK).
- Controlling the datapath to cause the first byte of the header to be output until acknowledged by the successor and then to allow subsequent bytes of the packet to be output (HOLD).

The read address counter provides byte-within-slot addressing to the buffer memory. This counter is held at zero when CLR is true. When CLR is false, the counter holds the count if CNT is false and counts by one if CNT is true. It provides an indication that the last byte of a slot is being addressed. In effect, if there are 57 bytes to transfer, then the allowed addresses are 0 through 56 and the counter provides the LAST signal when the count is 56. This is used to terminate the transfer of the packet.

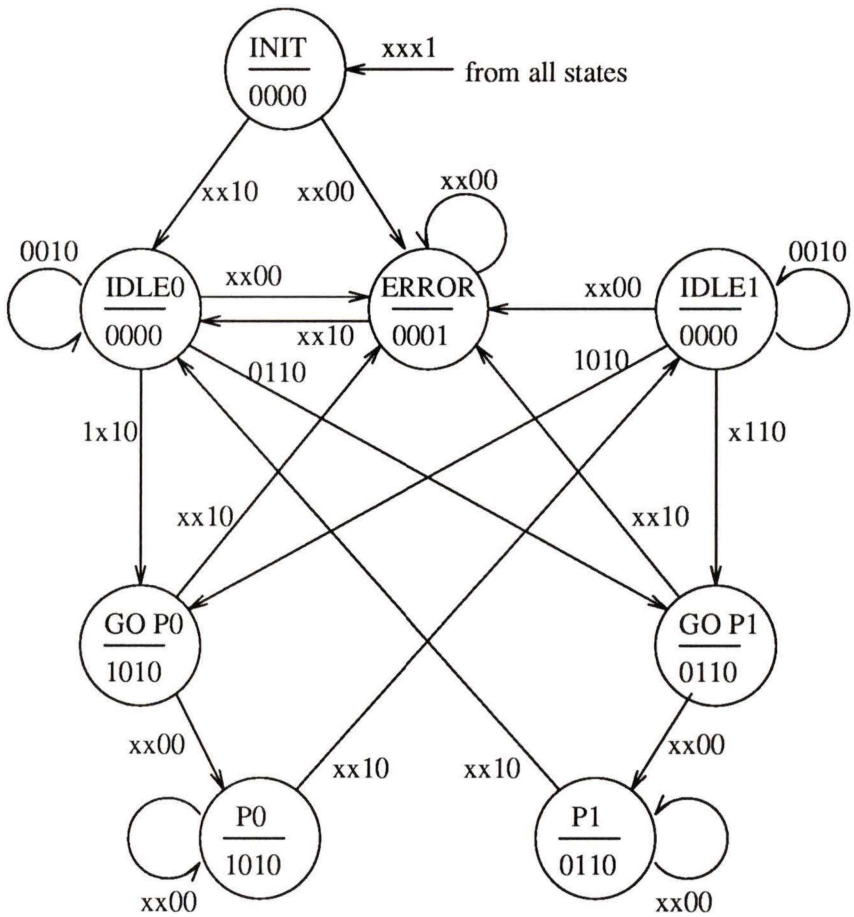
Figure 5.21 shows the state diagram for the Moore-type arbiter.

A RESET will cause the next state to be INIT. Once RESET returns to false, the next state is IDLE0, with one exception. For states INIT, IDLE0, and IDLE1, if DONE becomes false, the next state is ERROR, since DONE cannot be false before the arbitration sequence starts. This transition is included for fault detection. From the ERROR state, DONE must become true before the arbiter will return to IDLE0.

For this description, x and y refer to the two FIFO's, $0 \leq (x, y) \leq 1$. FIFO priority is encoded by the idle state. Assuming that FIFO x has priority over FIFO y, the state would be IDLEx. An arbitration is performed when either packet ready signal (PRx or PRy) is true. The transitions from IDLEx are such that if PRx is true, the next state is GOPx, regardless of the value of PRy. If PRx is false, then the next state is GOPy if PRy is true.

GOPx (GOPy) sets the START signal along with the appropriate read enable signals. The output port controller must respond by setting DONE to false. Failure of the DONE signal to become false results in the next state being ERROR, providing fault detection. If the DONE signal does become false, then the next state is Px (Py).

In state Px (Py), the arbiter waits for the DONE signal to become true. Once



Inputs: PR0,PR1,DONE,RESET

Outputs: RE0,RE1,START,ERR

Figure 5.21: State diagram for the arbiter.

true, the next idle state will give the higher priority to the FIFO which was not just served (Px $\rightarrow$ IDLEy, Py $\rightarrow$ IDLEx).

Figure 5.22 shows the state diagram for the Moore-type output port controller.

A RESET will cause the next state to be INIT. In the INIT state, DONE is set to true to indicate to the arbiter that the OPC is ready. DONE is also used as a clear signal to the read address counter, thereby initializing the counter. Once RESET returns to false, the next state is IDLE, provided the input signals START, LAST, and ACK all assume a false value. If any of these signals are true, then a fault exists and the next state is ERROR.

From the IDLE state, the OPC waits for the START signal from the arbiter. During this time, if either LAST or ACK become true, a fault exists and the next state is ERROR.

Upon receiving the START signal, the next state is BYTE0. At the same time as the state change to BYTE0, the output latch captures byte 0 of the outgoing packet. DONE is set to false, effectively providing feedback to the arbiter that the START signal has been seen. With DONE set to false, the read address counter is no longer held in clear. In addition, CNT, HOLD, and REQ are set to true. With CNT set true, the read address counter will be allowed to count on the next rising edge of its input clock. The HOLD signal will cause the header byte in the output latch (H1) to be held.

From state BYTE0, if an ACK appears, the next state is XFR where the read address counter is allowed to count and the output latch is allowed to latch subsequent bytes of the packet for transmission. If the ACK remains false, then the next state is WAIT. In the WAIT state, the read address counter is prevented from counting and the value in the output latch is held, pending acceptance by the successor SE.

From both states BYTE0 and WAIT, if the START signal goes false or the LAST signal goes true, a fault has been detected and the next state is ERROR.

In state XFR, all bytes of the packet are transmitted to the successor SE. When the last byte is latched into the output latch, LAST becomes true. When this occurs, the next state is LAST. If, before LAST becomes true, either START or ACK becomes false, then a fault exists and the next state is ERROR.

In state LAST, the REQ signal is set false to indicate to the successor SE that this is the last byte. Also, the DONE signal is set true to indicate to the arbiter

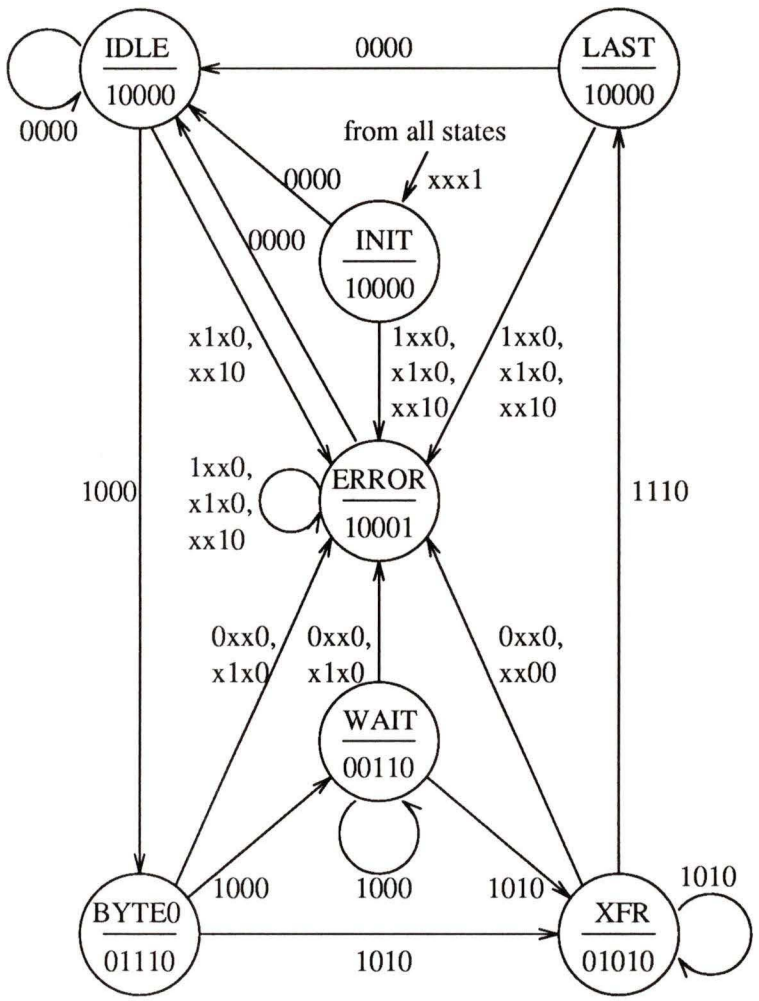


Figure 5.22: State diagram for the output port controller.

that the transfer is complete. The signals START, LAST, and ACK are expected to become false, which results in a transfer to the IDLE state. If any of these signals did not return to false, then a fault exists and the next state is ERROR.

If the ERROR state is entered, START, LAST, and ACK must become false to return to the IDLE state. While in the ERROR state, the ERR output is true.

5.7 Summary

In this chapter, the high-level design for a 2×2 SE to be used in the switch fabric of a BISDN node was presented. The design is completely synchronous, requiring a clock rate of about 25 MHz to achieve the data rates of SONET STS-3c (155.520 Mbits/sec).

The SE model used has four 4-slot buffers between the switching multiplexors to provide the best compromise between performance and memory requirements. Datapaths are nine bits wide, including a parity bit. Packets with errors detected in the header are dropped to avoid routing them to the incorrect destination. Parity errors in data bytes are logged by the SE for use in locating datapath faults. Virtual cut-through may be enabled or disabled by an external signal. When enabled, a packet can transit the SE in just three clock cycles.

It was recommended that the SE implementation include the IEEE Standard 1149.1 Test Access Port and Boundary-Scan Architecture for board-level in-circuit testing. Internal scan paths and memory BIST's would be accessible with this architecture.

Chapter 6

Conclusions

6.1 Summary of Results

This thesis began with a general overview of interconnection networks. Several types of network architecture were shown and methods of switching, controlling, and timing for networks were given.

Experiments in BISDN have concentrated on two switching and transmission methodologies—asynchronous time division (ATD) and fast packet switching (FPS). Due to the lack of standards, BISDN experiments have been performed with trunk controllers (TC's) which convert any desired network transmission protocol (ATD or FPS) to a unique protocol for the switch fabric being studied. Such a model was assumed in this thesis, with the switch fabric being a type of multistage interconnection network known as a delta network. Delta networks are composed of several interconnected stages of $a \times b$ SE's. Typically, networks are rectangular (number of inputs equals number of outputs), so $a \times a$ SE's are used.

Since delta networks are internally blocking, buffers are added to the SE's for storing arriving packets until the required output link is available. These buffers may be added before the switching multiplexors of the SE or between the switching multiplexors of the SE. Simulations performed in the literature have shown that SE's with buffers between the switching multiplexors outperform SE's with buffers before the switching multiplexors.

In this thesis, equations were developed to allow an analytical comparison of the two methods of buffer placement, as well as to investigate the effect of buffer

size and switch size on the throughput and delay performance of the network. The equations also allow the analysis of a proposed look-ahead routing scheme, where packets at a given stage are presorted according to the required outputs of SE's at subsequent stages. Furthermore, the equations allow for modeling local or global arbitration communication methods.

Simulations were used to determine the effect of arbitration scheme on the network's performance. Random, rotating, and buffer occupancy arbitration schemes performed about the same, so the simplest to implement should be used. An exception is the case where offered loads exceed 0.8 in a network with look-ahead routing. For this case, the buffer occupancy scheme degenerated in terms of delay performance, so it should not be used. Furthermore, there were no performance improvements associated with trying different packets in response to blocked requests. Thus, it was recommended that the selected packet continue to be resubmitted until successfully transmitted rather than go to the bottom of the arbitration priority list.

From simulations, it was found that dilation provided very little improvement in performance and, in terms of added hardware complexity, would not be justified. Virtual cut-through, on the other hand, provided reasonable improvements in the delay performance for offered loads less than 0.8 (as the offered load approaches 1.0, these performance gains diminish).

The analysis and simulations revealed that several architectural simplifications could be made if the offered load was kept below 0.8. Above an offered load of 0.8, minor improvements in performance are possible by increasing buffer size or adding look-ahead routing. It is unlikely that the added hardware complexity to achieve these minor performance improvements would be justified. Rather, it is recommended that the system be designed such that the offered load does not exceed 0.8.

The analysis also revealed that there was very little performance difference between local and global arbitration communication methods. Thus, the local method was recommended since it is easier to implement.

The high-level design of a 2×2 SE to be used in a BISDN switch was given. The SE has 4-slot buffers between the switching multiplexors, no look-ahead routing, single links, and virtual cut-through. The design is completely synchronous, requiring a system clock rate of 25 MHz to carry proposed BISDN data rates of

155.520 Mbits/sec while not exceeding 0.8 loading. Virtual cut-through may be enabled or disabled by an external signal. When enabled, a packet can begin exiting an SE as early as three clock cycles after entering the SE. Datapaths throughout the design consist of eight data bits and one parity bit. Packets with errors detected in the header bytes are dropped to avoid routing them to the incorrect destination. Parity errors in data bytes are logged, with potential use in locating datapath faults. Design for testability recommendations include internal scan paths, built-in self tests, and boundary-scan.

6.2 Further Work

Further work on the equations would be needed to address such issues as arbitration method, dilation, virtual cut-through, and resubmission of blocked requests. The model used in developing the equations assumed random arbitration, no dilation, no virtual cut-through, and no resubmission of blocked requests (i.e., blocked requests were dropped).

Bibliography

[1] CCITT, *CCITT Red Book: Integrated Services Digital Network (ISDN)*, vol. III, Fascicle III.5. Geneva, Switzerland: International Telecommunication Union, 1985.

[2] P. Kirton, J. Ellershaw, and M. Littlewood, "Fast packet switching for integrated network evolution," in *Proceedings of the IEEE International Switching Symposium*, pp. B6.2.1–B6.2.7, Mar. 1987.

[3] M. De Prycker, "Definition of network options for the Belgian ATM broadband experiment," *IEEE Journal on Selected Areas in Communications*, vol. 6, pp. 1538–1544, Dec. 1988.

[4] R. J. McMillen, "A survey of interconnection networks," in *IEEE Global Telecommunications Conference Proceedings*, pp. 105–113, Nov. 1984.

[5] G. Broomell and J. R. Heath, "Classification categories and historical development of circuit switching topologies," *ACM Computing Surveys*, vol. 15, pp. 95–133, June 1983.

[6] T. Feng, "A survey of interconnection networks," *Computer*, vol. 14, pp. 12–27, Dec. 1981.

[7] H. J. Siegel, R. J. McMillen, and P. T. Mueller Jr., "A survey of interconnection methods for reconfigurable parallel processing systems," in *Proceedings AFIPS National Computer Conference*, vol. 48, pp. 529–548, 1979.

[8] G. A. Anderson and E. D. Jensen, "Computer interconnection structures: Taxonomy, characteristics, and examples," *ACM Computing Surveys*, vol. 7, pp. 197–213, Dec. 1975.

[9] K. J. Thurber, "Interconnection networks—A survey and assessment," in *Proceedings AFIPS National Computer Conference*, pp. 909–919, May 1974.

[10] C. Clos, "A study of non-blocking switching networks," *Bell System Technical Journal*, vol. 32, pp. 406–424, Mar. 1953.

[11] V. E. Beneš, *Mathematical Theory of Connecting Networks and Telephone Traffic.* New York, NY: Academic Press, 1965.

[12] D. J. Farber and K. C. Larson, "The system architecture of the distributed computer system—the communications systems," in *Proc. Symposium on Computer-Communications Networks and Teletraffic*, vol. XXII, pp. 21–27, Brooklyn Polytechnic Press, Apr. 1972.

[13] C. C. Reames and M. T. Liu, "A loop network for simultaneous transmission of variable length messages," in *Proc. 2nd Annual Symposium on Computer Architecture*, pp. 7–12, Jan. 1975.

[14] C. Rieger, J. Bane, and R. Trigg, "A highly parallel multiprocessor," in *Proc. IEEE Workshop Picture Data Description Manage.*, pp. 298–304, 1980.

[15] M. Sato, H. Matsuura, H. Ogawa, and T. Iijima, "Multimicroprocessor system PX-1 for pattern information processing," in *Multicomputer Algorithms and Image Processing*, pp. 361–371, New York, NY: Academic Press, 1982.

[16] S. I. Saffer, D. J. Mishelevich, S. J. Fox, and V. B. Summerour, "NODAS—The net oriented data acquisition system for the medical environment," in *Proceedings AFIPS National Computer Conference*, vol. 46, pp. 295–299, 1977.

[17] R. J. Swan, S. H. Fuller, and D. P. Siewiorek, "CM*—A modular multi-microprocessor," in *Proceedings AFIPS National Computer Conference*, vol. 46, pp. 637–644, 1977.

[18] H. T. Kung, "The structure of parallel algorithms," in *Advances in Computers* (M. C. Yovits, ed.), vol. 19, pp. 65–112, New York, NY: Academic Press, 1980.

[19] E. M. Aupperle, "MERIT computer network: Hardware considerations," in *Computer Networks* (R. Rustin, ed.), pp. 49–63, Englewood Cliffs, NJ: Prentice-Hall, 1972.

[20] B. W. Arden and H. Lee, "Analysis of chordal ring network," *IEEE Transactions on Computers*, vol. C-30, pp. 291–295, Apr. 1981.

[21] D. A. Reed and R. M. Fujimoto, *Multicomputer Networks: Message-Based Parallel Processing.* Cambridge, MA: MIT Press, 1987.

[22] F. P. Preparata and J. Vuillemin, "The cube-connected cycles: A versatile network for parallel computation," *Communications of the ACM*, vol. 24, pp. 300–309, May 1981.

[23] H. S. Stone, "Parallel processing with the perfect shuffle," *IEEE Transactions on Computers*, vol. C-20, pp. 153–161, Feb. 1971.

[24] C. Wu and T. Feng, "The universality of the shuffle-exchange network," *IEEE Transactions on Computers*, vol. C-30, pp. 324–332, May 1981.

[25] G. M. Masson and B. W. Jordan, Jr., "Generalized multi-stage connection networks," *Networks*, vol. 2, no. 3, pp. 191–209, 1972.

[26] J. P. Ofman, "A universal automaton," *Transactions of the Moscow Mathematical Society*, vol. 14, pp. 200–215, 1965.

[27] N. J. Pippenger, "The complexity theory of switching networks," in *Rep. TR-487*, pp. 42–43, M.I.T. Res. Lab. of Electronics, 1973.

[28] C. D. Thompson, "Generalized connection networks for parallel processor intercommunication," *IEEE Transactions on Computers*, vol. C-27, pp. 1119–1125, Dec. 1978.

[29] B. Beizer, "The analysis and synthesis of signal switching networks," in *Proceedings of the Symposium on Mathematical Theory of Automata*, vol. XII, pp. 563–576, Apr. 1962.

[30] A. E. Joel, Jr., "On permutation switching networks," *Bell System Technical Journal*, vol. 47, pp. 813–822, May-June 1968.

[31] A. Waksman, "A permutation network," *Journal of the Association for Computing Machinery*, vol. 15, pp. 159–163, Jan. 1968.

[32] H. J. Siegel, *Interconnection Networks for Large-Scale Parallel Processing.* Lexington, MA: Lexington Books, 1985.

[33] T. Feng, "Data manipulating functions in parallel processors and their implementations," *IEEE Transactions on Computers*, vol. C-23, pp. 307–318, Mar. 1974.

[34] H. J. Siegel and S. D. Smith, "Study of multistage SIMD interconnection networks," in *Proc. 5th Annual Symposium on Computer Architecture*, pp. 223–229, Apr. 1978.

[35] D. S. Parker and C. S. Raghavendra, "The gamma network," *IEEE Transactions on Computers*, vol. C-33, pp. 367–373, Apr. 1984.

[36] L. R. Goke and G. J. Lipovski, "Banyan networks for partitioning multiprocessor systems," in *Proc. 1st Annual Symposium on Computer Architecture*, pp. 21–28, Dec. 1973.

[37] C. Wu and T. Feng, "On a class of multistage interconnection networks," *IEEE Transactions on Computers*, vol. C-29, pp. 694–702, Aug. 1980.

[38] K. E. Batcher, "The flip network in STARAN," in *Proceedings of the International Conference on Parallel Processing*, pp. 65–71, Aug. 1976.

[39] D. H. Lawrie, "Access and alignment of data in an array processor," *IEEE Transactions on Computers*, vol. C-24, pp. 1145–1155, Dec. 1975.

[40] M. C. Pease, "The indirect binary n-cube microprocessor array," *IEEE Transactions on Computers*, vol. C-26, pp. 458–473, May 1977.

[41] J. H. Patel, "Performance of processor-memory interconnections for multiprocessors," *IEEE Transactions on Computers*, vol. C-30, pp. 771–780, Oct. 1981.

[42] G. B. Adams and H. J. Siegel, "The extra stage cube: A fault-tolerant interconnection network for supersystems," *IEEE Transactions on Computers*, vol. C-31, pp. 443–454, May 1982.

[43] D. M. Dias and J. R. Jump, "Augmented and pruned NlogN multistage networks: Topology and performance," in *Proceedings of the International Conference on Parallel Processing*, pp. 10–12, 1982.

[44] K. Yoon and W. Hegazy, "The extra stage gamma network," in *Proc. 13th Annual International Symposium on Computer Architecture*, pp. 175–182, 1986.

[45] G. B. Adams III, D. P. Agrawal, and H. J. Siegel, "A survey and comparison of fault-tolerant multistage interconnection networks," *Computer*, pp. 14–27, June 1987.

[46] K. Pradmanabhan and D. H. Lawrie, "A class of redundant path multistage interconnection networks," *IEEE Transactions on Computers*, pp. 1099–1108, Dec. 1983.

[47] L. Ciminiera and A. Serra, "A connecting network with fault tolerance capabilities," *IEEE Transactions on Computers*, vol. C-35, pp. 578–580, June 1986.

[48] M. Jeng and H. J. Siegel, "A fault-tolerant multistage interconnection network for multiprocessor systems using dynamic redundancy," in *Proc. 6th International Conference on Distributed Computing Systems*, pp. 70–77, May 1986.

[49] R. J. McMillen and H. J. Siegel, "Performance and fault tolerance improvements in the inverse augmented data manipulator network," in *Proc. 9th Annual Symposium on Computer Architecture*, vol. 10, pp. 63–72, Apr. 1982.

[50] S. M. Reddy and V. P. Kumar, "On fault-tolerant multistage interconnection networks," in *Proceedings of the International Conference on Parallel Processing*, pp. 155–164, Aug. 1984.

[51] S. C. Kothari, G. M. Prabhu, and R. Roberts, "The kappa network, with fault-tolerant destination tag algorithm," *IEEE Transactions on Computers*, vol. 37, pp. 612–617, May 1988.

[52] V. P. Kumar and S. M. Reddy, "Design and analysis of fault-tolerant multistage interconnection networks with low link complexity," in *Proc. 12th An-*

nual International Symposium on Computer Architecture, pp. 376–386, June 1985.

[53] S. C. Kothari, G. M. Prabhu, and R. Roberts, "A multipath network with crosslinks," *Journal of Parallel and Distributed Computing*, vol. 5, pp. 185–193, Apr. 1988.

[54] C. S. Raghavendra and A. Varma, "INDRA: A class of interconnection networks with redundant paths," in *Proceedings of the IEEE Real-Time Systems Symposium*, pp. 153–164, Dec. 1984.

[55] N. Pippenger, "On rearrangeable and non-blocking switching networks," *Journal of Computer and System Sciences*, vol. 17, pp. 145–162, Oct. 1978.

[56] A. S. Tanenbaum, *Computer Networks.* Englewood Cliffs, NJ: Prentice-Hall, 1981.

[57] P. Kermani and L. Kleinrock, "Virtual cut-through: A new computer communication switching technique," *Computer Networks*, vol. 3, pp. 267–286, 1979.

[58] D. F. Wann and M. A. Franklin, "Asynchronous and clocked control structures for VLSI based interconnection networks," *IEEE Transactions on Computers*, vol. C-32, pp. 284–293, Mar. 1983.

[59] A. L. Fisher and H. T. Kung, "Synchronizing large VLSI processor arrays," in *Proc. 10th Annual International Symposium on Computer Architecture*, pp. 54–58, 1983.

[60] P. Belforte, E. Garetti, A. Notarstefano, F. Perardi, and M. Perassi, "Advanced switching techniques for integrated broadband communications," *CSELT Technical Reports*, vol. XV, pp. 297–306, June 1987.

[61] S. Giorcelli, C. Demichelis, G. Giandonato, and R. Melen, "Experimenting with fast packet switching techniques in first generation ISDN environment," in *Proceedings of the IEEE International Switching Symposium*, pp. B5.4.1–B5.4.7, Mar. 1987.

[62] G. Perucca, P. Belforte, E. Garetti, and F. Perardi, "Research on advanced switching techniques for the evolution of ISDN and broadband ISDN," *IEEE Journal on Selected Areas in Communications*, vol. SAC-5, pp. 1356–1364, Oct. 1987.

[63] G. W. R. Luderer, J. J. Mansell, E. J. Messerli, R. E. Staehler, and A. K. Vaidya, "Wideband packet technology for switching systems," in *Proceedings of the IEEE International Switching Symposium*, pp. B6.1.1–B6.1.7, Mar. 1987.

[64] M. De Prycker and M. De Somer, "Performance of a service independent switching network with distributed control," *IEEE Journal on Selected Areas in Communications*, vol. SAC-5, pp. 1293–1301, Oct. 1987.

[65] G. J. Anido and A. W. Seeto, "Multipath interconnection: A technique for reducing congestion within fast packet switching fabrics," *IEEE Journal on Selected Areas in Communications*, vol. 6, pp. 1480–1488, Dec. 1988.

[66] R. G. Bubenik and J. S. Turner, "Performance of a broadcast packet switch," *IEEE Transactions on Communications*, vol. 37, pp. 60–69, Jan. 1989.

[67] J. Y. Hui and E. Arthurs, "A broadband packet switch for integrated transport," *IEEE Journal on Selected Areas in Communications*, vol. SAC-5, pp. 1264–1273, Oct. 1987.

[68] A. Huang and S. Knauer, "Starlite: A wideband digital switch," in *IEEE Global Telecommunications Conference Proceedings*, pp. 121–125, 1984.

[69] C. P. Kruskal and M. Snir, "The performance of multistage interconnection networks for multiprocessors," *IEEE Transactions on Computers*, vol. C-32, pp. 1091–1098, Dec. 1983.

[70] M. Kumar and J. R. Jump, "Performance enhancement in buffered delta networks using crossbar switches and multiple links," *Journal of Parallel and Distributed Computing*, vol. 1, pp. 81–103, Aug. 1984.

[71] M. Kumar and J. R. Jump, "Performance of unbuffered shuffle-exchange networks," *IEEE Transactions on Computers*, vol. C-35, pp. 573–578, June 1986.

[72] H. Yoon, K. Y. Lee, and M. T. Liu, "Performance analysis and comparison of packet switching interconnection networks," in *Proceedings of the International Conference on Parallel Processing*, pp. 542–545, Aug. 1987.

[73] P.-Y. Chen, P.-C. Yew, D. Lawrie, and D. Padua, "Interconnection networks using shuffles," *Computer*, vol. 14, pp. 55–64, Dec. 1981.

[74] D. M. Dias and J. R. Jump, "Analysis and simulation of buffered delta networks," *IEEE Transactions on Computers*, vol. C-30, pp. 273–282, Apr. 1981.

[75] D. M. Dias and J. R. Jump, "Packet switching interconnection networks for modular systems," *Computer*, vol. 14, pp. 43–53, Dec. 1981.

[76] P.-Y. Chen, P.-C. Yew, and D. Lawrie, "Performance of packet switching in buffered single-stage shuffle-exchange networks," in *Proc. 3rd International Conference on Distributed Computing Systems*, pp. 622–627, Oct. 1982.

[77] P.-Y. Chen, "Multiprocessor systems: Interconnection networks, memory hierarchy, modeling and simulations," Tech. Rep. UIUCDCS-R-82-1083, Dept. of Computer Science, Univ. Illinois, Urbana, IL, January 1982.

[78] Y.-C. Jenq, "Performance analysis of a packet switch based on single-buffered banyan network," *IEEE Journal on Selected Areas in Communications*, vol. SAC-1, pp. 1014–1021, Dec. 1983.

[79] L. T. Wu, "Mixing traffic in buffered Banyan network," in *Proc. 9th Data Communications Symposium*, pp. 134–139, Sept. 1985.

[80] H. Yoon, K. Y. Lee, and M. T. Liu, "Performance analysis of multibuffered packet-switching networks in multiprocessor systems," *IEEE Transactions on Computers*, vol. 39, pp. 319–327, Mar. 1990.

[81] R. E. Walpole and R. H. Myers, *Probability and Statistics for Engineers and Scientists.* New York, NY: Macmillan Publishing Company, 4 ed., 1989.

[82] I. Olkin, L. J. Gleser, and C. Derman, *Probability Models and Applications.* New York, NY: Macmillan Publishing Company, 1980.

[83] D. W. L. Yen, J. H. Patel, and E. S. Davidson, "Memory interference in synchronous multiprocessor systems," *IEEE Transactions on Computers*, vol. C-31, pp. 1116–1121, Nov. 1982.

[84] T. N. Mudge, J. P. Hayes, G. D. Buzzard, and D. C. Winsor, "Analysis of multiple-bus interconnection networks," *Journal of Parallel and Distributed Computing*, vol. 3, pp. 328–343, 1986.

[85] Y.-C. Liu and C. Wang, "Analysis of prioritized crossbar multiprocessor systems," *Journal of Parallel and Distributed Computing*, vol. 7, pp. 321–334, 1989.

[86] F. El-Guibaly, "Design and analysis of arbitration protocols," *IEEE Transactions on Computers*, vol. 38, pp. 161–171, Feb. 1989.

[87] T1S1 Technical Sub-Committee, "Broadband aspects of ISDN baseline document," Draft T1S1.5/89-001, AT&T, Bellcore, BellSouth Services, GTE-Telops, Northern Telecom, 1989.

[88] D. Del Corso, H. Kirrmann, and J. D. Nicoud, *Microcomputer Buses and Links.* New York, NY: Academic Press, 1986.

[89] Y. Yeh, M. G. Hluchyj, and A. S. Acampora, "The Knockout Switch: A simple, modular architecture for high-performance packet switching," *IEEE Journal on Selected Areas in Communications*, vol. SAC-5, pp. 1274–1283, Oct. 1987.

[90] J. S. Turner, "Design of a broadcast packet switching network," *IEEE Transactions on Communications*, vol. 36, pp. 734–743, June 1988.

[91] M. Devault, J.-Y. Cochennec, and M. Servel, "The "Prelude" ATD experiment: Assessments and future prospects," *IEEE Journal on Selected Areas in Communications*, vol. 6, pp. 1528–1537, Dec. 1988.

[92] P. Plehiers, M. Fastrez, J. Bauwens, and M. De Prycker, "Evolution towards a Belgian broadband experiment," in *Proceedings of the IEEE International Switching Symposium*, pp. B5.3.1–B5.3.8, Mar. 1987.

[93] D. K. Pradhan, ed., *Fault-Tolerant Computing - Theory and Techniques.* Englewood Cliffs, NJ: Pretice-Hall, 1986.

[94] C. M. Maunder and R. E. Tulloss, *The Test Access Port and Boundary-Scan Architecture.* IEEE Computer Society Press, 1990.

[95] A. Mourad, B. Özden, and M. Malek, "Comprehensive testing of multistage interconnection networks," *IEEE Transactions on Computers*, vol. 40, pp. 935–951, Aug. 1991.

VITA

Surname:	Mund	Given Names:	Graeme Brad
Place of Birth:	Edmonton, Alberta	Date of Birth:	March 9, 1960

Educational Institutions Attended:

University of Victoria	1987 to 1992
University of Saskatchewan, Saskatoon, Saskatchewan	1978 to 1982

Degrees Awarded:

B.A.Sc. (Electrical Engineering) University of Saskatchewan 1982

Publications:

1. G. B. Mund and F. El-Guibaly, "A 2×2 switching element for broadband ISDN", in *IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, Victoria, BC, June 1–2, 1989, pp. 620–623.

PARTIAL COPYRIGHT LICENSE

I hereby grant the right to lend my thesis to users of the University of Victoria Library, and to make single copies only for such users or in response to a request from the Library of any other university, or similar institution, on its behalf or for one of its users. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by me or a member of the University designated by me. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Title of Thesis: SWITCHING ELEMENT DESIGN FOR BISDN

Author: [redacted]

(Signature)

GRAEME BRAD MUND

(Name in Block Letters)

September 17, 1992

(Date)