USC–SIPI REPORT #284

Design For a Shared Memory Computer With a Combining Optical Interconnection Network Using External Routing

by

Clare Waterson

May 1995

Signal and Image Processing Institute
UNIVERSITY OF SOUTHERN CALIFORNIA
Department of Electrical Engineering-Systems
3740 McClintock Avenue, Room 404
Los Angeles, CA 90089-2564 U.S.A.

Acknowledgements

Dr. B. Keith Jenkins — (Advisor) for technical advice and guidance throughout the long process of research and writing that led to this dissertation

Air Force Office of Scientific Research and National Science Foundation — for support of my Ph.D. research

The Aerospace Corporation — for support of education and research

Bob Bolender — for extensive assistance, support, time, and encouragement; and for nice computer equipment

Michael Elkins — for encouragement, motivation, patience, time, and computer assistance

Adam Goldstein — (my "Progress Advisor") for *interaction is a good thing* (hey, but *extremes are bad*), and for timely (and favorable) paper reviews

Adam Goldstein, Ching-Chu Huang, Kuang-Yu Li, Sabino Piazzolla — for mutual support

Kuang-Yu Li — for timely suggestions

Bill Woody — for technical advice and suggestions

Brian Primeau — for *gentle* humor

Ed Meinel — for support and patience

Russell Kurtz — for empathizing

Ken Chan — for encouragement

Jill Waterson — for keeping me sane?

Mucha, Tigger & Hobbes, Tsunami & Typhoon, Amber & Mello — for inspiration (μ!)

John & John *(They Might Be Giants)* — for *I will never say the word "procrastinate" again...*

Johnette Napolitano *(Concrete Blonde)* — for *Who did you think I would be? Hah! Well you got me instead!*

Brian Dewan — for *Feel the brain... slippery and smooth*

Douglas Adams — for *You've got to build bypasses*

Bach and KROQ — for the perfect music by which to write luculent lucubrations full of sesquipedalian terminology.

Douglas Hofstadter — Hofstadter's Law: *It always takes longer than you expect, even when you take Hofstadter's Law into account.*

Abstract

The design of a fine-grained MIMD computer architecture, the Shared Memory Optical/Electronic Computer (SMOEC), is presented. The SMOEC architecture consists of electronic processing elements and memory modules interconnected with a novel combining passive optical interconnection network which is controlled by a separate electronic routing processor. The hybrid system design emphasizes the strengths of the two technologies, using optics for communication and electronics for processing. Sample system implementation recipes exemplify the flexibility, realizability, and scalability of the SMOEC system architecture. The system-level focus on interrelated development of three main design facets—architecture, hardware, and control algorithms—has been crucial in designing a well-balanced high performance system.

The design of the optical interconnection network, the Free-space Interconnection with Externally-controlled Routing (FIER), is presented in detail. The network is implemented using exclusively passive optical elements, and it employs a shuffle–exchange topology. Each shuffle–exchange stage is optically cascadable. The FIER switch nodes are capable of broadcast and combine operations in additional to bypass and exchange. The FIER is designed to be circuit-switched by an external electronic routing processor. Although the FIER is designed to be used as a subsystem within the SMOEC, it may be used as a subsystem in other communication or computing architectures.

A review of Extended Generalized Shuffle (EGS) network theory is provided to give a solid theoretical background for the interconnection topology employed in the FIER and to prepare for presentation of a new routing algorithm.

A new algorithm, the Flexible Localized Algorithm for EGS-network Management (FLAEM), is presented. This algorithm was developed for circuit-switched combining regular simplified EGS networks (such as the FIER optical network); it parallelizes the routing process in a different way than the previously available method. The effectiveness of the FLAEM is illustrated by simulation results, which show that the algorithm time complexity is logarithmic in system size (N). The FLAEM extends the repertoire of routing techniques available for this class of EGS networks.

Contents

List of Figures

List of Tables

List of Acronyms

Parallel Computer Architecture Acronyms

MIMD Multiple Instruction streams over Multiple Data streams: *Parallel computer model employing independent processing elements.*

SMOEC Shared Memory Optical/Electronic Computer: *The parallel MIMD shared-memory computer system architecture presented herein.*

FIER Free-space Interconnection with Externally-controlled Routing: *The passive optical interconnection network within the* SMOEC.

MATSH Multifunctional Arbitrator of Traffic for Shuffle–exchange Hardware: *The single-stage recirculating electronic shuffle–exchange hardware used to control the* FIER *in a circuit-switched manner.*

FLAEM Flexible Localized Algorithm for EGS-network Management: *The parallel algorithm used on the* MATSH *to compute the bypass/exchange switch settings for the* FIER.

CSDI Control Signal Distribution & Interface: *The buffers and interface from the* MATSH *control unit to the* FIER *optical network.*

PE Processing Element: *One of N individual processors.*

MM Memory Module: *One of N modular blocks of the shared memory.*

MIN Multistage Interconnection Network: it A network with multiple stages of switching elements interconnected by an arbitrary interconnection pattern .

EGS Extended Generalized Shuffle: *A* MIN *with an interconnection topology invented by G. Richards* [Richards 91a] .

RS-EGS Regular Simplified EGS: *A particular set of additional restrictions on the general* EGS *definition, which are employed in the* FIER.

E-SE Electronic Shuffle–Exchange: *The single electronic shuffle–exchange stage within the* MATSH.

O-SE Optical Shuffle–Exchange: *One of the S_S passive optical shuffle–exchange stages within the* FIER.

FIER Optical Network Acronyms

LA Linear Array: *An array of regularly spaced uniformly polarized individual optical channels.*

CP Channel Pairs: *An array of superimposed pairs of orthogonally polarized optical channels.*

SLM Spatial Light Modulator: *A two-dimensional array of optical switching devices (pixels).*

TSLM Tri-state SLM: *An* SLM *with tri-state optical swiching elements, designed for use in the* FIER.

PSA Pixel Spacing Adjuster: *Optical setup designed for the* FIER *to adjust pixel spacings to enable cascadability of optical shuffle–exchange* (O-SE) *stages.*

HWP Half-Wave Plate: *Optical device capable of (e.g.) rotating a linear polarization by* 90° *when oriented at* 45° *to the incoming linear polarization axis.*

QWP Quarter-Wave Plate: *Optical device capable of (e.g.) turning linear polarization into circular polarizaiton, and vice-versa.*

PWP Pixellated Wave Plate: *Fixed wave plate divided into pixels of* HWP *and null effects.*

FLC Ferroelectric Liquid Crystal: *Material from which polarization-switching* SLMs *(and* TSLMs*) may be constructed.*

List of Notation

General RS-EGS Network Parameters

N		The number of inlets in a MIN. N is also the number of outlets when the MIN under consideration is an RS-EGS network.
n	$= \log_2 N$	The number of bits in an inlet address.
F		The amount of "fan-out" and "fan-in" in an RS-EGS network. $F=2^f$ for a restricted-F network.
f	$= \log_2 F$	The number of stages of 1×2 switches to implement a $1\times F$ "fan-out" section (or 2×1 switches for a $F\times1$ "fan-in" switch in an RS-EGS network; each stage is composed of a perfect shuffle and an array of 2×2 switches.
W	$= N \cdot F$	The "width" (number of links) of the main section of an RS-EGS network.
S_S	$= \mathcal{O}[\log N]$	The number of stages in the "main section" of an RS-EGS network (the main section consists of all the perfect shuffles and 2×2 switches, and excludes the "fan-out" and "fan-in" stages and the final F-shuffle).
S_T	$= S_S + 2$	The total number of stages in an RS-EGS network, when the "fan-out" and "fan-in" sections are counted as one stage each.
S_F	$= S_S + 2f$	The total number of stages in a restricted-F RS-EGS network, when the "fan-out" and "fan-in" sections are each composed of f stages of 1×2 or 2×1 switches, respectively.

Routing and Write Arbitration Parameters

D		Number of bits of data in an MM
B		Number of bits per submodule (word) within an MM
L	$=D/B$	Number of submodules (words) in an MM
		= Number of bits in a subaddress field
C		Number of wires (or fibers) per MATSH (E-SE) node
K	$=L/C$	Number of submodules transmitted per MATSH wire
U		Number of status bits for MATSH routing processing
E	$=n+f+S_S$	Number of bits in a Path Vector
J	$=E+U$	Number of bits per read operation

FIER Optical Network Switching Parameters

$\tau_{\text{O-Sw}}$		Time to simultaneously load and switch the optical switching components (FLC TSLMs, described in Section 4.2.6)
		= Time to simultaneously switch all the optical bypass/exchange switches
		= Optical reconfiguration time
r_{FIER}		Data rate through the FIER (bps)
T_{FIER}	$= D/r_{\text{FIER}}$	Total optical communication time: time to access (read from or write to) an entire MM

FIER Optical Network Power and Loss Parameters

$E_{\text{MIN}}^{(\text{Det})}$		Minimum energy required for bit discrimination at a detector in the FIER
$P_{\text{MIN}}^{(\text{Det})}$	$= E_{\text{MIN}}^{(\text{Det})} \cdot r_{\text{FIER}}$	Minimum power required for bit discrimination at a detector in the FIER, for a given bit rate
$P_{\text{INPUT}}^{(\text{LD})}$		Power from the laser diodes input into the FIER
$\mathcal{L}$		Total optical loss factor throughout the entire FIER optical network
$\mathcal{L}_{\text{TOL}}$	$= P_{\text{INPUT}}^{(\text{LD})} \ / \ P_{\text{MIN}}^{(\text{Det})}$	Tolerable optical loss factor in the FIER, without optical repeaters
$\mathcal{R}$		Number of repeaters required to be included in the FIER to keep the loss above a tolerable level

MATSH Routing Processor Switching Parameters

r_W	Wire (or fiber) data rate between E-SE stage cycles in the MATSH
$\vec{\tau}^{(READ)}_{XMIT}$	Time to transmit routing bits between E-SE stage cycles within the MATSH (in the PE→MM "forward" direction)
$\vec{\tau}^{(WRITE)}_{XMIT}$	Time to transmit routing and write arbitration bits between E-SE stage cycles within the MATSH (forward)
$\vec{\tau}^{(READ)}_{PROC}$	Time to process routing bits for one E-SE stage cycle within the MATSH (forward)
$\vec{\tau}^{(WRITE)}_{PROC}$	Time to process routing and write arbitration bits for one E-SE stage cycle within the MATSH (forward)
$\overleftarrow{\tau}^{(READ)}_{XMIT}$	Time to transmit routing bits between E-SE stage cycles within the MATSH (in the MM→PE "reverse" direction)
$\overleftarrow{\tau}^{(WRITE)}_{XMIT}$	Time to transmit routing and write arbitration bits between E-SE stage cycles within the MATSH (reverse)
$\overleftarrow{\tau}^{(READ)}_{PROC}$	Time to process routing bits for one E-SE stage cycle within the MATSH (reverse)
$\overleftarrow{\tau}^{(WRITE)}_{PROC}$	Time to process routing and write arbitration bits for one E-SE stage cycle within the MATSH (reverse)
$\tau^{(*)}_{PROC}$	Specially defined processing time: See Section 7.1.2.5 (forward or reverse)
$\vec{\Psi}_{READ}$	Number of forward read passes (routing bits only) in a routing cycle
$\vec{\Psi}_{WRITE}$	Number of forward write passes (write arbitration bits) in a routing cycle
$\vec{\Psi}_{READ}$	Number of reverse read passes (routing bits only) in a routing cycle
$\vec{\Psi}_{WRITE}$	Number of reverse write passes (write arbitration bits) in a routing cycle

MATSH Routing Processor Switching Parameters (cont.)

T_{XMIT} Time devoted to data transmission in the MATSH

$$= S_F(\quad \vec{\Psi}_{\text{READ}} \, \vec{\tau}^{(\text{READ})}_{\text{XMIT}} + \vec{\Psi}_{\text{WRITE}} \, \vec{\tau}^{(\text{WRITE})}_{\text{XMIT}} + \overleftarrow{\Psi}_{\text{READ}} \, \overleftarrow{\tau}^{(\text{READ})}_{\text{XMIT}} + \overleftarrow{\Psi}_{\text{WRITE}} \, \overleftarrow{\tau}^{(\text{WRITE})}_{\text{XMIT}} \quad)$$

T_{PROC} Time devoted to data processing in the MATSH

$$= S_F(\quad \vec{\Psi}_{\text{READ}} \, \vec{\tau}^{(\text{READ})}_{\text{PROC}} + \vec{\Psi}_{\text{WRITE}} \, \vec{\tau}^{(\text{WRITE})}_{\text{PROC}} + \overleftarrow{\Psi}_{\text{READ}} \, \overleftarrow{\tau}^{(\text{READ})}_{\text{PROC}} + \overleftarrow{\Psi}_{\text{WRITE}} \, \overleftarrow{\tau}^{(\text{WRITE})}_{\text{PROC}} \quad)$$

T_{MATSH} Total electronic control bit computation time in the MATSH

$$= T_{\text{XMIT}} + T_{\text{PROC}}$$

$$= S_F[\quad \vec{\Psi}_{\text{READ}} \, (\vec{\tau}^{(\text{READ})}_{\text{PROC}} + \vec{\tau}^{(\text{READ})}_{\text{XMIT}} \,) + \vec{\Psi}_{\text{WRITE}} \, (\vec{\tau}^{(\text{WRITE})}_{\text{PROC}} + \vec{\tau}^{(\text{WRITE})}_{\text{XMIT}} \,) + \overleftarrow{\Psi}_{\text{READ}} \, (\overleftarrow{\tau}^{(\text{READ})}_{\text{PROC}} + \overleftarrow{\tau}^{(\text{READ})}_{\text{XMIT}} \,) + \overleftarrow{\Psi}_{\text{WRITE}} \, (\overleftarrow{\tau}^{(\text{WRITE})}_{\text{PROC}} + \overleftarrow{\tau}^{(\text{WRITE})}_{\text{XMIT}} \,) \,]$$

Chapter 1

INTRODUCTION

In past years, optical computing research has often focussed on the design of individual computer hardware subsystems implemented with optical devices. However, integration of these subsystems into a complete computer system has often proven to be impractical since the hardware was often designed without consideration for system design requirements. For example, several optical interconnection networks were designed that looked attractive, yet lacked practical control mechanisms. In more recent years there has been a growing realization among designers of optical hardware that system concerns are essential to the usefulness of any proposed optical subsystem, so optical engineers are beginning to pay more attention to hitherto neglected (or postponed) aspects of computer system design such as interconnection network control techniques.

Over the course of the past decade, there has been a dawning realization among optical computing researchers that a hybrid combination of electronic computing and optical communication holds the most promise for powerful practical systems

[Jenkins 92, Lohmann 95, Schenfeld 95]; i.e., the trend is away from "optical computing" and toward "optics in computing." Thus the research presented herein reflects a system-level orientation to design decisions, incorporating electronic processing and optical communication subsystems that were conceived from the beginning to work together.

The hybrid shared memory computer system described in the following pages was designed to incorporate a passive optical interconnection network as an essential central component. The system was designed to emphasize the strengths and balance the weaknesses inherent in a passive optical design. Explicit routing algorithms and communication techniques were developed in concert with the system architecture and interconnectin network design so that these important system aspects would not limit performance. The intent was provide a detailed design for a general-purpose MIMD computer system architecture that exhibits flexibility, realizability, and scalability.

Parts of this work have been previously published as follows. Chapter 1 (Introduction), Chapter 2 (System Architecture), and Chapter 6 (Communication Algorithms) are primarily drawn from [Waterson 91] and [Waterson 94a]. Chapter 4 (Optical Interconnection Network Hardware) is taken from [Waterson 94b], except for Section 4.3 (Optical EGS Hardware Design) which is entirely new. An overview of Chapter 5 (Network Routing Algorithm) was published as [Waterson 95]. Chapter 3 (Interconnection Network Topology and Routing [Review]) and Chapter 7 (Architecture Performance) are newly presented here, and Chapter 8 (Conclusions) draws from all these sources.

1.1 Background

Parallel processors have been receiving a great deal of research interest largely because many desired computational applications continue to greatly exceed the capabilities of current uniprocessor computers [Beetem 85, Butler 93, Hwang 93]. Parallel processors may provide increased performance at the cost of increased hardware complexity. The ultimate goal of parallel computer design is to use N processing elements to reduce the execution time by a factor of N relative to that of a comparable uniprocessor. In general the speedup actually achieved by a parallel processor can be significantly less than N due primarily to data dependencies and communication contention [Stone 73]. Furthermore, the theoretical speedup obtained in an abstract parallel computation *model* may be significantly higher than in a parallel computer *architecture* (a realization of the model) [Jenkins 86].

Several abstract models of parallel computation have been developed and studied by the computer science and parallel processing communities [Fortune 78, Schwartz 80]. The shared memory models are among the most computationally powerful of these models. They benefit from substantial theoretical foundations, and many algorithms have been mapped onto these models in order to characterize theoretically optimum parallel performance [Schwartz 80, Borodin 85]. Thus, parallel machine architectures based on shared memory models should share these benefits. However, achieving fine-grained parallelism (very large numbers of relatively simple processing elements) based on a shared memory model is a difficult goal primarily due to the complexity of the interconnection network hardware and its associated control.

The shared memory model of computation [Fortune 78, Schwartz 80, Vishkin 83, Borodin 85, Jenkins 86] consists of a set of processing elements (PEs) that can all communicate (read and/or write) simultaneously with a shared memory in a single time step (Figure 1.1). Each PE is a uniprocessor capable of carrying out an independent program, so that the set of PEs can operate in a MIMD (Multiple Instruction stream over Multiple Data stream) manner. The shared memory comprises a set of memory cells. In this ideal model any PE can communicate with any memory cell in one time step. Therefore, all PEs can simultaneously access different cells of the shared memory. In addition, in the strongest version of the shared memory model (CRCW = Concurrent Read, Concurrent Write) multiple PEs can simultaneously access (read from and/or write to) the same individual memory cell [Hwang 93].

Several compromises of the shared memory model ideal must be made in order to allow a physical realization. For example, the interconnection network is a particularly critical element, and usually the limiting factor, of parallel architectures based on a shared memory computation model. Therefore, focussing design effort on the optimization of this element is essential.

Parallel access to memory is a key feature of a physical realization of a shared memory machine. By reducing addressing bottlenecks compared to a conventional von Neumann architecture a substantial improvement in performance can be achieved. This parallel memory access is commonly approximated by dividing the shared memory into memory modules (MMs), and providing an interconnection network between the PEs and MMs. This interconneciton network is used to perform read and write (and other) operations, requiring an amount of time which

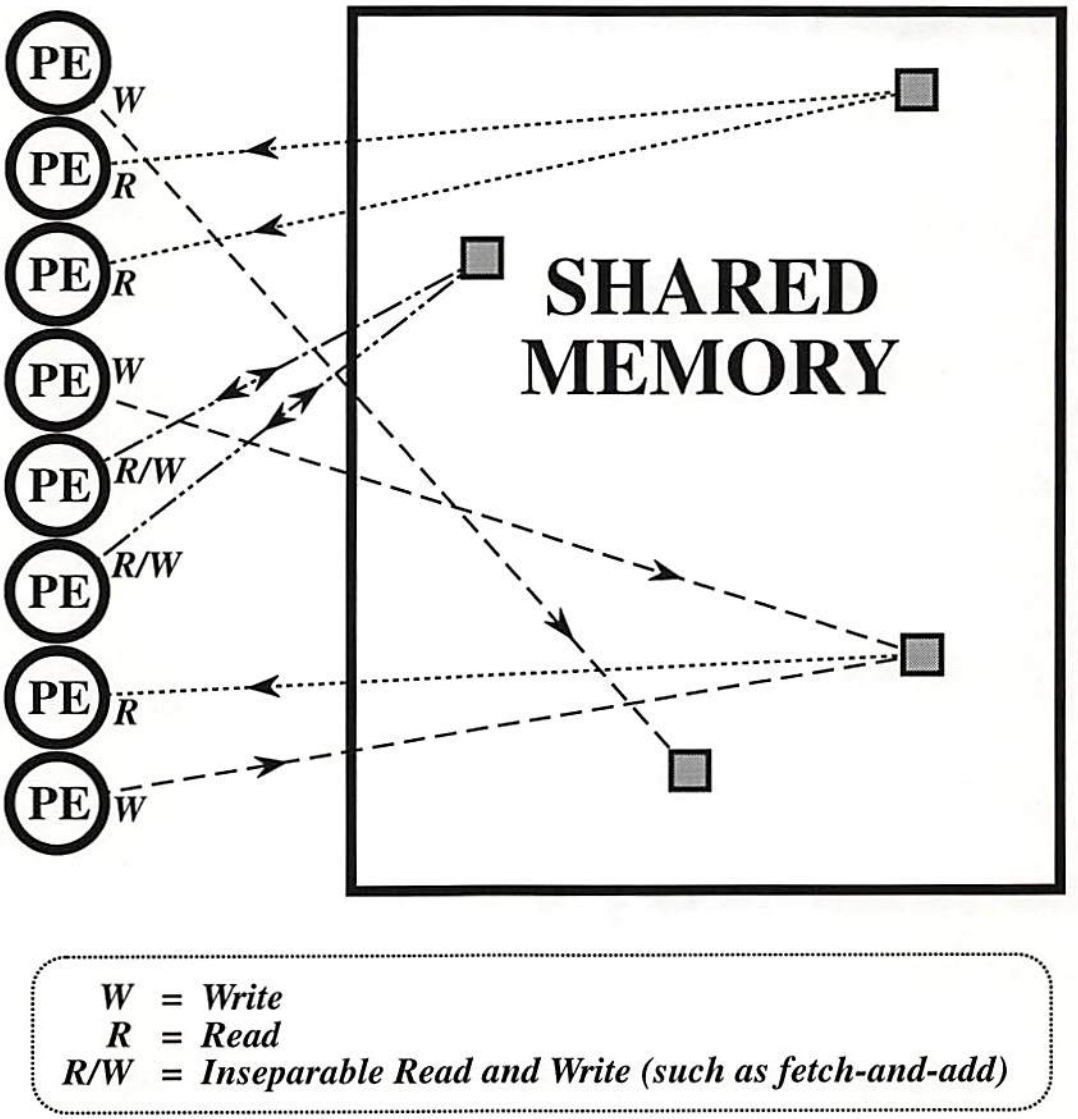


Figure 1.1: The Shared Memory Model Computer. PE, Processing Element.

is dependent on N, the number of PEs and MMs. Typically (as in the architecture proposed here) the number of PEs and the number of MMs are set to be equal to balance PE and MM communication requirements.

Highly parallel electronic processing architectures are primarily limited by the characteristics of their electronic interconnection network [Guha 90]. Unfortunately, complex global interconnection topologies can be extremely difficult to implement electronically due to complexity limitations of two-dimensional VLSI layouts or three-dimensional wire layouts (or even proposed three-dimensional VLSI layouts) [Giles 86, Feldman 88]. Therefore it is worth considering the use of optics in implementing the interconnection network in the proposed shared memory machine.

1.2 Motivation

The imaging, superposition, and three-dimensional nature of optics allow complex interconnection topologies to be implemented optically with reduced complexity from electronic implementations [Giles 86, Barakat 87]. Optical imaging allows massive parallelism when interconnecting from pixel to corresponding pixel, without requiring fabrication of a separate physical path for each connection. Optical superposition allows light beams to pass essentially unaffected through each other, allowing reduced hardware complexity by eliminating the necessity to reserve separate physical space for each data path in a network [Jenkins 88]. The three-dimensional nature of optics allows further reduced complexity over the planar constraints of VLSI [Giles 86]. Optical hardware also has the advantage of

being capable of operation at extremely high bandwidths [Guha 90, Feldman 88, Hartman 86].

Although optical superposition is used in optical interconnection networks to reduce complexity, it is precisely this difficulty of getting optical beams to interact that makes optical switching a less mature technology than electronic switching. Thus the trend in recent years has been away from designing all-optical computer systems and instead towards designing hybrid computer systems with electronic processing and optical interconnecitons [Jenkins 92, Lohmann 95, Schenfeld 95]. Therefore, in the design of the parallel computer architecture presented herein, all nonlinear processing operations (computations) will be performed electronically. Similarly, although optical memory is somewhat further developed, electronic memories will also be used. A novel optical interconnection network is designed to satisfy the communications requirements for this computer system.

Since the interconnection network is critical to the overall machine performance, its design and control is central to the design of a shared memory computer. However, specifying an efficient high performance complex interconnection network as an integral part of a parallel computer architecture does not provide sufficient basis for believing the architecture to be an improvement over other computers with less capable interconnection networks. A valid comparison *cannot* be made between architectures or networks without careful consideration of the associated control algorithms. Whereas an apparently "more powerful" network might be capable of great flexibility in its routing (with perhaps even redundant paths available for fault tolerance), such a network can prove to be undesirable due to the associated cumbersome and slow control algorithms. In fact, it is quite possible for a control

algorithm to have worse time complexity behavior than the total time delay of data transfer through the network.

One important aspect of interconnection network control is the method of resolving routing conflicts. Buffered networks (typically used in electronic networks) provide one very common type of conflict resolution. In a buffered network conflicts in a switching node are resolved by routing one message and buffering (saving in a local memory stack) the other message, which is sent later. However, a major drawback of buffering is the possibility of network performance degradation due to "hot spot" formation [Pfister 85b, Kumar 86, Lee 86, Thomas 86, Yew 86]. Hot spots in a buffered network are formed when a large number of simultaneous references are made to the same "hot" memory location. Requests may stack up and overflow the buffer sizes, with backups propagating backward through the interconnection network in a tree pattern, until the backup affects the flow through most or all of the network (see Figure 1.2.)

Non-buffered networks eliminate the possibility of hot spot formation, but require the problem of routing conflict resolution to be dealt with in some other manner. A non-buffered network is employed in the interconnection network presented here, and allows passive optical switching to be used for its implementation. The resulting network can achieve greater throughput and can incorporate a novel conflict resolution technique implemented in a centralized controller (discussed in Chapter 5).

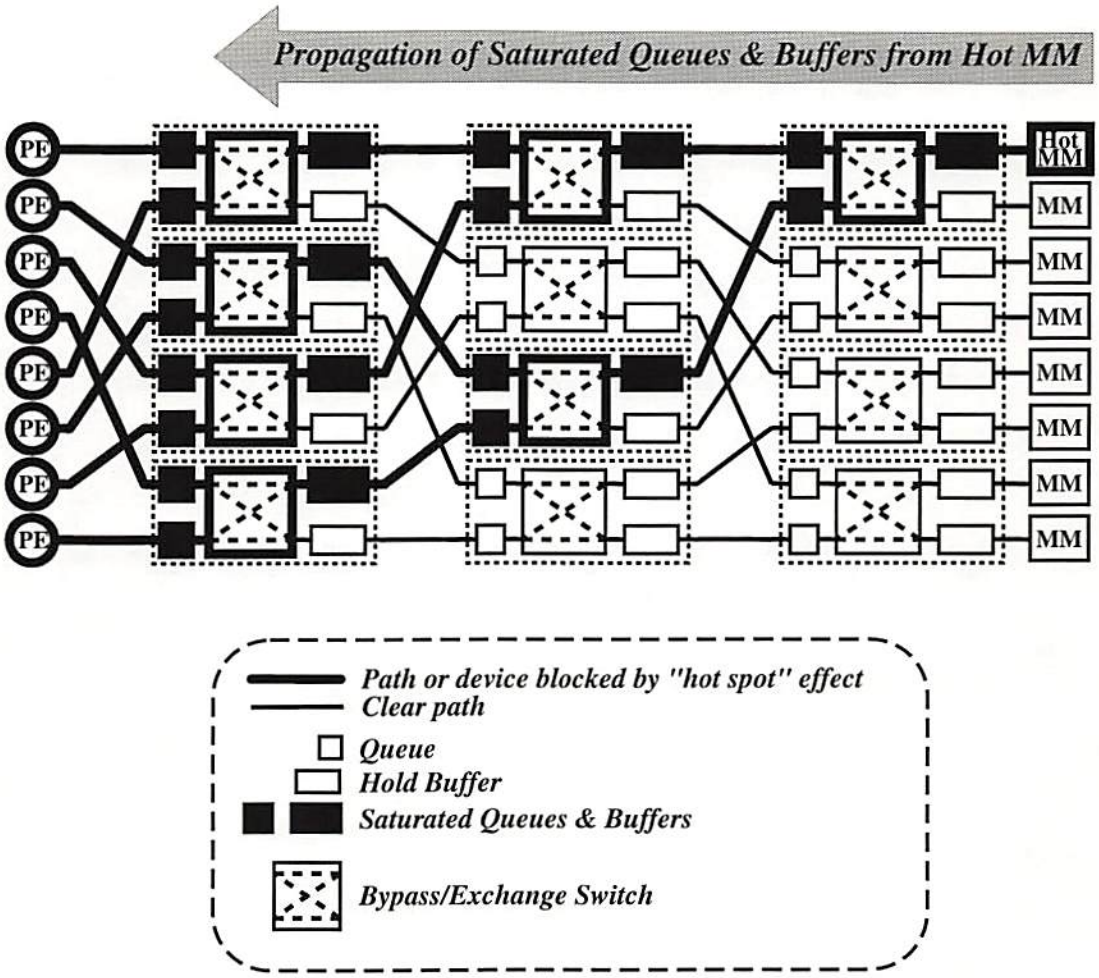


Figure 1.2: Hot Spot Formation (*from* [Kumar 86]). PE, Processing Element; MM, Memory Module.

1.3 Design Philosophy

Tradeoffs between control algorithm complexity and interconnection network characteristics must be an integral part of the design process. Control algorithm design must be undertaken simultaneously with interconnection network design to obtain true performance improvement. Similarly, hardware implementation must also be considered (at least at a high level) simultaneously with computer design so that the design may be physically realized. Control algorithm design also affects the hardware decisions, and vice versa, since control signals must be able to reach the actual control sites. Therefore this paper presents the design of the Shared Memory Optical/Electronic Computer (SMOEC) as a design consisting of three essential facets: architecture, control algorithm, and hardware. These three facets are like the three legs of a tripod; if one leg is weak, the whole tripod is weak (see Figure 1.3).

For example, the triple focus of the design philosophy allowed the choice of a passive optical interconnection network to impact the control algorithm and system architecture design. In this way, the SMOEC was designed with an optical interconnetion network as an integral element of the design. This method is in contrast to taking the idea of an optical network and inserting it in a preexisting electronic parallel computer design that was intended to be used with a fundamentally different electronic interconnection network.

The triple focus of this research on architecture, hardware, and control is part of a trend towards systems design among those designing hybrid optical/electronic systems which incorporate novel optical subsystems. Some recent work in the "optics in computing" field explicitly advocates attention to system-level design

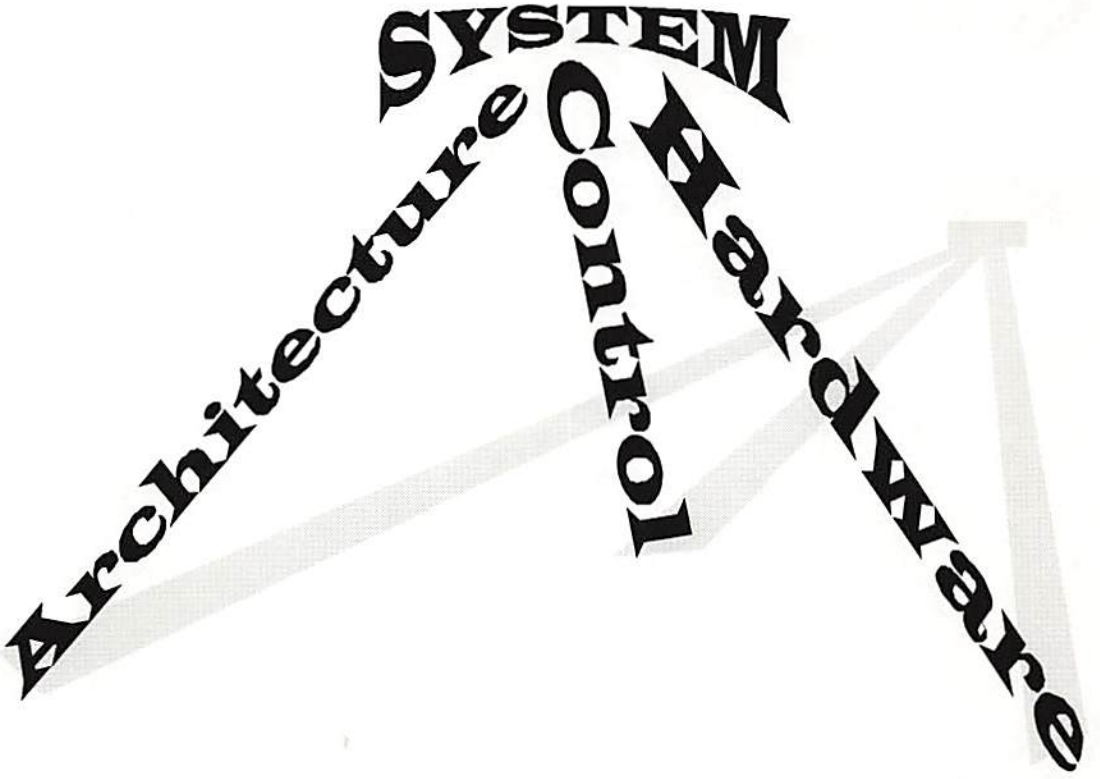


Figure 1.3: System Design Tripod: System viability is supported by three essential design facets.

concerns for hybrid systems [e.g. Pinkston 92, Schenfeld 95]. Part of the research presented herein, published previously in [Waterson 91], was an early proponent of a system-level design focus. This dissertation, along with other previously published parts of this work [Waterson 94a, Waterson 94b, Waterson 95], contribute further support and advocacy of a system-level design emphasis for hybrid optical/electronic systems.

Chapter 2

SYSTEM ARCHITECTURE

Optimizing the balance of compromises required for physical implementation of a shared memory model machine is the driving force behind the Shared Memory Optical/Electronic Computer (SMOEC) architecture. The SMOEC is designed to satisfy a number of desired features, as summarized in Table 2.1. These desired features were chosen knowing that a passive optical interconnection network would be employed in the SMOEC. Thus the characteristics of the optical interconnection network can be maximally exploited by the overall SMOEC architecture, and at the same time the overall architecture accomodates the weaknesses of the optical interconnection network with minimal effect on computational performance.

2.1 SMOEC Architecture Overview

The functional architecture of the SMOEC is shown in Figure 2.1. It consists of a bank of N processing elements (PEs) connected to a bidirectional optical

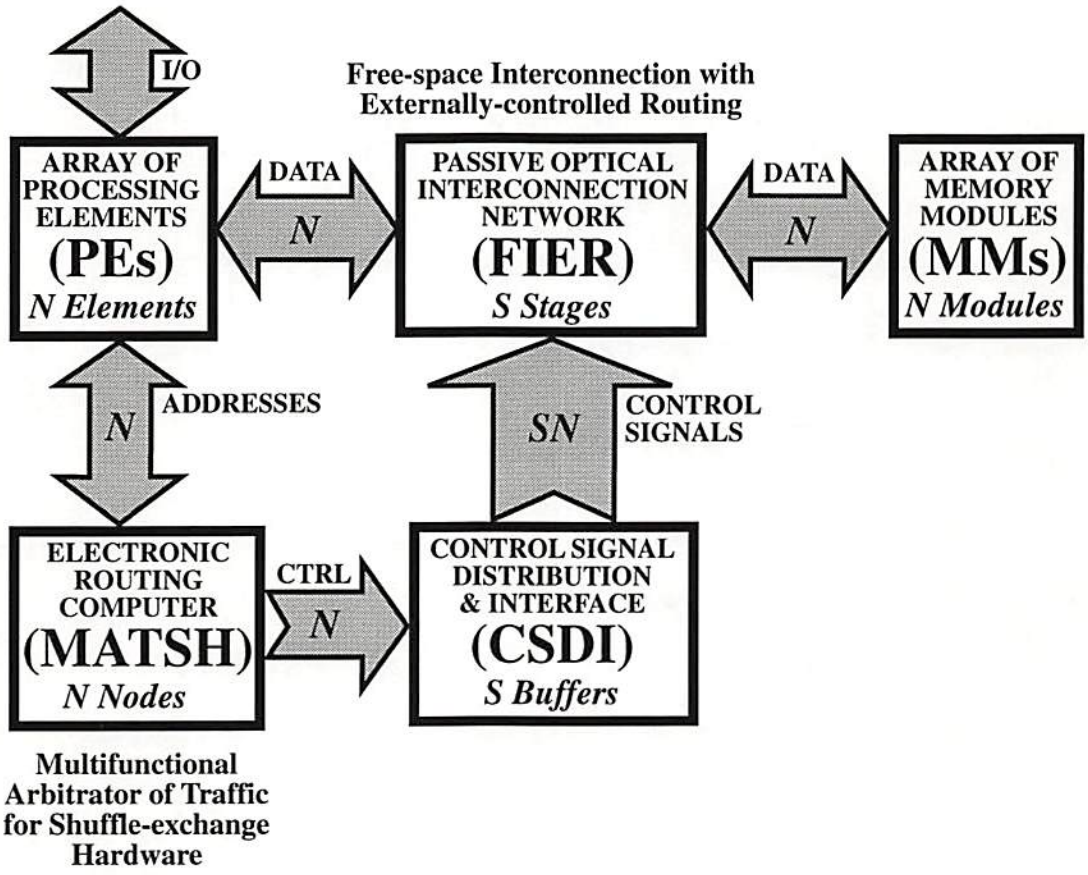


Figure 2.1: SMOEC: Block Diagram. Labels on arrows indicate number of communication lines.

- ▷ Shared memory computation model as architectural basis
- ▷ MIMD (Multiple Instruction, Multiple Data stream) operation
- ▷ Physically parallel (not buffered), simultaneous memory reads and writes
- ▷ Memory access completion within $\mathcal{O}[1]$ passes through the network (A network pass may take $\mathcal{O}[\log N]$ time)
- ▷ High throughput between any PE (or set of PEs) and MM
- ▷ Parallel fetch-and-add (see Appendix 6.3) capability (to restricted MM cells)
- ▷ Prevention of hot spot formation
- ▷ Fine-grained parallelism ($N \sim 10^3 - 10^6$)

Table 2.1: Desired Features for the SMOEC.

interconnection network called the Free-space Interconnection with Externally-controlled Routing (FIER), which is connected at its other end to a bank of N Memory Modules (MMs). The FIER optical network consists of a sequence of S shuffle–exchange stages, where $S = \mathcal{O}[\log N]$. The bank of PEs is also connected to the inlet links of a single recirculating electronic shuffle–exchange stage. This is the Multifunctional Arbitrator of Traffic for Shuffle/exchange Hardware (MATSH) which processes address bits from the PEs to compute the control settings for the FIER switches. The Control Signal Distribution & Interface (CSDI) buffers the control bits until it is time to switch the FIER optical network to accomodate new routing paths. The optical switches are then all set at the same time, opening the FIER for bidirectional communication between connected PEs and MMs. In order to achieve high bandwidth data transfer, the FIER is designed to be optically

passive, and to use near-term optical technology. The choice of passive optical hardware has substantial architectural and control implications.

To illustrate a typical mode of operation and to introduce how the SMOEC subsystems interact, a simplified overview of the handling of simultaneous read requests is provided. Communications in the SMOEC are handled in separate phases, each consisting of a single type of request (such as read or write) to facilitate conflict resolution. Because of these globally coordinated communication phases, the SMOEC is not a pure MIMD (Mulitple Instruction, Multiple Datastream) computer in the most strict sense, but is considered a "primarily MIMD" computer. During the read phase, each PE that requires information from a MM forms a read request consisting of the MM index (address) it desires to read from. These addresses are fed in parallel to the MATSH control unit. Using the routing algorithm (explained in Chapter 5) , the MATSH sequentially computes the appropriate switch settings for the FIER optical network and passes them to the CSDI buffering unit. When all settings are computed, all bits are sent simultaneously from the CSDI to the appropriate FIER switches. Once the FIER switches are set, two way communication channels are established between the PEs and the desired MMs (though only one way communication is used for reading). The MMs then each simultaneously transmit their entire contents serially as a stream of intensity modulated light. The optical signals pass through the FIER network, with fan-out as necessary, to the requesting PEs. Fan-out is required to satisfy multiple PE requests of the same MM.

The specific types of hardware selected to implement the SMOEC were chosen with the intention of feasible implementation in the near term. As previously

discussed, the hardware is divided into five subsystems: electronic Processing Elements (PEs), electronic Memory Modules (MMs), an optical interconnection network (FIER), an electronic routing control unit (MATSH), and an electronic Control Signal Distribution & Interface (CSDI). Units listed as electronic actually include an electronic/optical interface, and thus must be considered as having a partially hybrid composition. The architecture of this system is shown in Figure 2.2. An overview of the functionality of each of the units is presented in the following sections. Later chapters will examine the theory and design of the major subsystems in greater detail.

2.2 Processing Elements (PEs)

The PEs are intended to be simple electronic processors, such as a basic microprocessor. The complexity of individual PEs is a design parameter for the SMOEC; the architecture permits flexibility in this aspect. Located with each PE is a local memory at least equivalent to the size of a MM.

Each processor must have two types of Input/Output (I/O): electrical and optical. The optical I/O to the PEs is implemented by fiber optic techniques, one transmitter and receiver per PE. Since no addressing or multiplexing of resources (I/O signals) is needed, a common interface bottleneck is avoided. Optical I/O (laser sources and detectors) is used to send data to and receive data from the FIER optical network. The electrical I/O is used to send addresses (and sometimes small amounts of data; see Appendix 6.3) to the MATSH control unit to compute the switch settings for the FIER.

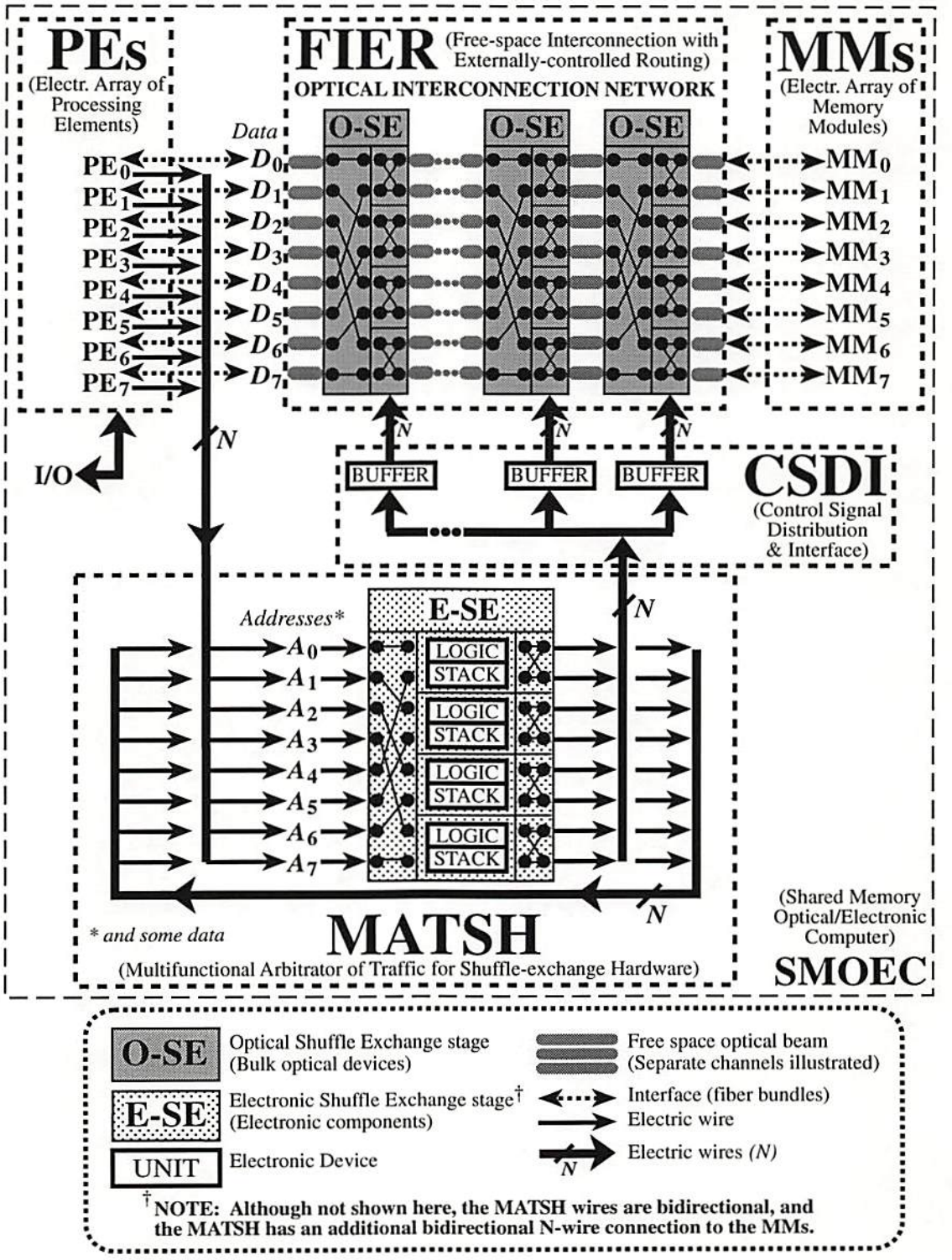


Figure 2.2: SMOEC: Architecture.

2.3 Memory Modules (MMs)

The size of the electronic MMs is somewhat dictated by the architecture. Since the read and write processes operate on the entire MM contents, the MM size must not be too large. However, they must be of sufficient size for useful computation to be practical. MM size has a strong impact on the selection of values for other design parameters (as discussed in Section 7.1.1). The Memory Modules will work best if each is conceived of as a large recirculating shift register, capable of reading and writing data quickly in sequence throughout its entire capacity. A small amount of random-access memory is also included in each MM, to provide locations for fetch-and-add operations (see Appendix 6.3). In the same manner as for PEs, the MMs also have both optical and electronic I/O to the FIER optical network.

Although electronic MMs are envisioned in the near term, optical technologies may provide improved performance in the longer term. Examples include optical disk memory with multiple write heads and parallel optical readout [Lee 88, Psaltis 90], as well as volume holographic media [Lee 88, Li 92, Li 95]. However, in the current design, optical intensity is sent into the FIER optical network after being serially modulated by the electronically-stored information in the MMs.

2.4 Optical Interconnection Network (FIER)

The optical interconnection network, called the Free-space Interconnection with Externally-controlled Routing (FIER), is a novel design based on passive optical switching using polarization-based routing. The FIER optical network features a broadcast/combine capability and employs an external circuit-switched control

system. The topology and theoretical underpinnings of the FIER are elaborated in Chapter 3, and the optical design of the FIER and the combination of optical hardware used to implement it are explained in detail in Chapter 4. The self-contained nature of the FIER means that it may also be used as a component in systems other than the SMOEC.

A fundamental choice in the design of an optical interconnection network is active vs. passive switching. In an active switch, the switch state must be physically set upon the arrival of each data bit. Therefore a switch delay is incurred before the corresponding output data bit is generated. In a passive switch, the switch state is set beforehand by a separate control signal, then data arriving at the switch pass through to the preselected outlet link without any switching delays. Passive switch connections effectively act as if a direct wire from inlet link to outlet link had been installed inside the switch. Passive optical switching was selected for the FIER to permit high speed data throughput in the network. Passive switching also enables circuit-switched (see Section 2.5) control of the network, which permits additional functionality to be incorporated in the control algorithms. However, decreased optical power after each stage in the FIER is a tradeoff that must be considered when implementing a passive network. This limits the number of optical stages that may be cascaded without regeneration of the optical power.

A shuffle–exchange interconnection network topology [Stone 71, Parker 80] was chosen for the FIER since it provides hardware simplicity (ease of optical implementation) while still exhibiting sufficient topological generality. The shuffle–exchange stages are incorporated in an Extended Generalized Shuffle (EGS) network [Richards 91a] which was chosen because of its ability to provide nonblocking

access between the sets of PEs and MMs. The bypass/exchange switches designed for use in the FIER are capable of the usual "bypass" and "exchange" settings, as well as the extended "combine" and "broadcast" settings. The extended switch settings are not often used by other shuffle–exchange network designs. In the FIER, these extended settings are used to allow data to be routed together when appropriate. Because of this enhanced switch capability, write request combining and broadcast of MM contents to multiple requesting PEs may be performed by the FIER.

The FIER is implemented using classical free-space optics with cascaded shuffle–exchange stages consisting of polarizing prisms, fixed waveplates, bulk lenses, lenslet arrays, and optically addressed polarization-switching ferroelectric liquid crystal Spatial Light Modulators (SLMs). The write beams to the SLMs carry the control bits to set the SLM pixels (individual optical switches) to individually modify the polarization of individual incoming optical data channels. These SLM write beams are shown as electonic in Figure 2.2, however, they could instead be optical write beams. The read beams (data channels) carrying intensity-modulated data receive a polarization change (or no change) which allows the data channels to be sent in different directions (routed) using polarizing prisms. The CSDI buffering unit sends its control bits either electronically or optically in parallel to the SLMs in the FIER, so that the network is reconfigured by switching all pixels in all the SLMs simultaneously. The optical switch states in the FIER may be maintained by utilizing bistability of some SLM devices, or by using the memory in the CSDI buffering unit to refresh the SLM states.

The interfaces between the optical and electronic signals are designed to be fully parallel (no addressing schemes) to avoid bottlenecks at these conversion points. Optical fibers are used to transfer signals from the FIER to the PEs and MMs, and vice versa. Bundles of optical fibers are used to format the signals into pixel arrays for entry and exit to the FIER. These bundles allow the fibers to be unbundled at their opposite end (connecting to the PEs or MMs) to allow I/O to individually controlled, physically separate devices. In this manner, the PEs (and MMs) may be located on separate electronic component boards, with one or more PEs (or MMs) per board, and with each PE (or MM) having its own laser diode and detector. Thus the data rate through the passive FIER is primarily limited by the speeds of the lasers and detectors. However, noise sources due to the various optical components may be additional limiting factors on the data throughput rate. The actual data rate through the FIER may be treated as a design parameter.

Several elements in the list of desired parallel computer features (Table 2.1) motivate the choice of the optically-implemented FIER over an electronic interconnection network for the SMOEC. The passive nature of the FIER permits high speed PE$\leftrightarrow$MM throughput. The ability of the FIER to combine and broadcast messages permits physically parallel simultaneous memory reads and writes at these high data rates, which contributes strongly to its utility as the central subsystem (when teamed with the MATSH, described below) of a shared memory architecture. Fine-grained parallelism is facilitated through the use of bulk optics and optical devices with simple replicated structures such as lenslets, and through the reduced complexity (as compared with electronics) due to the three-dimensional nature and superposition principle of optics (optical beams may pass

through each other unaffected). Electronic interconnection networks designed for shared memory machines were not designed with physically parallel data throughput as a goal, nor is their complexity well suited for application to fine-grained parallelism. The SMOEC was designed to incorporate the FIER as an essential element from the very beginning of its design, so that a novel type of shared memory machine with different strengths and improvements based on the advantages of optics would be the result.

2.5 Electronic Routing Control Unit (MATSH)

Passive switching elements require control bit computation to be carried out physically separate from data passing through the switches. Non-buffered networks may not store any data within a switching node. Therefore, the FIER is circuit switched. Circuit switching consists of setting a dedicated communication channel between a source and destination pair for the duration of a communication cycle. All switches in a circuit-switched network are set simultaneously. In contrast, in packet switching there is no dedicated channel; switches are individually set (using only local information within the switch) upon the arrival of a data block at each intermediate switching node.

The Multifunctional Arbitrator of Traffic for Shuffle–exchange Hardware (MATSH) was designed to implement circuit switching control for the FIER optical network. It was also designed to perform additional data combining and arbitration functions that the passive optical data paths cannot.

Within the MATSH is an electronic implementation of a single shuffle pattern and an array of bypass/exchange switches, forming a single Electronic Shuffle–Exchange stage (E-SE). This single stage, with the addition of bidirectional feedback connections that connect its outlet and inlet links, can mimic the operation of the FIER optical network. Electronic I/O to this single stage is provided to the PEs, the CSDI buffering unit (and the MMs). (The MATSH$\leftrightarrow$MM I/O is used solely for implementing fetch-and-add, as described in Appendix 6.3.) This setup allows the MATSH to mimic the operation of the FIER optical network, albeit at a lower bandwidth, as described in Chapter 5.

The bypass/exchange boxes in the MATSH are required to be capable of performing write request combining and arbitration (as discussed in Section 6.2) in addition to computing control bits for the FIER. Write request combining and arbitration require that each electronic node be capable of performing a "process-and-exchange" function, in which the action performed during the "process" part involves additional computation steps that are dependent on which communication phase is being carried out. Each bypass/exchange switch node in the MATSH therefore incorporates additional logic hardware and some local memory to allow it to perform such operations as data comparisons, data additions, bit cycling, **AND**, **OR**, and storage of a limited amount of data. The data storage is provided by a register stack of depth $\mathcal{O}[\log N]$.

2.6 Control Signal Distribution & Interface (CSDI)

The Control Signal Distribution & Interface (CSDI) is responsible for sending the optical switch settings to the appropriate locations in the FIER optical network at the appointed time. Electronic buffers store the control information until all control signals are present, then it is transmitted simultaneously to the FIER so that the optical devices may all switch at once. Simultaneous setting of all optical switches reduces the amount of delay introduced by the optical array (SLM) switching times (which are relatively slower than electronic switching times).

Chapter 3

INTERCONNECTION NETWORK TOPOLOGY AND ROUTING (REVIEW)

The interconnection network employed in the SMOEC is required to be capable of performing arbitrary connection patterns between the inlets and outlets of the network. The connection patterns required by the SMOEC involve many-to-one connections in the PE→MM direction, which perform conversely as one-to-many connections in the MM→PE direction. This chapter elucidates the topology underlying the FIER optical network, and provides its theoretical underpinnings. Although this chapter is primarily a review of relevant work by other research groups, a few new points and results are introduced.

Note on notation: Many interconnection parameters are frequently used both directly and as logarithms. To facilitate the presentation and use of these parameters, this report employs the convention that lowercase parameters are the

logarithm (usually base 2) of the corresponding uppercase parameters. Occasionally uppercase script letters will be used to differentiate related parameters that do not follow this convention.

When discussing interconnection networks it is important to consider their capability to satisfy multiple connection requests. This capability is discussed in terms of the blocking of network connections. This terminology varies somewhat in its usage, so the particular terminology employed herein (adapted from [Hinton 93a]) is summarized below, in order of increasing capability.

Partial Access Networks have limited function; larger networks are often built from smaller, partially connected network pieces.

Full Access Networks provide the connectivity such that any inlet can be connected to any outlet. These networks can be blocking. In these networks there is at least one path from any inlet to any outlet.

Rearrangeably Nonblocking Networks are those in which any idle inlet may be connected to any idle outlet provided that we may rearrange existing connections. They are capable of any permutation of inlets to outlets.

Wide-Sense Nonblocking Networks have some paths available through the network that may block other connections, but if certain rules are adhered to, the network may operate as if it were strictly nonblocking. For example, crossbar networks may be wide-sense nonblocking (instead of strictly nonblocking) due to the existence of "pathological" paths, which are paths that snake through multiple switching nodes and consequently may block other connections.

Strictly Nonblocking Networks are those in whicn any idle inlet may be connected to any idle outlet, regardless of how many other connections are already established and regardless of the specific paths used by the other connections.

Redundant Strictly Nonblocking Networks have more than enough elements to be strictly nonblocking; these networks are designed to maintain an acceptable level of connectivity even when elements within the network fail.

3.1 Shuffle–Exchange Topology

As previously mentioned, a shuffle–exchange interconnection network topology [Stone 71, Parker 80] was chosen for the FIER optical network since it provides hardware simplicity (ease of optical implementation) while still exhibiting sufficient topological generality. A basic shuffle–exchange network is a multistage cascadable network, where each stage consists of a "perfect shuffle" followed by an array of bypass/exchange switches. More general shuffle–exchange networks may be composed of $a \times b$ switching nodes interconnected by a generalized version of the perfect shuffle, called a Q-shuffle. The resulting "shuffle–exchange" stage is a building block for several types of networks. The parts of shuffle–exchange networks are detailed below.

3.1.1 The Perfect Shuffle

The perfect shuffle is named after the common riffle shuffle used when playing cards in which a deck of cards is divided in half and combined by interleaving cards from

each deck half. A "perfect" shuffle is therefore one in which the deck is first divided precisely in half, and then single cards are exactly interleaved. Examination of this process on a linear array of nodes with binary indices (see Figure 3.1) reveals that information is sent to the node whose index is the left cyclic shift of the source node. That is, given $N = 2^n$ inlet nodes, if the index of an inlet node i is represented as a sequence of n binary digits (bits) b_j, as:

$$i = (b_{n-1}\, b_{n-2}\, \cdots\, b_1\, b_0), \qquad 0 \leq i \leq N-1 \tag{3.1}$$

where

$$i = \sum_{j=0}^{n-1} b_j\, 2^j, \quad \text{with } b_j \in \{0,1\}, \tag{3.2}$$

then the perfect shuffle performs the operation $\mathcal{SH}_N(i)$, where:

$$\mathcal{SH}_N(i) = (b_{n-2}\, \cdots\, b_1\, b_0\, b_{n-1}) \tag{3.3}$$

$$= \textbf{LEFT-CYCLICAL-SHIFT}(i) \tag{3.4}$$

3.1.2 Q-Shuffles

A Q-shuffle of $N = QR$ playing cards can be defined as follows (from [Patel 81]). Divide the deck of QR cards into Q piles of R cards each; top R cards in the first pile, next R cards in the second pile, and so on. Now pick the cards, one at a time from the top of each pile; the first card from top of pile one, second card from the top of pile two, and so on in a circular fashion until all cards are picked up. This new order is defined as the Q-shuffle of the QR cards. Observe that a perfect shuffle is a 2-shuffle in this notation. An 8-shuffle is illustrated in Figure 3.2.

The Q-shuffle of $N = QR$ objects performs the operation $Q\text{-}\mathcal{SH}_N(i)$ on an object with index i. This can be expressed mathematically in two ways [Patel 81], either

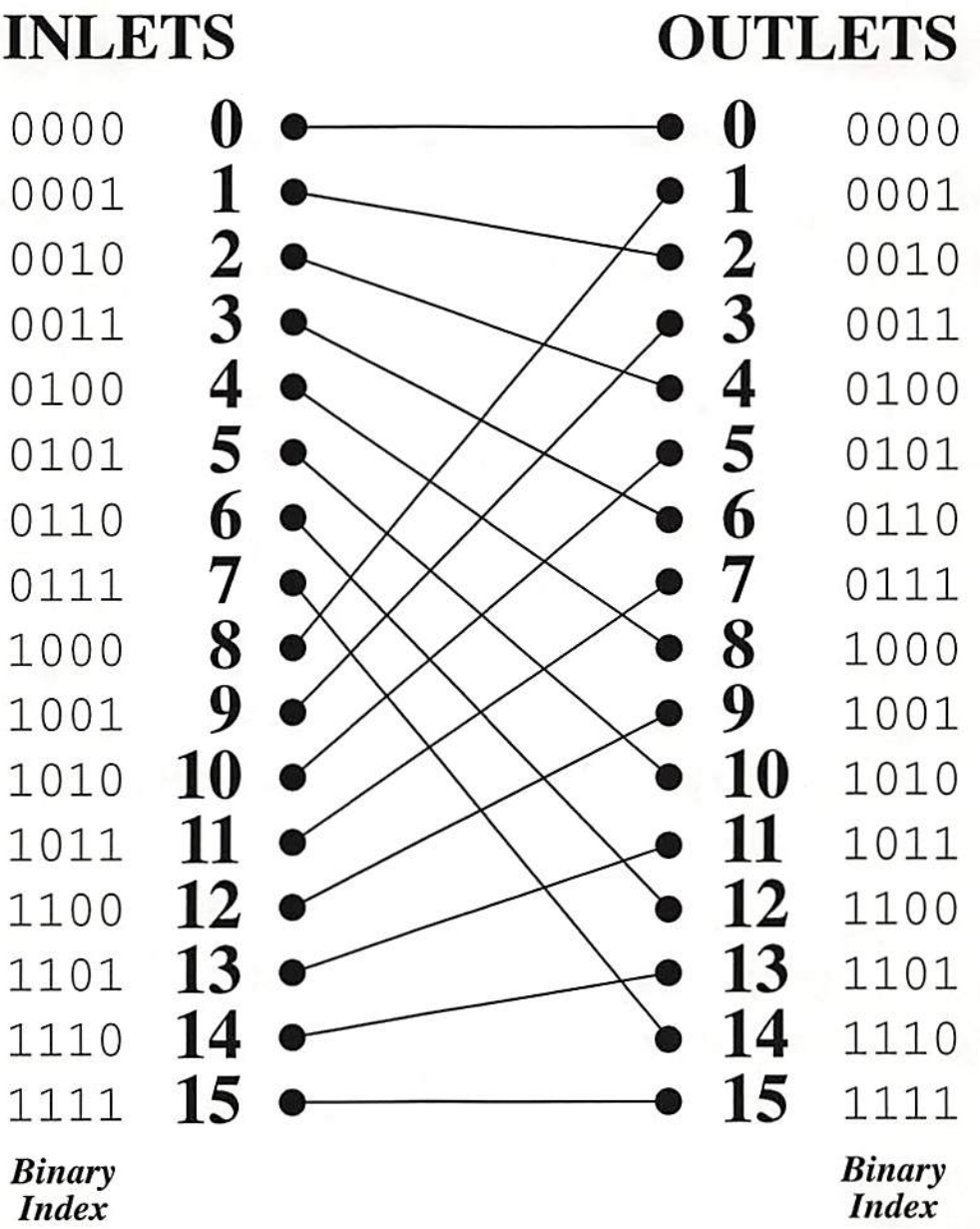


Figure 3.1: A Perfect Shuffle.

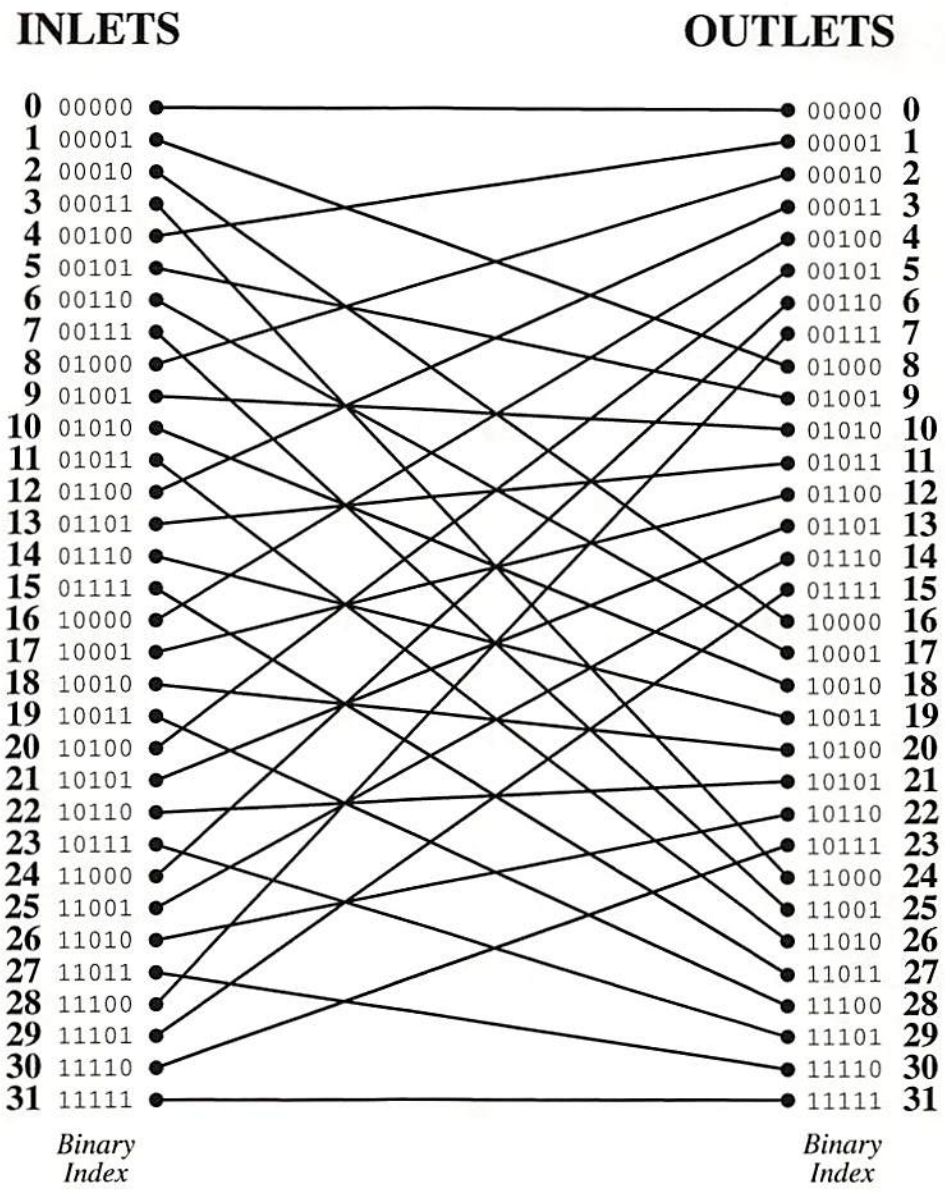


Figure 3.2: An 8-shuffle on 32 nodes.

as

$$Q\text{-}\mathcal{SH}_N(i) = \begin{cases} i\,Q \bmod (QR-1) & 0 \le i < QR-1 \\ i & i = QR-1 \end{cases} \tag{3.5}$$

or as

$$Q\text{-}\mathcal{SH}_N(i) = \left(i\,Q + \left\lfloor \frac{i}{R} \right\rfloor\right) \bmod QR. \qquad 0 \le i \le QR-1 \tag{3.6}$$

where $\lfloor u \rfloor$ is the *floor* function (the greatest integer $\le u$), and $u \bmod v$ is the modulo operation (the remainder when u is divided by v). .

3.1.2.1 b-Shuffle on b^k objects

For the case of $N = b^k$, the b-shuffle has a simple form [Patel 81]. If the index of an inlet node i is represented as a sequence of k base-b digits d_j, as:

$$i = (d_{k-1}\; d_{k-2} \cdots d_1\; d_0)_b, \qquad 0 \le i \le b^k - 1 \tag{3.7}$$

where

$$i = \sum_{j=0}^{k-1} d_j\, b^j, \quad \text{with } d_j \in \{0, \ldots, b-1\}, \tag{3.8}$$

then the b-shuffle performs the operation $b\text{-}\mathcal{SH}_{b^k}(i)$, where:

$$b\text{-}\mathcal{SH}_{b^k}(i) = (d_{k-2} \cdots d_1\; d_0\; d_{k-1})_b \tag{3.9}$$

$$= \textbf{LEFT-CYCLICAL-SHIFT}_b(i). \tag{3.10}$$

Thus the b-shuffle of an index is a left cyclical shift of the index digits represented in base b.

3.1.2.2 Q-Shuffle on c^n objects

If $N = c^n$ a special result for the decomposition of the Q-shuffle may be obtained. (To the author's knowledge neither this general result nor the special case for $c=2$

have been previously published, so the derivation is presented here). In this case (it is necessarily true that) $Q = c^q$ and $R = c^r$. (However it is not necessarily true that there exists a k such that $N = Q^k$, so the result from Section 3.1.2.1 does not directly apply.)

First, represent the index i as a sequence of base-c digits a_j, as:

$$i = (a_{n-1}\, a_{n-2} \cdots a_1\, a_0)_c, \qquad 0 \le i \le c^n - 1 \tag{3.11}$$

where

$$i = \sum_{j=0}^{n-1} a_j\, c^j \quad \text{with } a_j \in \{0, \ldots, c-1\}. \tag{3.12}$$

Observe that for the simpler case of a c-shuffle on the $N = c^n$ objects, the result from Section 3.1.2.1 does apply, so using Equation 3.10 yields that the c-shuffle performs the operation $c\text{-}\mathcal{SH}_{c^n}(i)$, where:

$$c\text{-}\mathcal{SH}_{c^n}(i) = (a_{n-2} \cdots a_1\, a_0\, a_{n-1})_c \tag{3.13}$$

$$= \textbf{LEFT-CYCLICAL-SHIFT}_c(i). \tag{3.14}$$

Thus the c-shuffle operation on an index i is a left cyclical shift of the index digits represented in base c.

Now, returning to the general case of a Q-shuffle on $N = c^n = QR$ objects, plug $Q = c^q$, $R = c^r$, and $n = q+r$ into the Q-shuffle definition, equation 3.6, which yields:

$$Q\text{-}\mathcal{SH}_{c^n}(i) = \left(i\, c^q + \left\lfloor \frac{i}{c^r} \right\rfloor\right) \bmod c^n \qquad 0 \le i \le N-1 \tag{3.15}$$

$$\triangleq (\mathcal{U} + \mathcal{V}) \bmod c^n. \tag{3.16}$$

Now observe that, for i represented as a sequence of n base-c digits as in equations 3.11 and 3.12,

$$\begin{aligned}
\mathcal{U} &= i\,c^q && (3.17)\\
&= i \textbf{ LEFT-SHIFTED-BY } q \text{ digit-positions} && (3.18)\\
&= (a_{n-1}\,a_{n-2}\,\cdots\,a_1\,a_0\,\underbrace{0\,0\,\cdots\,0\,0}_{q \text{ zeros}}), && (3.19)
\end{aligned}$$

and

$$\begin{aligned}
\mathcal{V} &= \left\lfloor \frac{i}{c^r} \right\rfloor && (3.20)\\
&= i \textbf{ RIGHT-SHIFTED-BY } r \text{ digit-positions} && (3.21)\\
&= (\underbrace{a_{n-1}\,a_{n-2}\,\cdots\,a_r}_{q \text{ digits}}), && (3.22)
\end{aligned}$$

so

$$\mathcal{U}+\mathcal{V} = (a_{n-1}\,a_{n-2}\,\cdots\,a_1\,a_0\,\underbrace{a_{n-1}\,a_{n-2}\,\cdots\,a_r}_{q \text{ digits}}), \qquad (3.23)$$

which yields

$$\begin{aligned}
(\mathcal{U}+\mathcal{V}) \bmod c^n &= \textbf{RIGHTMOST-}n\textbf{-DIGITS of } (\mathcal{U}+\mathcal{V}) && (3.24)\\
&= (\underbrace{a_{r-1}\,\cdots\,a_1\,a_0}_{n-q \text{ digits}}\,\underbrace{a_{n-1}\,a_{n-2}\,\cdots\,a_r}_{q \text{ digits}}), && (3.25)
\end{aligned}$$

and thus

$$\begin{aligned}
Q\text{-}\mathcal{SH}_{c^n}(i) &= \textbf{LEFT-CYCLICAL-SHIFT}_c^q(i) && (3.26)\\
&= c\text{-}\mathcal{SH}^q_{c^n}(i). && (3.27)\\
& && (3.28)
\end{aligned}$$

Thus the Q-shuffle operation on $N = c^n$ nodes (with $Q = c^q$ identified) is equivalent to q successive applications of the c-shuffle operation.

3.1.2.3 Q-Shuffle on 2^n objects

The case of a Q-shuffle on 2^n objects is a special case of the general result derived in the previous section. However, it is worth particular attention, because the FIER optical network is designed to have $N = 2^n$ nodes. In this case, the digits a_j in the equations become binary digits (bits), and the general result given in Equation 3.28 simplifies to the following:

$$Q\text{-}\mathcal{SH}_{2^n}(i) = \textbf{LEFT-CYCLICAL-SHIFT}^q(i) \tag{3.29}$$

$$= \mathcal{SH}^q_{2^n}(i). \tag{3.30}$$

$$\tag{3.31}$$

Thus the Q-shuffle operation on $N = 2^n$ nodes when $Q = 2^q$ is equivalent to q successive applications of the perfect shuffle operation.

As an example of this $c=2$ case, the 8-shuffle shown in Figure 3.2 is now seen to be equivalent to 3 applications of the perfect shuffle (since $8=2^3$), as illustrated in Figure 3.3. That is, for

$$i = (b_4\, b_3\, b_2\, b_1\, b_0), \qquad 0 \leq i \leq 31 \tag{3.32}$$

where b_j are binary digits (bits), have

$$8\text{-}\mathcal{SH}_{32}(i) = (b_1\, b_0\, b_4\, b_3\, b_2). \tag{3.33}$$

3.1.3 Bypass/Exchange Switch

The basic definition of a bypass/exchange (see Figure 3.4) switch is one with two settings that allow two inputs to pass straight through ("bypass"), or swap positions ("exchange"). Extensions to the bypass/exchange switch shown in Figure 3.4

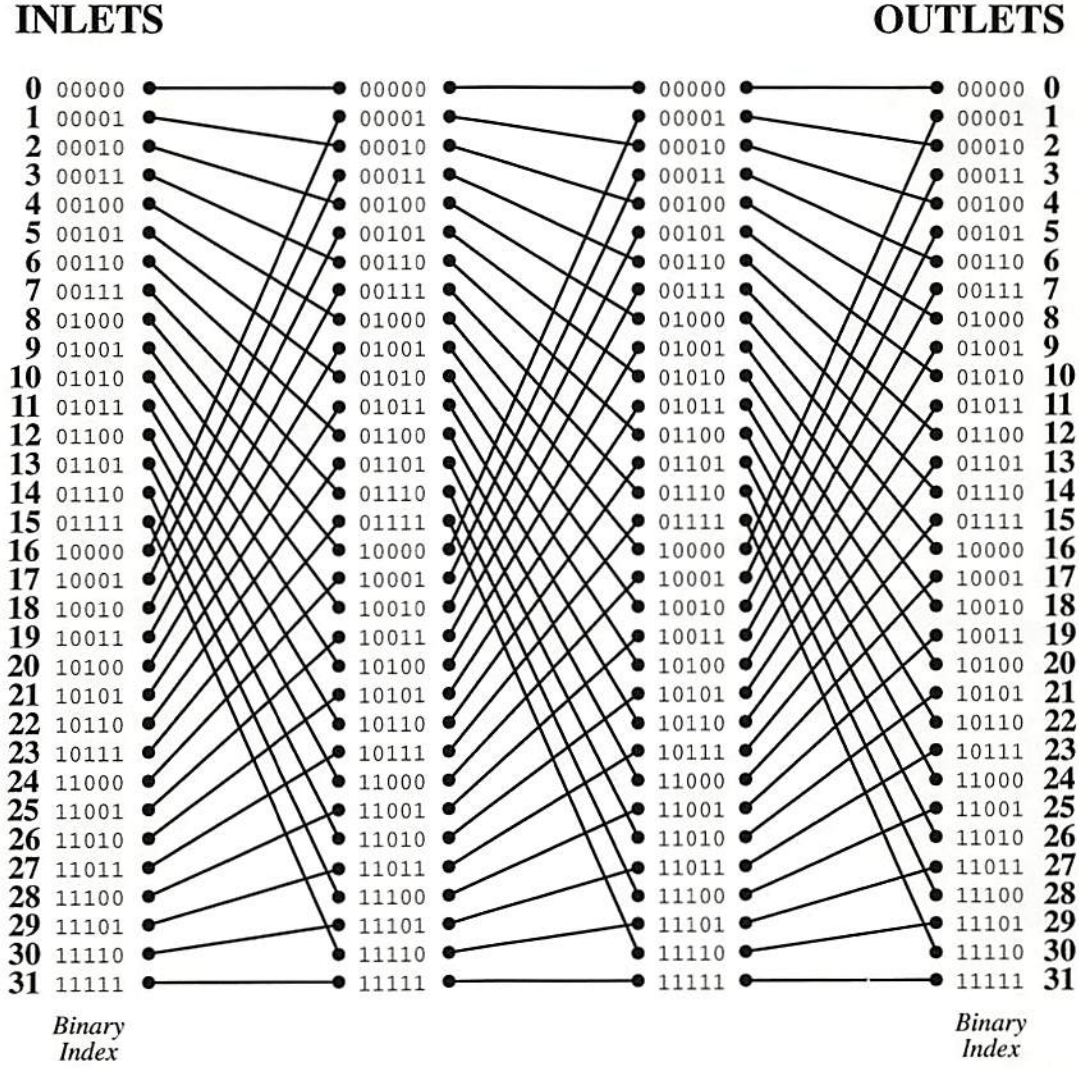


Figure 3.3: Three consecutive perfect shuffles on 32 nodes.

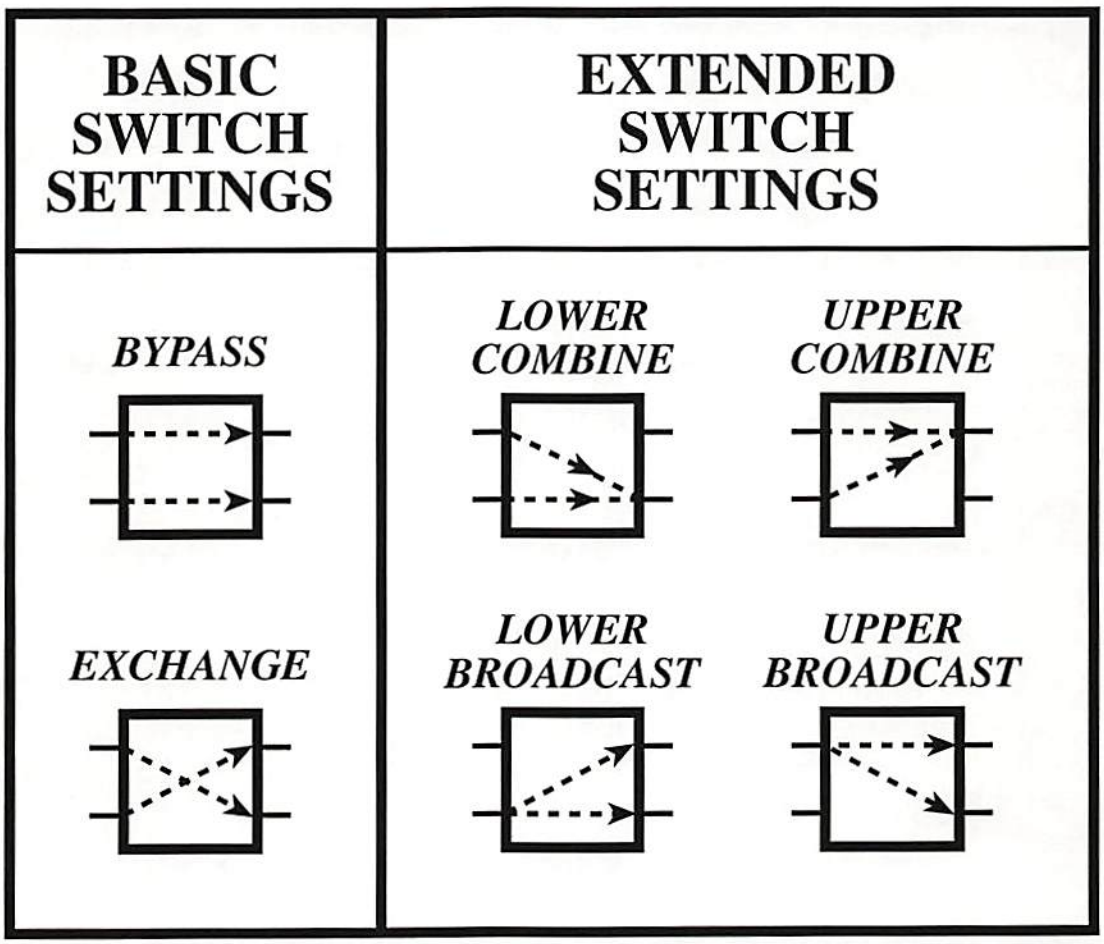


Figure 3.4: Bypass/Exchange Switch Settings.

include upper and lower broadcast and upper and lower combine. These extended states are each implemented unidirectionally in the FIER optical network, allowing combine states solely in the "forward" direction (the processor-to-memory direction when used in the SMOEC) and broadcast states solely in the "reverse" direction. Note that a bypass/exchange switch with inlet link indices $2i$ and $2i+1$ routes each of its two inputs to an outlet link with either the identical index, or an index that is identical except that the last bit is inverted. For i represented by binary digits (bits) as in equations 3.1 and 3.2, the exchange operation on i can be expressed as

$$\mathcal{EX}(i) = (b_{n-1}\, b_{n-2} \cdots b_1\, \overline{b_0}). \tag{3.34}$$

3.2 Shuffle–Exchange Networks

The shuffle–exchange topology is popular in the literature because it is flexible and well characterized. Many types of networks may be designed utilizing this topology. Those networks relevant to an understanding of the FIER optical network are presented in the following sections.

3.2.1 Ω Networks

The simplest form of a shuffle–exchange network is the omega (Ω) network [Parker 80], as illustrated in Figure 3.5. An Ω network with $N = 2^n$ inlet nodes and N outlet nodes is defined as a sequence of $n = \log_2 N$ shuffle–exchange stages (without the extended bypass/exchange states). The binary index of each node contains $n = \log_2 N$ bits, so $\log_2 N$ perfect shuffles will return all information from N nodes to their original location. If, however, the exchange of adjacent nodes is

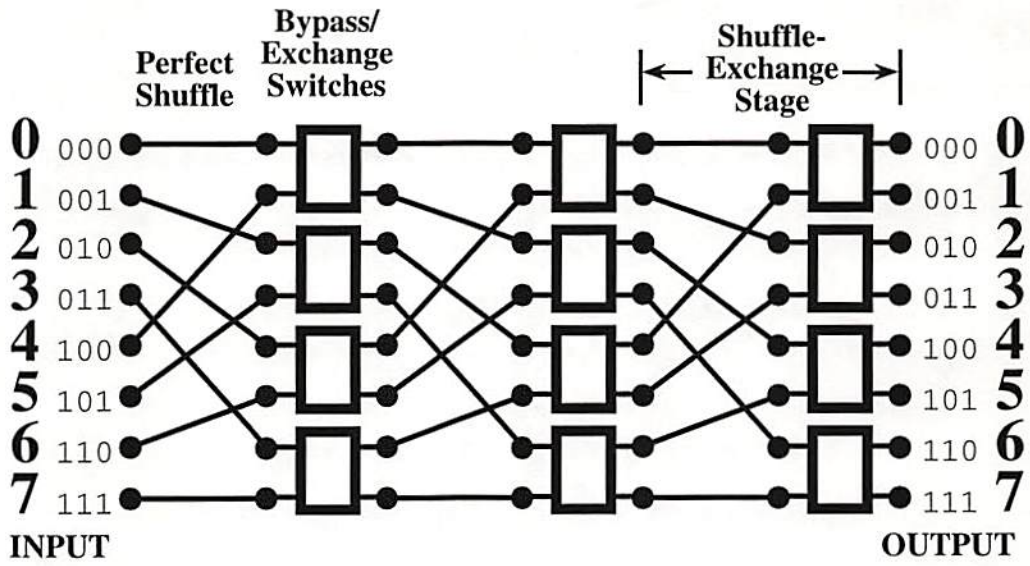


Figure 3.5: An Omega Network. Illustrated for $N = 8$ nodes.

permitted between shuffles, then it is possible to route information from any source node to any destination node. This is understood by noting that the repeated shuffles consecutively bring each bit of the node address to the Least Significant Bit (LSB), and the exchange function complements the LSB of the node address. This technique provides a very simple and local routing algorithm.

The simple algorithm for an Ω network works fine if only one source node is to be routed. However, if several nodes need to be routed simultaneously, it is likely that conflicts in switch settings will result. From the simple algorithm described above, it is apparent that there is only one route available between a given source-destination pair. A second source-destination pair may require a different setting for a bypass/exchange switch that was used in routing the first pair. Thus, although any individual inlet may be connected to any outlet using $\log_2 N$ shuffle–exchange stages (i.e. it is a full access network), conflicts may arise in the bypass/exchange switch settings, making some permutations of inlets to

outlets unimplementable. The typical solution to such routing conflicts is to buffer requests at the node within the Ω network when a conflict arises. However, as previously mentioned, data buffering is not allowed in the SMOEC. Therefore, a network more capable than an Ω network is used in the FIER optical network.

If the 2×2 nodes in an Ω network are capable of the extended bypass/exchange switch settings listed in Figure 3.4 (with the unidirectional settings of forward combine and reverse broadcast as previously described), then the full-access capability of the Ω network can be used to provide extended access to single nodes. Using forward combine switch settings, any set of inlet nodes may simultaneously communicate with any single outlet node (this assumes that the set of requests from the inlet nodes are combinable, and ignores the possibility of other conflicting communication requests). Conversely, the reverse broadcast behavior of these same switch settings allows the single outlet node to simultaneously communicate with the set of inlet nodes. This ability to incorporate a broadcast and combine capability is characteristic of all the shuffle–exchange networks described later in this chapter as well.

3.2.2 Delta Networks

The Ω network is a subset of a class of networks called delta networks [Patel 81]. Delta networks are full access networks with a very general class of link patterns between stages. Although delta networks provide increased topological flexibility over that of Ω networks, they have essentially the same blocking characterists. A delta network is defined to be an $a^k \times b^k$ switching network with k stages, in which

each stage consists of $a \times b$ switches (crossbar modules).* The interconnection pattern used in the delta network consists of either a a-shuffles between consecutive stages or any (topologically equivalent) set of patterns which will result in a full-access network. The special case of N-inlet/N-outlet delta networks ($N = b^k$) comprising k stages interconnected with b-shuffles is summarized here.

Routing in a k-stage N-inlet/N-outlet delta network (where $N = b^k$) can be expressed mathematically as follows (from [Patel 81]). If the destination D is expressed in base-b notation as:

$$D = (d_{k-1}\, d_{k-2} \cdots d_1\, d_0)_b, \qquad 0 \le i \le b^k - 1 \tag{3.35}$$

where

$$D = \sum_{j=0}^{k-1} d_j\, b^j, \quad \text{with } d_j \in \{0, \ldots, b-1\}, \tag{3.36}$$

then the base-b digit d_i controls the crossbar modules of stage $(n-i)$, thus providing a local routing algorithm for delta networks. A setting of d_i for a given crossbar module means that the the crossbar outlet with index d_i is chosen for the path being routed.

3.2.3 Nonblocking Shuffle–Exchange Networks

One way to increase the number of realizable connection patterns for shuffle–exchange networks beyond those achievable by Ω or delta networks is to increase the number of shuffle–exchange stages beyond the minimum for full access. In an Ω network, for each additional shuffle–exchange stage that is added beyond

*Notational note: the "$\times$" symbol is used to describe individual switches or whole switching networks; a $U \times V$ switch (or switching network) has U inlets and V outlets. This notation looks the same, but is entirely distinct from that used to describe an A by B array as an "$A \times B$ array."

$S = \log_2 N$, the number of paths between a given inlet and outlet is doubled. Researchers have shown that $2\log_2 N - 1$ shuffle–exchange stages are sufficient to implement all possible permutations without conflicts (i.e. to implement a "rearrangeably nonblocking" network) [Abdennadher 92], but the required control algorithm is much more complex [Parker 80, Wu 81] than that for the Ω or delta networks, and is costly to implement.

Another way to increase the number of realizable connection patterns for shuffle–exchange networks is to increase the width of the network. If each inlet passes first through a stage of $1{\times}F$ switches, into a shuffle–exchange network of width $W = N \cdot F$, then finally through a stage of $F{\times}1$ switches, this can increase the number of paths available by a factor of F. The Extended Generalized Shuffle (EGS) networks presented in the next section employ both of these methods to increase the number of realizable connection patterns, thus enabling operation with low probability of blocking, or even nonblocking operation.

3.3 Extended Generalized Shuffle Network Topology

Extended Generalized Shuffle (EGS) networks were invented by Gaylord Richards of AT&T [Richards 91b, Richards 91a]. EGS networks provide extended capability and flexibility beyond that of conventional shuffle–exchange networks (such as Ω or delta networks) by removing restrictions on the specific interconnection patterns used, and they provide a theoretical framework for making tradeoffs of network length and width. Such tradeoffs can be tailored to fit the requirements of a particular implementation, so EGS networks provide a practical framework

for design flexibility. EGS networks are also particularly suited to optical implementation, due to the many optically implementable shuffle-equivalent topologies [Cloonan 94]. EGS networks may be thought of as a generalization of delta networks, having an unrestricted number of stages and variable switching stage compositions (number and size of switches).

3.3.1 EGS Definition

A (fully general) Multistage Interconnection Network (MIN) is shown in Figure 3.6 and is defined by the following five conditions (from [Richards 91a]):

1. A MIN has some arbitrary number S stages of nodes.
2. There are r_i nodes in stage i, each having n_i inlet links and m_i outlet links.
3. Nodes in different stages may have different values of n_i and m_i.
4. For $1 \leq i \leq S-1$, the outlets from nodes in stage i are connected (via links) to the inlets of nodes in stage $i+1$.
5. $r_i m_i = r_{i+1} n_{i+1}$ for $1 \leq i \leq S-1$.

An Extended Generalized Shuffle (EGS) network [Richards 91a] is a MIN with a particular specified link interconnection pattern. Thus, an EGS network is also illustrated by Figure 3.6. The basic EGS interconnection pattern is defined as follows. Stages $i-1$ and i are interconnected by an n_i-shuffle, which effectively assigns outlet links from stage $i-1$ consecutively to switches in stage i. However, an EGS network is defined more broadly to include networks which are topologically equivalent to the basic pattern just given. Thus, EGS networks may be made

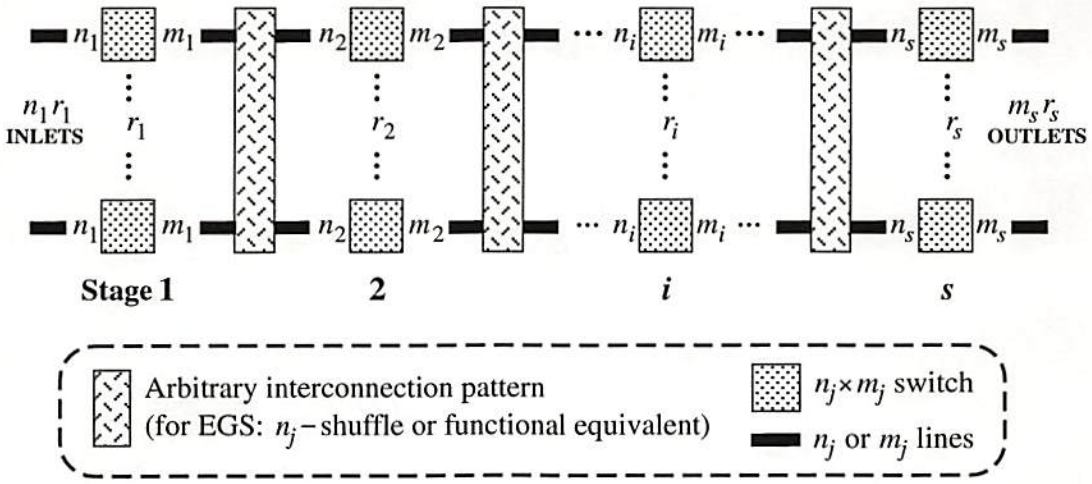


Figure 3.6: Multistage Interconnection Network (MIN) general definition (including EGS network definition).

with [Hinton 93b] the crossover interconnect [Jahns 88] , the banyan interconnect [Wu 80], the n-cube interconnect [Wu 80], the modified data manipulator interconnect [Feng 74], or the flip interconnect [Wu 80]. Therefore all of these networks are subsets of the EGS class of networks.

3.3.2 Regular Simplified EGS (RS-EGS) Definition

In the work presented here, a Regular Simplified class of EGS networks (RS-EGS networks) is considered, as shown in Fig. 3.7. Four simplications of the general EGS definition are employed for RS-EGS networks:

1. There are an equal number (N) of inlets and outlets, which is restricted to be a power of two ($N = 2^n$),

2. The first stage consists of $1 \times F$ switches, and the final stage consists of $F \times 1$ switches,

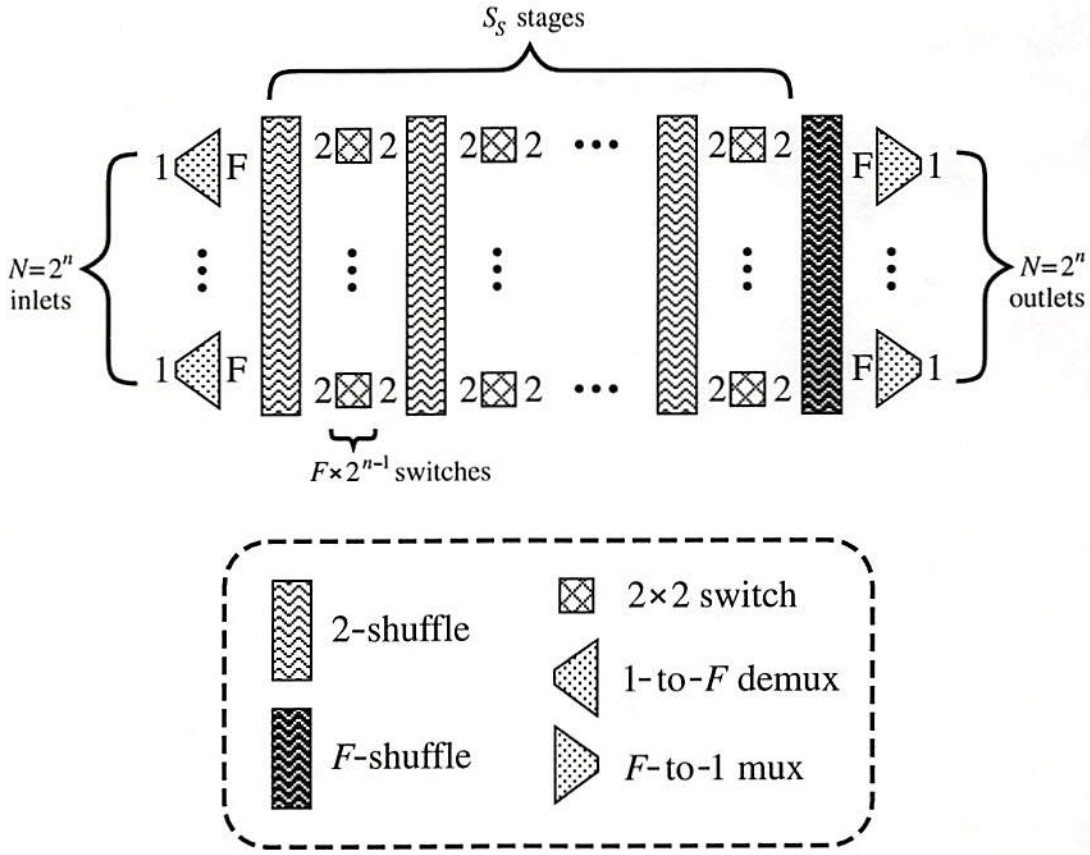


Figure 3.7: Regular simplified class of EGS networks (RS-EGS networks)

3. Each of the S_S stages between the first and last (the "main section") is identical (i.e. the network is homosyndetic), consisting of 2×2 switches.

4. All S_S main section stages are interconnected with 2-shuffles, and the final interconnection from the last main section stage to the F×1 switches is an F-shuffle.

Therefore, in a RS-EGS network the data progress as follows. First, the data from $N = 2^n$ network inlets pass through the stage of $1 \times F$ switches which act as demultiplexers (called the "fan-out" stage), emerging on $W = NF$ data lines. Next, the data pass through the main section of S_S shuffle–exchange stages. Finally, the data pass through an F-shuffle, followed by the stage of F×1 switches which act as a set of F×1 multiplexers (called the "fan-in" stage). Ultimately the data emerge from the "fan-in" stage at the N network outlets.

An example RS-EGS network is shown in Figure 3.8. This example illustrates a special case of RS-EGS networks (used in the SMOEC architecture) in which F is a power of two (the "restricted-F" case: $F = 2^f$). In this case, the "fan-in" and "fan-out" stages may each be implemented by f stages of 1×2 switches or 2×1 switches as shown. Also, when $F = 2^f$ the final F-shuffle before the "fan-in" stage may be implemented by f consecutive ordinary 2-shuffles (as proved in Section 3.1.2.3), however this is not illustrated in Figure 3.8. This may prove advantageous for implementations (such as the FIER optical network) where 2-shuffles are particularly easy to implement.

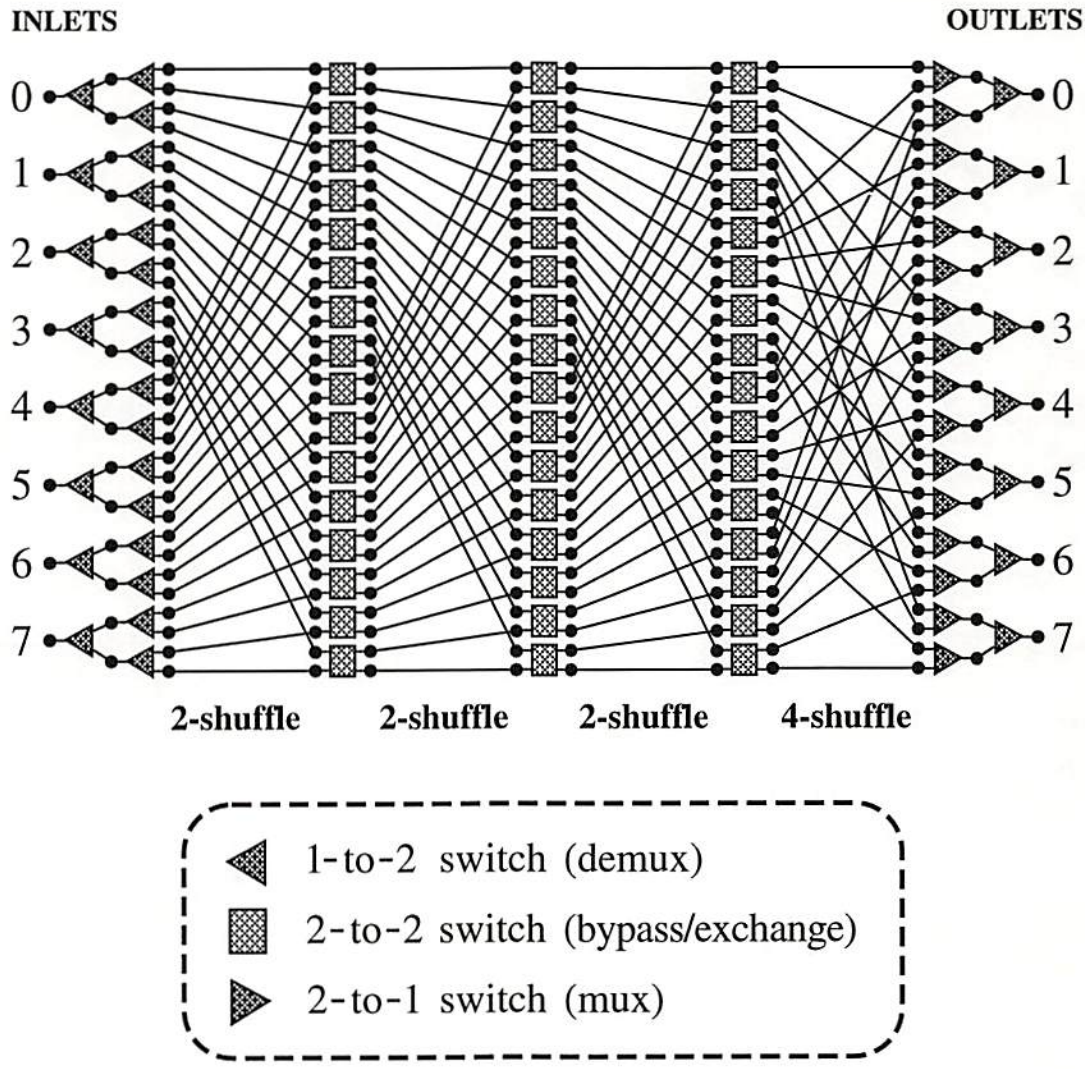


Figure 3.8: RS-EGS network example: $N = 8 = 2^3$, $F = 4 = 2^2$, $S_S = 3$, $W = NF = 32$.

	S_S Even	S_S Odd
$S_S \leq n$	$F \geq 2^{n-S_S}(1.5 \times 2^{\frac{S_S}{2}} - 1)$	$F \geq 2^{n-S_S}(2^{\frac{S_S+1}{2}} - 1)$
$n < S_S \leq 2n-1$	$F \geq 2^{n-S_S}(1.5 \times 2^{\frac{S_S}{2}}) + S_S - n - 1$	$F \geq 2^{n-S_S}(2^{\frac{S_S+1}{2}}) + S_S - n - 1$

Table 3.1: Richards' formulae: nonblocking conditions for $N \times N$ RS-EGS networks (from [Richards 93]).

3.3.3 RS-EGS Theory

Relations between the RS-EGS parameters n, S_S, and F for nonblocking network operation are provided in Table 3.1 (from [Richards 93]). The table lists relations for four cases of S_S, when $S_S \leq 2n-1$. Another parameter of interest in such networks is the number of paths P available between an arbitrary inlet-outlet node pair. For RS-EGS networks, P is independent of which particular inlet and outlet are chosen. P is given by:

$$P = \frac{2^{S_S} F}{N} = F 2^{S_S - n} \tag{3.37}$$

Several special cases deduced from Richards' formulae in Table 3.1 and equation 3.37 are listed in Table 3.2.

3.3.4 RS-EGS Design Parameter Analysis

As the number of stages S_S is increased from 1 to $2n-3$, the minimal required F is decreased. However, raising S_S above $2n-3$ does not result in a further decrease in the minimal required F. To illustrate this, and to further study the behavior of these parameters, the first four columns of Tables 3.3, 3.4, and 3.5 provide specific

$S_S = 1$	$F \geq \frac{1}{2}N$	$P \leq 1$
$S_S = 2$		$P \leq 2$
$S_S = n$	$F \geq \begin{cases} \frac{3}{2}\sqrt{N} - 1 & n \text{ odd} \\ \sqrt{2N} - 1 & n \text{ even} \end{cases}$	$P = F$
$S_S = 2n - 3$	$F \geq n$	$P \leq n\,2^{n-3} = \frac{1}{8}N \log_2 N$
$S_S = 2n - 2$		$P \leq n\,2^{n-2} = \frac{1}{4}N \log_2 N$
$S_S = 2n - 1$		$P \leq n\,2^{n-1} = \frac{1}{2}N \log_2 N$

Table 3.2: Special cases of Richards' formulae

values of P and the minimal required F for $2 \leq N \leq 11$ and $1 \leq S_S \leq 2n-1$. Column five of these tables lists the "device cost" parameter D (scaled by N for ease of comparison). The device cost is based on the number of switching elements required to implement a particular network. For the purposes of computing D, 2×2 swiches are assigned a cost of 1, and $1{\times}F$ and $F{\times}1$ switches are assigned a cost of $F-1$ (since they can be implemented with $F-1$ 1×2 or 2×1 switches, respectively). The device cost is thus:

$$D = N[F(S/2+2) - 2]. \tag{3.38}$$

In the first five columns of Tables 3.3, 3.4, and 3.5, the special cases $S_S = n$ and $S_S = 2n-3$ are noted, and the cases with minum device cost for each N are highlighted. The last three columns in these tables, F', P', and D'/N, present the "restricted-F" case. For this special case, the tables show that the new minimal device cost D' may occur at different values F' and S'_S. These new values may be directly calculated for a given n by rounding $F = n$ up to the next highest

Network Size	S_S	F	P	D/N	F'	P'	D'/N
$N=4$ $(n=2)$	$2n-3=$ **1**	**2**	**1**	**3.0**	**2**	**1**	**3.0**
	$n=2$	2	2	4.0	2	2	4.0
	3	2	4	5.0	2	4	5.0
$N=8$ $(n=3)$	**1**	**4**	**1**	**8.0**	**4**	**1**	**8.0**
	2	4	2	10.0	4	2	10.0
	$2n-3=n=$ 3	3	3	8.5	4	4	12.0
	4	3	6	10.0	4	8	14.0
	5	3	12	11.5	4	16	16.0
$N=16$ $(n=4)$	1	8	1	18.0	8	1	18.0
	2	8	2	22.0	8	2	22.0
	3	6	3	19.0	8	4	26.0
	$n=4$	5	5	18.0	8	8	30.0
	$2n-3=$ **5**	**4**	**8**	**16.0**	**4**	**8**	**16.0**
	6	4	16	18.0	4	16	18.0
	7	4	32	20.0	4	32	20.0
$N=32$ $(n=5)$	1	16	1	38.0	16	1	38.0
	2	16	2	46.0	16	2	46.0
	3	12	3	40.0	16	4	54.0
	4	10	5	38.0	16	8	62.0
	$n=$ **5**	7	7	29.5	**8**	**8**	**34.0**
	6	6	12	28.0	8	16	38.0
	$2n-3=$ **7**	**5**	**20**	**25.5**	8	32	42.0
	8	5	40	28.0	8	64	46.0
	9	5	80	30.5	8	128	50.0
$N=64$ $(n=6)$	1	32	1	78.0	32	1	78.0
	2	32	2	94.0	32	2	94.0
	3	24	3	82.0	32	4	110.0
	4	20	5	78.0	32	8	126.0
	5	14	7	61.0	16	8	70.0
	$n=6$	11	11	53.0	16	16	78.0
	7	8	16	42.0	**8**	**16**	**42.0**
	8	7	28	40.0	8	32	46.0
	$2n-3=$ **9**	**6**	**48**	**37.0**	8	64	50.0
	10	6	96	40.0	8	128	54.0
	11	6	192	43.0	8	256	58.0

N, network size (number of PEs and number of MMs); $n = \log_2 N$; S_S, number of shuffle–exchange stages in the "main section" of the RS-EGS network; F, inlet fan-out and outlet fan-in factor; P, number of paths available between a given inlet and outlet; D, device count (number of switches) required for network. Primed quantities are for a Restricted-F RS-EGS network, in which F is required to be a power of two.

Table 3.3: RS-EGS network parameters for $n=2$ through $n=6$

Network Size	S_S	F	P	D/N	F'	P'	D'/N
$N=128 \quad (n=7)$	1	64	1	158.0	64	1	158.0
	2	64	2	190.0	64	2	190.0
	3	48	3	166.0	64	4	222.0
	4	40	5	158.0	64	8	254.0
	5	28	7	124.0	32	8	142.0
	6	22	11	108.0	32	16	158.0
	$n=7$	15	15	80.5	16	16	86.0
	8	12	24	70.0	16	32	94.0
	9	9	36	56.5	16	64	102.0
	10	8	64	54.0	**8**	**64**	**54.0**
	$2n-3=$ **11**	**7**	**112**	**50.5**	8	128	58.0
	12	7	224	54.0	8	256	62.0
	13	7	448	57.5	8	512	66.0
$N=256 \quad (n=8)$	1	128	1	318.0	128	1	318.0
	2	128	2	382.0	128	2	382.0
	3	96	3	334.0	128	4	446.0
	4	80	5	318.0	128	8	510.0
	5	56	7	250.0	64	8	286.0
	6	44	11	218.0	64	16	318.0
	7	30	15	163.0	32	16	174.0
	$n=8$	23	23	136.0	32	32	190.0
	9	16	32	102.0	16	32	102.0
	10	13	52	89.0	16	64	110.0
	11	10	80	73.0	16	128	118.0
	12	9	144	70.0	16	256	126.0
	$2n-3=$ **13**	**8**	**256**	**66.0**	**8**	**256**	**66.0**
	14	8	512	70.0	8	512	70.0
	15	8	1024	74.0	8	1024	74.0
$N=512 \quad (n=9)$	1	256	1	638.0	256	1	638.0
	2	256	2	766.0	256	2	766.0
	3	192	3	670.0	256	4	894.0
	4	160	5	638.0	256	8	1022.0
	5	112	7	502.0	128	8	574.0
	6	88	11	438.0	128	16	638.0
	7	60	15	328.0	64	16	350.0
	8	46	23	274.0	64	32	382.0
	$n=9$	31	31	199.5	32	32	206.0
	10	24	48	166.0	32	64	222.0
	11	17	68	125.5	32	128	238.0
	12	14	112	110.0	**16**	**128**	**126.0**
	13	11	176	91.5	16	256	134.0
	14	10	320	88.0	16	512	142.0
	$2n-3=$ **15**	**9**	**576**	**83.5**	16	1024	150.0
	16	9	1152	88.0	16	2048	158.0
	17	9	2304	92.5	16	4096	166.0

Table 3.4: RS-EGS network parameters for $n=7$ through $n=9$

Network Size	S_S	F	P	D/N	F'	P'	D'/N
$N=1024$	1	512	1	1278.0	512	1	1278.0
$(n=10)$	2	512	2	1534.0	512	2	1534.0
	3	384	3	1342.0	512	4	1790.0
	4	320	5	1278.0	512	8	2046.0
	5	224	7	1006.0	256	8	1150.0
	6	176	11	878.0	256	16	1278.0
	7	120	15	658.0	128	16	702.0
	8	92	23	550.0	128	32	766.0
	9	62	31	401.0	64	32	414.0
	$n=10$	47	47	327.0	64	64	446.0
	11	32	64	238.0	32	64	238.0
	12	25	100	198.0	32	128	254.0
	13	18	144	151.0	32	256	270.0
	14	15	240	133.0	**16**	**256**	**142.0**
	15	12	384	112.0	16	512	150.0
	16	11	704	108.0	16	1024	158.0
	$2n-3=$ **17**	**10**	**1280**	**103.0**	16	2048	166.0
	18	10	2560	108.0	16	4096	174.0
	19	10	5120	113.0	16	8192	182.0
$N=2048$	1	1024	1	2558.0	1024	1	2558.0
$(n=11)$	2	1024	2	3070.0	1024	2	3070.0
	3	768	3	2686.0	1024	4	3582.0
	4	640	5	2558.0	1024	8	4094.0
	5	448	7	2014.0	512	8	2302.0
	6	352	11	1758.0	512	16	2558.0
	7	240	15	1318.0	256	16	1406.0
	8	184	23	1102.0	256	32	1534.0
	9	124	31	804.0	128	32	830.0
	10	94	47	656.0	128	64	894.0
	$n=11$	63	63	470.5	64	64	478.0
	12	48	96	382.0	64	128	510.0
	13	33	132	278.5	64	256	542.0
	14	26	208	232.0	32	256	286.0
	15	19	304	178.5	32	512	302.0
	16	16	512	158.0	**16**	**512**	**158.0**
	17	13	832	134.5	16	1024	166.0
	18	12	1536	130.0	16	2048	174.0
	$2n-3=$ **19**	**11**	**2816**	**124.5**	16	4096	182.0
	20	11	5632	130.0	16	8192	190.0
	21	11	11264	135.5	16	16384	198.0

Table 3.5: RS-EGS network parameters for $n=10$ and $n=11$

power of 2 (F'), then working backwards from the equations in Table 3.1 to find the new $S'_S \leq S_S$. This restricted-F case is of particular interest (as previously mentioned in Section 3.3.2) since it may be implemented with full trees of 1×2 and 2×1 switches. In Tables 3.3, 3.4, and 3.5, the minimal device cost for the parameter sets $(S_S, F', P', D'/N)$ are also highlighted.

Examination of the minimum device cost (highlighted) for the unprimed quantities (the unrestricted-F case) in Tables 3.3, 3.4, and 3.5 reveals that for this case the minimum device cost always occurs at $S_S = 2n-3$ except for the (anomalous) case of $n=3$. However, for the primed quantities (the $F=2^f$ case), the minimum device cost occurs at $S_S \leq 2n-3$ (again, except for the cases of $n=2$ and $n=3$), with equality occuring only when n is itself a power of two. Table 3.6 provides a summary of these highlighted minimum device cost cases, including an additional column comparing the minimum device costs.

Table 3.6 reveals that for all these minimum device cost cases (except for the anomalous case of $n=3$), $D' \geq D$, although often by rather small amounts. Thus the simplicity of implementation of the $F=2^f$ case does not in general impose a great cost penalty over the unrestricted minimal F cases. Table 3.6 is broken up into sections of equal F' values to better reveal parameter trends.

3.4 Restricted-F RS-EGS Network Routing

Routing through restricted-F RS-EGS networks has a particularly simple easy-to-use form. Given a restricted-F RS-EGS network with N inlets and outlets, F "fan-in/out", and S_S main section stages, make the following definitions. For any inlet-outlet pair, number the P available paths with indices $0 \leq \mathcal{P} \leq P-1$. Then

n	N	S_S	F	P	D/N	S'_S	F'	P'	D'/N	D'/D
2	4	1	2	1	3.0	1	2	1	3.0	1.00
3	8	3	3	3	8.5	1	4	1	8.0	0.94
4	16	5	4	8	16.0	5	4	8	16.0	1.00
5	32	7	5	20	25.5	5	8	8	34.0	1.33
6	64	9	6	48	37.0	7	8	16	42.0	1.14
7	128	11	7	112	50.5	10	8	64	54.0	1.07
8	256	13	8	256	66.0	13	8	256	66.0	1.00
9	512	15	9	576	83.5	12	16	128	126.0	1.51
10	1024	17	10	1280	103.0	14	16	256	142.0	1.38
11	2048	19	11	2816	124.5	16	16	512	158.0	1.27
12	4096	21	12	6144	148.0	19	16	2048	182.0	1.23
13	8192	23	13	13312	173.5	21	16	4096	198.0	1.14
14	16384	25	14	28672	201.0	23	16	8192	214.0	1.06
15	32768	27	15	61440	230.5	26	16	32768	238.0	1.03
16	65536	29	16	131072	262.0	29	16	131072	262.0	1.00
17	131072	31	17	278528	295.5	26	32	16384	478.0	1.62
18	262144	33	18	589824	331.0	29	32	65536	526.0	1.59
19	524288	35	19	1245184	368.5	31	32	131072	558.0	1.51
20	1048576	37	20	2621440	408.0	33	32	262144	590.0	1.45
21	2097152	39	21	5505024	449.5	35	32	524288	622.0	1.38
22	4194304	41	22	11534336	493.0	37	32	1048576	654.0	1.33
23	8388608	43	23	24117248	538.5	39	32	2097152	686.0	1.27
24	16777216	45	24	50331648	586.0	41	32	4194304	718.0	1.23

Table 3.6: RS-EGS parameters for two minimal device cost cases: D, general case; D', restricted-F case.

[Richards 93, Hinton 93a, Hinton 93b] for a given inlet $\mathcal{X}$, outlet $\mathcal{Y}$, and path number $\mathcal{P}$, the path $(\mathcal{X},\mathcal{P},\mathcal{Y})$ will pass through the stage i switch $s_i(\mathcal{X},\mathcal{P},\mathcal{Y})$, where:

$$s_i(\mathcal{X},\mathcal{P},\mathcal{Y}) = \left\lfloor \frac{\mathcal{X}F2^{S_S} + \mathcal{P}N + \mathcal{Y}}{2^{S_S+1-i}} \right\rfloor_{\bmod NF/2} \qquad 1 \le i \le S_S. \tag{3.39}$$

This expression can be broken down and interpreted on the bit level as follows. A given inlet $\mathcal{X}$, outlet $\mathcal{Y}$, and path number $\mathcal{P}$ may be represented as binary digits (bits) as:

$$\mathcal{X} = (x_{n-1}\ x_{n-2}\ \cdots\ x_2\ x_1\ x_0), \qquad 0 \le \mathcal{X} \le N-1 \tag{3.40}$$

$$\mathcal{Y} = (y_{n-1}\ y_{n-2}\ \cdots\ y_2\ y_1\ y_0), \qquad 0 \le \mathcal{Y} \le N-1 \tag{3.41}$$

$$\mathcal{P} = (p_{p-1}\ p_{p-2}\ \cdots\ p_2\ p_1\ p_0), \qquad 0 \le \mathcal{P} \le P-1 \tag{3.42}$$

Now, examine the numerator of Equation 3.39, which will be referred to here as the *path vector* $\mathcal{V}$:

$$\mathcal{V} = \mathcal{X}F2^{S_S} + \mathcal{P}N + \mathcal{Y}. \tag{3.43}$$

When viewed as a sequence of bits, the path vector has a separable form. Using $n = \log_2 N$ and $f = \log_2 F$ and $p \triangleq \log_2 P = f + S_S - n$, the path vector (of $f+S_S+n$ bits) takes the form:

$$\mathcal{V} = (\underbrace{x_{n-1}\ \cdots\ x_1\ x_0}_{n \text{ bits}}\ \underbrace{p_{p-1}\ \cdots\ p_1\ p_0}_{f+S_S-n \text{ bits}}\ \underbrace{y_{n-1}\ \cdots\ y_1\ y_0}_{n \text{ bits}}). \tag{3.44}$$

The index $\mathcal{F}$ is introduced in this work to further clarify the meaning of the bits in the path vector, and to facilitate the use of the path vector for routing. Number the F available paths through each "fan-out" switch with indices $0 \le \mathcal{F} \le F-1$, where the binary digits (bits) for $\mathcal{F}$ are given by

$$\mathcal{F} = (f_{f-1}\ f_{f-2}\ \cdots\ f_2\ f_1\ f_0). \qquad 0 \le \mathcal{F} \le F-1 \tag{3.45}$$

This expression may be inserted into the path number portion of the path vector, which yields:

$$\mathcal{V} = (\underbrace{x_{n-1} \cdots x_1\ x_0}_{n \text{ bits}}\ \underbrace{f_{f-1} \cdots f_1\ f_0}_{f \text{ bits}}\ \underbrace{p_{p-f-1} \cdots p_1\ p_0}_{S_S-n \text{ bits}}\ \underbrace{y_{n-1} \cdots y_1\ y_0}_{n \text{ bits}}). \tag{3.46}$$

The expression in Equation 3.39 can now be identified as a sliding window on the path vector, as follows:

$$s_i(\mathcal{X}, \mathcal{P}, \mathcal{Y}) = \left\lfloor \frac{\mathcal{V}}{2^{S_S+1-i}} \right\rfloor_{\text{mod } NF/2} \tag{3.47}$$

$$= \textbf{RIGHTMOST-}(n+f-1)\textbf{-BITS of} \tag{3.48}$$

$$\left\{\mathcal{V} \textbf{ RIGHT-SHIFTED-BY } S_S+1-i \text{ bit-positions}\right\}$$

$$= \textbf{the boxed portion of } \mathcal{V} \textbf{ below:} \text{ (arbitrary offset shown)}$$

$$(x_{n-1} \cdots x_1 \underbrace{\boxed{x_0\ f_{f-1} \cdots f_1\ f_0\ p_{p-f-1}}}_{n+f-1 \text{ bits}} \underbrace{\cdots p_1\ p_0\ y_{n-1} \cdots y_1\ y_0}_{S_S+1-i \text{ bit-positions}}). \tag{3.49}$$

Thus, every switch in the restricted-F RS-EGS network has an index composed of $n+f-1$ bits, and the stage i switch passed through by path vector $\mathcal{V}$ has index found by extracting $n+f-1$ bits located S_S+1-i bit-positions from the LSB (Least Significant Bit) of the path vector. More information can be gleaned from the path vector, such as the switch setting, and the link index that exits from the switch; these will be introduced and explanined along with the example below.

To illustrate the extraction of routing information from the path vector, consider the example illustrated in Figure 3.9 and Table 3.7. The case of $N=8$, $F=4$, $S_S=4$, and $P=8$ is shown. In Figure 3.9 the $P=8$ paths from inlet node 1 (001) to outlet node 5 (101) are marked. In addition, the path $\mathcal{P}=5$ (101) is highlighted for particular attention. Table 3.7 lists the information that may be extracted from the path vector for path 5, $\mathcal{V}$=(001101101). Obviously, from construction,

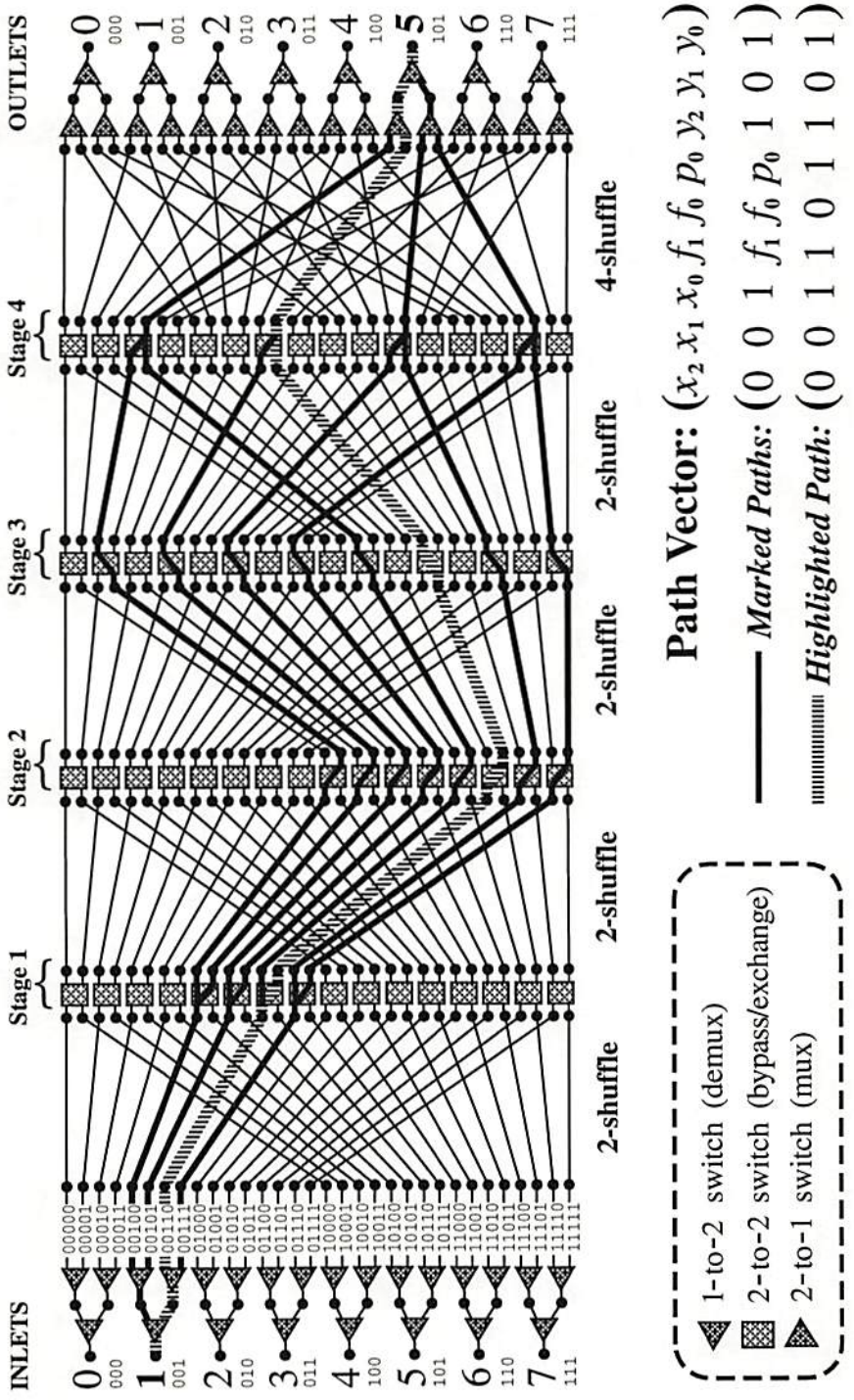


Figure 3.9: Path vector illustration; $N=8$, $F=4$, $S_S=4$, $P=8$

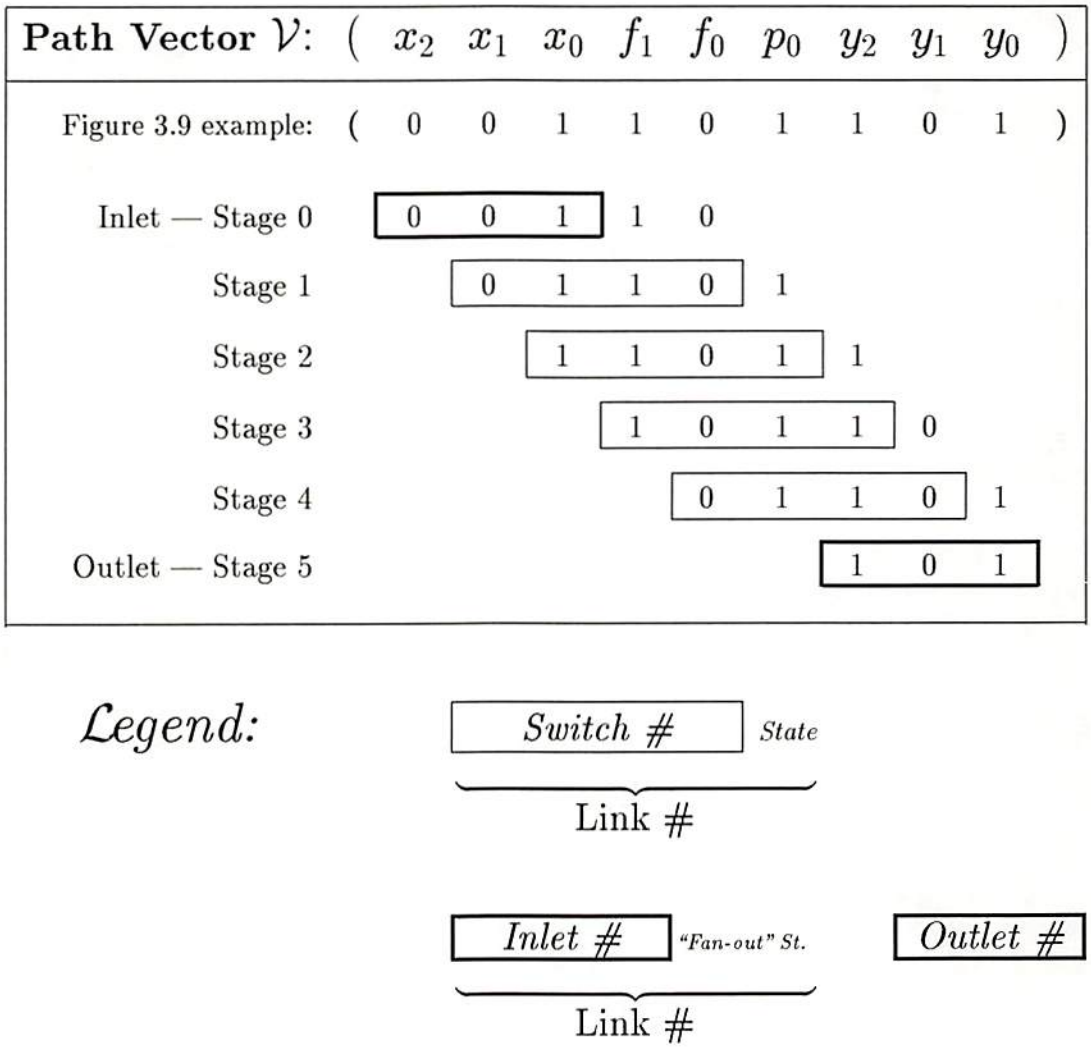


Table 3.7: Switch and link indices extracted from the path vector; $N=8$, $F=4$, $S_S=4$, $P=8$

the inlet node is the first $n=3$ bits of $\mathcal{V}$ and the outlet node is the last $n=3$ bits of $\mathcal{V}$. Additionally, the switch indices of $n+f-1=4$ bits for the main section stages are shown, where the index for the switch used in stage i is shifted left in the path vector by $S_S+1-i = 5-i$ bit-positions. The switch indices are not directly labelled in Figure 3.9 (to eliminate diagram clutter), but can be ascertained by numbering each switch in a stage from zero at the top of the diagram. The switch indices are listed in Table 3.7 for stages 1 through 4 as the boxed quantities. The bit immediately following the stage i switch index indicates the setting of the switch (the switch *state*); a 0 bit indicates that the path exits the upper switch outlet, while a 1 bit indicates that the path exits the lower switch outlet. The table lists these switch states for stages 1 through 4 as the unboxed quantities following the switch indices. A simple arithmetic operation on the switch index and switch state (twice the switch index plus the switch state) yields the link index upon which the path exits the switch. The link indices are listed in the Figure after the "fan-out" stage, although each set of links is numbered independently from zero at the top of the diagram in each stage immediately following the switch nodes. The link indices take one more bit than the switch indices, which is $n+f=5$ bits for this example. Table 3.7 shows the link indices for stages 1 through 4, as 5-bit numberes composed of the boxed switch index bits concatenated with the following unboxed switch state bit. Additionally, the $\mathcal{F}$ bits may be identified following the inlet node bits, which indicates which route the path takes through the "fan-out" stage. The inlet node index concatenated with the following $\mathcal{F}$ bits compose the link index by which the path exits the "fan-out" stage (stage 0) and enters the "main section" of the network.

From this example, the effect of the $\mathcal{F}$ part of the $\mathcal{P}$ path number is seen. The $\mathcal{F}$ bits $(f_{f-1} \cdots f_1 \; f_0)$ represent the state of the "fan-out" switches, and any remaining bits $(p_{p-f-1} \cdots p_1 \; p_0)$ from the path number $\mathcal{P}$ represent routing flexibility within the main section of the RS-EGS network stages. The number of bits in $(p_{p-f-1} \cdots p_1 \; p_0)$, $(p-f)$, indicates the number of stages in the main section which have paths to the destination outlet node via both the upper and lower switch exit links.

3.5 Summary and Contributions

This chapter has presented a review of the theory behind shuffle–exchange interconnection network topologies and routing. EGS networks were selected for special attention because they have a rich topology capable of strict nonblocking operation, and because they may be tailored to fit a given application through tradeoffs of network length and width (S_S and F). A restricted-F RS-EGS network is chosen as the basis of the FIER optical network since it provides strictly nonblocking operation, and paths may be computed by a simple local routing method. RS-EGS networks are specified instead of general EGS networks so that simple repeated hardware may be used in building the network (i.e. it is homosyndetic), and the restricted-F version is specified for simplicity of both routing and hardware (the "fan-in" and "fan-out" switch trees).

Although this chapter is primarily review of the work of other researchers, contributions are made in the areas of terminology, concept refinement, and new theoretical results. The term "path vector" is applied to a particularly useful routing bit sequence. The meaning of this path vector is refined by the explicit

identification of the bits controlling the "fan-out" switches. A particularly useful subset of EGS networks that Richards has identified [Richards 91a, Richards 93] is given the term RS-EGS networks to simplify references to this subset. In addition, restricted-F RS-EGS networks are also defined, and a new analysis of their design characteristics is presented here. New theoretical results presented here include the decomposition of c^q-shuffles on c^n nodes, and the practical result that a Q-shuffle on 2^n nodes may be implemented as $q = \log_2 Q$ consecutive perfect shuffle operations.

Chapter 4

OPTICAL INTERCONNECTION NETWORK HARDWARE

This chapter presents the design of an optical interconnection network, the Free-space Interconnection with Externally-controlled Routing (FIER). The passive bidirectional optical hardware implementation of the FIER is illustrated with block diagrams and raytraces. The individual interconnection stages (shuffle–exchanges) are designed to be optically cascadable. A novel method of performing broadcast and combine operations within the FIER is discussed in detail.

The FIER is a self-contained subsystem. Therefore it may also be used as a component in systems other than the SMOEC. The FIER may be used as a switching element in distributed computing systems. Multimedia communication

systems may also employ the FIER as a switching element to utilize its potentially very high data throughput rate. However, the FIER is restricted to applications where a circuit-switched network is appropriate, and in which combining or permutation-restricted operation is applied.

4.1 Passive Hardware Implications

Current state-of-the-art optical array switching is slower than electronic switching . The effect of relatively slower optical switching speeds is minimized by allowing all FIER switches to change state simultaneously, and then sending large amounts of data very rapidly through the network. Active optical switching (setting the switch after the data arrive, and thereby detecting and regenerating the data signals) would make each network transit acquire a delay proportional to the product of S (the number of stages in the FIER) and the active optical device switching time. Therefore passive optical switching is used in the FIER, enabling high speed reversible data transfer.

Passive optical switching requires that switching decisions must be carried out physically separate from the data passing through the switches. Therefore, a separate routing processing subsystem (MATSH) was designed to control the network when used in the SMOEC. In applications other than the SMOEC, different control bit computation and distribution techniques may be developed to fit differing requirements.

Since the passive optical data paths are externally controlled, the interconnection network is circuit switched. Circuit switching sets a dedicated communication

channel between a source and destination pair for the duration of the communication. In contrast, in packet switching there is no dedicated channel. Each data block in the communication is separately stored and forwarded at each intermediate switching node. Different data blocks in the same communication may take different source-destination paths, which never happens with circuit switching.

4.2 Optical Shuffle–Exchange Hardware Design

Optical switching and permuting in the FIER is polarization-based, while information content is carried by serially-modulated optical intensity. The optical data format within the FIER is termed Channel Pairs (CP). A CP consists of the data from two orthogonally polarized adjacent channels which are superimposed in location and direction. Thus a CP is an optical beam carrying two information streams, one horizontally polarized and the other vertically polarized. This format provides ease of channel manipulation. The external input and output format for the FIER consists of the channels lined up separately with identical polarizations in a format termed Linear Array (LA).

The FIER consists of several subsystems. These subsystems are designed to be optically cascadable, with optical switching arrays integrated into the optical shuffle design (instead of being separately located). In the following presentation of the optical subsystems, the terms "pixels" and "optical channels" will be used interchangeably to emphasize the interconnection function of these systems.

4.2.1 FIER Input/Output

The interfaces between the optical and electronic signals are designed to be fully parallel (no addressing schemes) to avoid bottlenecks at these conversion points. Optical fibers can route signals to and from external discrete electronic or optical devices, and format the signals into pixel arrays for entry and exit to the FIER. In addition, if optically addressed SLMs are used in the FIER, optical fiber bundles can also be used to provide the control bits to the SLMs in parallel optical pixel arrays.

4.2.2 The LA↔CP Format Converter

The Linear Array to Channel Pair (LA↔CP) format converter employs a polarizing prism to merge adjacent pairs of channels in the Linear Array input to form orthogonally polarized Channel Pairs. An overview of the converter design is shown in Figure 4.1. (Note: in all figures labelled "Overview," optical image inversions are omitted for simplicity). A detailed raytrace is shown in Figure 4.2. The Linear Array of vertically polarized light channels (channels 0–7 shown) is first given spatially alternating polarizations by using a fixed Pixellated Wave Plate (PWP) containing pixels of half-wave retardation alternating with pixels of no retardation. The propagation angles of the pixels are then modified in the Fourier plane of a bulk lens by a Wollaston prism, giving the alternately polarized pixels slightly different angular offsets. (Note: Wollaston prisms will provide constant angular offsets for angles that meet the small angle approximation, as shown in A). These offsets are chosen (along with the lens focal length) so that adjacent channels will

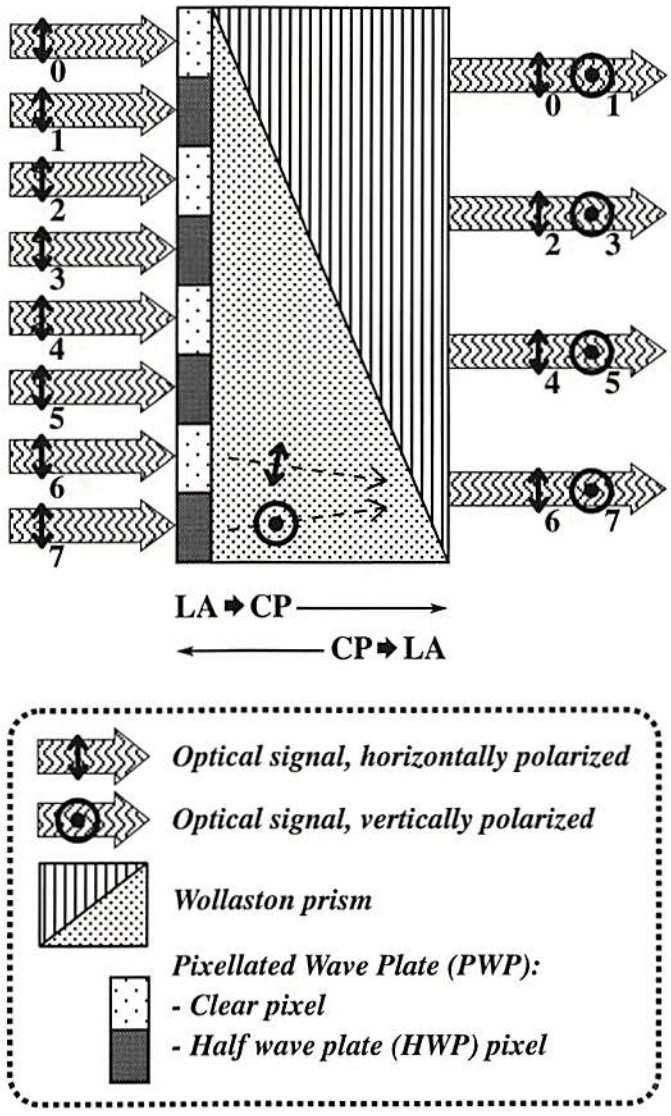


Figure 4.1: Overview: Linear Array $\leftrightarrow$ Channel Pairs Format Converter.

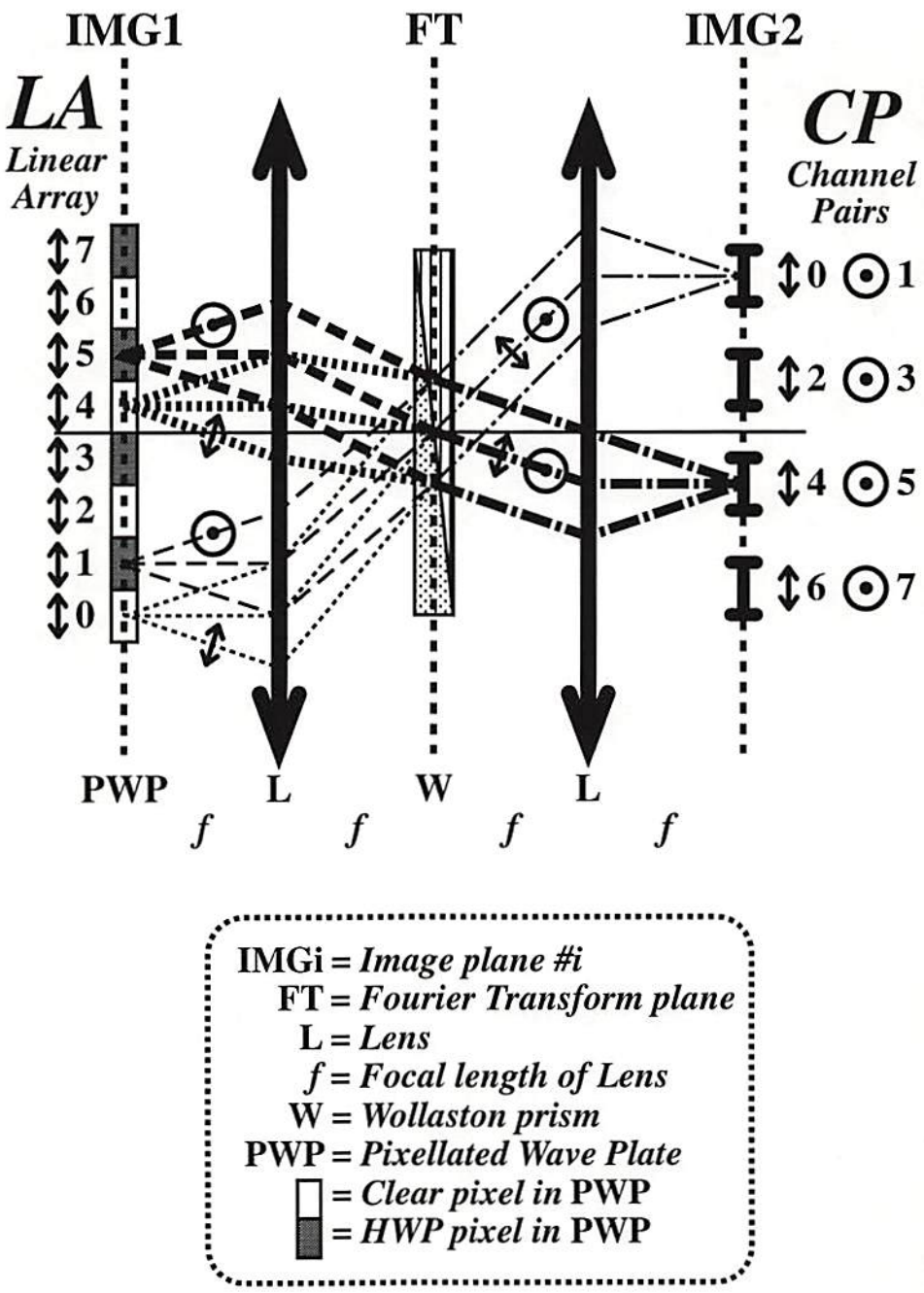


Figure 4.2: Raytrace: LA $\leftrightarrow$ CP Format Converter.

be adjusted to the same propagation angles. A second lens retransforms the adjusted array, converting angular offsets to linear offsets. Thus, adjacent pixels are superposed to form Channel Pairs, propagating colinearly with orthogonal polarizations. Since the angles are modified in the Fourier plane, only the positions of the output pixels are changed from the original positions; the pixel propagation and divergence angles are preserved. It is important to note that each Channel Pair output of the converter has the same pixel width and angular spread as an input Linear Array pixel, but unlike the Linear Array input, the output pixels have twice the center-to-center spacing.

Figure 4.1 also illustrates the convention for channel indices for the LA and CP formats. The LA format uses $\updownarrow$-polarized light, indexed sequentially. The indices for a channel pair in CP format are: the even index i is $\updownarrow$-polarized and the odd index $i+1$ is $\odot$-polarized.

The passive nature of the optical design ensures that this converter will work oppositely in the reverse direction, splitting appropriately spaced Channel Pairs into separate channels, and then rectifying the polarizations to form the Linear Array. It is essential that the array of Channel Pairs input to the "CP$\leftrightarrow$LA" converter have adjacent pairs separated by one pixel width in order for the converter to maintain separation of the channels.

4.2.3 Pixel Spacing Adjuster

The output from the LA$\leftrightarrow$CP converter has extra interpixel spacing that must be eliminated for it to mesh with other components in the FIER. Similarly, for cascadability of the shuffle–exchange stages (discussed in the next sections), a

Pixel Spacing Adjuster (PSA) will be included in the design of the CP exchange. Figure 4.3 shows a raytrace diagram of the PSA optical system. The system employs two bulk lenses (one with twice the focal length of the other: $f_3 = 2f_4$) to provide overall demagnification of the pixel array to half its original size. The two lenslet arrays [Ostermayer 83] (one with twice the focal length of the other: $2f_5 = f_6$) then magnify the individual pixels to fill out the reduced size array. This choice of optical system preserves incoming pixel size, shape, angles, and polarization. Of course, this setup will work in the reverse direction as well, adding space between the pixels. It is useful to note that for flexibility of implementation, an alternate ordering of optical elements is equally valid, as shown in Figure 4.4.

4.2.4 The CP Unshuffle

The development of the optical shuffle system will be presented in this section in the reverse, as an "unshuffle" or "inverse shuffle." Therefore the "forward" direction (mentioned in Section 3.1) is actually right-to-left (and "reverse" is left-to-right) in Figures 4.5, 4.6, 4.7, 4.8, and 4.12. The optical shuffle system is presented in this manner to better illustrate the development of its components. The unshuffle–exchange has merely to be reversed for incorporation into the FIER as a shuffle stage.

The perfect shuffle of the optical data in the channel pairs array is performed by a concatenation of systems (albeit modified) that have been already discussed. An overview of the CP unshuffle is shown in Figure 4.5. The unshuffle consists of a modified CP↔LA format converter, followed by a LA↔CP format converter, followed by a PSA system. Note that the format of the output of the unshuffle

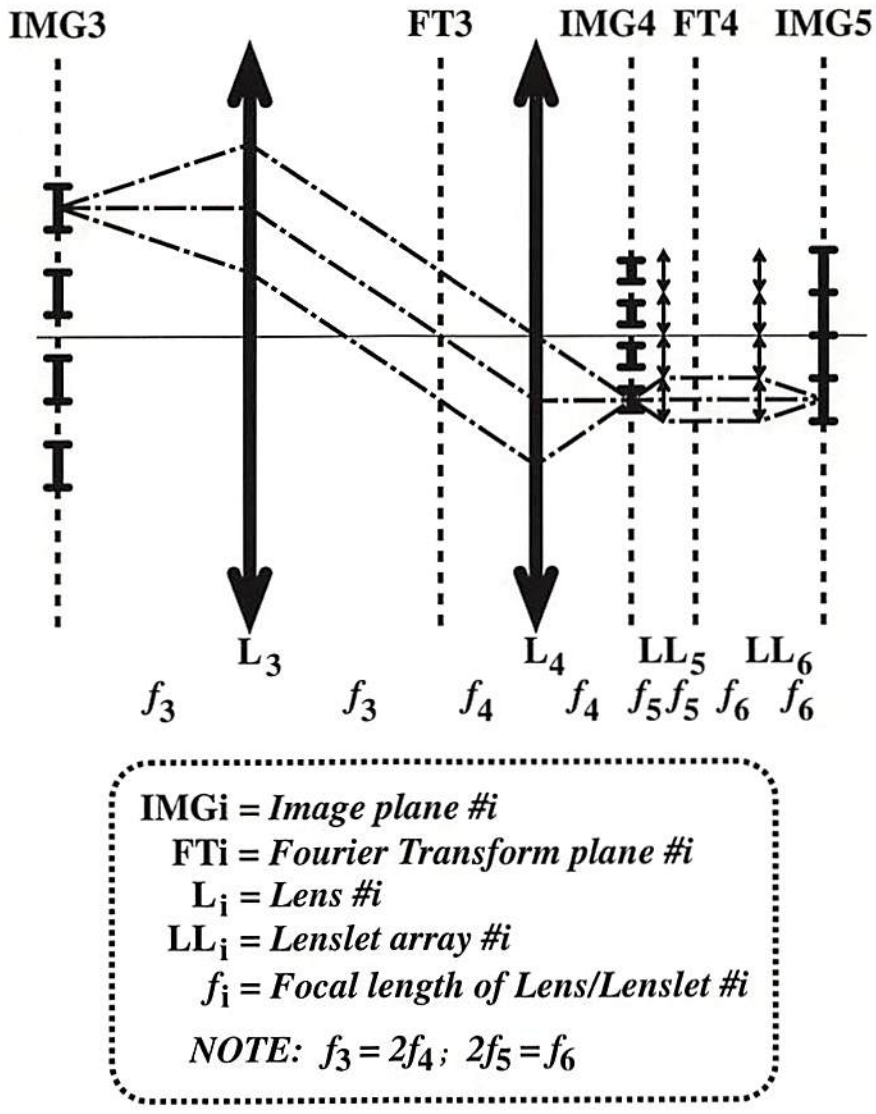
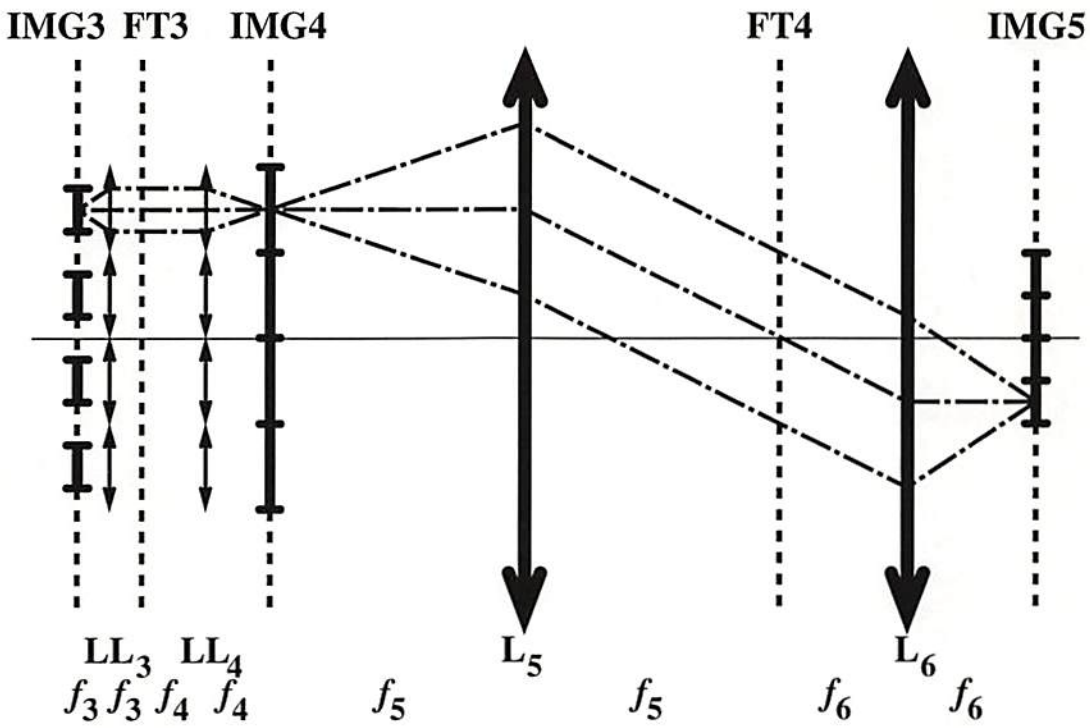


Figure 4.3: Raytrace: Pixel Spacing Adjuster (PSA) — Design #1.



IMGi = *Image plane #i*
FTi = *Fourier Transform plane #i*
L_i = *Lens #i*
LL_i = *Lenslet array #i*
f_i = *Focal length of Lens/Lenslet #i*

NOTE: $2f_3 = f_4$; $f_5 = 2f_6$

Figure 4.4: Raytrace: Pixel Spacing Adjuster (PSA) — Design #2.

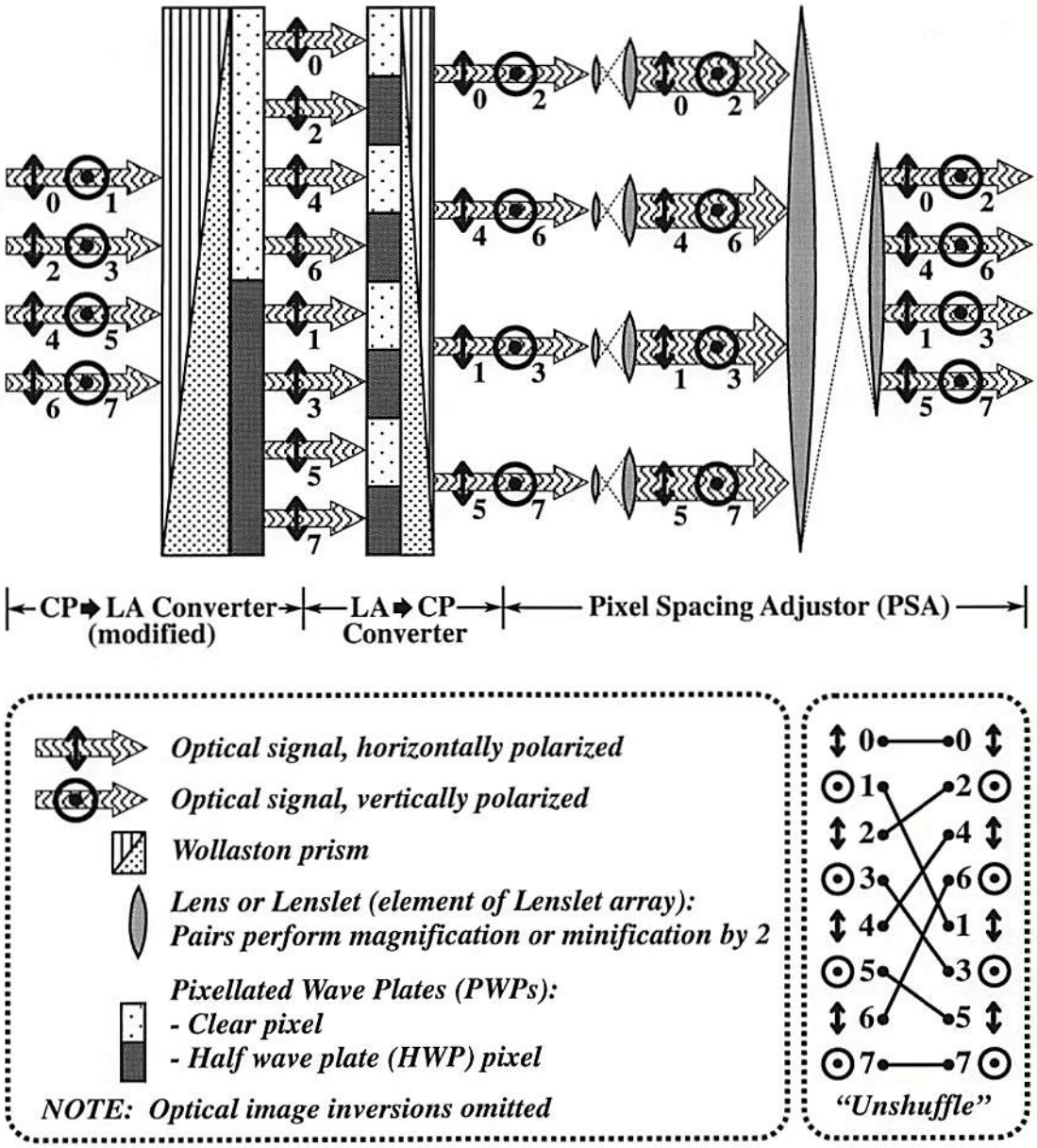


Figure 4.5: Overview: The Channel Pair Unshuffle.

is identical to its input, allowing cascadability (since the CP Exchange does not alter the data format, as discussed in the next section). The modification of the CP↔LA format converter for use in the CP unshuffle requires that the Wollaston prism in the converter be extra thick (as compared to the prism in the original design); it must be thick enough to provide an offset equal to the width of the input CP array. It is also possible to combine the fixed retardation plates (the two PWPs) into one plate in the implementation.

A development of the raytrace of the CP unshuffle will now be presented. The unshuffle (or inverse shuffle) is the operation of dealing cards alternately into two piles then stacking the piles to form a new deck. Figure 4.6 shows the raytrace of the first two parts of the CP unshuffle (no PSA), without any Wollaston prisms. It is simply two sequential $4f$ imaging systems. The figure has $f_1 \neq f_2$ for flexibility of implementation. In Figure 4.7, the Wollaston prisms have been added, and additional ray paths are traced to show the different paths the horizontally and vertically polarized light rays take throughout the system. The first Wollaston prism, W_1, provides an upward angular offset for all $\odot$-polarized rays and a downward angular offset for all $\updownarrow$-polarized rays. The offsets are large enough for the two polarizations to entirely separate from each other, and come to a focus at the Pixellated Wave Plate (PWP). The PWP is the combined retardation plate as previously discussed. Now the unshuffle is basically complete, but it must be put back into Channel Pairs format. Thus a LA↔CP format converter follows after the PWP to recombine the channels. Now the basic shuffle is complete except to fix the pixel spacing (by means of a PSA) for cascadability. The complete CP Unshuffle is shown in Figure 4.8.

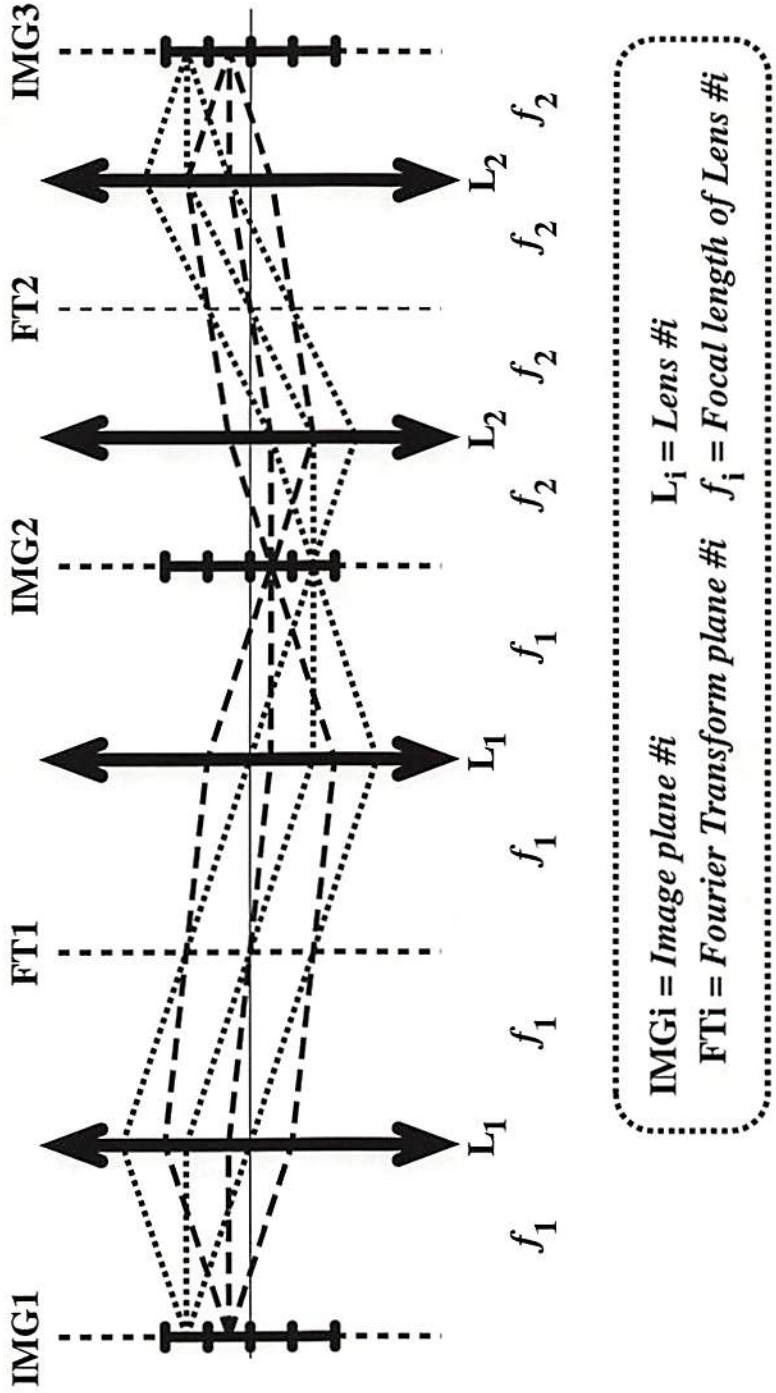


Figure 4.6: Raytrace: CP Unshuffle w/o Wollaston Prisms (w/o PSA).

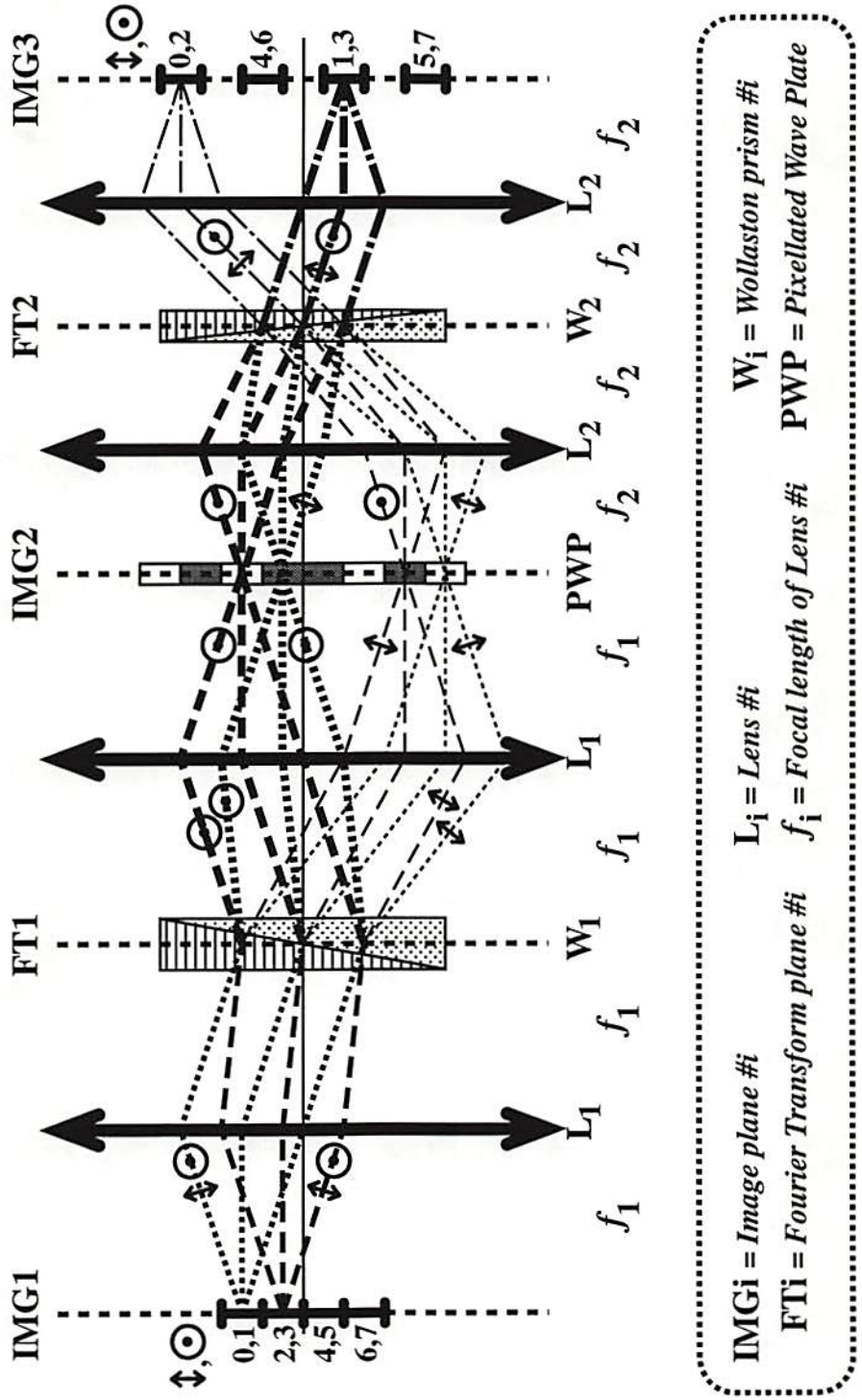


Figure 4.7: Raytrace: CP Unshuffle with Wollaston Prisms (w/o PSA).

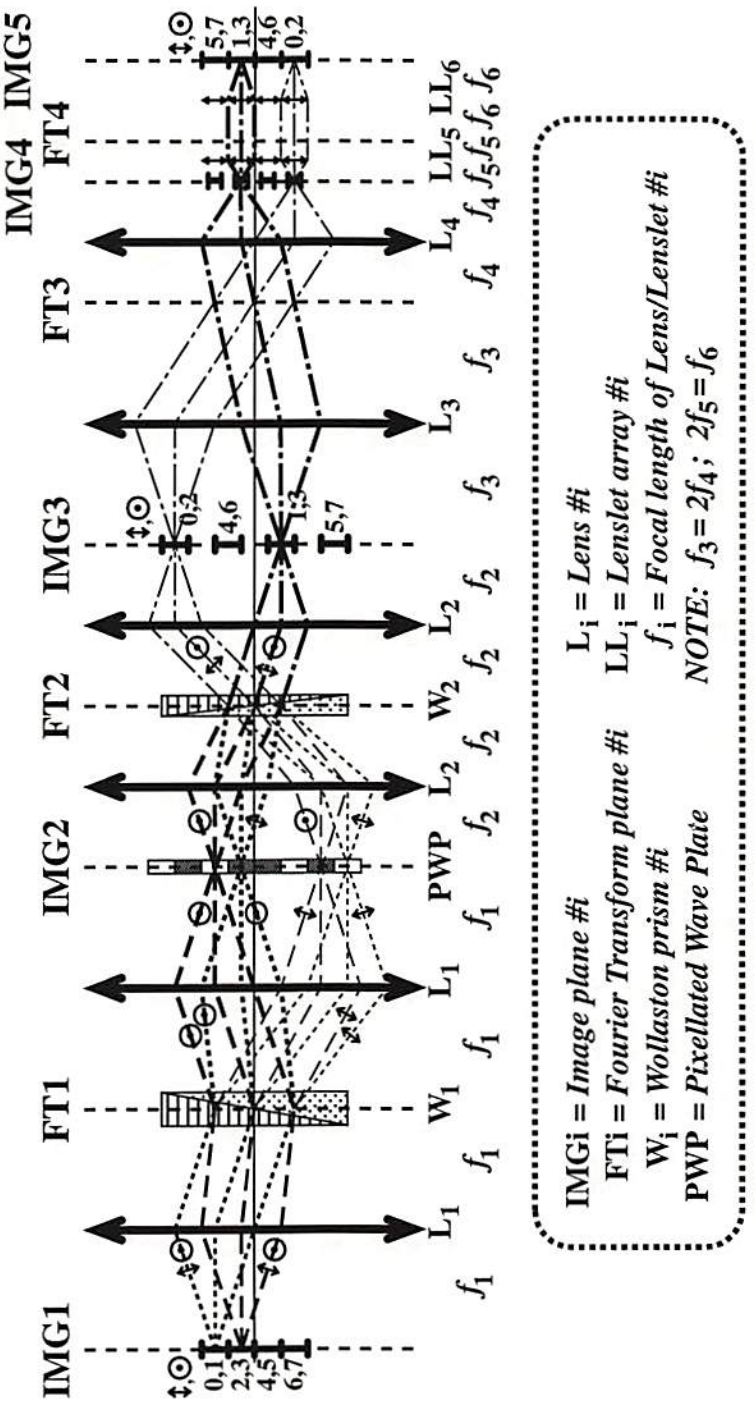


Figure 4.8: Raytrace: CP Unshuffle (Complete).

4.2.5 The CP Exchange

The CP exchange reveals the motivation for the Channel Pair format. Figure 4.9 shows the resulting simplicity of the exchange of a pair of channels. If a Channel Pair passes through a Half-Wave Plate (HWP) the polarization of each channel is rotated 90°, resulting in the effective exchange of the channels. Therefore what is needed for each channel is a switchable single-pixel HWP.

An example of a device that provides an array of switchable HWP pixels is the Ferroelectric Liquid Crystal (FLC) Spatial Light Modulator (SLM) [Patel 87, Johnson 87, Johnson 88, McAdams 90]. This SLM provides an array of switchable pixels, each of which is capable of operating as a switchable HWP. The typical FLC SLM operates in transmission so the optical layout is not complicated by the necessity to extract reflected information. FLC SLMs may be designed to employ either electronic or optical addressing. Either may be used in this implementation, but switching speeds and addressing bottlenecks must be carefully considered before making a choice of the SLM addressing type.

4.2.6 Broadcasting and Combining using TSLMs

In many applications (e.g. a shared memory model computer such as the SMOEC), the shuffle–exchange network must be capable of combining messages and broadcasting results. This requires either buffering and combining, or extended bypass/exchange switches with enhanced control. The FIER described herein employs the second of these options, since the one of the design goals (for the SMOEC) was to satisfy all communication requests in $\mathcal{O}[1]$ network passes.

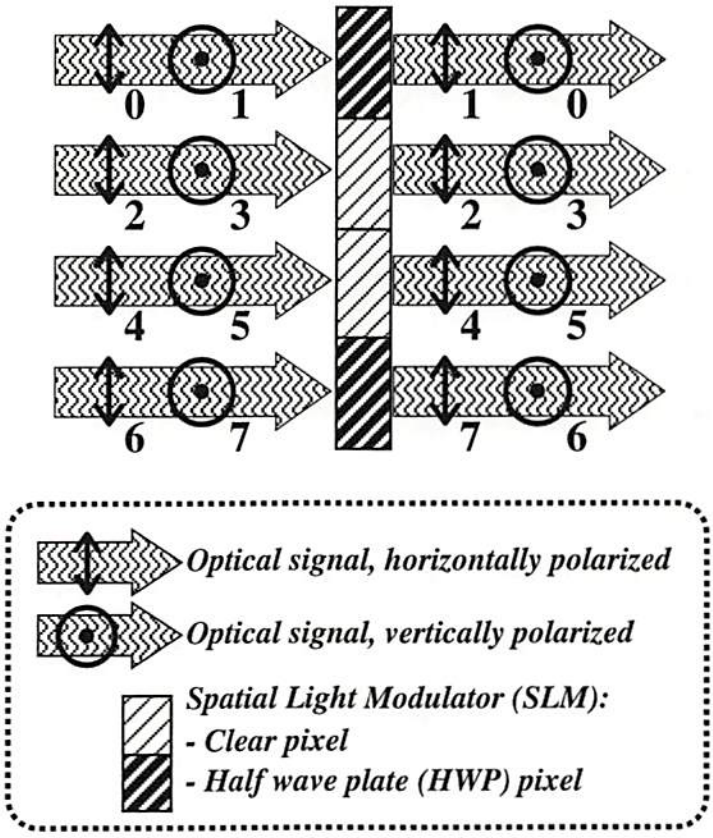


Figure 4.9: Overview: The Channel Pair Exchange.

The optical bypass/exchange switches in the shuffle–exchange network must thus be capable of carrying out some of the "extended" switching operations described in Figure 3.4, specifically upper and lower combining in the forward (shuffle) direction, and upper and lower broadcasting in the reverse (unshuffle) direction. These operations may be viewed as complementary functions since the reverse of a combine operation is the corresponding broadcast operation. The passive optical implementation of these operations exploits the bidirectionality of the optical system, so that the same optical switch setting is used to implement a forward combine and a reverse broadcast.

The optical unshuffle may be modified by the addition of two Tri-state SLMs (TSLMs) to route light from (or to) more than one optical channel. This creates an unshuffle–exchange stage (the reverse of a shuffle–exchange stage). The TSLM is designed to have three states of operation: NULL, MIX, and SWITCH. The NULL state is simply a clear pixel. The SWITCH state is simply the HWP pixel as described in Section 4.2.5. The MIX state is a third state which must be capable of sending light from one input channel to both of the output channels.

Either of two different optical devices are capable of implementing the MIX state: a SLM with Quarter-Wave Plate (QWP) pixels or a SLM with HWP pixels oriented at 22.5° to either the horizontal or vertical axis. To construct a SLM with QWP pixels, a ferroelectric liquid crystal device may be designed similarly to the HWP SLM described previously. The ferroelectric HWP SLM is made by choosing the device thickness d such that $\Delta n\, d = \lambda/2$ (where Δn is the material birefringence and λ is the vacuum wavelength of the light being used) [Handschy 87]. This causes the liquid crystal material to be capable of acting as a HWP or passing the light

basically unaffected. A ferroelectric QWP SLM may be designed in an analogous manner, by selecting the device thickness d such that $\Delta n\, d = \lambda/4$.

Researchers have recently designed and fabricated SLMs with sandwiched layers of FLC material which acts as multi-state modulator [Freeman 92]. Using this technique, the desired TSLM can be made from either a sandwich of two QWP FLC layers (described below), or a sandwich of one QWP layer and one HWP layer, or a sandwich of two layers of HWP oriented at 22.5° and 45°.

An overview of the operation of this modified shuffle–exchange stage is shown in Figure 4.10, illustrating the "forward" combine operation. A combine operation is performed in the following manner. The first TSLM (set to MIX) performs the function of "mixing" the light in a channel pair. Given that optical signals a and b are present in the input channels U and V, then the situation after passing through TSLM1 is that each channel U and V contains the combined signal $a+b$ (each at 50% power). The figure illustrates the MIX state that results from the double HWP TSLM implementation option. (Note that no optical power will actually be combined to form $a+b$ when the FIER is used in the SMOEC, for reasons discussed later in this section). The second TSLM is used to provide two separate tri-state pixels (L and R), and is sandwiched between a pair of polarizers. The pair of polarizers are oriented in parallel so that if a TSLM2 pixel is switched to the SWITCH state, the channel is blocked. The Polarizer/TSLM2/Polarizer sandwich is preceded by a Pixellated Wave Plate (PWP) which orients the incoming alternately-polarized optical channels to the U-axis to match the polarizers. By switching one pixel of the TSLM2 pixel pair to SWITCH and the other to NULL the combined signal $a+b$ is routed to the desired channel of the channel pair. Note

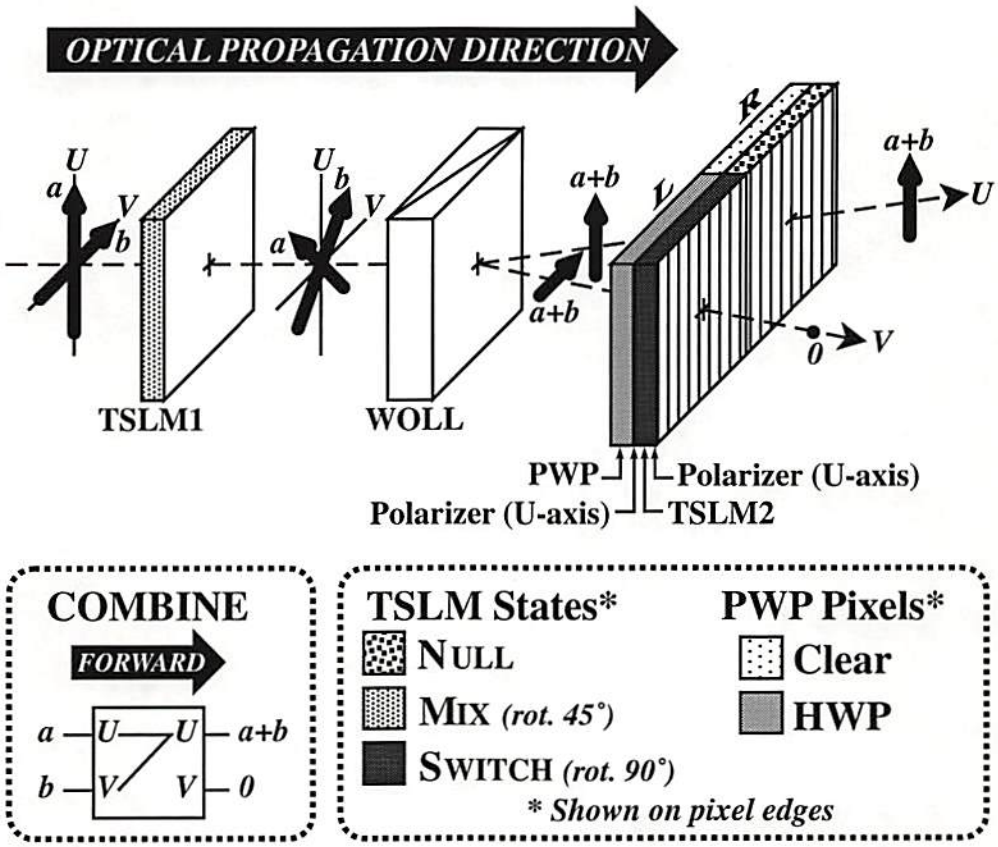


Figure 4.10: Overview: Shuffle–Exchange Stage Operation: Combine.

that since the implementation of the combine operation is embedded within the shuffle operation, channels U and V will not be recombined into a Channel Pair, but will be routed to different locations for the next stage.

The combine operation, when performed in reverse, results in a broadcast operation. Figure 4.11 provides an overview of a modified shuffle–exchange stage performing a broadcast operation. The broadcast is illustrated in the figure in the reverse (unshuffle) direction to emphasize the fact that it operates using the same optical hardware as the combine operation, but with the optical signals propagating in the reverse direction. The signal a in the input channel U is passed through TSLM2-R by setting it to NULL. It is then diverted onto the channel pair optical axis by the Wollaston prism. The TSLM1 is set to MIX to route 50% of the light to each output channel U and V. The horizontal and vertical components of the optical signal are thus 50% of the input a.

In a symmetric fashion, the design of the architecture forces the 50% loss per stage to occur in both directions of the optical network. The 50% loss due to the broadcast is obviously necessary in a passive network. Since the signal $a+b$ is sent to two channels, the output level is 50% of the input power. Since both broadcast and combine operations require 50% loss, it is required that all operations experience 50% loss so that the output signal levels from a pass through the FIER will be predictable. Therefore the "bypass" and "exchange" switch settings must have 50% loss added in. This is accomplished by setting the TSLM2 (both L and R pixels) to the MIX state for both cases.

Since in the envisioned application (the SMOEC) the external control of the FIER performs a "pre-arbitration" function on the messages to be combined, there

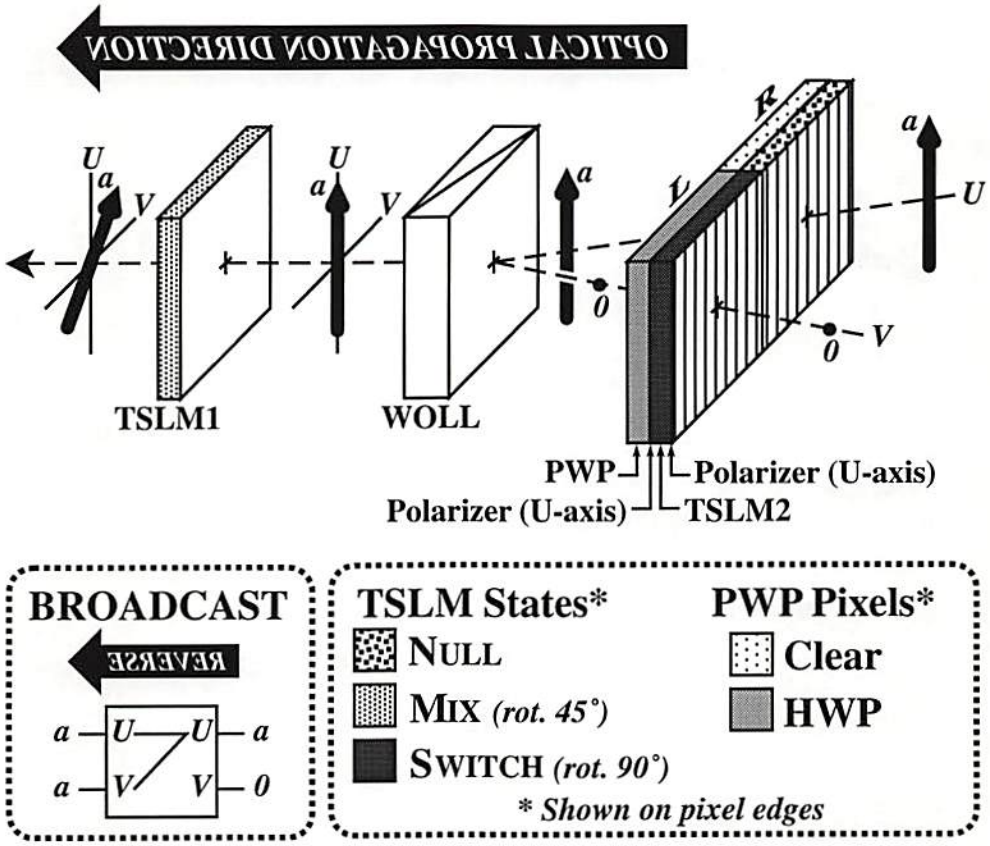


Figure 4.11: Overview: Shuffle–Exchange Stage Operation: Broadcast.

is no need for actual optical signals to be superimposed. The result of combining signals a and b to form $a+b$ is thus effectively a **OR** b (i.e., a and b are never both simultaneously "1"). Thus, in the "forward" direction where combining is taking place, no actual optical intensities are combined.

If the FIER is being built solely for certain applications where only permutations of input data lines are required (no broadcasts or combines), then the FIER may be simplified (called the basic-FIER). In this case, the original shuffle design depicted in Figure 4.8 with no TSLMs and only one HWP SLM placed between stages (in place of TSLM1) is sufficient. No additional loss is necessary.

Figure 4.12 shows where the two TSLMs are added to the optical unshuffle to enable the combine and broadcast operations. The bypass/exchange switch is embedded in unshuffle operation. The devices TSLM1 and Polarizer/TSLM2/Polarizer are used to implement the switch. The TSLM2 is sandwiched between two polarizers, next to the PWP. Each of these two polarizers consists of two halves, with the upper half set to pass the $\odot$-polarized channels in the upper half of the input at plane IMG2, and the lower half set to pass the $\updownarrow$-polarized channels in the lower half of the input at plane IMG2.

The Pol/TSLM2/Pol sandwich and PWP used in Figure 4.12 is shown in expanded form in Figure 4.13a. The TSLM2 has been described as having twice as many pixels as as TSLM1, operating in pairs $L;R$ which are always set to either NULL;SWITCH, SWITCH;NULL, or MIX;MIX. However, the second state (R) in each of these pairs is obtainable from the first (L) using a HWP pixel. Thus, instead of providing a TSLM2 with twice as many pixels as TSLM1 (Figure 4.13a), a fixed PWP$'$ with alternating HWP and clear pixels may be used to

Figure 4.12: Raytrace: An Unshuffle–Exchange Stage with Broadcast and Combine Capability.

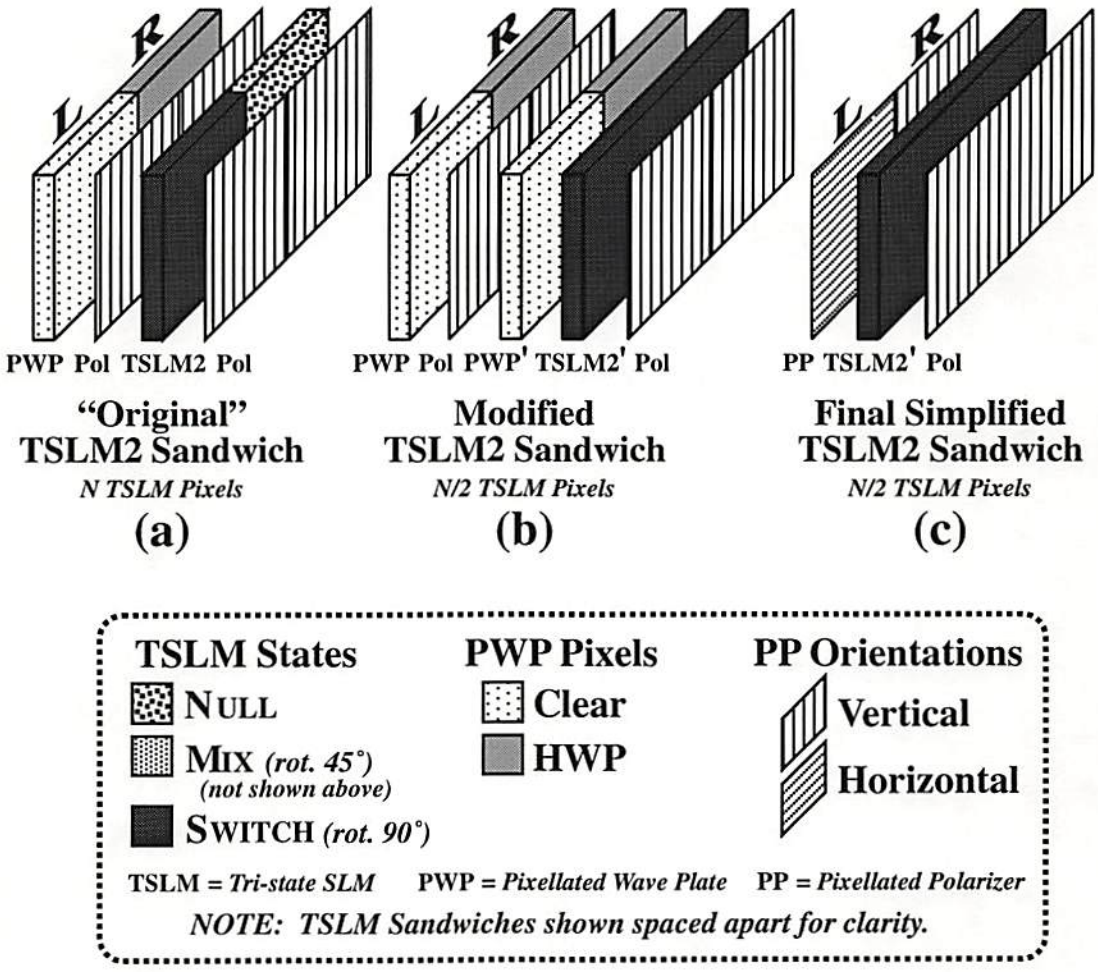


Figure 4.13: Two Implementations for TSLM2.

Operation	TSLM1	TSLM2-L	TSLM2-R
Bypass *Forward/Reverse*	NULL straight through	MIX 50% loss	MIX 50% loss
Exchange *Forward/Reverse*	SWITCH swap signals	MIX 50% loss	MIX 50% loss
Forward **Upper Broadcast** *Reverse* **Upper Combine**	MIX split/combine signals	NULL pass signal	SWITCH block signal
Forward **Lower Broadcast** *Reverse* **Lower Combine**	MIX split/combine signals	SWITCH block signal	NULL pass signal

Table 4.1: TSLM settings.

provide the necessary pixel pair from a single TSLM2′ pixel (Figure 4.13b). In this case, both TSLM1 and TSLM2′ have an identical number of pixels ($N/2$), and the added PWP′ has twice as many pixels (N). This modified sandwich can be modified into a final simplified form (Figure 4.13c) by combining the effect of the PWP/Polarizer/PWP′ combination into a Pixellated Polarizer (PP). The PP has alternating pixels of vertical and horizontal polarizing material. No wave plates are needed. The settings of the TSLM devices to perform the bypass/exchange operations are summarized in Table 4.1. Although the TSLM2-R and TSLM2-L (R & L pixels of a switch pair) settings are listed separately in the table, they are actually implemented in a single pixel, as described above.

4.3 Optical EGS Hardware Design

The previous sections have described the optical implementation of a basic conventional shuffle–exchange network. This section describes the additional specialized

optical subsystems necessary to implement an optical RS-EGS network (see Section 3.3.2). In addition to a "main section" of simple shuffle–exchange stages as previously described, an optical RS-EGS network requires special "fan-out" and "fan-in" hardware, and an implementation of F-shuffles.

For the case of the FIER, since the (forward) combine and (reverse) broadcast operations are unidirectional, the "fan-out" stage (following the inlet nodes) is not a true fan-out but a $1\times F$ demultiplexer. However, the "fan-in" stage (preceding the outlet nodes) is a true fan-in stage, which is permanently set to combine incoming signals (and broadcast signals in the reverse direction).

4.3.1 EGS "Fan-out" (demultiplexer)

The $1\times F$ "fan-out" stage in an RS-EGS network is implemented in the FIER as a sequence of $f=\log_2 N$ stages of 1×2 switches. This "fan-out" stage is also used in the FIER to perform the initial LA→CP conversion of input signals into the network. Thus the FIER "fan-out" is composed of $f-1$ stages of LA-format 1×2 switches followed by one stage of LA→CP 1×2 switches.

An overview illustration of the two states of a single LA-format 1×2 switch is provided by Figure 4.14. The LA-format 1×2 switch is composed of a CP→LA converter (the reverse of the LA→CP converter of Figure 4.1) preceded by a HWP SLM pixel (switch). This switch takes a LA-format optical signal, and performs either of two operations, Divert-UP or Divert-DOWN, sending the signal into one of two outlet LA-format channels.

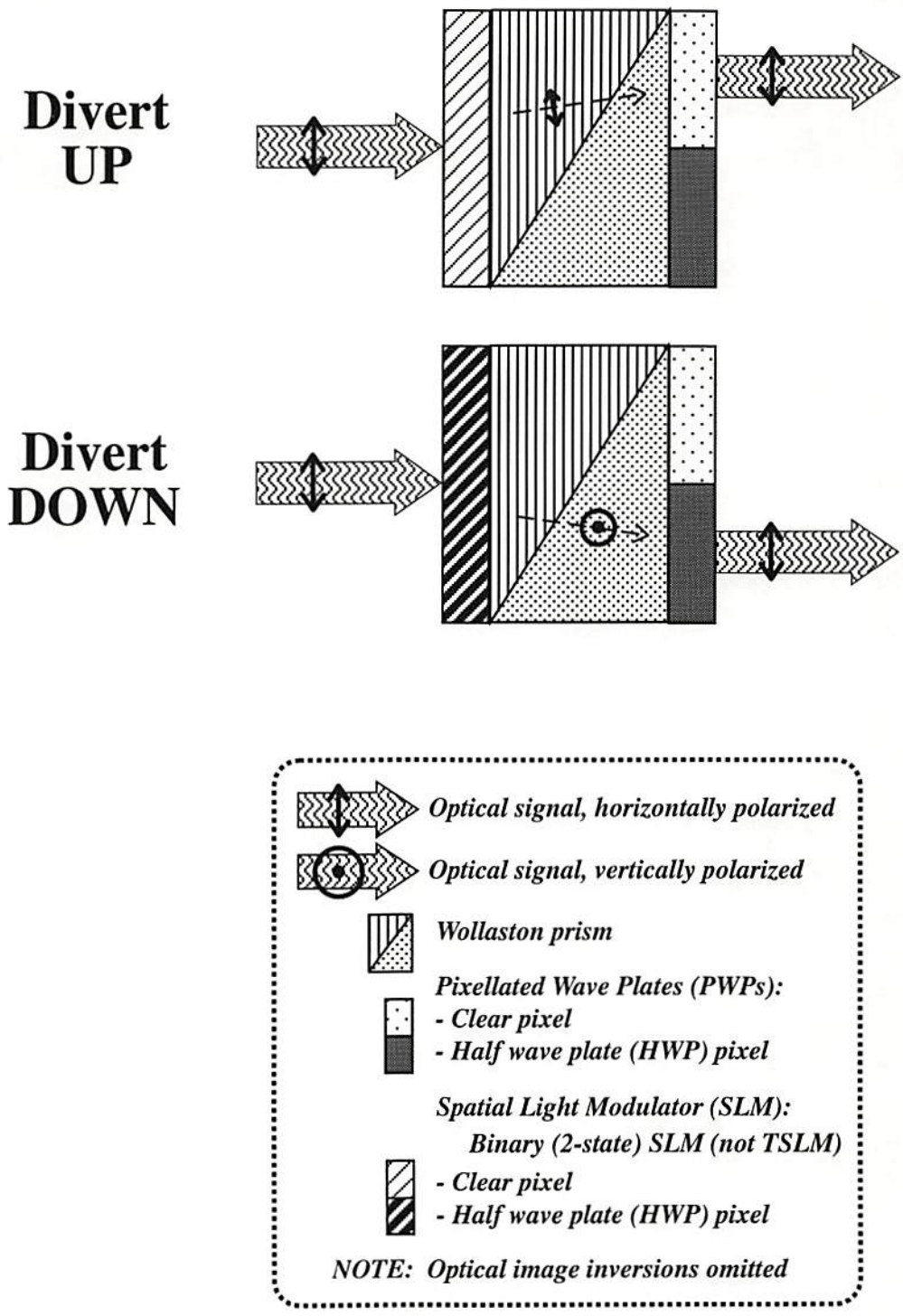


Figure 4.14: An LA-format 1×2 switch, used in "fan-out" implementation.

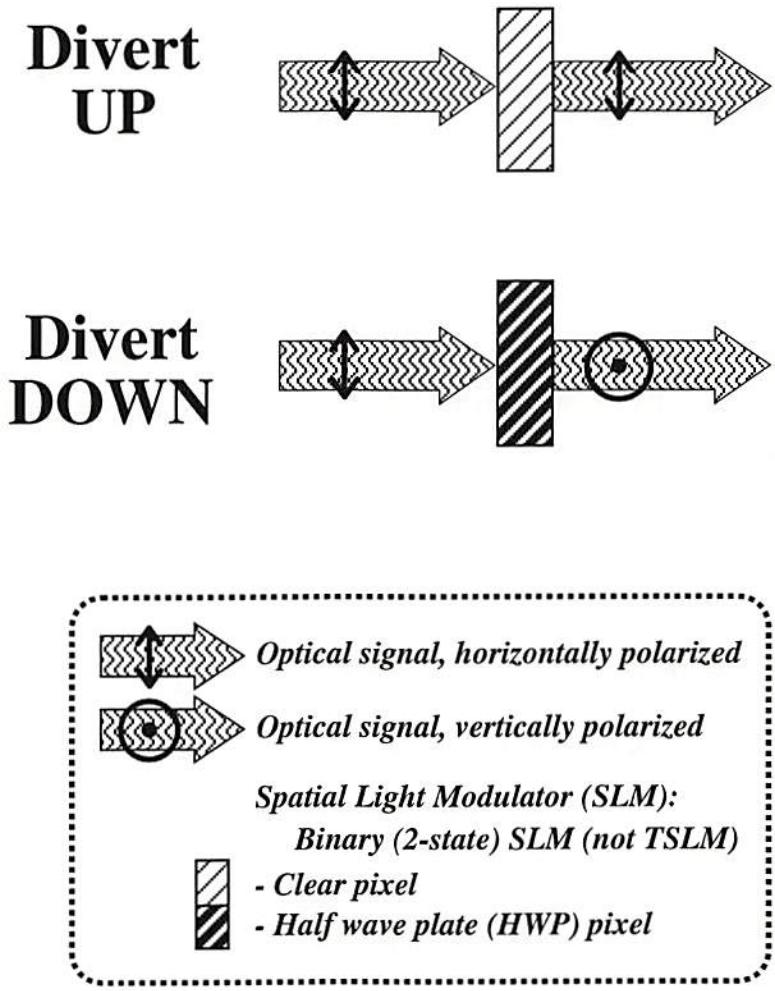


Figure 4.15: An LA→CP-format 1×2 switch, used in "fan-out" implementation.

Figure 4.15 provides an overview illustration of the two states of a single LA→CP 1×2 switch. This switch is merely a single HWP SLM pixel, which performs the necessary signal diversion in place (since Divert-UP or Divert-DOWN refer to the channel index number, not to physical location). Thus the switch takes a LA-format optical signal and sends it into one of two outlet CP-format channels.

A full $1\times F$ "fanout" (demultiplexer) stage consists of $f-1$ stages of the LA-format 1×2 switches of Figure 4.14 (with varied widths to achieve the appropriate offsets for each stage), followed by a single stage of the LA→CP-format 1×2 switches of Figure 4.15. Figure 4.16 illustrates a full 1×8 "fan-out" stage. The select lines shown in the figure are used to directly select which output channel is desired.

4.3.2 EGS F-shuffle

The implementaion of the F-shuffle in the FIER employs the result proved earlier in Section 3.1.2.3 that an F-shuffle operation on $N = 2^n$ nodes when $F = 2^f$ is equivalent to f successive applications of the perfect shuffle. Thus the FIER contains only the relatively easy to implement perfect shuffles (S_S stages in the main section and f stages for the final F-shuffle) instead of a complicated additional apparatus designed specifically for the F-shuffle.

4.3.3 EGS Fan-in

The final fan-in stage of $F\times1$ switches required for the FIER has a simple implementation when the F-shuffle is implemented as a sequence of f perfect shuffle stages. Compare the connectivities shown in Figure 4.17 and Figure 4.18. These

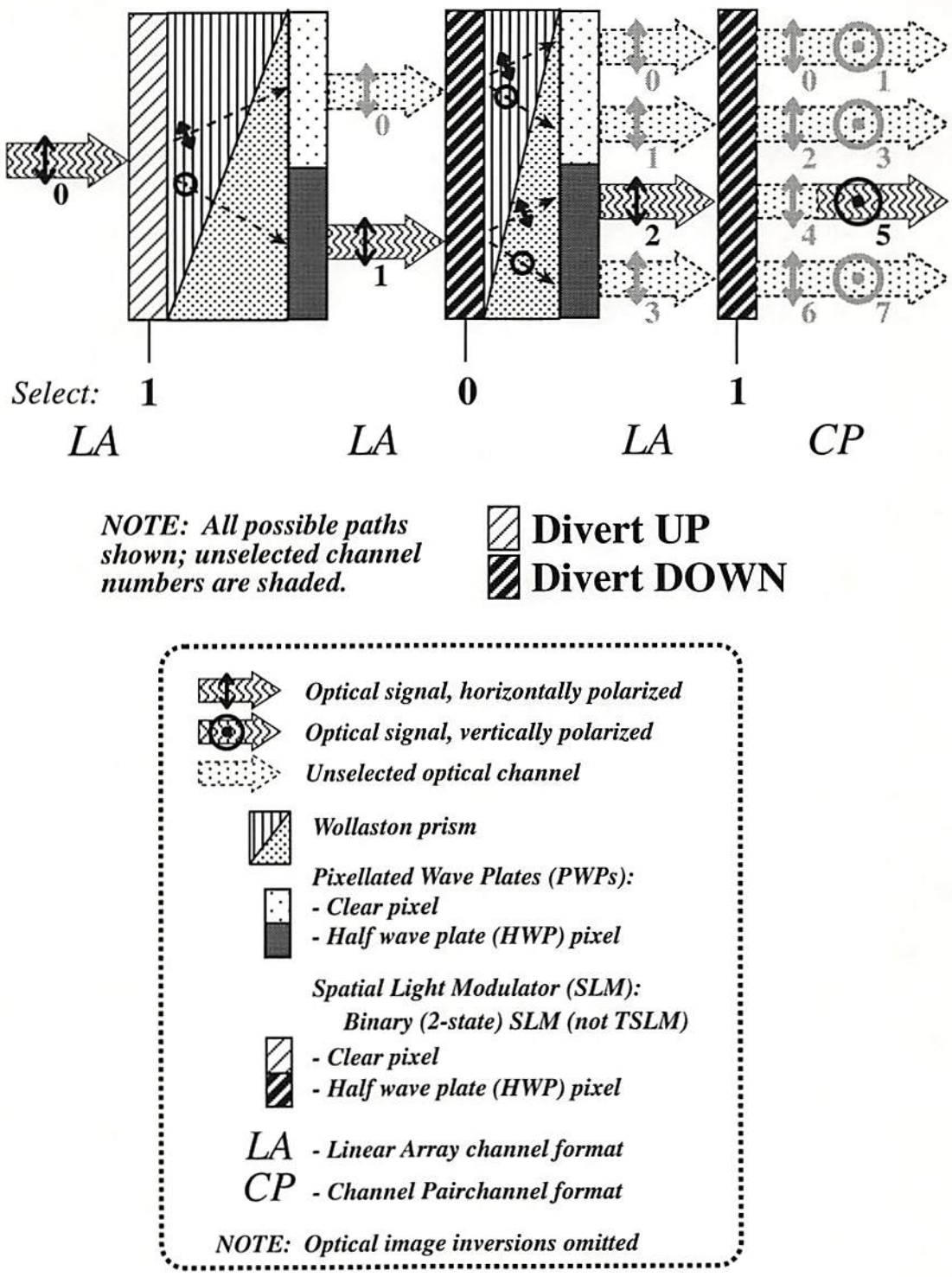


Figure 4.16: A full 1×8 "fanout" (demultiplexer) stage

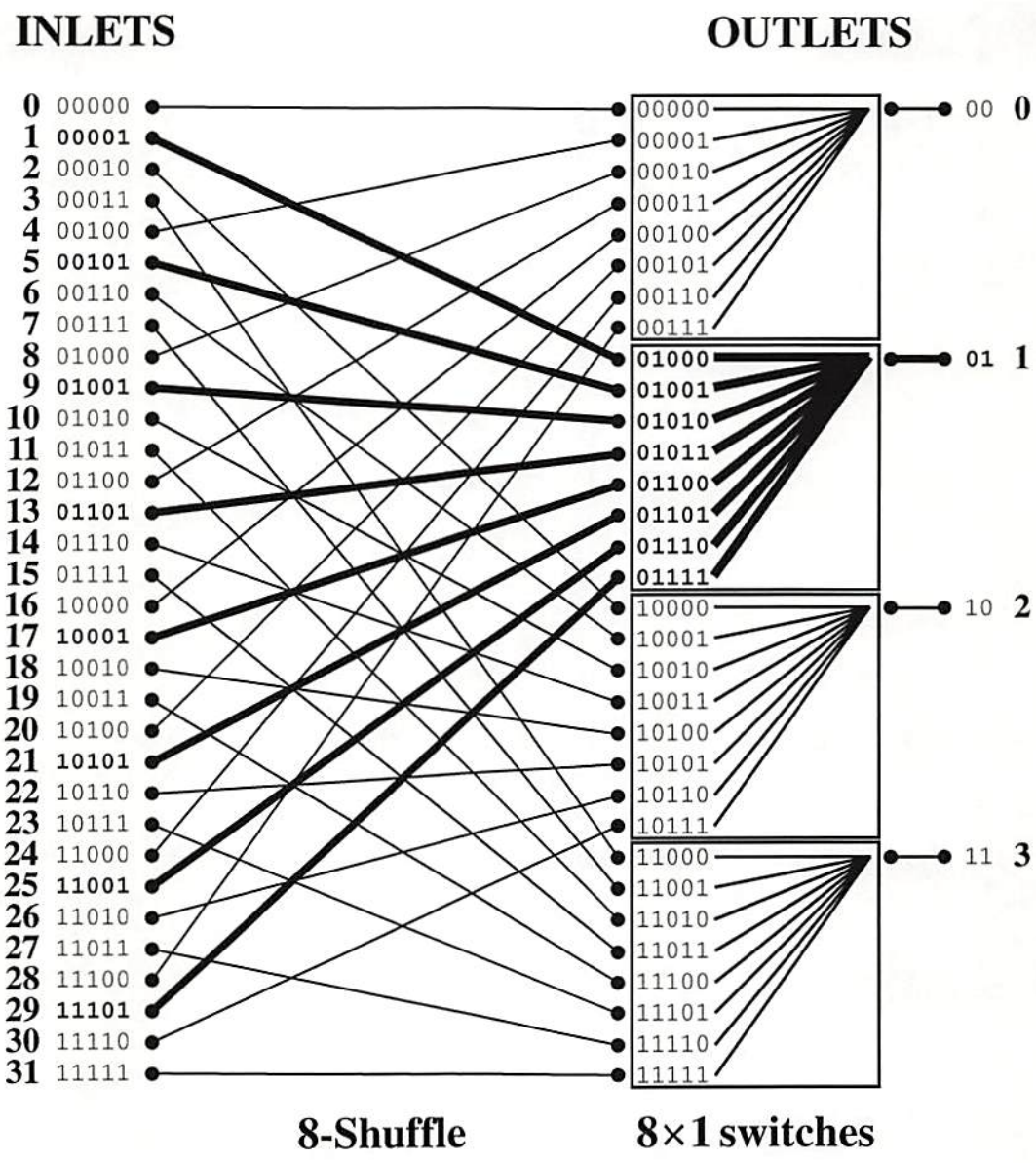


Figure 4.17: An 8-shuffle on 32 nodes followed by an 8×1 fan-in switch

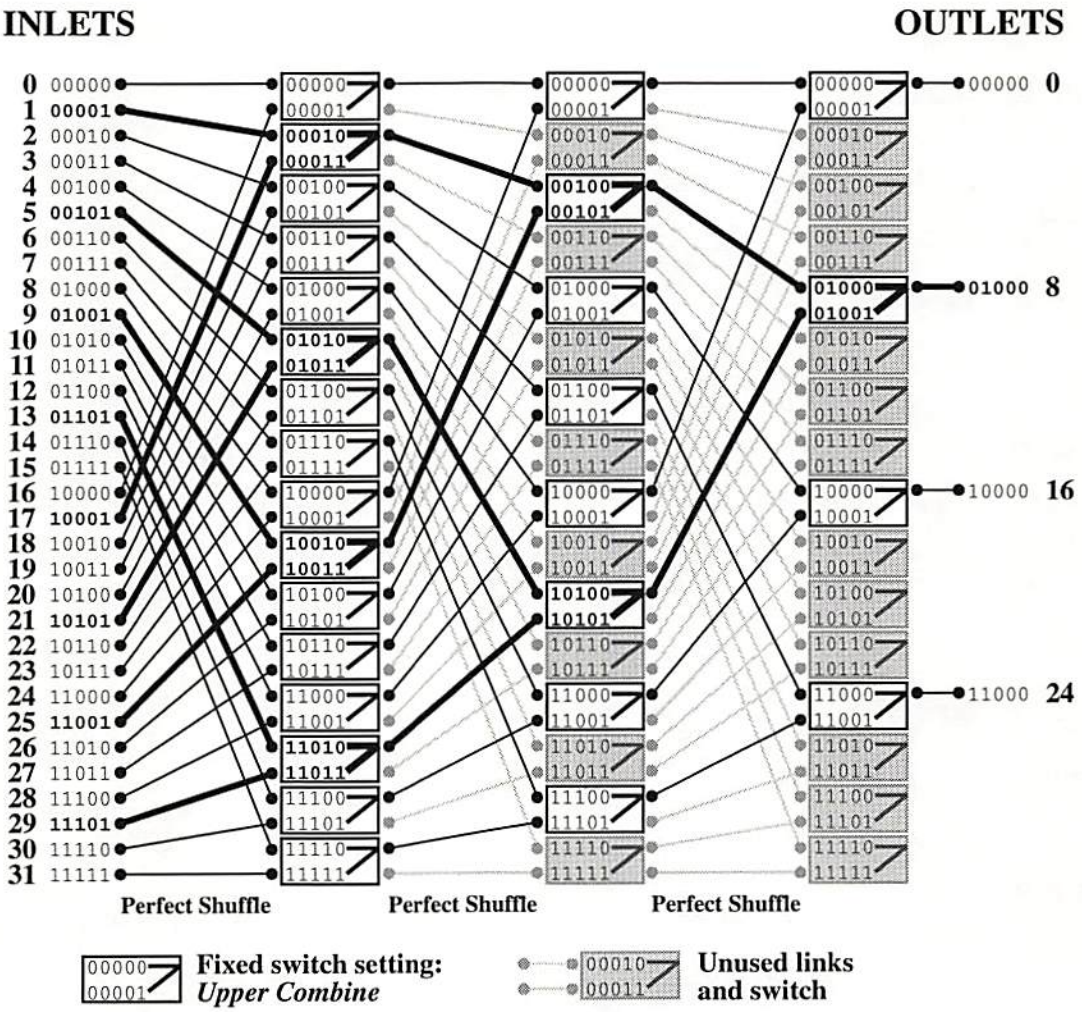


Figure 4.18: Three perfect shuffles on 32 nodes interspersed with pairwise combine switches

figures illustrate that parwise upper combine operations between f perfect shuffle stages effectively performs the same operation as an F-shuffle followed by a $F \times 1$ switch stage. This is proven by noting that a F-shuffle operation followed by a $F \times 1$ fan-in operation on an n-bit index i selects the rightmost $n-f$ bits from i. This is effectively equivalent to the f consecutive applications of the set of operations of perfect shuffle and pairwise upper combine, which results in an address composed of the rightmost $n-f$ bits from i followed by f zeros. Thus the combination of the F-shuffle and $F \times 1$ fan-in required for the FIER is implemented by f shuffle exchange stages which are identical to those in the "main section" of the FIER, except that these stages are set permanently to combine. No TSLMs are needed for these stages, since this is a fixed setting; all that is needed is a fixed QWP plate instead of the TSLM1 in Figure 4.12 and a fixed PWP plate instead of the TSLM2.

4.4 Optical Implementation Issues

Optical losses throughout the FIER depend strongly on S_T, the number of shuffle–exchange stages in the network. Define s to be the number of multiples of $\log_2 N$ shuffle–exchange stages, such that $S_T = s \log_2 N$. If broadcast and combine operations are allowed, the architecturally required optical loss A in each stage is 50%, as described in the previous section. However, there will of course be additional loss in each stage due to surface reflections and optical absorption in the optical components. Let the additional loss factor per stage be L (so that the total loss per stage is AL). Define further $\ell = 1/L$. Note that since $0 < L \leq 1$, then $\ell \geq 1$,

thus $log_2 \ell \geq 0$. Then the total optical loss factor throughout the entire network becomes:

$$\begin{aligned}
\mathcal{L} &= [AL]^{S_T} && (4.1)\\
&= \left[\tfrac{1}{2\ell}\right]^{s \log_2 N} \qquad \text{(applying definitions)} && (4.2)\\
&= 2^{s \log_2 N \log_2[\frac{1}{2\ell}]} \qquad \text{(by identity: } a^x = b^{x \log_b a}\text{)}\\
&= N^{s \log_2[\frac{1}{2\ell}]} \qquad \text{(collapsing: } N = 2^{\log_2 N}\text{)}\\
&= \left[\tfrac{1}{N}\right]^{s \log_2(2\ell)}\\
&= \left[\tfrac{1}{N}\right]^{s(1+\log_2 \ell)} && (4.3)\\
&\leq \left[\tfrac{1}{N}\right]^{s} && (4.4)
\end{aligned}$$

Consider the example of an Omega network, but enhanced with switches requiring $A = 50\%$. Since $s = 1$ for an Omega network by definition, assuming no additional optical component loss ($L = 1$) yields total loss $\mathcal{L} = 1/N$. If, however, this enhanced Omega network also has $L = 50\%$, then $\ell = 2$, and the total loss factor becomes $\mathcal{L} = 1/N^2$. For the case of the more general FIER, neglecting optical component loss ($L = 1$) yields $\mathcal{L} = \left[\frac{1}{N}\right]^s$. But, if L is 50%, then the total loss factor becomes $\mathcal{L} = \left[\frac{1}{N^2}\right]^s$. Therefore, minimization of the optical loss due to practical causes such as surface reflections and absorption is essential.

Considering the loss due to A alone, it is clear from equation 4.4 that there will be a minimum of a factor of $1/N$ loss for every multiple of $\log_2 N$ shuffle–exchange stages in the FIER. Therefore the addition of optical repeater hardware may need to be considered if the total loss factor $\mathcal{L}$ is unacceptably high.

Optical crosstalk in the network is due to two primary factors: SLM-induced crosstalk and channel spillover. Crosstalk induced by the SLMs will affect the signals present in channel pairs due to the nonideality of the HWP pixels. The Wollaston prisms will contribute negligible crosstalk to the channel pairs as discussed in [Johnson 88]. This reference also states that up to 31 HWP SLM passes may be cascaded with distinguishable output levels. Crosstalk in a potential QWP SLM has not yet been studied.

Channel spillover is a result of misalignment and focussing errors and scattering in the optical system. Of primary concern is error propagation throughout the system. If errors are multiplied as they pass from stage to stage, the system will not be able to function. Small errors in ray angles, pixel spacing, and pixel size/shape must be prevented from propagating through the system. If this is done properly these quantities are effectively restored to the same values at each pass through a shuffle–exchange stage, thus ensuring cascadability. Two masks will eliminate the propagation of such small errors. A mask containing pixel-sized holes at or between the lenslet arrays shown in Figure 4.12 will eliminate propagation of small ray angle errors. A mask containing pixel-sized holes at either image plane that has double-spaced pixels (those between L_2 & L_3 and between L_4 & LL_3) will eliminate pixel spacing, size, and shape errors.

The FIER is homosyndetic; that is, it is based on repeated identical stages. In this way, it can be more flexible to optical implementations than non-homosyndetic but functionally equivalent multistage networks. The optical implementation of a homosyndetic network may incorporate multiple stages within a single set of interconnection optics (including switching arrays) by partitioning the arrays into

separate regions for each stage, and including mirrors in the optical setup to relay the output of one stage (array region) to the input of the next. For example, a single 256×256 (256^2 pixels) FIER stage can be redesigned to be partitioned into 32 strips 256×8 containing $N=2048$ pixels. Then choosing $S = 2\log_2 N$ stages gives 22 stages. Since 22<32, a full network can be constructed using only a single set of interconnection optics. This leaves 10 unused 256×8 pixel strips which can be used for other purposes, such as potentially adding fault tolerance capabilities. Such a switch would be commonly referred to as a 2048×2048 switch, referring to the ability to connect 2048 inlets arbitrarily to 2048 outlets (note that this nomenclature is unrelated to the pixel array dimensions used elsewhere in this paragraph). In this way, the homosyndetic property of the FIER may be exploited for flexibility of implementation, allowing choice for the appropriate partition of S individual FIER shuffle–exchange stages between 1 to S physically separate sets of interconnection optics.

The previous paragraph mentions two-dimensional shuffle–exchange stages; however, the optical shuffle–exchange stages presented herein are only one-dimensional. The one-dimensional shuffle–exchange stages may be converted to two–dimensional shuffles by using techniques described in [Cheng 92]. The authors describe a 2-D separable shuffle which is composed of a 1-D shuffle along one dimension followed by a 1-D shuffle along the other dimension. The authors describe how to make a full access network using the 2-D separable shuffle and sets of horizontal and vertical 1-D pairwise bypass/exchange switches. The explicit analysis and mapping to apply the results for the network detailed in [Cheng 92] to the conversion of restricted-F RS-EGS networks to 2-D operation and the consequent modification

of the optical hardware used in the FIER optical network to a 2-D geometry are left for future work.

The FIER is designed to be ultimately scalable to large array sizes. The use of lenslet arrays means that diffraction of unfocussed beams will not limit scalability to large arrays. Limitations of large-scale implementations will be SLM resolution, optical aberrations, and alignment sensitivities. SLM speed will not be a progressively worse concern as larger systems are considered if optically addressed SLMs are used. Optically addressed SLMs are free from pixel-addressing bottlenecks which would scale poorly with system size.

4.5 Comparison

The FIER optical shuffle–exchange network design presented herein is only one of many possible implementations. Several optical shuffle implementations [Lohmann 86a, Lohmann 86b, Lin 87, Sawchuk 87, Stirk 88, Brenner 88, Sawchuk 88, Jutamulia 89, Sheng 89] have been proposed. However, for use in a passive optical shuffle–exchange network several requirements must be met. The network must reduce light waste (such as light lost at a mask) as much as possible and practical. The network must be scalable to reasonably large values of N. The most important requirement is that it be cascadable—that is, any slight change in format in spacing, angular spread, pixel size and shape, and optical wavelength from input to output of a stage must be corrected at each stage. The optical shuffles referenced above were not designed with a passive multi-stage implementation in mind, so they do not satisfy these requirements: all but [Sawchuk 87] use masks which lose at least 75% of the light (50% in the 1-D shuffles), and none reproduce

both pixel spacing and pixel angular spread. The basic-FIER stages (the FIER with one HWP SLM instead of two TSLMs) are cascadable and have no architecturally required losses. The normal FIER stages are cascadable, but in order to incorporate the broadcast and combine capabilities a 50% loss per stage has been incorporated into the design.

Another consideration in optical shuffle design is compatibility with a practical exchange switch technique. Most of the optical shuffle implementations mentioned above (all but [Lohmann 86b, Sheng 89], which both suffer the 75% architecturally required loss) are intended to be used with electronic exchange switches. The shuffle described in this paper is designed to work with a passive optical switching technique. Switchable half-wave plates (HWPs) are employed in the FIER because the Channel Pair configuration is designed for this type of exchange switch. The extended switch settings are also easily implemented using TSLMs as discussed previously.

One optical interconnection network that is similar to the FIER is the optical Benes network [Noguchi 91]. A Benes network is composed of butterfly connections of varying sizes, and has similar capabilities to a shuffle–exchange network. The optical Benes network is a passive implementation with cascadable interconnection stages. The Benes network is not homosyndetic (i.e. it does not employ identical stages), so there is no corresponding added flexibility for optical implementations. The optical Benes network does not implement broadcast or combine operations, unlike the FIER. Both networks are circuit-switched by external electronic means. The polarization multiplexing of channels in the optical Benes network is the same as the Channel Pair technique used in the FIER. The pixels in the optical Benes

network are collimated light beams, so this leads to a simpler optical setup. However, the scalability of the system is limited because both the number and length of optical channels is limited (pixels cannot be too small or propagate too long due to diffraction). The FIER uses focussed pixels, so it is capable of employing greater numbers of smaller pixels over longer distances. Thus the FIER is more suited to the goal of implementing a fine-grained system such as the SMOEC.

4.6 Summary

The described optical interconnection network, the Free-space Interconnection with Externally-controlled Routing (FIER), can provide parallel high bandwidth data throughput using passive switching with an external circuit-switching control technique. Reconfiguration times are limited to the time for one TSLM to switch, since all can switch in parallel. A shuffle–exchange topology was selected because it is flexible and well characterized. The optical shuffle–exchange stage was designed to be optically cascadable to permit all-optical bidirectional data flow between source and destination nodes. Novel aspects of the FIER include the ability to combine and broadcast signals in predefined directions, and a design that was based on practical control algorithms. The FIER was designed for use in the Shared Memory Optical/Electronic Computer (SMOEC) system, but since it is self-contained it may also be used as a circuit-switched subsystem in other parallel or distributed computing systems.

Chapter 5

NETWORK ROUTING ALGORITHM

As previously stated, the design philosophy for the SMOEC was a triple focus on architecture, hardware, and control algorithms. The interplay between concerns in these three areas was essential to the design of a feasible system, lest the optimization of one or two of these areas impose impractical requirements on the third area. Since control algorithms are an essential element of parallel computer design, with wide impact on the resulting computer system, sufficient detail about these algorithms must be provided for the system description to be complete. In novel architectures, such as the proposed hybrid optical/electronic system, the control algorithms need especial attention to tailor them to unusual features of the design. Thus this chapter and the following chapter are devoted to a detailed exposition of the control algorithms developed for the SMOEC.

The control algorithms used in the interconnection network in the SMOEC can be broken down into two aspects: routing and communication. The routing

algorithm specifies how data from one node (or multiple nodes) are sent through the network to arrive at a destination node (or vice versa). The communication algorithms (discussed in the next chapter) are built on the routing algorithm to provide necessary facilities, such as multiple reads and multiple writes. This chapter focusses on the routing algorithm itself, covering the design impact of the MATSH control unit on the routing algorithm (and vice versa), the theory behind the routing algorithm, and the routing algorithm simulation and results.

5.1 Interrelation of the MATSH Control Unit and Routing Algorithm Design

The structure and connections between the FIER and the MATSH are illustrated in Figure 2.2. The FIER consists of a sequence of S_F Optical Shuffle–Exchange stages (O-SEs). The MATSH consists of a single Electronic Shuffle–Exchange stage (E-SE) with bidirectional feedback connections. A forward pass through the MATSH is defined as S_F forward cycles through the E-SE, and the reverse pass is defined correspondingly. A memory stack within each switch node in the E-SE allows it to provide dedicated memory space to correspond to each switch in the FIER, creating the effect of S_F independent switch stages in the MATSH which are accessed sequentially forward or backward by pushing or popping information to and from the stacks. The MATSH is thus defined to consist of a set of S_F stages of *virtual* switches (to simplify the following discussions of algorithms) in which the $m^{\underline{\text{th}}}$ stage of virtual switches in the MATSH corresponds to the $m^{\underline{\text{th}}}$ actual cycle through the E-SE. (However, for the purpose of simplifying the discussion of

the routing algorithm in the next section, the MATSH will be discussed there as if it consists of S_T stages, consisting of a stage of virtual $1 \times F$ switches followed by the regular main section of S_S stages of virtual 2×2 switches followed by a stage of virtual $F \times 1$ switches.)

A pass through the MATSH is thus designed to topologically mimic a single pass through the FIER in either the forward or reverse direction, with the difference that the switches within the MATSH are capable of active logical switching decisions whereas the switches within the FIER are passive switches which require external control. Complex routing and communication algorithms (e.g. finding unblocked paths or combining write requests) are implemented on the MATSH by employing multiple forward and reverse passes, in which the final pass through the MATSH produces the required switch states for the FIER. Since the MATSH operates by topologically mimicking the FIER, the resulting control bits computed by the MATSH are exactly those needed for setting the FIER bypass/exchange switches. In particular, when the MATSH performs its final pass to complete the routing algorithm and finalize the connections and switch states, the $m^{\underline{\text{th}}}$ pass through the E-SE computes the control bits for the $m^{\underline{\text{th}}}$ O-SE of the FIER. The full set of control bits for the entire FIER are thus computed by the MATSH (and stored in the CSDI buffering unit) in a sequential stage-by-stage manner, but the full set of control bits is sent from the CSDI to the FIER in parallel, thus circuit-switching (see Section 2.5) the FIER.

The control bit computation in the E-SE switches is designed to rely solely on local information within the switches. Since control bit computation is localized, algorithms inspired by those that are used to route data on a packet-switching

shuffle–exchange network are appropriate for the MATSH, which in turn operates the FIER in a circuit-switched manner. Since the passive FIER prohibits data buffering within its nodes, the MATSH also mimics this aspect by operating without buffering as well. Therefore only packet switching types of algorithms that do not use data buffering may be employed in the MATSH.

Another motivation for the use of a packet-switching type of algorithm in the MATSH is the extra data manipulation required by some of the communication algorithms. In general, the MATSH works by operating solely on the destination addresses of the data. However, small amounts of data, and special *subaddress fields* are operated on by the MATSH to perform write request combining and arbitration (which will be explained in Section 6.2 and 6.3). To permit such functionality, a routing algorithm is required that can incorporate the necessary local pairwise operations. Existing packet switching algorithms may be modified to accomodate these operations, because the locality of pairwise routing decisions in packet switching algorithms fits easily with additional local pairwise address- or data-based decisions.

The extended bypass/exchange switch settings are used to alleviate multiple-access (output port) conflicts in the FIER. If several PEs are trying to communicate (in any given phase) with the same MM, the routing algorithm assumes (requires) that these requests are combiniable. The routing algorithm implementation on the MATSH provides that these combinable requests will meet in a pairwise manner at separate virtual switches in the MATSH. These switches are then set to the upper or lower combine states, thus providing a binary tree-shaped path combining the several PEs' outputs to a single inlet at the desired MM. Conversely, in the

MM→PE direction, this tree operates to give a broadcast path from the MM to the several requesting PEs. The combining/broadcasting tree path provided in this manner is completely passive; it acts like a fan-in wire from the PEs to the MM (and fan-out in the reverse direction). The communication algorithms in the SMOEC are designed to make use of these passive combining/broadcasting paths (see Section 6).

In certain applications, it is known in advance that only a limited set of one-to-one connection patterns (permutations) will be utilized (see Section 7.2). In these types of applications, the cycling of addresses through the MATSH (for the purpose of switch state computation) can be completely eliminated, and previously stored control bit sets can be read directly into the switches in the FIER. If the computer is to be used only for this class of applications, a modified SMOEC may be designed that replaces the MATSH with a control bit set storage bank. (In this case, the FIER may be simplified as well, as discussed in Section 7.2.) If the applications do not require extremely frequent changing of the connection patterns this simplified SMOEC may be particularly practical. For these types of applications, a SMOEC may be further simplified to operate solely in a SIMD mode if desired. For more information, see Section 7.2.

5.2 The FLAEM Routing Algorithm

A nonblocking EGS network has a multitude of paths available between each inlet-outlet pair. Richards' path hunt algorithm [Richards 91b] completes a single routing request in constant time. It parallelizes the routing of multiple requests by pipelining the requests and by processing a small constant number of these pipelined

requests in parallel. Thus Richards' algorithm takes $\mathcal{O}[N]$ time to process N simultaneous routing requests. A new parallel algorithm, the Flexible Localized Algorithm for EGS-network Management (FLAEM), is presented here which processes each of N routing requests in parallel for a combining RS-EGS network. Since each routing request is processed in approximately $\mathcal{O}[\log N]$ time, the FLAEM routing algorithm can control a circuit-switched combining RS-EGS network in approximately $\mathcal{O}[\log N]$ time.

The following sections will explain how the FLAEM was designed, expound the inner workings of the algorithm, and trace through a specific example to illustrate the routing process.

5.2.1 Motivation

Richards' path hunt algorithm was designed to operate a network in which connection requests arrive asynchronously. This algorithm is particularly appropriate for networks where individual connections are maintained for relatively long times so that only relatively small numbers of simultaneous connection requests will occur (such as in telecommunications applications). These small numbers of requests are handled very fast since each request is processed in $\mathcal{O}[1]$ time (i.e. constant time). Richards' algorithm was not designed to deal with N simultaneous requests. In contrast, the SMOEC is designed to simultaneously request N new connections with every communication cycle. Thus the SMOEC required the design of a new routing algorithm with reduced complexity from the $\mathcal{O}[N]$ time of Richards' path hunt algorithm.

The FLAEM routing algorithm was designed to operate on an electronic routing processor that would have $\mathcal{O}[\log N]$ stages. The goal was to create an algorithm that would satisfy N simultaneous connection requests to a N-inlet/N-outlet network in a small constant number of passes through the routing processor, which would result in an algorithm that processs N requests in $\mathcal{O}[\log N]$ time. Such an algorithm would then be preferred to Richards' path hunt algorithm in cases where greater than $\mathcal{O}[\log N]$ simultaneous requests need to be processed.

The $\mathcal{O}[\log N]$ FLAEM was designed by analyzing the way the path hunt process works in RS-EGS networks, and the algorithm exploits the multitude of paths available between each inlet-outlet pair by trying many candidate paths in parallel. Optimization of this technique led to to the particular design of the FLAEM (presented in the following section). The data from repeated simulation runs show that the FLAEM does require only a small number (5 for the simulated cases) of forward and reverse routing passes through the routing processor.

The FLAEM routing algorithm was designed to be implemented in parallel using the MATSH routing processor. Each node in the MATSH is a multifunctional switch containing an enhanced bypass/exchange switch, some memory, and some elementary logic functions. Nodes in the MATSH are capable of true fan-out (in the forward direction) so that multiple copies of routing requests may be produced and used to try multiple paths in parallel. Once the full set of switch settings is calculated by the FLAEM, these settings are sent in parallel to circuit-switch the FIER optical network.

To facilitate the explanation of the FLAEM, the MATSH will be described as if it consists of S_T stages, consisting of a stage of virtual $1 \times F$ switches followed

by the regular main section of S_S stages of virtual 2×2 switches followed by a stage of virtual $F \times 1$ switches. (This differs from the convention employed in the previous section which described the MATSH as consisting of $S_F = S_S + 2f$ stages, which mimics the FIER optical network more explicitly and indicates more exactly how the MATSH operates since it actually uses S_F cycles.) Processing by the MATSH is therefore restricted to parallel operations across the switches within a single stage, and progressing (forward or reverse) to other stages sequentially stage by stage.

Each virtual switch of the MATSH routing processor provides a small amount of dedicated memory space to store information associated with the requests it is processing. Information stored for each connection request includes the path vector (which incorporates the source and destination indices), a priority value, and various routing state flags.

In the SMOEC, multiple requests that are destined for the same outlet node are always assumed to be combinable. Thus the FIER optical network is capable of implementing many-to-one connection patterns in the forward direction. This combinability is an essential assumption of the FLAEM method. Thus, the FLAEM is applicable only to EGS networks that can work with such a strong combinability assumption. However, a trivial case of this assumption (zero requests destined for the same outlet node) yields the useful result that the FLAEM is also applicable to EGS networks that are externally restricted to process only permutation (one-to-one) connection patterns.

5.2.2 Exposition

For specificity, several terms used in explaining the FLAEM procedure are defined here. A *connection pattern* consists of N individual *connection requests*, one for each of the N inlet nodes. Multiple connection requests may refer to the same outlet node. The FLAEM procedure makes F *request copies* of each individual connection request (with randomly selected path numbers and priorities), and routes these F request copies in parallel until each request copy either reaches a switch conflict, is combined with another request copy, or succeeds at reaching its destination outlet node. Virtual switches in the MATSH are *marked* by setting a state variable within each virtual switch. The state of each virtual switch is initialized to FREE. Individual request copies then set these state variables as they progress through the network. The state variable settings are RUN, THRU, FIXED, FREE, COMBINED, and CONFLICT. Two sets of settings are associated with each virtual switch, one for each outlet link from the switch. Thus the following discussion will discuss storing settings within links.

The Basic-FLAEM procedure to satisfy N connection requests is presented in three parts: P1, an initial forward routing pass to process in parallel F request copies of each connection request; P2, a reverse pass to communicate back to the inlet nodes which requests made it through to the outlet nodes, while erasing unsuccessful request copies; and P3, a forward pass to reroute F new request copies for each connection request that had zero successful request copies, and to finalize one selected (winning) successful request copy for each connection request that had a nonzero number of successful request copies, while erasing non-winning successful request copies. P2 and P3 are repeated as a unit until all requests are

satisfied. The flow chart in Figure 5.1 illustrates how these parts fit together. Each of these three FLAEM procedure parts is now explained in detail, followed by a description of the Enhanced-FLAEM algorithm, in which a single connection request may generate more than F request copies to be processed in parallel during each forward pass.

For purposes of evaluating the success of the FLAEM routing algorithm (discussed later), the execution of P1 and P2 count as the first *try*, and any subsequent executions of P3 followed by P2 count as additional *tries*. The final P3 forward routing pass when no unsuccessful connection requests are being rerouted is not counted as a *try*; this last forward pass is a finalization of all the switch settings. Thus, a *try* is counted every time new request copies are generated and routed through the network.

5.2.2.1 Initial Forward Routing Pass (P1)

First, the MATSH links are initialized to FREE. Then F copies of each connection request (marked RUN) are made by the initial stage of 1-to-F fan-out switches in the MATSH control unit. Each request copy is assigned a random unique *priority* value from 0 (highest priority) to $F-1$ (lowest priority). Each request copy is also assigned a path vector (see Section 3.4) which is composed of the inlet and outlet node indices plus a randomly selected EGS path number. This path vector specifies one of the many paths available from the inlet node to its destination outlet node. The priorities, path vectors, and state value (e.g. RUN) are stored in the memories associated with the links (using the indices that immediately follow the switches in the forward direction) as the request copy is routed.

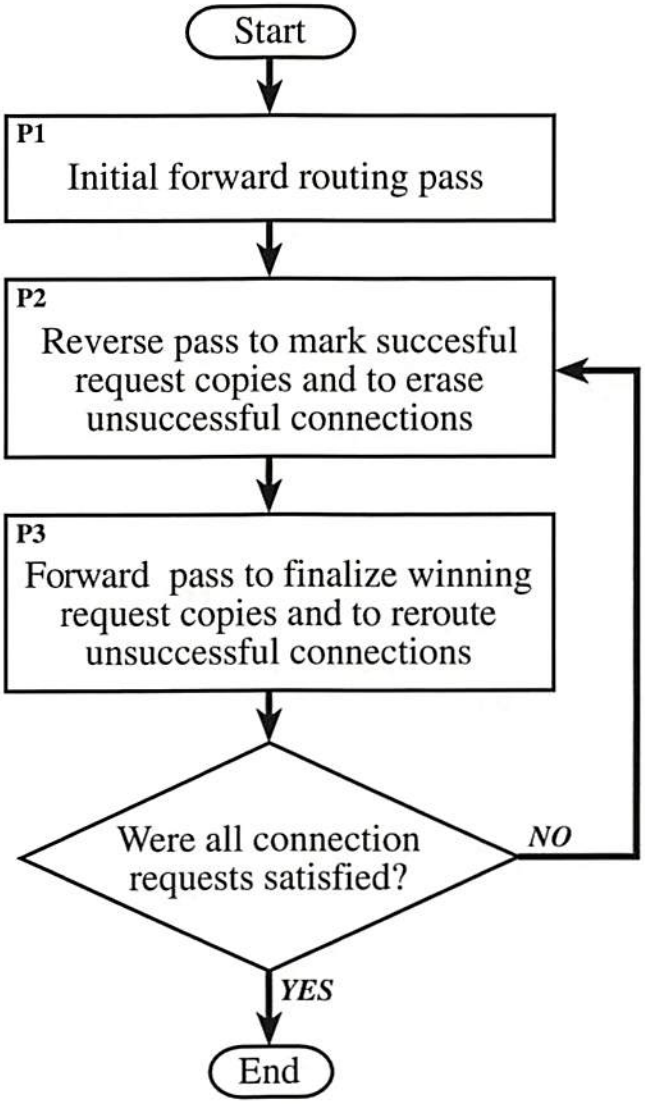


Figure 5.1: FLAEM Flow Chart

Stage by stage, the request copies are propagated forward whenever possible. The two possible inputs to each link in the MATSH are processed sequentially in random order, so that a link may be FREE or occupied on this first pass. Let L_n denote the link in the next stage that was specified by the path vector of a request copy at link L_c in the current stage. The request copy at L_c may be either simply routed forward, combined with another request copy, or aborted due to conflict with another request copy, as follows. That request copy is routed forward if the link L_n is marked FREE. If L_n is occupied by another request copy that is destined for the same outlet node, the request copy at L_c may be combined with it. In this case, the link L_c is marked COMBINED and not individually routed further. The request copy at L_n is given the highest of the two priorities (i.e. the minimum numerical value of the two priorities). The later return pass of P2 will further process the COMBINED request copy as approprate. If L_n is occupied by another request copy with an incompatible destination, its priority is compared the priority of the request copy at L_c, and the one with the higher priority wins while the other is aborted (marked CONFLICT). If L_c is a stage that is early enough in the MATSH such that it still has more than one path available to its destination outlet node (this is termed a *flexible* stage), an aborted request copy may then instead try to propagate or combine with the other available link that it can reach in the next stage (this is called the *modpath* technique). The modpath technique entirely prevents conflicts from occuring within the flexible stages in the network.

5.2.2.2 Reverse Marking/Erasing Pass (P2)

After the first pass through the MATSH, all request copies that made it to the final stage are marked THRU. A reverse pass through the MATSH is now performed, propagating THRU-marked request copies in the reverse direction (including request copies that were combined with the THRU-marked request copies), and erasing any aborted CONFLICT request copies (including request copies that were combined with aborted request copies) and setting the previously occupied link to FREE.

5.2.2.3 Forward Finalizing/Rerouting Pass (P3)

After this reverse pass, each inlet node has some number (possibly zero) of THRU request copies. An inlet node is termed *successful* if it has a nonzero number of THRU request copies. A single arbitrarily-selected winning THRU request copy (the lowest priority request copy, i.e. maximum numerical value of the priority variable, was selected for the FLAEM simulation) for each successful inlet node is marked FIXED. Inlet nodes with FIXED request copies are termed *satisfied.* If all inlet nodes with connection requests are either previously successful or newly satisfied, this execution of P3 will be the final pass. If some inlet nodes had zero THRU request copies (and were not previously satisfied), then F new RUN request copies are generated for each unsuccessful inlet node connection request. Each set of F new RUN request copies is assigned new random unique priorities and individual path vectors as previously described for Part(1). The FIXED request copies are now propagated forward through the MATSH (changing winning THRU request copies to FIXED at each stage), while non-winning THRU request copies

are erased. At the same time, the new RUN request copies are routed as previously described for P1, except that no new request copy can displace a FIXED request copy, although combining with a FIXED request copy copy is permitted. The full connection pattern of N individual connection requests is considered satisfied after each inlet node has a FIXED (satisfied) request copy.

5.2.2.4 The Enhanced-FLAEM Procedure

The previous subsections described the Basic-FLAEM routing procedure. The computer simulation and results presented later in this chapter use this Basic-FLAEM procedure. However, an Enhanced-FLAEM routing procedure which incorporates the technique of *splitting* may provide increased performance. Splitting is the generation of additional (new) copies of request copies that encounter extra free nodes at specific early stages in the network during all forward routing passes. Splitting allows more than F request copies of each connection request copy to be tried in parallel during each forward routing pass.

The path vector for an RS-EGS network has $p = \log_2 P$ bits devoted to the path number that identifies a specific path between given inlet and outlet nodes. As seen in Section 3.4, the first $f = \log_2 F$ bits of the path number are devoted to the fan-out switch setting. Thus the F request copies generated from each connection request occupy all possible states of these f bits in the path vector. However, there still remain $p-f$ unspecified bits in the path vector after the F request copies are generated. (Note: $p-f > 0$ for all cases of practical interest.) These $p-f$ bits correspond to the switch settings in stages 1 through $p-f$. The Basic-FLAEM procedure takes advantage of these unrestricted switch settings to implement the

modpath procedure previously described in P1. However, additional advantages from these unrestricted switches may be pursued.

Stages 1 through $p-f$ are termed *flexible* stages. The splitting technique is employed solely in these flexible stages. Because of the *modpath* technique (described previously) that is employed in these flexible stages, there can be no conflict between any pair of request copies that meet in a switch in a flexible stage. Thus there are only two ways for a spare link to appear in a flexible stage, as the result of a combine operation, or during a later forward pass through the network when there are many free nodes. The Enhanced-FLAEM splitting technique is designed to take advantage of these otherwise unused links.

Within a switch in stage s (where $1 \leq s \leq p-f$) there are two possible ways to apply the splitting technique to generate additional request copies within the network:

S1: **Simple split** When a single incoming request copy to a switch encounters no other request copies, that incoming request copy may be split to produce two outgoing request copies from the switch. In this case, one randomly chosen outgoing request copy is assigned the original priority T and the other outgoing request copy is assigned the priority $T+F2^s$. This ensures that new request copies generated by this splitting operation will always have lower priorities than any request copies generated by the fan-out switch.

S2: **Combine and split** When the two incoming request copies to the switch are to be combined, the resulting combined request copy is split to produce two outgoing request copies, with one request copy sent out of each of the two outlet links from the switch. Given that the two incoming request copies had

priorities T_1 and T_2, two possibilities for setting the priorities of the outgoing two request copies are:

a: Randomly assign priority T_1 to one outgoing request copy and priority T_2 to the other outgoing request copy. This conserves the two priority values.

b: To allow the comparison described in the following paragraph, randomly assign one outgoing request copy the original highest (minimum numerical value) priority T and assign the other request copy the priority $T+F2^s$. This ensures that new request copies generated by this splitting operation will always have lower priorities than any request copies generated by the fan-out switch.

These splitting procedures require additional bookkeeping to be performed in both the forward and reverse directions when passing through the flexible stages.

The Enhanced-FLAEM technique using splitting techniques S1 and S2b necessarily results in performance that is at least good as and probably noticeably better than that of the Basic-FLAEM technique. The Enhanced-FLAEM technique only introduces additional request copies that have lower priorities (higher numerical values) beyond those provided by the Basic-FLAEM technique. Since request copies with lower priorities can never win in a conflict against request copies with higher priorities, no connection established by the Basic-FLAEM technique can be jeopardized by the Enhanced-FLAEM technique. Furthermore, the additional copies provided by splitting technique provide additional chances for request copies to get through to the desired outlet node, so the Enhanced-FLAEM technique should result in increased algorithm performance. Splitting technique

S2a is not rigorously guaranteed to not affect connections that would have been generated by the Basic-FLAEM method, but it more closely follows the design methodology of the Basic-FLAEM method in that it conserves priority numbers. The Enhanced-FLAEM method using splitting techniques S1 and S2a would thus likely be quite successful as well.

5.2.3 Illustrative Example

This section presents an example which uses a small non-minimal RS-EGS network to illustrate the methods used in the FLAEM routing algorithm. The example RS-EGS network (showing an unfolded MATSH) has $N = 4$ inlets and outlets, $F = 4$ fan-out and fan-in, and $S_S = 3$ stages in the main section, resulting in $p - f = 1$ flexible stage. These parameters were chosen to give the smallest example that would permit most aspects of the FLAEM algorithm to be illustrated.

The FLAEM routing procedure to satisfy a single connection pattern is illustrated in Figures 5.2, 5.3, 5.4, 5.5, and 5.6. The connection pattern being processed is illustrated above the network diagram. The routing paths of all request copies from a given inlet are given a particular grey tone or pattern to identify them. A request copy that was produced by combining two other request copies is arbitrarily given the marking of one of the two inlet nodes that produced the original request copies. The links are individually labelled with indices immediately following the switch stages to show where the link addresses are assigned. The switches in each stage are indexed from 0 at the top to 7 at the bottom of the figure; however, these switch indices are not explicitly shown in the figures. The priority of each individual request copy is listed inside of the circle next to the link node index. This

is the location where all information about the request copy is stored, including the priority, state value, and the path vector. Each link is also marked with the priority of the request copy in a circle immediately preceding (to the left of) each switch stage to elucidate the processing from input to outlet at each switch. These circles are also sometimes used instead to show some of the state values instead of the priorities. Although the figures show trees of 1×2 and 2×1 switches to implement the fan-out and fan-in in the MATSH routing processor, they will still be discussed as $1 \times F$ and $F \times 1$ switches.

The five figures, Figures 5.2, 5.3, 5.4, 5.5, and 5.6, show the specific case of the Basic-FLAEM routing algorithm working to satisfy the connection pattern: (0→2, 1→3, 2→1, 3→3) in two *tries*. A case that took two tries was selected for this example to illustrate most of the facets of the FLAEM routing procedure. (Note: it is quite difficult to find a connection pattern for the given network that takes two tries to complete the processing, since the simulation using the computer program described later in this chapter found that only 17 out of 10,000 (0.17%) randomly chosen patterns for this network took two tries, while the remaining 9988 patterns took only one try. Thus this example has a few special coincidences for arbitrarily chosen routing parameters.) For this example case, the first try consists of the initial forward pass (P1), followed by a reverse pass (P2), with one connection request remaining unsatisfied after this first try, so a second try is performed. The second try consists of a forward pass (P3) followed by a reverse pass (P2). After this second try, all connection requests are satisfied, so a final pass (P3) is made to finalize the connection settings, and the routing operation is complete. Each of these five passes of the Basic-FLAEM procedure and the corresponding figure is

now discussed in detail, followed by an illustration (with a corresponding figure) of how the Enhanced-FLAEM proecedure would work on the same connection pattern.

5.2.3.1 Try #1: Forward Pass (P1)

Figure 5.2 illustrates the first forward pass (P1) of try #1 to process the connection pattern (0→2, 1→3, 2→1, 3→3).

First, examine the four request copies generated by inlet node 0 (marked light grey in the figure). Immediately following the fan-out stage, the four request copies are observed to contain priorities zero through three in the randomly selected order (2, 0, 1, 3). These four request copies then progress through the network, with all four successfully reaching the desired outlet node 2.

In a similar manner, the four request copies generated by inlet node 2 (marked with a spotted pattern in the figure) also progress through the network to successfully reach the desired outlet node 1.

The four request copies generated by inlet node 1 (marked dark grey in the figure) have more involved processing. These four request copies emerge from the fan-out stage with the priorities in the randomly selected order (2, 0, 1, 3). (Note: this connection pattern example has the peculiar feature that this same priority pattern was randomly assigned to sets of request copies produced by three of the inlet nodes.) At switch 5 of stage 1, two request copies from inlet nodes 1 and 3 are combined, emerging at link 11. This combined request copy was randomly chosen to be marked with a path vector indicating its origin at inlet node 1. Thus the incoming request copy from inlet node 1 proceeds on in the normal manner, and the

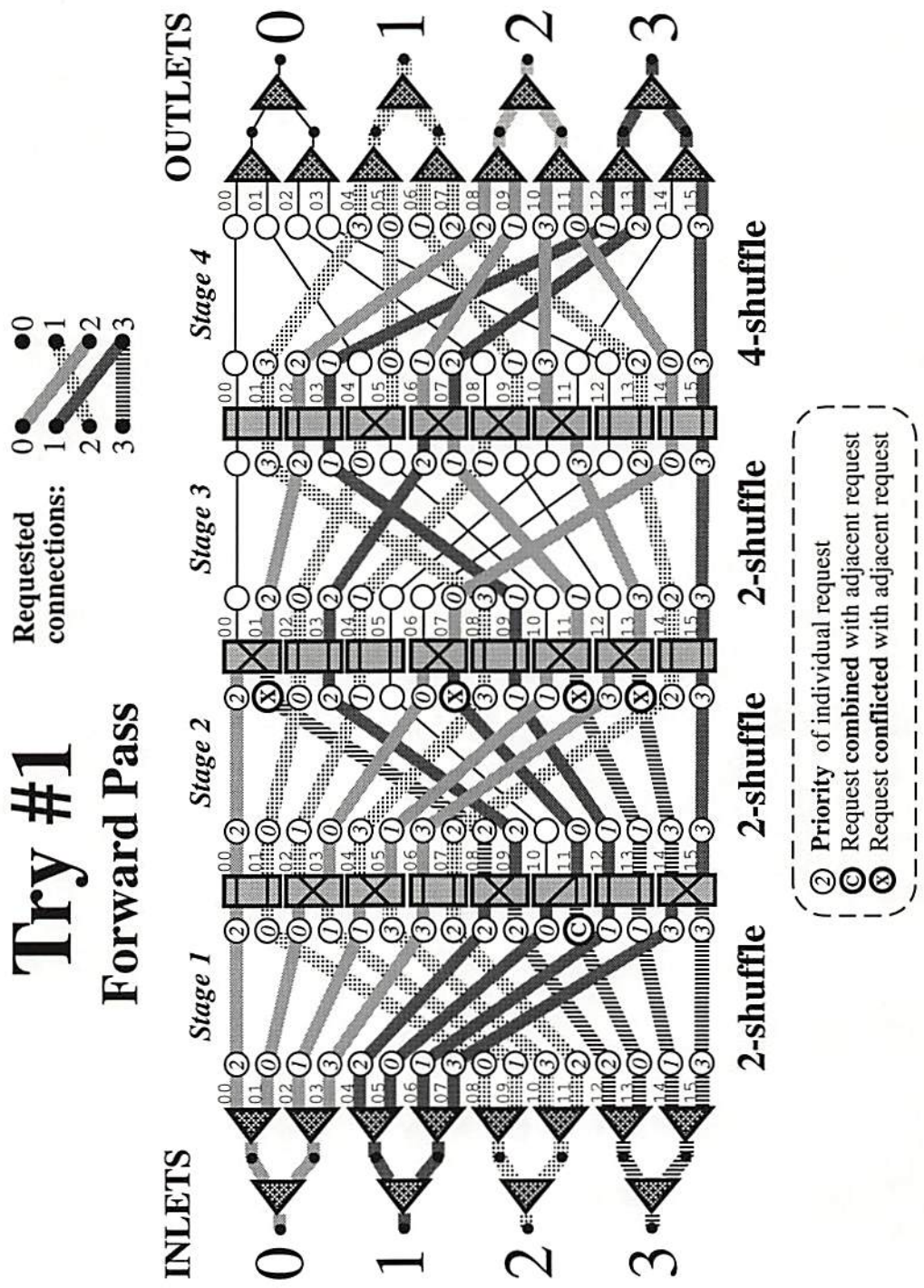


Figure 5.2: The Basic-FLAEM procedure: Try #1: Forward Pass (P1)

incoming request copy from inlet node 3 is marked COMBINED and its processing is aborted for now. Thus all four request copies from inlet node 1 emerge from stage 1. At stage 2, three of these request copies proceed unaffected, but the request copy entering switch 3 encounters a conflict. The two request copies entering switch 3 are a request copy from inlet node 0 at priority 0 and a request copy from inlet node 1 at priority 0. Both incoming request copies need to emerge at link 7, but they are uncombinable since they are each bound for incompatible outlet node destinations. Since they have identical priorities, one is randomly chosen to be the winner, and the other is marked CONFLICT, and aborted. In this case, the request copy from inlet node 0 is chosen to win, and the request copy from inlet node 1 is aborted. Thus three of the original four request copies from inlet node 1 emerge from stage 2. These three request copies encounter no further complications as they progress through the rest of the network, so three request copies successfully reach the desired outlet node 3.

The four request copies generated by inlet node 3 (marked with a striped pattern in the figure) turn out to be unsuccessful. One request copy (the one with priority 0) is combined with a request copy from inlet node 1 (as previously mentioned), but that request copy is ultimately unsuccessful. The three other request copies each make losing encounters with conflicting requests at equal or higher priorities, so each marked CONFLICT and aborted.

5.2.3.2 Try #1: Reverse Pass (P2)

Figure 5.3 illustrates the reverse pass (P2) of try #1 to continue the processing the previously mentioned connection pattern. First the successful connections are

Figure 5.3: The Basic-FLAEM procedure: Try #1: Reverse Pass (P2)

marked THRU (labelled as T in the figure) immediately after the fan-in stage (at the right of the figure). As processing proceeds in the reverse direction through the network, successful request copies (i.e. those that generated a THRU request on the preceeding forward pass) are marked THRU including request copies that were combined to produce a THRU request (although there are no cases of ultimately successful combines in this example). Additionally, aborted CONFLICT requests are erased.

The result of this reverse pass is that four successful THRU-marked request copies are found at inlet nodes 0 and 2, three successful THRU-marked request copies are found at inlet node 1, and no successful request copies are found at inlet node 3. Since inlet node 3 is entirely unsuccessful, a new try at the connection pattern must be made.

5.2.3.3 Try #2: Forward Pass (P3)

Figure 5.4 illustrates try #2 to process the previously stated connection pattern (0→2, 1→3, 2→1, 3→3). This the second forward pass (P3), which will fix in place the successful connections (0→2, 1→3, 2→1) and reroute the unsucessful connection (3→3).

Since inlet nodes 0, 1, and 2 contain THRU request copies from the previous pass, each of these inlet nodes is successful, and an arbitrarily-chosen winning THRU request copy is chosen for each inlet node and marked FIXED immediately following the fan-out stage. The remaining unchosen (non-winning) THRU request copies are erased following the fan-out stage.

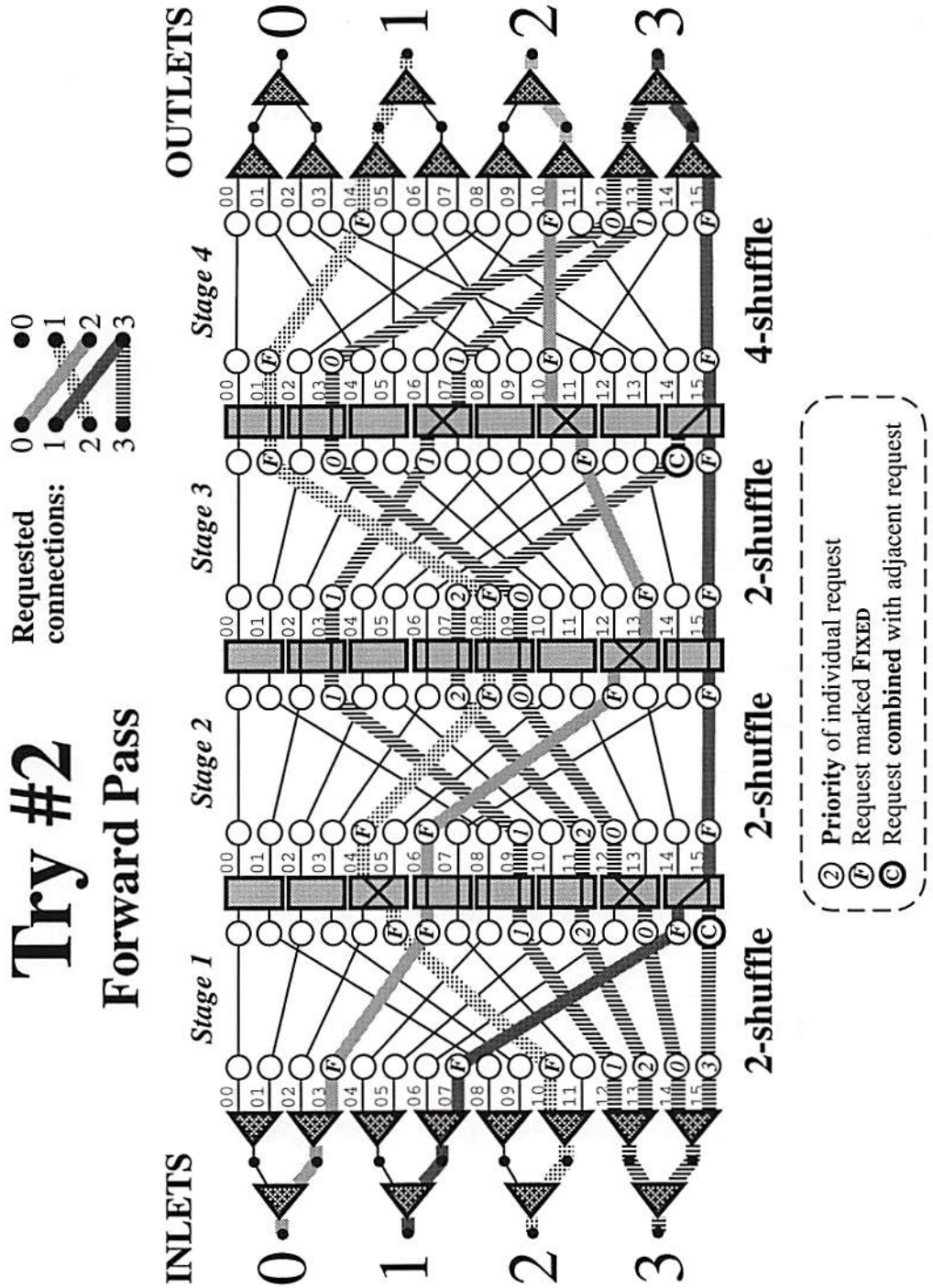


Figure 5.4: The Basic-FLAEM procedure: Try #2: Forward Pass (P3)

Inlet node 3 was entirely unsuccessful during the previous try, so a new set of F request copies is generated (marked with a striped pattern in the figure), each marked RUN, and each with a randomly chosen assignment of the priorities from 0 to 3, which for this case turns out to be (1, 2, 0, 3).

As the forward pass for try #2 progresses, the winning THRU request copies are marked FIXED non-winning THRU request copies are erased (the links are marked FREE), and the newly generated RUN-marked request copies are routed through the network. In this case, two of the RUN-marked request copies (with priorities 2 and 3) are combined with the FIXED connection from inlet node 1 which is destined for the same outlet node 3. The completion of the forward pass for try #2 illustrated in Figure 5.4 reveals that two of the newly-generated request copies from inlet node 3 successfully arrived independently at outlet node 3, and the other two request copies were each earlier assured of success since they combined with FIXED connections.

Figure 5.4 illustrates an important aspect of the FLAEM procedure that reveals why it is so successful. In general, during the first forward pass of the FLAEM procedure, most connection requests are satisfied, leaving only a small number of connection requests to be rerouted. During the second forward pass (as just discussed) the previously satisfied connection requests are set to FIXED within the network, and thus each only occupy a single path through the network. This frees up a large number of paths in the network, which results in the greater amount of free space available to the connection requests being rerouted. This free space in the network (which is really only hinted at in this small example) is responsible for

the very high success rate for the second try (as will be shown by the simulation results later).

5.2.3.4 Try #2: Reverse Pass (P2)

Figure 5.5 illustrates the reverse pass (P2) of try #2 to continue the processing the previously mentioned connection pattern. The successful connections at outlet node 3 are marked THRU (labelled as T in the figure) immediately after the fan-in stage (at the right of the figure). These successful THRU request copies are propagated backwards, as before. However, each FIXED request must also be propagated backward so that although the FIXED request itself will remain unaffected, any RUN requests that combined with a FIXED request must be found and marked THRU as well. Thus all four request copies that were generated at inlet node 3 end up with a THRU mark. Since inlet node 3 is entirely successful, no new try at the connection pattern needs be made. Therefore the next forward pass will be the final P3 pass.

5.2.3.5 Final Forward Pass (P3)

Figure 5.6 illustrates the final forward pass (P3) to finalize the newly satisfied connection (3→3) to complete the desired connection pattern (0→2, 1→3, 2→1, 3→3). The bottom request copy from the fan-out from inlet node 3 was chosen to win, so it was marked FIXED, and the other three request copies were erased during this final forward pass. This final connection pattern that would be downloaded into the FIER optical network includes: all fan-out settings set to *down* (a coincidence), four switches set to *bypass*, three switches set to *exchange*, two switches set to *combine*, (the final fan-in switches are permanently set to *combine*).

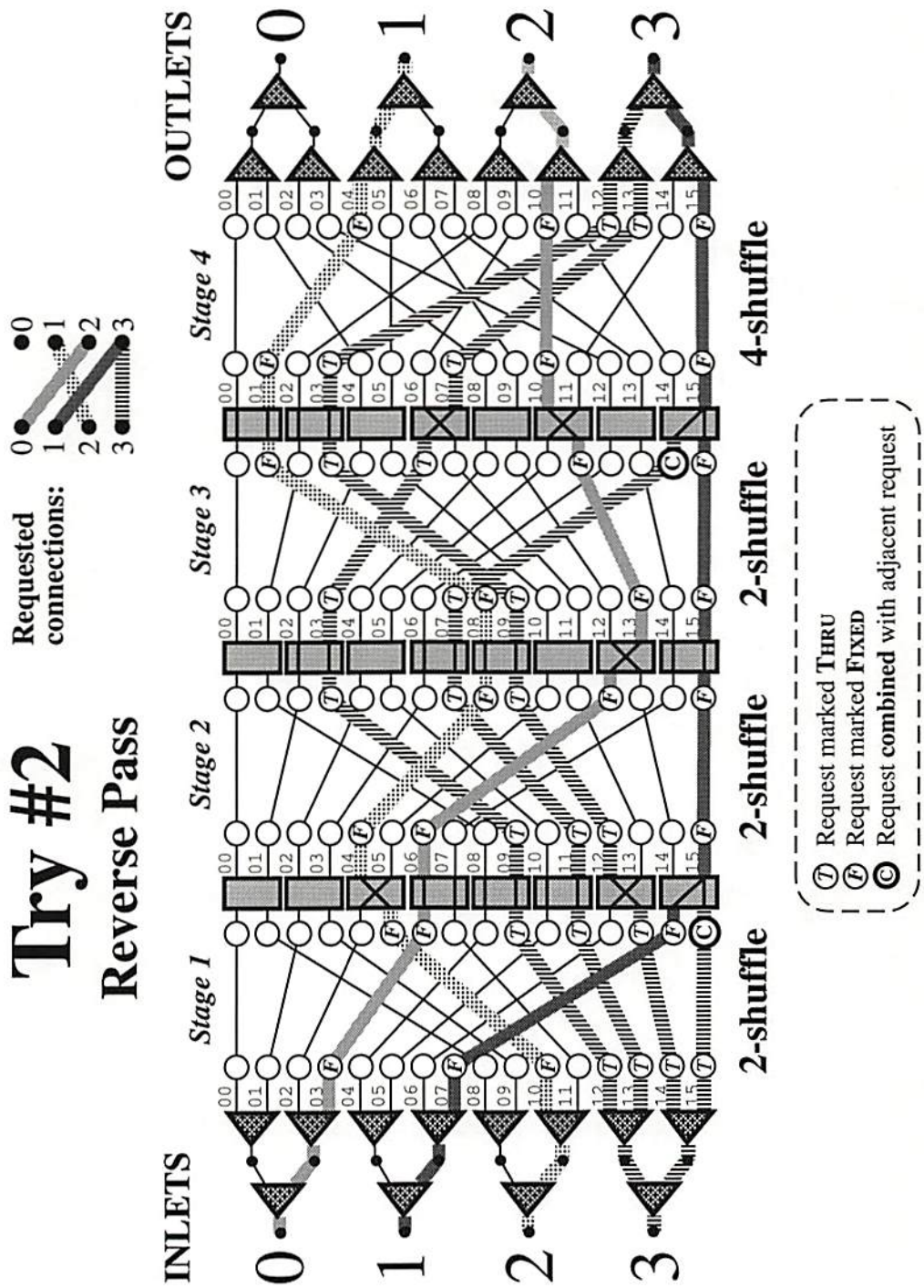


Figure 5.5: The Basic-FLAEM procedure: Try #2: Reverse Pass (P2)

Figure 5.6: The Basic-FLAEM procedure: Final Forward Pass (P3)

5.2.3.6 Enhanced-FLAEM procedure illustrations

Figure 5.7 illustrates the combine-and-split technique S2b, showing the first forward pass of try #1 for the Enhanced-FLAEM procedure. This figure is the same as that of Figure 5.2 which showed the first forward pass of try #1 for the Basic-FLAEM procedure, except that an additional specially-marked request copy is added for Figure 5.7. Stage 1 of this network is a *flexible* stage since for this network $p-f=1$ (since $P=8$ and $F=4$). At switch 5 of stage 1, instead of just sending the combined request copy out at link 11, an additional combined request copy is generated and sent out at link 10 (marked with thick black stripes in the figure). Splitting technique S2b is illustrated; the newly assigned priority is 4. If splitting technique S2a were employed, the newly assigned priority would be 0, resulting in the same set of priorities entering and leaving the switch. For the case of this connection pattern, the additional request copy generated by the combine-and-split operation is ultimately successful, and would result in this connection pattern being satisfied in one try by the Enhanced-FLAEM algorithm.

Figure 5.8 illustrates the simple-split technique S1, showing the second forward pass (try #2) for the Enhanced-FLAEM procedure, (which is shown as if connection request 3→3 was still unsatisfied after the first pass). This figure is the same as that of Figure 5.4 which showed the second forward pass (try #2) for the Basic-FLAEM procedure, except that additional specially-marked request copies are added for Figure 5.8. At the flexible switch stage 1, the request copies being rerouted are split to generate request copies at both outgoing links from the switches. However, the request copy entering switch 7 is not split, since it is combined with a FIXED request and thus is considered automatically successful.

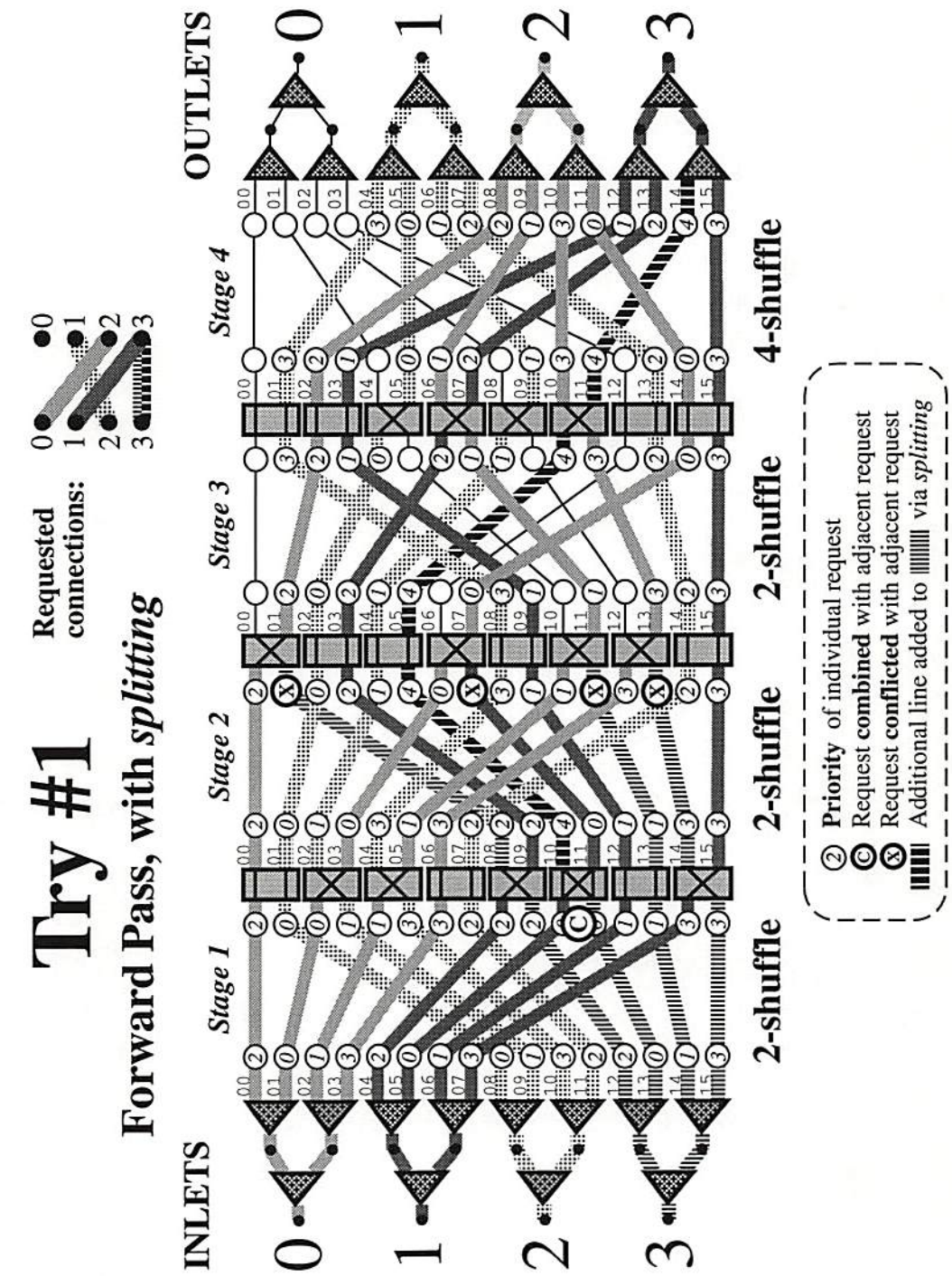


Figure 5.7: The Enhanced-FLAEM procedure: Combine-and-Split (S2)

Figure 5.8: The Enhanced-FLAEM procedure: Simple Split (S1)

Splitting this request copy would only occupy unnecessary space in the network. The three new request copies generated in the flexible stage 1 emerge at links 8, 10, and 13, with priorities set equal to the original priority plus 4 ($4 = F \cdot 2^s$, since $s = 1$). Although the extra request copies generated by this simple-split operation do not reduce the number of tries necessary to complete the routing process in this example, an additonal request copy does make it through to the outlet node 3, showing that this technique can increase the odds of success in general.

5.3 The FLAEM Simulation

The computer simulation of the Basic-FLAEM procedure (sim-FLAEM) is a C program which takes as input the EGS network parameters N, F, and S_S, and generates various types of data about the routing process for evaluation and analysis. One type of output from the sim-FLAEM program is a stage-by-stage trace of the routing process for a randomly-generated or user-supplied connection pattern. Such traces were used to verify program correctness and to generate the examples presented herein. The other type of output from the sim-FLAEM program is a summary report listing the performance of the algorithm on large numbers of randomly-generated connection patterns or permutations. This section provides an overview of the sim-FLAEM C program itself, then presents the simulation results obtained from this program.

5.3.1 Sim-FLAEM Program Development

The sim-FLAEM program was developed from a series of simpler routing simulation programs that implemented less sophisticated routing algorithms. Each

program in the development series represented an improvement in the routing performance. The FLAEM procedure was developed from this series of programs by refining the routing process until the algorithm was effective enough to be practical.

5.3.1.1 The First Routing Program

The first routing program was a simple router that would randomly generate a set of path vectors for a given connection pattern, and proceed forward through the network until a conflict occurred. Only local routing data were used for this algorithm. At this point, the processing was aborted and another try at routing was started, with an entirely new set of randomly-generated path numbers. This first program was not very successful; it was only able to successfully route connection patterns for extremely small networks, usually taking large numbers of tries for each success.

5.3.1.2 The Second Routing Program

A modification was added to the first program to allow it to complete its forward pass, remember which individual connection requests conflicted, and then on subsequent tries only these formerly conflicted requests received new randomly-generated path numbers while the successful connection requests were left alone. A global array of variables, one for each inlet node, was used to communicate information about conflicts back to the inlet nodes, with special bookkeeping to communicate information about requests that had combined with other requests that ultimately ended at a conflict. This second program was a definite improvement over the first, although it could still take a large number of tries to complete a single connection pattern.

5.3.1.3 The Third Routing Program

The third program had the *modpath* modification that allowed path vectors to be altered to avoid conflict when the conflict would have occurred in a flexible stage. This produced a subtle improvement in the routing results, because very small networks with only a single flexible stage were used to test the program.

5.3.1.4 The Fourth Routing Program: Basic-FLAEM

The fourth program implements the idea of trying F copies of each connection request in parallel. This program also incorporated modifications that allowed it to more closely simulate the operation of the MATSH control unit by operating solely on the local data that would be available in the virtual switches of the MATSH; no more global communication was permitted. This program was the first to incorporate an explicit reverse pass to communicate information about successful and conflicted requests backward to the inlet nodes. The backward pass was essential to return the information about successful and conflicted request copies along the complex combining treelike paths that were created within the network. The simplest way to do the complicated bookkeeping required turned out to have the advantage that is most closely approximated the intended actual routing procedure for the MATSH control unit.

This fourth program produced an enormous improvement in performance, due to the many paths tried in parallel, and the relative emptiness of the network during the second try at a given pattern. This fourth program thus became the basis for the Basic-FLAEM algorithm, and is the program that was later called the sim-FLAEM program.

Detailed information about the sim-FLAEM program is provided in Appendix B. The appendix includes the sim-FLAEM C program listing and sample trace data output for the small network example illustrated earlier in Section 5.2.3.

5.3.1.5 The Fifth Routing Program: Enhanced-FLAEM

The basis for an envisioned fifth routing program was conceived during the implementation of the fourth routing program. However, the fourth program (the sim-FLAEM program implementing the Basic-FLAEM routing algorithm) was so successful that the idea for a fifth program was postponed for potential future implementation. This fifth routing program is the basis for the Enhanced-FLAEM algorithm described previously.

5.3.2 Simulation Results

The performance of the Basic-FLAEM routing algorithm is demonstrated by using the sim-FLAEM program to perform the routing of large numbers of randomly-generated connection patterns. Table 5.1 presents the results from using the sim-FLAEM program to route connection patterns on various sizes of restricted-F RS-EGS networks. The first group of five columns lists the RS-EGS parameters for each network, including the number of flexible stages $(p-f)$ available in each network configuration. Each set of network parameters used employs the minimal number of stages for a restricted-F RS-EGS network for each value of n, as derived from Richards' formulae which are listed in Table 3.1. The next column lists the total number of randomly-generated patterns that were routed, listing both unrestricted connection patterns ("Unres.") and permutation connection patterns

RS-EGS Parameters					Total Number of	% of Routed Patterns** *Number of Tries:*				Average Tries per
n	N	F	S_S	$p-f$	Patterns*	*1*	*2*	*3*	*4+*	Pattern
4	16	4	5	1	10000 Unres.	74.03	25.67	0.27	0.03	1.2630
5	32	8	5	0	10000 Unres.	84.38	15.61	0.01	0	1.1563
6	64	8	7	1	10000 Unres.	65.44	34.56	0	0	1.3456
7	128	8	10	3	10000 Unres.	33.93	66.01	0	0	1.6595
8	256	8	13	5	10000 Unres.	2.85	96.86	0.29	0	1.9744
9	512	16	12	3	10000 Unres.	73.53	26.47	0	0	1.2647
10	1024	16	14	4	1000 Unres.	34.9	65.1	0	0	1.651
11	2048	16	16	5	1000 Unres.	3.6	96.4	0	0	1.964
12	4096	16	19	7	1000 Unres.	0	100	0	0	2.000
13	8192	16	21	8	1000 Unres.	0	100	0	0	2.000
4	16	4	5	1	10000 Perm.	72.43	27.48	0.09	0	1.2766
5	32	8	5	0	10000 Perm.	85.56	14.43	0.01	0	1.1443
6	64	8	7	1	10000 Perm.	69.65	30.35	0	0	1.3035
7	128	8	10	3	10000 Perm.	39.15	60.85	0	0	1.6085
8	256	8	13	5	10000 Perm.	3.75	96.21	0.04	0	1.9621
9	512	16	12	3	10000 Perm.	79.43	20.57	0	0	1.2057
10	1024	16	14	4	1000 Perm.	46.7	53.3	0	0	1.533
11	2048	16	16	5	1000 Perm.	6.6	93.4	0	0	1.934
12	4096	16	19	7	1000 Perm.	0	100	0	0	2.000

* Unres. $\Rightarrow$ Unrestricted connection patterns (many-to-one allowed);
Perm. $\Rightarrow$ Permutation connection patterns (restricted to one-to-one only).

** Each of the four columns lists the number of patterns (expressed as a percentage of the total number of patterns) that took a particular number of tries (1, 2, 3, or ≥ 4).

N, network size (number of PEs and number of MMs); $n = \log_2 N$; S_S, number of shuffle–exchange stages in the "main section" of the RS-EGS network; F, inlet fan-out and outlet fan-in factor; $p-f$, number of flexible stages.

Table 5.1: FLAEM simulation results.

("Perm.") . The number of routed patterns is decreased for the larger values of N, since the simulation time grew too large to simulate 10,000 patterns for these cases. The same set of RS-EGS parameters was used for both the unrestricted connection pattern case and the permutation pattern case, however the $n=13$ ($N=8192$) case simulated only for the unrestricted connection pattern case because the simulation was run using borrowed time on a computer that had sufficient memory to run this large case and there was only enough time to run one of the two cases. The next group of four columns lists the number of patterns (expressed as a percentage of the total number of patterns) that took a particular number of tries (1, 2, 3, or ≥ 4). The final column lists the average number of tries per pattern, calculated by dividing the sum of the tries required for all patterns by the total number of patterns. For example, for the $n=6$ unrestricted patterns case the Average number of tries per pattern is computed from $(1 \cdot 6544 + 2 \cdot 3456)/10000 = 1.3456$. Table 5.1 is divided into two parts, first listing simulation runs using unrestricted connection patterns followed by simulation runs using permutation connection patterns. These two parts are further subdivided into groups containing identical fan-out values F. These subdivisions serve to highlight the patterns in the data.

5.3.2.1 Analysis of Simulation Data

In RS-EGS networks, the cases of networks with small n are special cases that are often not representative of the behavior of networks with larger n. The smallest cases, $n=2$ and $n=3$ ($N=4$ and $N=8$) have minimal networks that contain only one main section switching stage, and are therefore not included in the simulation. The cases for the minimal restricted-F RS-EGS network for $n=4$ ($N=16$) listed

in Table 5.1 is also a special case. It has one flexible stage ($p-f=1$), but with only a fan-out of $F=4$, there is not much "space" within the network. Thus this case is observed to result in the largest number of tries necessary for single patterns to be completed, with a small number of patterns requiring four or more tries for completion. The case of $n=4$ is included here to show that the operation for small networks may not be indicative of the operation for larger networks

For the cases of restricted-F RS-EGS networks with $n \geq 5$ listed in Table 5.1, the data show excellent behavior. For the range of parameters listed, the maximum number of tries required for any connection is nearly 2. Cases that require 3 tries are very rare, and do not significantly contribute to the resulting average number of tries per pattern. It is this maximum number of tries that is important to the FLAEM routing procedure, since it is most practical to allow a fixed number of passes through the network to accomplish a routing task.

The percentage of routed unrestricted connection patterns that took each number of tries is graphed in Figure 5.9. Each bar in the graph represents 100% of the patterns for the listed value of N, subdivided into the percentage that took 1 through 3 tries. Since the number of patterns that took 3 tries to route is so small, and is 0 for most values of N, only a small thickening may be noted at the top of the $n=8$ ($N=258$) data bar to represent the 0.29% of patterns (29 out of the 10,000 unrestricted connection patterns simulated) for this case that took 3 tries. It is interesting to note that for the $n=12$ and $n=13$ ($N=4096$ and $N=8192$) cases that all of the routed patterns took 2 tries to complete. However, in the small number of test runs not included in the table or graph, a few (2) patterns out of approximately 400 additonal patterns simulated for the $n=13$ ($N=8192$) case

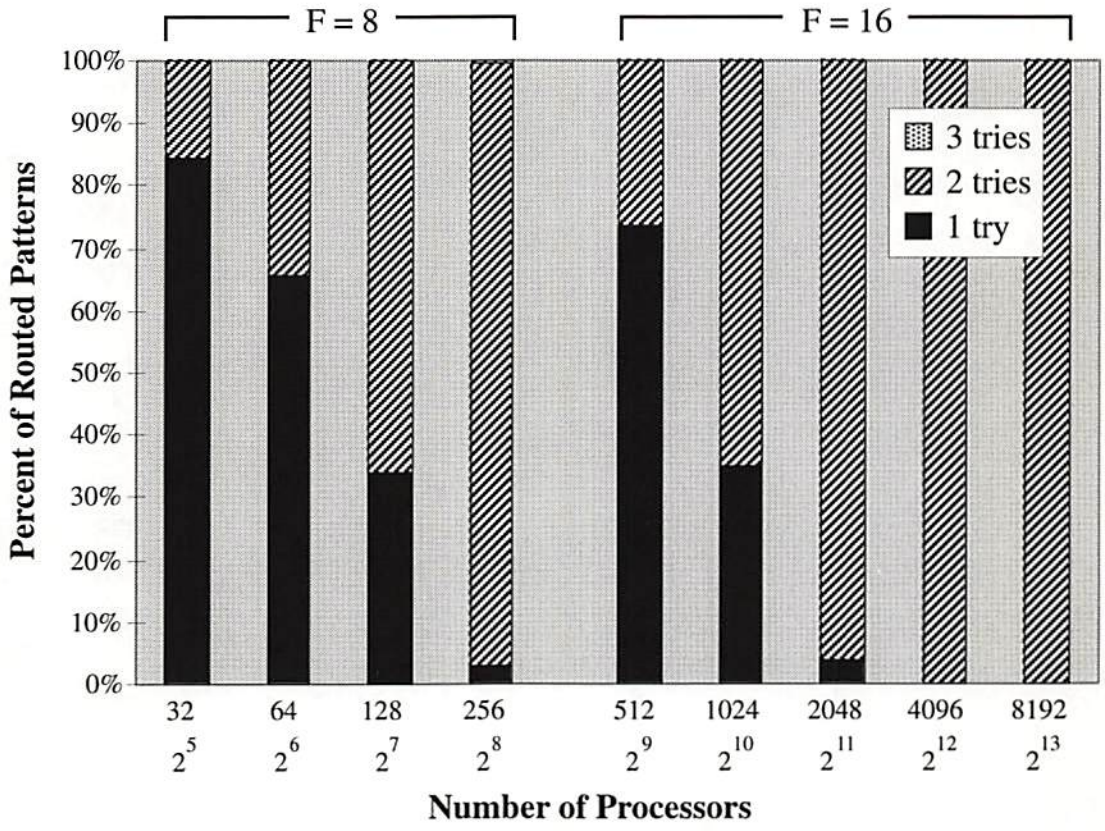


Figure 5.9: Percentage of routed unrestricted connection patterns that took specified numbers of tries.

took 3 tries. However no patterns taking 3 tries were observed in any test runs for the $n = 12$ ($N = 4096$) case. Unfortunately, patterns for $n \geq 14$ ($N \geq 16,384$) were prohibitively large to simulate (primarily in terms of memory requirements), so the table ends at $n = 13$ ($N = 8192$). It appears that the maximum number of tries for $n \geq 13$ may be trending slowly towards 3, but these data suggest that it is likely to remain bounded at 3 for several following higher values of n.

The data for the unrestricted patterns used in the SMOEC system leave open the question that a large factor in the success of the FLAEM routing algorithm might the simulation of large numbers of patterns that required combining. That is, the unrestricted patterns that were randomly generated (by independently randomly selecting an outlet node from 0 to $N - 1$ for each of the N inlet nodes) may favor significant amounts of combining and the FLAEM routing algorithm may require this combining for efficient operation. Thus a series of simulations were performed that routed large numbers of randomly-generated permutation (one-to-one) patterns. These data are listed in the lower half of Table 5.1. As previously mentioned, insufficient computer access was available to run the $n = 13$ ($N = 8192$) case for permutation patterns. Figure 5.10 is a graph of these results that shows the data from the unrestricted pattern case and the permutation case as a pair for each value of n for comparison purposes. The graph shows that for all values of n except $n = 12$ ($N = 4096$) the percentage of patterns that took 1 try for the permutation patterns case increased slightly from the corresponding unrestricted pattern case. Thus it is not true that the FLAEM routing algorithm requires significant amounts of combining for efficient operation.

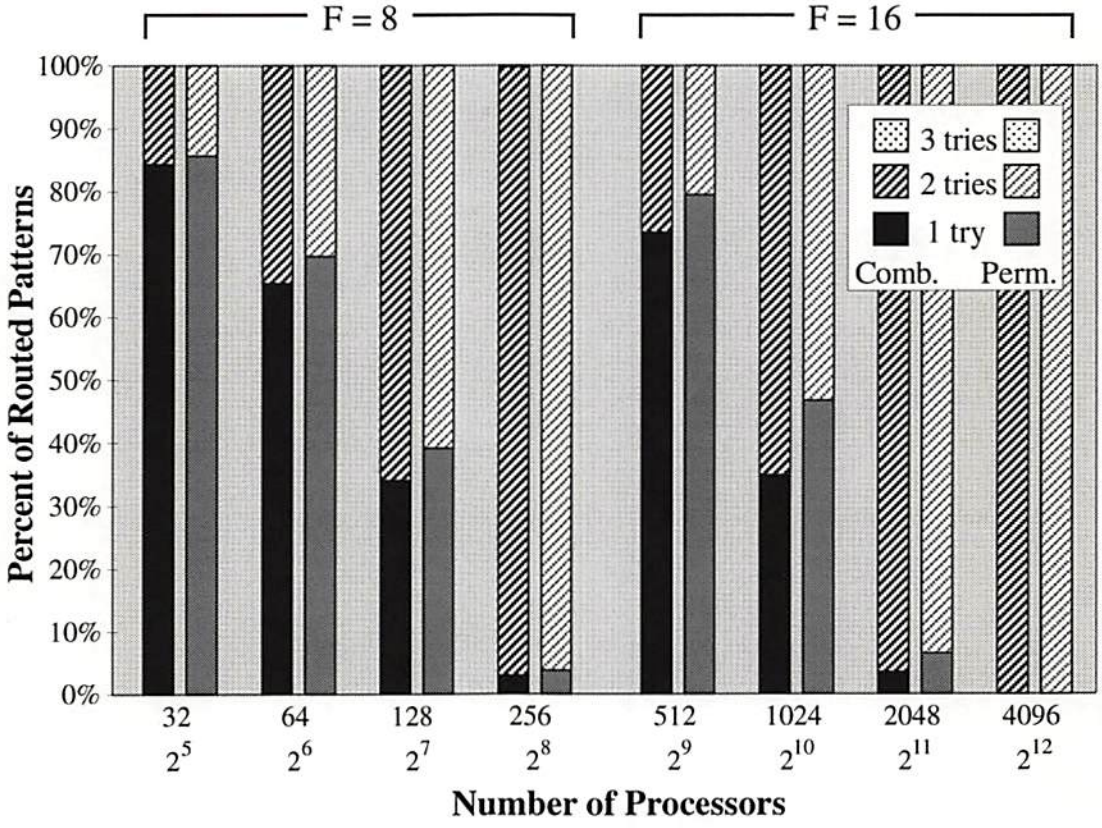


Figure 5.10: Percentage of routed connection patterns that took specified numbers of tries: comparison.

From examination of Table 5.1, it is clear that for a given value of F, the average number of tries per pattern increases as n increases. The average number of tries per pattern drops when F is increased. The average number of tries per pattern is graphed in Figure 5.11, which explicitly reveals this trend in the data. Both sets of data are graphed; the values for both the unrestricted pattern case and the permutation pattern case are plotted. The data for both cases show the same trends, and are quite close in values, however, as in the case of directly comparing numbers of tries, the average number of tries per pattern for the permutation pattern case are always slightly below the values for the unrestricted pattern case. The data with $F = 8$ and $F = eq16$ are grouped separately to show the trends within the data at constant F. The data for $F = 8$ trend upwards towards 2.0, until the minimal required F is changed. The data for $F = 16$ trend upwards, and appear to asymptotically reach the 2.0 level. Data from Table 3.6 reveal that $F = 16$ remains constant for $9 \leq n \leq 16$ (i.e. $N \in \{128, 256, 512, \ldots, 65536\}$). Thus F remains constant for several values of n after the largest $n = 13$ shown in the graph. Therefore there will be no discontinuity similar to that shown between $F = 8$ and $F = 16$ in the graph until $n = 17$ ($N = 131{,}072$). It appears that the average number of tries per pattern is either trending upward extremely slowly or not at all for the largest values of n plotted at the right hand side of the graph in Figure 5.11.

5.3.2.2 Conclusions From Simulation Data

The main result from the FLAEM simulation is that the maximum number of tries necessary for a complete routing appears empirically to be a small near-constant

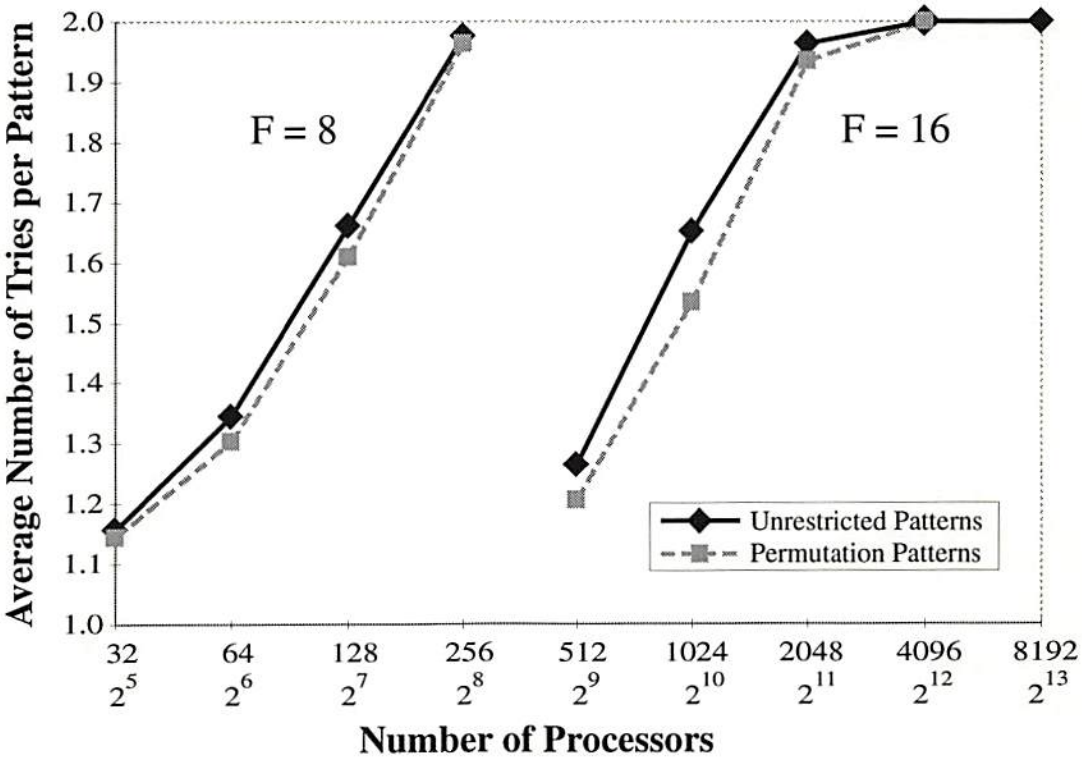


Figure 5.11: Average number of tries per pattern, both cases shown.

value (since results for $n \gtrsim 20$ ($N \gtrsim 10^6$) are unlikely to be of practical interest). For the data simulated $5 \leq n \leq 13$, the maximum number of tries is (very nearly) 2. Results do not substantially differ for routing permutation connection patterns and unrestricted connection patterns.

The Basic-FLAEM technique works as well as it does because it satisfies most of its N routing requests on the first try, leaving only a tiny number of requests for rerouting. The second try is even more successful (very few failures of the second try are noted in the data) since there are so many free paths in the network available to the small number of copies of the rerouted (initially unsuccessful) requests. During a second try all successful requests occupy but one path through the network instead of the F new paths attempted by unsuccessful request copies being rerouted. Since each try takes (slightly more than) 2 passes through the MATSH control unit (plus an extra final pass), and a MATSH pass takes $\mathcal{O}[\log N]$ time, the apparent small constant bound to the routing tries indicates the FLAEM technique completes a full pattern routing in approximately $\mathcal{O}[\log N]$ time.

5.4 Summary

Essential routing and control aspects of interconnection network design are often not explored in sufficient detail to ensure that these aspects will not unduly limit system performance. Control issues for the FIER optical network are presented here so that they can be addressed during the design process rather than postponing them and leaving the system liable to future control difficulties. The FLAEM routing algorithm was designed to control a regular simplified combining (or permutation-restricted) EGS network in a circuit-switched manner in $\mathcal{O}[\log N]$

time. The sim-FLAEM computer program provides an accurate simulation of the Basic-FLAEM algorithm, and results from the simulation indicate that the algorithm can indeed route connection patterns in approximately $\mathcal{O}[\log N]$ time. The FLAEM routing algorithm is a useful new method for RS-EGS networks, which are particularly suited for passive optical implementation. The FLAEM routing algorithm is also potentially applicable to other appropriately tailored circuit-switched multistage interconnection networks.

Chapter 6

COMMUNICATION ALGORITHMS

The PE↔MM communication algorithms are built upon the routing algorithm. In the process of computing the routing bits, the additional functionality (extended bypass/exchange switch states, local stack memory) of the MATSH is used to facilitate multiple reads and to arbitrate and combine multiple writes.

Communication within the SMOEC is divided into separate "phases" to ensure that multiple communication requests may be handled simultaneously. Each phase consists of communication requests of the same type. Phases used in the SMOEC include: read phase, write phase, fetch-and-add phase, and data sort phase. Such separate phases greatly simplify the complexity of the request combining process within each phase.

6.1 The Read Algorithm

A read phase consists solely of read requests from each PE to its requested MM, and the subsequent transfer of data from each requested MM to its requesting PE(s). Multiple PEs may request information from the same MM without conflict. Each requesting PE forms a read request consisting of the MM index (address) it desires to read from. No (sub)address within the requested MM is required, since the entire MM contents will be transferred during the read phase. These MM addresses are fed directly into the MATSH, with each address going to the corresponding node in the MATSH (the MATSH node with the same index as the requesting PE). Using the routing algorithm, the addresses are repeatedly cycled through the single-stage Electronic Shuffle–Exchange (E-SE), enabling sequential computation of the appropriate switch settings for each Optical Shuffle–Exchange (O-SE) in the FIER. Since there are S_F O-SEs, S_F cycles of the E-SE will provide all the required control bits. These bits are buffered in the CSDI (Control Signal Distribution and Interface), where they are held until all bits are ready to be sent simultaneously to their destinations in the FIER switches.

Read requests from multiple PEs to the same MM are combined (routed together) within the MATSH using the upper and lower combine switch settings. The MATSH sends these switch settings to the FIER, where they will act as broadcast switch settings in the MM→PE direction. Each requested MM then generates an optical signal which is serially modulated by its entire memory content, and subsequently routed back through the FIER to the desired requesting PE(s). Data distribution (fan-out) to multiple requesting PEs is implemented using the broadcast switch settings in the FIER.

6.2 The Write Algorithm

The write technique designed for the SMOEC must be capable of simultaneous writes from multiple PEs to a single MM. This section provides an examination of four write technique candidates, a discussion of the serialization principle (which is essential to the technique used in the SMOEC), and a detailed presentation of the combining technique designed for the SMOEC.

6.2.1 Four Write Technique Candidates

The case of multiple PEs attempting to write simultaneously to a given MM is a difficult problem to solve. There are several techniques available, none of which is the most desirable in all cases. Four candidate techniques are presented here, and their suitability for use in the SMOEC is addressed.

6.2.1.1 W1: Permutations Only

Allow only groups of write requests that are known a priori to be permutations. This technique is extremely restrictive, but could work for algorithms that have deterministic, regular data flow.

6.2.1.2 W2: Arbitrate Write Requests

If a switch in the interconnection network receives conflicting requests, one request will be postponed. A sample method is: Address requests and priority levels are sent from each PE to the MATSH. Switches resolve conflicts by eliminating the lower priority request, and not routing it any further. After S_F stages, the MATSH sends a 1 to the MMs whose addresses weren't eliminated. The optical switches are

then updated, and the MMs that received a 1 send a 1 back through the network to notify the successful requesting PEs. Now the write data is sent through the optical network. Thus some write requests may be forced to wait and try again during the next write phase, since buffering *within* the FIER has been forbidden by design (passive optical switching). Multiple requests to the same MM are thus handled in a serial fashion, in separate write phases. The buffering of write requests before entry to the network allows an analog of "hot spots" (as discussed in Section 1.2) to form, creating bottlenecks that can ultimately completely clog the FIER. *This is not a practical technique for the* SMOEC, *so it is not used.*

6.2.1.3 W3: Combine Write Requests in FIER

Using the serialization principle (described below), write requests may be combined in a useful and consistant manner. The technique involves pairwise combining and/or arbitrating of write requests in the MATSH, followed by the proper setting of the FIER. Write request combining is the primary method used in the SMOEC, and will be discussed in detail later in this section. Note: this ability to combine write requests was a major factor in the choice of a passive optical implementation for the interconnection network in the SMOEC.

6.2.1.4 W4: Combine Write Requests in MATSH

As a special case, for very short write requests the MATSH can be used as a packet-switched network. For message types that may always be combined using the serialization principle (desribed below) combining in the MATSH may be preferable. "Short" requests are those that containing small amounts of data that can be passed through the MATSH without unacceptably slowing down the total

electronic control bit computation time (T_{MATSH}). Examples of combinable messages are those such as setting, incrementing or adding an amount to a stored value, or **OR**ing data values. This method is also preferable in the SMOEC for very short read/write requests (used for fetch-and-add type requests, which will be described in 6.3).

6.2.1.5 Write Algorithm Technique Conclusions

In summary, Techniques W1 and W2 are *not* used in the SMOEC (although Technique W1 may be used for a special restricted version of the SMOEC discussed in Chapter 7.2.1). Techniques W3 and W4, which employ write request combining and are more computationally powerful, are the techniques of choice for the SMOEC.

6.2.2 The Serialization Principle

The serialization principle [Gottlieb 83a, Gottlieb 83b] is a basic idea that allows data to be routed very efficiently. Simply put, it is the requirement that if two requests are being sent to a memory location simultaneously, the result must be the same as if the two requests had ocurred in some unspecified order. The result of two reads to the same memory location is straightforward. The result of two "simultaneous" writes (that do not involve any reads) is as if one is just ignored. One of the two write requests need never be sent. More complicated simultaneous memory requests invovling inseparable read/write combinations such as fetch-and-add (which fetches a datum from a memory location, then adds a value to the

datum and stores the new value in the same location, see 6.3) can be handled using this principle as well.

Although the serialization principle appears to discard information in the case of simultaneous writes, the operation makes sense when placed in the context of the ideal shared memory model. In the ideal model, asynchronous communications with the shared memory occur in unit time. So two *nearly* simultaneous write requests to the same location behave as if the first write request was simply discarded, if there were no intervening read requests. This behavior is what is expected of the ideal model; the serialization principle takes this behavior to its logical conclusion. It remains the job of the software to ensure that simultaneous write requests make sense. However, applying the serialization principle to the case of simultaneous fetch-and-add requests provides a very useful method of synchronizing disparate processes (applications of fetch-and-add are described in 6.3).

6.2.3 Write Request Combining

Write request combining is an extremely useful technique for satisfying an entire array of simultaneous write requests. Each write request sent to the MATSH consists of a MM address and a subaddress field. No actual data is sent to the MATSH — it will be sent through the FIER after the MATSH has combined the requests. Each MM is functionally divided into d "submodules" (where $d \leq D$). The subaddress field contains d bits which serve to mark specified submodules within a MM. The bit (or bits) in the subaddress field that correspond to the desired write locations within the MM are set to 1, while the remaining bits in the

field are set to zero. Note that the d can be a variable quantity set by software, and may even be changed during the course of an algorithm.

Figure 6.1 illustrates the subaddress field idea for the example of a MM that has been divided into 8 submodules. For the purposes of this example, assume that two write requests, $\mathcal{A}$ and $\mathcal{B}$, meet in a switch node, and that both requests are destined for the same MM. Each request has its own 8 bit subaddress field, denoted A and B respectively. The idea is to combine the two requests using the serialization principle such that both write requests are satisfied. Clearly the resulting write request, $\mathcal{A}'$ must contain the address of the same MM as the incoming requests, but it must also contain a subaddress field A' that properly reflects all requested submodules. The two incoming subaddress fields A and B are saved in local registers (actually implemented as a stack of depth $\mathcal{O}[S]$) in the same manner as described in 6.3 for later use in the final part of the write request combining procedure. The subaddress field A' that is sent out from the node is calculated by $A' = A \,\mathbf{OR}\, B$. In this manner every subaddress within the MM that was requested by any PE will ultimately receive a 1 in the subaddress field of the combined write request that emerges from the forward trip (S_F cycles) through the MATSH.

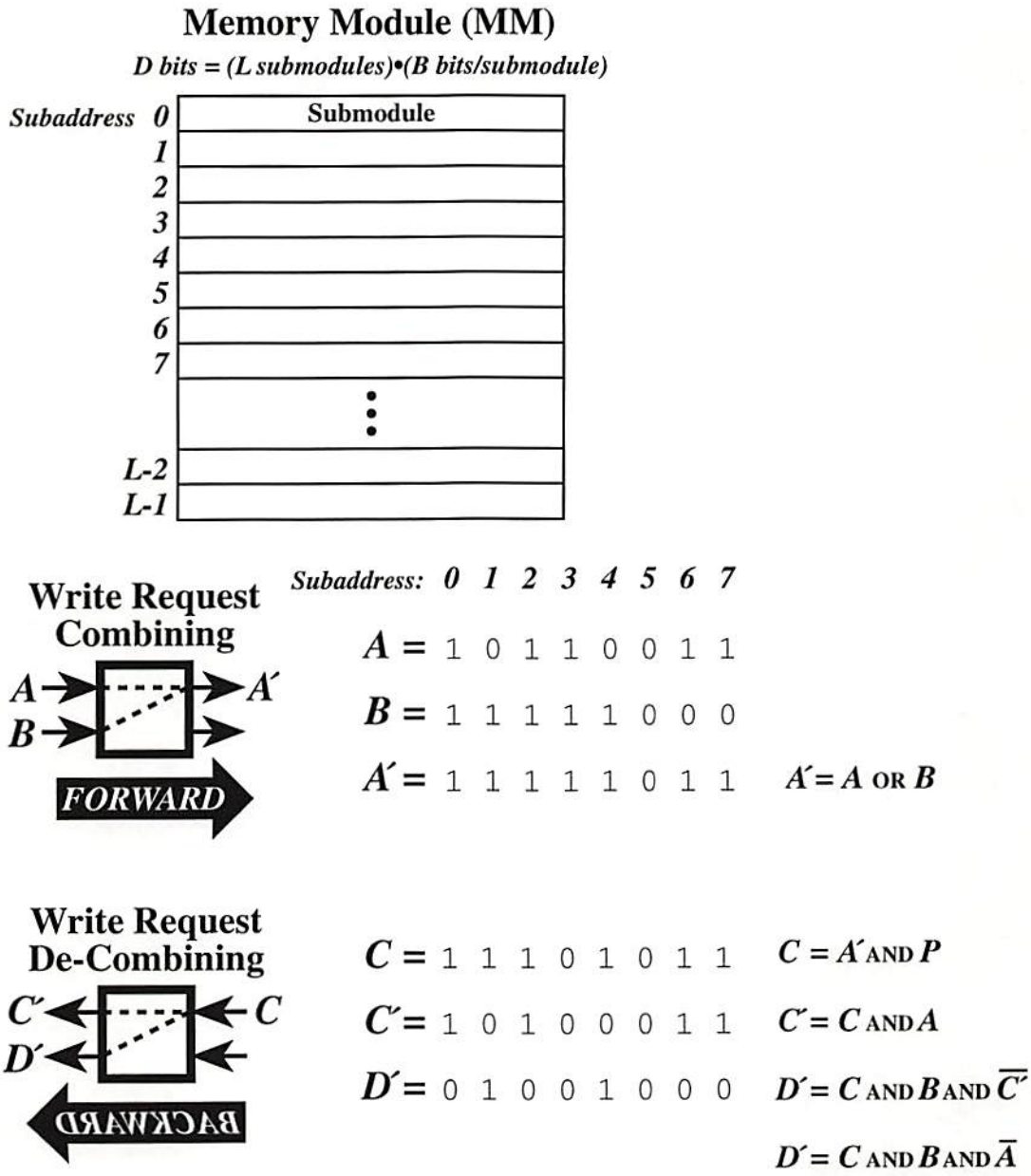


Figure 6.1: The Subaddress Field. First 8 bits illustrated.

After S_F cycles of subaddress field combining in the MATSH, the final combined write request contains a subaddress field consisting of the combined subaddress fields of all the appropriate requesting PEs (those PEs not eliminated using the serialization principle). This final combined subaddress field now identifies which locations within the MM are to receive data. This combined request then makes a return trip through the MATSH to perform write request "de-combining". The de-combining is necessary to inform each of the requesting PEs which data it is to send.

The de-combining of requests is accomplished as shown in the second part of Figure 6.1. Assume that the returning write request subaddress field C is to be de-combined into two write request subaddress fields C' and D'. Assume for discussion purposes that this de-combining operation is ocurring at the same node that was previously discussed (in which A and B were combined to produce A'). The goal is to divide the "1" bits in C between C' and D' such that C' **AND** $D' = 0$. This will ensure that none of the requesting PEs transmit data simultaneously. Another restriction on C' and D' is that C' **OR** $D' = C$, so that all necessary data will be transmitted. One way to obtain this result is by defining $C' = C$ **AND** A, and $D' = C$ **AND** B **AND** $\overline{C'}$. Note that the equation for D' simplifies to $D' = C$ **AND** B **AND** $\overline{A}$, so that it may be executed in parallel with the computaion of C'. Since Figure 6.1 illustrates the process of write request combining in a node at an arbitrary stage within the network (actually the MATSH cycle number), the expression for C is given as $C = A'$ **AND** P. The field P ("permission") is not a special field; it is included here merely to summarize the impact of de-combining from previous layers. P reflects the fact that the returning request from

the previous layer may have already been de-combined one or more times, so it may have already had its number of "1" bits reduced. Note that if the node under consideration were in the last stage of the network next to the MMs, then $P = 1$.

After working backwards through S_F cycles of the MATSH, the subaddress fields of the de-combined write requests arrive at the requesting PEs. These are used by the PEs to determine which data is to be sent ("pre-arbitration"). The appropriate data is then sent time-sequentially through the FIER. The PEs send data synchronously, so that a given time slot corresponds to a particular submodule in each MM. Each PE sends data during the time allocated for a given submodule only when it has a 1 in its returned write request subaddress field. In this way, only one requesting PE (for each MM) may be transmitting data at any given instant of time, and optical intensity combining is avoided. The upper and lower combine switch settings are used within the FIER to provide pathways that merge the pre-arbitrated write data streams as they flow to the destination MM.

For large MMs ($\gtrsim$100 Kbyte) the required subaddress field length would be much more than could be managed in the MATSH. In this case, the subaddress field method needs to be replaced with subaddress ranges instead. A subaddress range consists of lists of subaddress pairs indicating ranges of data to be written. Instead of simple **AND**s and **OR**s used to implement subaddress field combining, subaddress range combining requires the MATSH switching nodes to have more complex logic that can perform unions and splits of address ranges.

6.3 The Fetch-and-Add Algorithm

Fetch-and-add (F&A) is an inseparable combination of a read and write operation. This operation is cited in the literature [Gottlieb 83a, Gottlieb 83b] as "an important coordination primitive." The idea is to retrieve a value from a memory location X, add a predetermined integer increment value v to it (often $v = 1$), and store the new value back into the same memory location. A common notation for this is F&A(X,v).

6.3.1 F&A Applications

F&A(X,v) is used in parallel algorithms (for example) to allow separate processing elements to have an indication of when a procedure is complete by requiring each processor to execute F&A(Y,-1) when it completes its processing duties. If Y has been properly initialized, then when the value stored in Y reaches 0, all processors have finished. Another application is to use the F&A(Z,1) to assign unique integers to each PE as they become available (which is useful for assigning different loop iterations to specific PEs, for example). In this case, each PE executes F&A(Z,1) when it is available; the retrieved value tells it which loop iteration to perform. Because each PE receives a unique value, no two PEs execute the same loop iteration, and all loop iterations are assigned efficiently.

6.3.2 F&A Illustrations

Figure 6.2 shows the basic procedure involved in combining two F&A requests: F&A(X,ℓ) from PE_1 and F&A(X,r) from PE_2. This figure illustrates two PEs

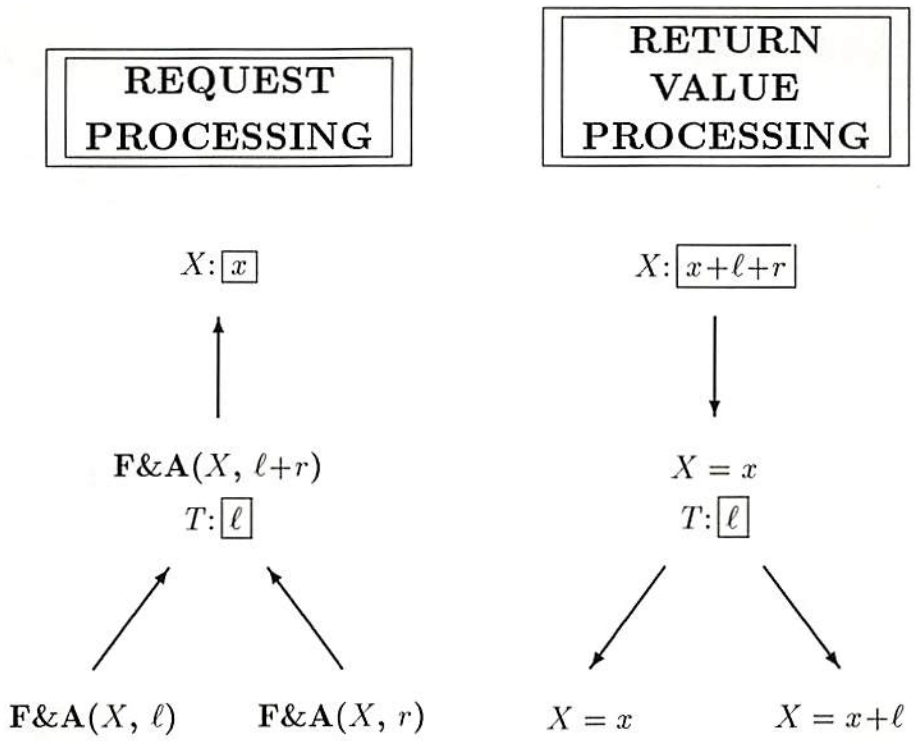


Figure 6.2: Illustration of Fetch-and-Add (F&A).

(bottom) connected to one MM (top) via a switch node (middle). For simplicity a network with only one switching stage is shown. It is very important to keep the serialization principle in mind in order to understand what must be done. Given that the original content of the memory location X is x, clearly the ultimate result of the two operations is that the memory location X must then contain the sum $x+\ell+r$. Furthermore, the two PEs each receive the values x and $x+\ell$. However, this is not the only possible outcome of combining these two requests. The output could equally as well have been $x+r$ and x (not shown in the Figure). The exact procedure is illustrated in two phases, "Request Processing", and "Return Value Processing".

The Request Processing phase begins with the two requests, F&A(X,ℓ) and F&A(X,r) entering the network. The requests meet at a switch node. In the examples shown here, the convention is that the request arriving from the left node is considered to have arrived first for the purposes of the serialization principle. Therefore the left increment value, ℓ, is saved in the temporary storage location T within the node. A new F&A request is then formed that combines the two requests: F&A(X,$\ell+r$). This request is sent to the memory location X.

The Return Value Processing phase occurs next. First the value stored in that location, $X = x$ is returned, then the increment value $\ell+r$ is added. Thus X ends up containing the sum $x+\ell+r$. Next the returned value $X = x$ arrives at the switch node. It must be "de-combined" to produce the appropriate distinct return values to the next level. This is accomplished by using the value stored in T within the node. The returned value $X = x$ is sent to the left node, and $X = x+\ell$ is sent to the right node, completing the F&A process. Two outcomes of this process should be noted: (1) the value saved from the request from the left node ends up being sent to the right node, and (2) the end result is the same as if the left request had been processed first. Note also that different conventions are possible for implementation of the F&A procedure, producing different but equally acceptable results.

A multilayer example of the F&A combining process is shown in Figure 6.3. This example illustrates how multiple combining and distributing can be used to assign a unique integer to each PE. Given that the value of V is initially 0, the process results in $V = 4$, and each PE receives a unique integer v in the range $(0 \leq v < 4)$.

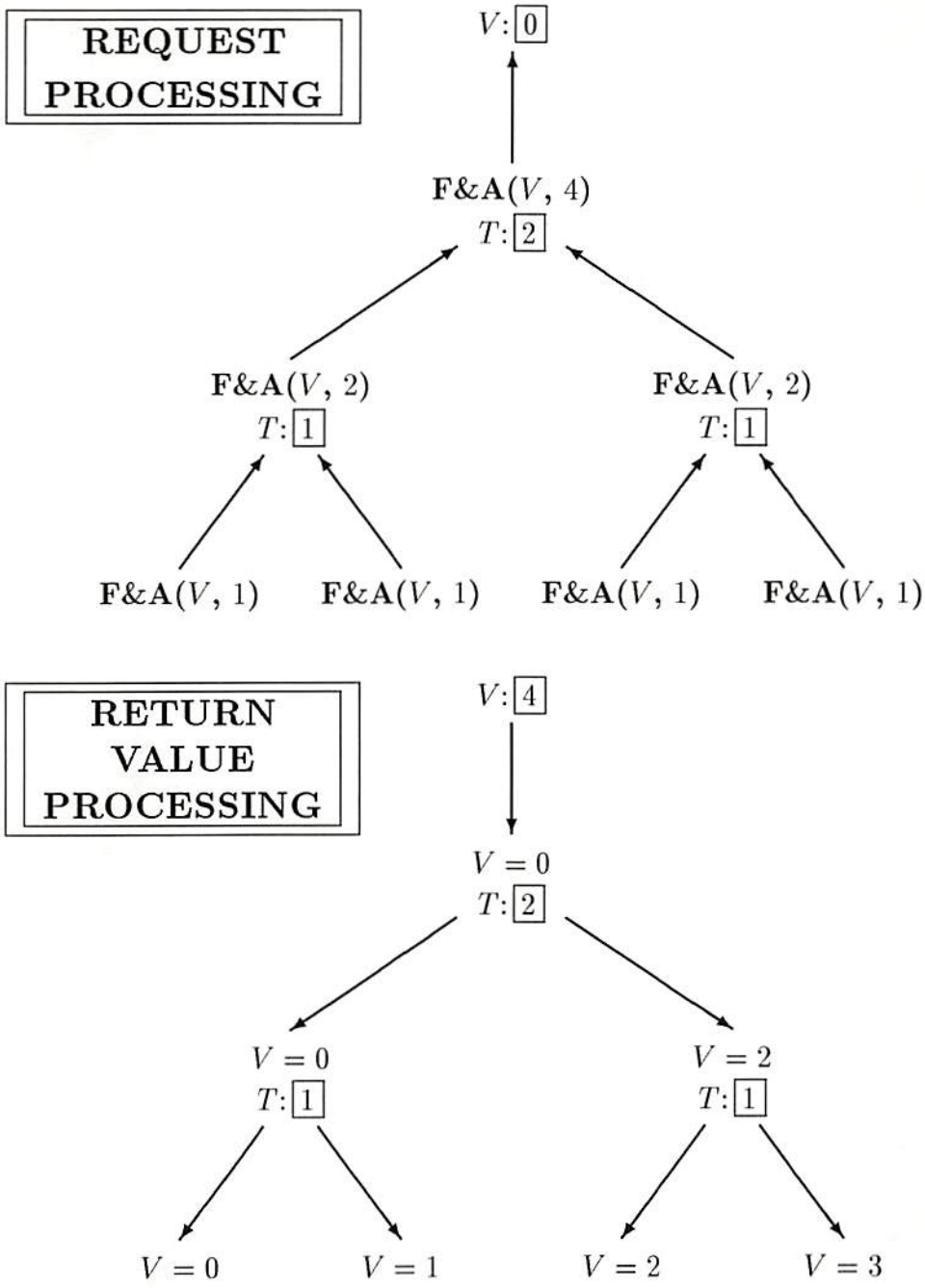


Figure 6.3: F&A Index Assignment Example.

6.3.3 How F&A works in the SMOEC

The F&A requests in the SMOEC are satisfied during a "F&A phase" during which only F&A requests will be processed. Due to reasons described below, there must be only a small number (variable in software) of F&A-able memory locations within the MMs, located in the same area of each MM. The local storage stack within each MATSH switch node is used to store the data values needed during the F&A operation. During the Request Processing phase, the ℓ values are pushed onto the stack within the switch. Then during the Return Value Processing phase the appropriate ℓ values are popped off the stack, and added to the memory returned value that is sent down the right branch.

It is important to note that a F&A phase uses only the MATSH for data communication; the FIER is not used at all. Therefore, since the MATSH is not designed for large data throughput (because of architecture and hardware design choices), it is important that only a small number (as small as is feasible given the application) of F&A-able locations be active at any one time. The maximum number of F&A-able locations active at any given time is also limited by the amount of local memory provided in the MATSH switch nodes. Since the F&A capability is intended for algorithm coordination purposes, and not as a main data communication mechanism, such a limitation is not likely to be overly restrictive.

A potentially useful extension of the F&A capability of the SMOEC is the ability to implement F&Φ algorithms. F&Φ is described in [Gottlieb 83a, Gottlieb 83b] as an extension of the F&A idea wherein the add is replaced by any commutative, associative, binary function Φ. Useful examples of Φ include **OR**, **AND**, and **MULTIPLY**. The ability of the SMOEC to implement any F&Φ is limited only by

the requirements that the MATSH switches contain the necessary logic to compute $u\Phi v$ and that each different type of function Φ_i be carried out in its own separate F&Φ_i phase. Given an appropriately powerful MATSH, the number of Φ_i functions available simultaneously is selectable by software as needed.

6.4 Special Purpose Algorithms

The SMOEC as described so far is a general-purpose machine in the Turing sense. However, some operations that do not parallelize efficiently on the SMOEC (as described so far) may occur very frequently in certain application areas. Therefore it is worthwhile to make available additional special-purpose algorithms to allow more efficient computation for some applications. Two such algorithms are presented in this section: data sorting and matrix transposition.

6.4.1 The Data Sort Algorithm

Sorting is easily accomplished on this architecture by sending the key data (data to be sorted) instead of MM addresses, and having the electronic "process-and-exchange" nodes base the switch decision on the value of the key data. A sorting phase would consist of sorting data blocks based on key data in specified PEs or MMs. The switches in the MATSH must contain the appropriate type of data comparison facility if a data sort phase is to be implement.

6.4.2 The Matrix Transpose Facility

The matrix transposition operation is frequently used in certain applicatons, such as in some scientific computing. A facility to expedite the transposing of a $k \times k$ matrix that is stored in multiple MMs (1 row per MM), can be easily added to the SMOEC. The addition of a serial-in, parallel-out (SIPO) shift register N units wide located across the bank of MMs will speed up this process (assuming that the number and size of MMs make switchable readout of rows and columns impractical.)

To perform the matrix transposition using the SIPO shift register requires the folowing steps, as illustrated ($k = 4$ shown) in Figure 6.4. First (Fig. 6.4a), the FIER is set to straight through ($PE_i \longleftrightarrow MM_i$). Next (Fig. 6.4b), all matrix data are sent in parallel from the MMs to the PEs. Third (Fig. 6.4c), the FIER is set to route data from all PEs to the first MM, allowing the data to be fed into the serial input end of the SIPO shift register. The PEs then take turns (Fig. 6.4d) sending each row of the matrix data in reverse order into the SIPO shift register. When the arrival of a row in the SIPO register is complete, it is immediately clocked in parallel from the SIPO register into the MMs. Ultimately (Fig. 6.4e), the MMs contain the transpose of the original matrix data in place. This $k \times k$ matrix transpose algorithm requires two FIER reconfiguration times, plus $\mathcal{O}[k^2]$ high-speed data transfer time. If no such hardware facility (the SIPO shift register) is provided, a matrix transpose takes k FIER reconfiguration times, which is significantly longer.

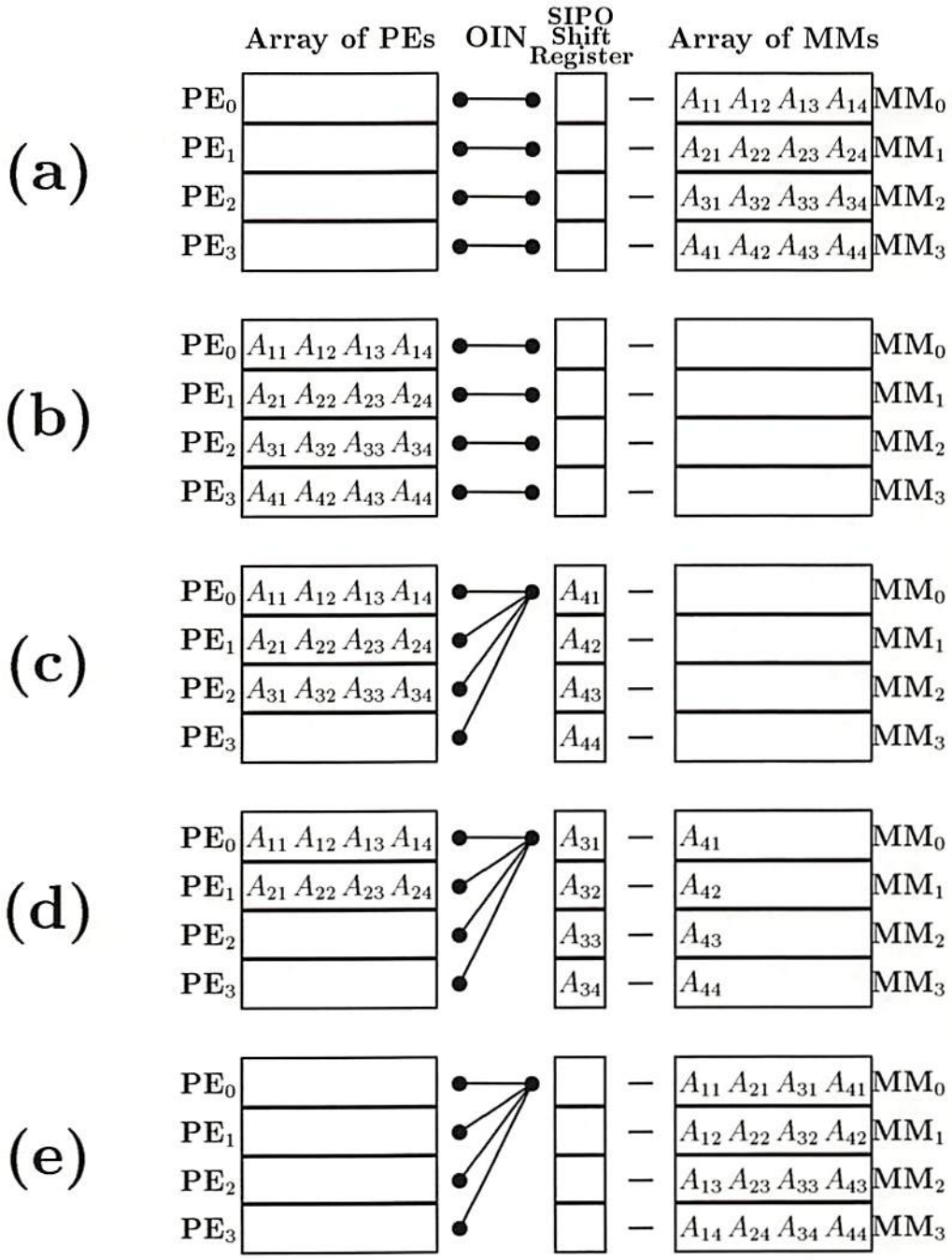


Figure 6.4: The Matrix Transpose Facility. SIPO, Serial-In Parallel Out.

Chapter 7

ARCHITECTURE PERFORMANCE

The SMOEC architecture is designed to provide a practical, flexible, scalable design for a shared memory computer system. This chapter elucidates these important system attributes by delving into implementation issues and providing specific examples of system recipes. Fitness of the SMOEC design for varied applications is discussed, and comparisons to similar systems are addressed.

7.1 Implementations

This section focusses on discussion of issues relevant to implementation of the SMOEC architecture, and the development of nine example SMOEC implementation recipes. These implementation recipes provide system attributes for various levels of technology (from current to future capabilities) and for widely differing system sizes, which illustrates the flexibility and scalability of the SMOEC system

design. Each SMOEC implementation recipe supplies a set of compatible values for all parameters given in the List of Notation preface section (page xiii).

7.1.1 Architecture Implementation Considerations

A primary assumption made in designing the SMOEC architecture is that for current and (likely) near-term future hardware, electronic logic device speeds significantly exceed optical array switching speeds. A passive optical network such as the one under consideration can allow high data rates since the data pass through the optical network without any delays due to optical array (SLM) switching times. Pumping large sized blocks of data through a single configuration of the optical network compensates for the slower reconfiguration time. Define three parameters: $\tau_{\text{O-Sw}}$ is the time required to simultaneously switch the optical switches (SLMs) in the FIER optical network, T_{FIER} is the total optical communication time required to access (read from or write to) an entire MM, and T_{MATSH} is the total control bit computation and write arbitration time in the MATSH electronic routing processor. Ultimately the desire is to balance these quantities, $\tau_{\text{O-Sw}}$, T_{FIER}, and T_{MATSH}, through the choice of system parameters so that hardware resources are not wasted and bandwidth matching concerns are satisfied.

One set of relations that provides such a balance is

$$\textbf{MATSH cycle time} \lesssim \textbf{FIER cycle time} \tag{7.1}$$

with

$$T_{\text{FIER}} \approx \tau_{\text{O-Sw}}. \tag{7.2}$$

The first relation, Equation 7.1, specifies that during the time taken to reconfigure the FIER optical network and to send data through the FIER, the next control

bits may be computed by the MATSH routing procesor. Since the FLAEM routing algorithm takes a significant amount of processing time to compute all the necessary switch settings, the option of equality given in the above relation is quite advantageous. The second relation, Equation 7.2, given above provides that the time penalty due to the relatively slow optical reconfiguration time $\tau_{\text{O-Sw}}$ is compensated by the use of high speed parallel optical data communication to provide a payoff of massive data throughput. When communication cycle procedures are established and the various fundamental hardware parameters are specified, these relations ultimately provide general restrictions on other system design parameters such as D (MM size), B (submodule size), and C (number of wires or fibers per MATSH routing processor node). System design concerns such as speed, size, cost, device availability, and intended applications must be weighed to select appropriate values for the system parameters.

7.1.2 Proposed SMOEC Implementations

When specifying an implementation of the SMOEC system the goal is to select values for the parameters listed in the List of Notation preface section (page xiii). It is essential that the selected parameter values be compatible with each other to produce a practical implementable system, and that the parameter values be appropriate for the envisioned system applications (i.e. general-purpose or specialized). This section presents a development of three example general-purpose implementations: the first using current technology (System $\mathbf{I_C}$), the second using anticipated near-term (near future) technology (System $\mathbf{I_N}$), and the third using

(medium-term) future technology (System $\mathbf{I_F}$). Each of these three example systems will be developed for three different system sizes, n=10 (N=1024), n=15 (N=32,768), and n=20 (N=1,048,576), ultimately yielding nine different implementation recipes.

7.1.2.1 Interconnection Network Communication Cycle Composition

The first step in devising a specific SMOEC implementation is to specify the components of the interconnection network communication cycle. Many possibilities exist for the design of the network cycle for the SMOEC system, ranging from fixed simple phases to complex variable adaptive phases. The network cycle may be a fixed or variable combination of read phases, write phases, fetch-and-add phases, and other phases. For specificity and simplicity, the example implementations developed herein will use a general-purpose fixed identical design for a basic read/write cycle, as described below.

For these example systems, let the same FIER configuration (connection pattern) be used to implement a read/write cycle, which is composed of a read cycle followed by a write cycle. So, define a FIER cycle time to be composed of three parts, as illustrated in Figure 7.1(a). The *Switch* component of the FIER cycle time is the optical reconfiguration time, $\tau_{\text{O-SW}}$, which is the time for all the SLMs in the FIER to switch simultaneously. The *Read* component of the FIER cycle time is the total optical communication time, T_{FIER}, which is the time required for an entire MM to dump its contents through the passive network to a PE. The *Write* component of the FIER cycle time is also T_{FIER}, since the time for a set of

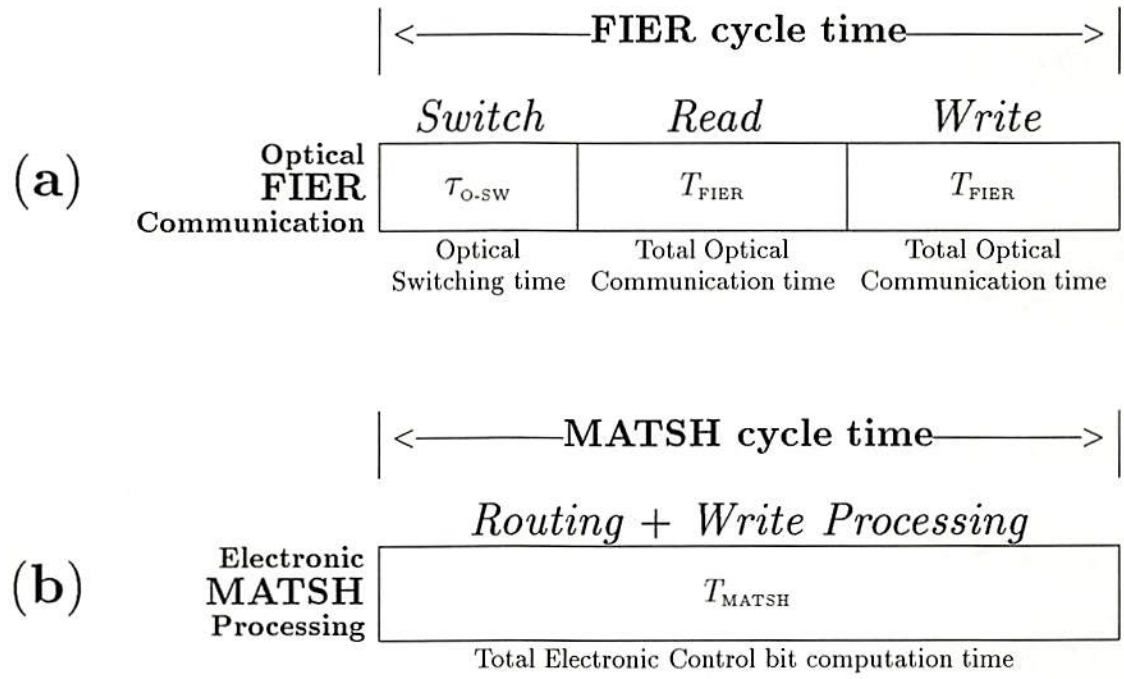


Figure 7.1: Read/write phase components: (a) FIER communication; (b) MATSH processing.

PEs to sequentially load an entire MM is equivalent to the *Read* time. Thus for the example under consideration,

$$\textbf{FIER Cycle Time} = \tau_{\text{O-SW}} + 2T_{\text{FIER}}. \tag{7.3}$$

The cycle time for the MATSH is illustrated in Figure 7.1(b). Each MATSH cycle must compute the switch settings for the FIER and perform write request arbitration using subaddress fields. Thus,

$$\begin{aligned}\textbf{MATSH Cycle Time} &= T_{\text{MATSH}} \\ &= S_F[\quad \vec{\Psi}_{\text{READ}} \ (\vec{\tau}^{(\text{READ})}_{\text{PROC}} + \vec{\tau}^{(\text{READ})}_{\text{XMIT}}) \\ &\quad + \vec{\Psi}_{\text{WRITE}} \ (\vec{\tau}^{(\text{WRITE})}_{\text{PROC}} + \vec{\tau}^{(\text{WRITE})}_{\text{XMIT}}) \\ &\quad + \overleftarrow{\Psi}_{\text{READ}} \ (\overleftarrow{\tau}^{(\text{READ})}_{\text{PROC}} + \overleftarrow{\tau}^{(\text{READ})}_{\text{XMIT}}) \\ &\quad + \overleftarrow{\Psi}_{\text{WRITE}} \ (\overleftarrow{\tau}^{(\text{WRITE})}_{\text{PROC}} + \overleftarrow{\tau}^{(\text{WRITE})}_{\text{XMIT}}) \ ]\end{aligned} \tag{7.4}$$

where the indices on the τ parameters indicate times for the single cycle of the E-SE stage (within the MATSH routing processor) to process (PROC) or transmit (XMIT) the necessary bits for a read (READ) or write (WRITE) operation in the forward ($\rightarrow$) or reverse ($\leftarrow$) directions, and the Ψ parameters indicate how many forward ($\rightarrow$) and reverse ($\leftarrow$) passes through the MATSH are required, and which type of data is acted upon, read-type (READ) or write-type (WRITE) data. (Recall that a MATSH pass is defined to consist of S_F cycles through the E-SE within the MATSH.)

Finally, inserting Equations 7.3 and 7.4 into the bandwidth-balancing relation of Equation 7.1 yields the fundamental balancing relation for the three SMOEC implementations developed herein:

$$T_{\text{MATSH}} \lesssim \tau_{\text{O-SW}} + 2T_{\text{FIER}} \tag{7.5}$$

From Section 5.3.2.2, the Basic-FLAEM routing algorithm (almost) always requires 2 "tries" to complete the routing of an arbitrary connection pattern. These 2 tries translate to 5 (3 forward plus 2 reverse) MATSH passes. The simple fixed read/write cycle illustrated in Figure 7.1 may be implemented by incorporating the write request processing into the final pass of the Basic-FLAEM routing algorithm. First, 4 passes of the MATSH are completed, operating solely on path vector data. These 4 passes are thus read-type MATSH passes. Then the final forward MATSH pass (pass #5) is performed, carrying out all the switch finalization procedures required in the Basic-FLAEM routing algorithm, while performing the first forward pass using the subaddress field data to perform the forward write arbitration pass. The subsequent reverse MATSH pass (pass #6) performs the reverse write arbitration pass, resulting in the completed de-combined arbitrated write requests arriving at the requesting PEs as needed. The PEs are ready to write once they've received these de-combined arbitrated write requests. These last 2 passes are the final 2 passes of the 6 MATSH passes required to generate the FIER switch settings and arbitrate the write requests. These two final MATSH passes are write-type passes, since they operate on subaddress fields instead of path vectors. Thus, for the example implementations developed herein, the numbers of MATSH passes required for the simple read/write process are: two

$$\begin{aligned} \vec{\Psi}_{\text{READ}} &= 2 \\ \overleftarrow{\Psi}_{\text{READ}} &= 2 \\ \vec{\Psi}_{\text{WRITE}} &= 1 \\ \overleftarrow{\Psi}_{\text{WRITE}} &= 1 \end{aligned} \tag{7.6}$$

Thus, Equation 7.4 becomes:

$$\begin{aligned} T_{\text{MATSH}} = S_F[\quad & 2\overrightarrow{\tau}^{(\text{READ})}_{\text{PROC}} + 2\overrightarrow{\tau}^{(\text{READ})}_{\text{XMIT}} \\ & + \overrightarrow{\tau}^{(\text{WRITE})}_{\text{PROC}} + \overrightarrow{\tau}^{(\text{WRITE})}_{\text{XMIT}} \\ & + 2\overleftarrow{\tau}^{(\text{READ})}_{\text{PROC}} + 2\overleftarrow{\tau}^{(\text{READ})}_{\text{XMIT}} \\ & + \overleftarrow{\tau}^{(\text{WRITE})}_{\text{PROC}} + \overleftarrow{\tau}^{(\text{WRITE})}_{\text{XMIT}} \ ] \end{aligned} \tag{7.7}$$

7.1.2.2 Development of Specific System Implementation Recipes

In the following three sections the development is presented for the specific values of parameters for the example systems. The order of parameter determination is important, and is determined by which parameters are particularly important or troublesome. The order of development used herein is merely one of many possible methods to determine parameters in a reasonable order given various hardware characteristics. Other sequences of system parameter determination may be employed, proceeding from known hardware characteristics or desired application constraints to determine the necessary bounds on the remaining parameters. For the examples under consideration, the following three-part order of presentation is employed. First, development is presented for parameters that vary solely among the three system types. Next, development is presented for parameters that vary solely among the three system sizes. Finally, individual developments are completed for parameters that vary among both system type and system size, producing nine system implementation recipes.

7.1.2.3 Three System Types: $\mathbf{I_C}$, $\mathbf{I_N}$, and $\mathbf{I_F}$

The switching time of the ferroelectric liquid crystal TSLMs in the FIER is a critical parameter for any SMOEC implementation. For the implementations under development here, let the optical switching time characterize the three system types. Thus, define [McKnight 95, Stevens 95, Kompanets 95]:

$$\tau_{\text{O-SW}} = \begin{cases} 50\mu\text{s} & \text{System } \mathbf{I_C} \\ 10\mu\text{s} & \text{System } \mathbf{I_N} \\ 3\mu\text{s} & \text{System } \mathbf{I_F} \end{cases} \tag{7.8}$$

Now, start from the relation given in Equation 7.2, $T_{\text{FIER}} \approx \tau_{\text{O-Sw}}$, which states that the granularity of the communication time is chosen to balance the optical switching time. Modify this to "T_{FIER} is within a small multiple (call it ξ) of $\tau_{\text{O-SW}}$," where ξ is "a small number," which may be expressed as:

$$\tfrac{1}{\xi}\tau_{\text{O-SW}} \lesssim T_{\text{FIER}} \lesssim \xi\tau_{\text{O-SW}}. \tag{7.9}$$

Choose (arbitrarily) $\xi \approx 4$, yielding:

$$\begin{array}{lll} 12\mu\text{s} \lesssim T_{\text{FIER}} \lesssim 200\mu\text{s} & & \text{System } \mathbf{I_C} \\ 2.5\mu\text{s} \lesssim T_{\text{FIER}} \lesssim 40\mu\text{s} & & \text{System } \mathbf{I_N} \\ 0.75\mu\text{s} \lesssim T_{\text{FIER}} \lesssim 12\mu\text{s} & & \text{System } \mathbf{I_F} \end{array} \tag{7.10}$$

Now, assume that the individual laser diodes and detectors can do:

$$r_{\text{FIER}} = \begin{cases} 1\text{GHz} = 10^9\text{bps} & \text{System } \mathbf{I_C} \\ 10\text{GHz} = 10^{10}\text{bps} & \text{System } \mathbf{I_N} \\ 50\text{GHz} = 5\times10^{10}\text{bps} & \text{System } \mathbf{I_F} \end{cases} \tag{7.11}$$

Using these values for r_{FIER}, and plugging in $D = T_{\text{FIER}} r_{\text{FIER}}$ gives for the size D of the MMs:

$$\begin{array}{lll} 12{,}500 \text{ bits} \lesssim D \lesssim 200{,}000 \text{ bits} & & \text{System } \mathbf{I_C} \\ 25{,}000 \text{ bits} \lesssim D \lesssim 400{,}000 \text{ bits} & & \text{System } \mathbf{I_N} \\ 37{,}500 \text{ bits} \lesssim D \lesssim 600{,}000 \text{ bits} & & \text{System } \mathbf{I_F} \end{array} \tag{7.12}$$

Choose for definiteness in the subsequent calculations:

$$D = \begin{cases} 2^{14} = 16{,}384 \text{ bits} & \text{System } \mathbf{I_C} \\ 2^{17} = 131{,}072 \text{ bits} & \text{System } \mathbf{I_N} \\ 2^{19} = 524{,}288 \text{ bits} & \text{System } \mathbf{I_F} \end{cases} \tag{7.13}$$

which after using $T_{\text{FIER}} = D/r_{\text{FIER}}$, yields:

$$T_{\text{FIER}} = \begin{cases} 16\mu s & \text{System } \mathbf{I_C} \\ 13\mu s & \text{System } \mathbf{I_N} \\ 10\mu s & \text{System } \mathbf{I_F} \end{cases} \tag{7.14}$$

And finally, using the relation from Equation 7.5, $T_{\text{MATSH}} \lesssim \tau_{\text{O-SW}} + 2T_{\text{FIER}}$, the values from Equations 7.8 and 7.14 yield:

$$T_{\text{MATSH}} = \begin{cases} 83\mu s & \text{System } \mathbf{I_C} \\ 36\mu s & \text{System } \mathbf{I_N} \\ 24\mu s & \text{System } \mathbf{I_F} \end{cases} \tag{7.15}$$

Next, the parameters are developed that quantify how the D bits in the MMs are divided up and transported through the MATSH routing processor. Each MM is divided up into words or "submodules" of size B. For the three systems, choose:

$$B = \begin{cases} 32 & \text{System } \mathbf{I_C} \\ 64 & \text{System } \mathbf{I_N} \\ 128 & \text{System } \mathbf{I_F} \end{cases} \tag{7.16}$$

So, for each MM, there are L words, where

$$L = \begin{cases} 512 & \text{System } \mathbf{I_C} \\ 2048 & \text{System } \mathbf{I_N} \\ 4096 & \text{System } \mathbf{I_F} \end{cases} \tag{7.17}$$

Now if the recirculating shuffle connection for each node in the E-SE (within the MATSH) is interconnected with C wires or fibers, the data rate per wire or fiber (K) may be reduced by a factor of C. For the systems under discussion, choose:

$$C = 8, \tag{7.18}$$

which yields the number of subaddress bits to be transmitted per fiber

$$K = L/C = \begin{cases} 64 & \text{System } \mathbf{I_C} \\ 256 & \text{System } \mathbf{I_N} \\ 512 & \text{System } \mathbf{I_F} \end{cases} \tag{7.19}$$

Thus K is the number of bits that need to be transmitted per wire or fiber in the MATSH routing processor during a write operation. The number of bits necessary for a read operation will be discussed later in this section. However, both read and write operations will need a few status bits to facilitate routing decisions, so allocate a few status bits U to be used for both operation types. For the examples under discussion, specify:

$$U = 4. \tag{7.20}$$

Now, specify the data rate r_w through the individual wires or fibers that comprise the recirculating shuffle connection in the E-SE within the MATSH routing processor. Select:

$$r_w = \begin{cases} 300\text{MHz} = 3\times10^8\text{bps} & \text{System } \mathbf{I_C} \\ 1\text{GHz} = 10^9\text{bps} & \text{System } \mathbf{I_N} \\ 5\text{GHz} = 5\times10^9\text{bps} & \text{System } \mathbf{I_F} \end{cases} \tag{7.21}$$

Assuming that the time for forward write data transmission is equal to the time for reverse write data transmission, these parameters may now be calculated:

$$\overrightarrow{\tau}^{(\text{WRITE})}_{\text{XMIT}} = \overleftarrow{\tau}^{(\text{WRITE})}_{\text{XMIT}} = (K+U)/r_w = \begin{cases} 210\text{ns} & \text{System } \mathbf{I_C} \\ 260\text{ns} & \text{System } \mathbf{I_N} \\ 100\text{ns} & \text{System } \mathbf{I_F} \end{cases} \tag{7.22}$$

In addition, the simple case of reverse read data transmission may be simply calculated:

$$\overleftarrow{\tau}^{(\text{READ})}_{\text{XMIT}} = U/r_w = \begin{cases} 13\text{ns} & \text{System } \mathbf{I_C} \\ 4.0\text{ns} & \text{System } \mathbf{I_N} \\ 0.80\text{ns} & \text{System } \mathbf{I_F} \end{cases} \tag{7.23}$$

7.1.2.4 Three System Sizes: $n=10$, $n=15$, and $n=20$

The SMOEC architecture is designed for "fine grained computing," so the appropriate system sizes are:

$$N \in \{10^3 \text{ to } 10^6\} \approx \{2^{10} \text{ to } 2^{20}\}. \tag{7.24}$$

Three choices for system size that span this range are selected, and labelled with the value of $n=\log_2 N$. The three system sizes under consideration are:

$$N = \begin{cases} 2^{10} = & 1024 \approx 10^3 & (n=10) \\ 2^{15} = & 32{,}768 \approx 10^{4.5} & (n=15) \\ 2^{20} = & 1{,}048{,}576 \approx 10^6 & (n=20) \end{cases} \tag{7.25}$$

This set of values for N (i.e. n) will be maintained throughout the development of the implementation recipes so that the impact of system size on the resulting SMOEC system scalability will be clearly revealed.

Current ferroelectric liquid crystal SLMs [McKnight 95, Stevens 95, Kompanets 95] have resolutions up to 512×512 pixels. A 512×512 pixel SLM has $(2^9)^2 = 2^{18}$=262,144 pixels available. For such a device, the maximum size restricted-F RS-EGS main section stage (requiring $W = N \cdot F$ pixels) would be $N = 2^{14} = 16{,}384$ with $F = 2^4 = 16$. Future SLMs with greater pixel resolutions are expected to be developed, particularly because of the push to create high-resolution display media for HDTV (high-definition television) applications, so larger SMOEC implementations are expected to be quite feasible in the relatively near future.

Now, the parameters that depend on the characteristics of the RS-EGS network may be determined. For a restricted-F RS-EGS with minimal device cost, the "fan-out" F and number of main section stages S_S may be determined from the values

for N by using Richards' formulae (listed in Table 3.1 on page 47):

$$F = \begin{cases} 16 & (n=10) \\ 16 & (n=15) \\ 32 & (n=20) \end{cases} \tag{7.26}$$

so

$$f = \log_2 F = \begin{cases} 4 & (n=10) \\ 4 & (n=15) \\ 5 & (n=20) \end{cases} \tag{7.27}$$

and

$$S_S = \begin{cases} 14 & (n=10) \\ 26 & (n=15) \\ 33 & (n=20) \end{cases} \tag{7.28}$$

Which gives for the total number of stages in the FIER optical network (and recirculating passes through the E-SE in the MATSH routing processor):

$$S_F = S_S + 2f = \begin{cases} 22 & (n=10) \\ 34 & (n=15) \\ 43 & (n=20) \end{cases} \tag{7.29}$$

In the forward direction, a read operation is required to transmit the path vector for the connection in progress, plus the few status bits (U) previously mentioned. The number of bits in the path vector (E) is:

$$E = n + f + S_S = \begin{cases} 28 & (n=10) \\ 45 & (n=15) \\ 58 & (n=20) \end{cases} \tag{7.30}$$

which results in the number of bits per read (J) as:

$$J = E + U = \begin{cases} 32 & (n=10) \\ 49 & (n=15) \\ 62 & (n=20) \end{cases} \tag{7.31}$$

7.1.2.5 Nine System Recipes: Three Types With Three Sizes

Now the development of the example systems is completed by finalizing the parameters for each of the nine combinations of system type and size.

The remaining forward read transmit time ($\overrightarrow{\tau}^{(\text{READ})}_{\text{XMIT}}$) may now be specified using the nine cases since the number of bits per read (J) varies with system size and the MATSH data rate (r_{W}) varies with system type:

$$\overrightarrow{\tau}^{(\text{READ})}_{\text{XMIT}} = J/r_{\text{W}} = \begin{cases} 110\text{ns} & \text{System } \mathbf{I_C}(n=10) \\ 160\text{ns} & \text{System } \mathbf{I_C}(n=15) \\ 210\text{ns} & \text{System } \mathbf{I_C}(n=20) \\ 32\text{ns} & \text{System } \mathbf{I_N}(n=10) \\ 49\text{ns} & \text{System } \mathbf{I_N}(n=15) \\ 62\text{ns} & \text{System } \mathbf{I_N}(n=20) \\ 6.4\text{ns} & \text{System } \mathbf{I_F}(n=10) \\ 9.8\text{ns} & \text{System } \mathbf{I_F}(n=15) \\ 12\text{ns} & \text{System } \mathbf{I_F}(n=20) \end{cases} \tag{7.32}$$

The four MATSH data transmit time per stage parameters may now be combined with the number of stages to yield the total amount of time in the MATSH that is devoted to transmitting information:

$$T_{\text{XMIT}} = S_F[2\overrightarrow{\tau}^{(\text{READ})}_{\text{XMIT}} + \overrightarrow{\tau}^{(\text{WRITE})}_{\text{XMIT}} + 2\overleftarrow{\tau}^{(\text{READ})}_{\text{XMIT}} + \overleftarrow{\tau}^{(\text{WRITE})}_{\text{XMIT}}] \tag{7.33}$$

$$= \begin{cases} 15\text{ns} & \text{System } \mathbf{I_C}(n=10) \\ 27\text{ns} & \text{System } \mathbf{I_C}(n=15) \\ 37\text{ns} & \text{System } \mathbf{I_C}(n=20) \\ 13\text{ns} & \text{System } \mathbf{I_N}(n=10) \\ 21\text{ns} & \text{System } \mathbf{I_N}(n=15) \\ 28\text{ns} & \text{System } \mathbf{I_N}(n=20) \\ 4.8\text{ns} & \text{System } \mathbf{I_F}(n=10) \\ 7.7\text{ns} & \text{System } \mathbf{I_F}(n=15) \\ 9.9\text{ns} & \text{System } \mathbf{I_F}(n=20) \end{cases} \tag{7.34}$$

so the time left in the MATSH to be devoted to processing information within the E-SE nodes is:

$$T_{\text{PROC}} = T_{\text{MATSH}} - T_{\text{XMIT}} = \begin{cases} 68\text{ns} & \text{System } \mathbf{I_C}(n=10) \\ 56\text{ns} & \text{System } \mathbf{I_C}(n=15) \\ 46\text{ns} & \text{System } \mathbf{I_C}(n=20) \\ 23\text{ns} & \text{System } \mathbf{I_N}(n=10) \\ 15\text{ns} & \text{System } \mathbf{I_N}(n=15) \\ 8.5\text{ns} & \text{System } \mathbf{I_N}(n=20) \\ 19\text{ns} & \text{System } \mathbf{I_F}(n=10) \\ 16\text{ns} & \text{System } \mathbf{I_F}(n=15) \\ 14\text{ns} & \text{System } \mathbf{I_F}(n=20) \end{cases} \tag{7.35}$$

These values for T_{PROC} must be divided between the various types of processing within the nodes in the E-SE in the MATSH, as follows:

$$T_{\text{PROC}} = S_F\,[\,2\overrightarrow{\tau}^{(\text{READ})}_{\text{PROC}} + \overrightarrow{\tau}^{(\text{WRITE})}_{\text{PROC}} + 2\overleftarrow{\tau}^{(\text{READ})}_{\text{PROC}} + \overleftarrow{\tau}^{(\text{WRITE})}_{\text{PROC}}\,]. \tag{7.36}$$

To evaluate the practicality of the T_{PROC} values listed above, make some general assumptions about the E-SE processing times:

$$\overleftarrow{\tau}^{(\text{READ})}_{\text{PROC}} \ll \overrightarrow{\tau}^{(\text{READ})}_{\text{PROC}} \tag{7.37}$$

and

$$\tau^{(*)}_{\text{PROC}} \triangleq \overrightarrow{\tau}^{(\text{WRITE})}_{\text{PROC}} = \overleftarrow{\tau}^{(\text{WRITE})}_{\text{PROC}} \approx (\overrightarrow{\tau}^{(\text{READ})}_{\text{PROC}} + \overleftarrow{\tau}^{(\text{READ})}_{\text{PROC}}). \tag{7.38}$$

Thus, Equation 7.36 becomes:

$$T_{\text{PROC}} = 4\,S_F\,\tau^{(*)}_{\text{PROC}}\,, \tag{7.39}$$

from which the final result for time available within each E-SE cycle for processing:

$$\tau_{\text{PROC}}^{(*)} = T_{\text{PROC}}/4S_F = \begin{cases} 770\text{ns} & \text{System } \mathbf{I_C}(n=10) \\ 410\text{ns} & \text{System } \mathbf{I_C}(n=15) \\ 260\text{ns} & \text{System } \mathbf{I_C}(n=20) \\ 270\text{ns} & \text{System } \mathbf{I_N}(n=10) \\ 110\text{ns} & \text{System } \mathbf{I_N}(n=15) \\ 50\text{ns} & \text{System } \mathbf{I_N}(n=20) \\ 220\text{ns} & \text{System } \mathbf{I_F}(n=10) \\ 120\text{ns} & \text{System } \mathbf{I_F}(n=15) \\ 80\text{ns} & \text{System } \mathbf{I_F}(n=20) \end{cases} \tag{7.40}$$

These values are quite reasonable for the small numbers of simple operations required for read and write processing within the E-SE nodes.

7.1.2.6 Loss Calculations for Implementation Examples

It is essential to examine the optical loss encountered in the multistage implementations just specified. Calculations are now performed to determine the number of optical repeaters required within the FIER optical network to permit an acceptable amount of loss.

In order to perform these calculations, three assumptions are made. First, require a minimum of 1000 photons per bit at the detector for adequate bit discrimination. From this, the minimum energy required at the detector is approximately

$$E_{\text{MIN}}^{(\text{Det})} = 0.5\,\text{fJ} = 5\times10^{-16}\text{ Joules.} \tag{7.41}$$

Second, assume that the various input power levels from the laser diodes are:

$$P_{\text{INPUT}}^{(\text{LD})} = \begin{cases} 100\text{mW} & \text{System } \mathbf{I_C} \\ 300\text{mW} & \text{System } \mathbf{I_N} \\ 1\text{W} & \text{System } \mathbf{I_F} \end{cases} \tag{7.42}$$

Third, assume 50% loss per stage.

Now calculate the required minimum power at the detector, $P_{\text{MIN}}^{(\text{Det})}$:

$$P_{\text{MIN}}^{(\text{Det})} = E_{\text{MIN}}^{(\text{Det})} \cdot r_{\text{FIER}}. \tag{7.43}$$

Recalling the values for the optical data rate through the FIER optical network given in Equation 7.11:

$$r_{\text{FIER}} = \begin{cases} 1\text{GHz} = 10^9\text{bps} & \text{System } \mathbf{I_C} \\ 10\text{GHz} = 10^{10}\text{bps} & \text{System } \mathbf{I_N} \\ 50\text{GHz} = 5\times10^{10}\text{bps} & \text{System } \mathbf{I_F} \end{cases} \tag{7.44}$$

yields for the minimum power at the detector:

$$P_{\text{MIN}}^{(\text{Det})} = \begin{cases} 0.5\mu\text{W} & \text{System } \mathbf{I_C} \\ 5\mu\text{W} & \text{System } \mathbf{I_N} \\ 25\mu\text{W} & \text{System } \mathbf{I_F} \end{cases} \tag{7.45}$$

Now, calculate the total tolerable optical loss factor (without repeaters) in the FIER, from laser diodes to detectors:

$$\mathcal{L}_{\text{TOL}} = \frac{P_{\text{INPUT}}^{(\text{LD})}}{P_{\text{MIN}}^{(\text{Det})}} = \begin{cases} 2\times10^5 \gtrsim 2^{17} & \text{System } \mathbf{I_C} \\ 6\times10^4 \gtrsim 2^{15} & \text{System } \mathbf{I_N} \\ 4\times10^4 \gtrsim 2^{15} & \text{System } \mathbf{I_F} \end{cases} \tag{7.46}$$

So, with the assumed 50% loss per stage, the number of stages that may be cascaded without repeaters is:

$$S_{\text{TOL}} = \log_2(\mathcal{L}_{\text{TOL}}) = \begin{cases} 17 \text{ stages} & \text{System } \mathbf{I_C} \\ 15 \text{ stages} & \text{System } \mathbf{I_N} \\ 15 \text{ stages} & \text{System } \mathbf{I_F} \end{cases} \tag{7.47}$$

Recall that the total number of stages in the FIER optical network is:

$$S_F = \begin{cases} 22 & (n=10) \\ 34 & (n=15) \\ 43 & (n=20) \end{cases} \tag{7.48}$$

Thus the number of repeaters, $\mathcal{R}$, required by the nine system recipes is:

$$\mathcal{R} = \begin{cases} 1 & \text{System } \mathbf{I_C}(n=10) \\ 1 & \text{System } \mathbf{I_C}(n=15) \\ 2 & \text{System } \mathbf{I_C}(n=20) \\ 1 & \text{System } \mathbf{I_N}(n=10) \\ 2 & \text{System } \mathbf{I_N}(n=15) \\ 2 & \text{System } \mathbf{I_N}(n=20) \\ 1 & \text{System } \mathbf{I_F}(n=10) \\ 2 & \text{System } \mathbf{I_F}(n=15) \\ 2 & \text{System } \mathbf{I_F}(n=20) \end{cases} \tag{7.49}$$

Therefore each of the nine system recipes specified herein requires either one or two optical repeaters to be inserted in the FIER optical network.

The optical repeaters to be used in the FIER optical network must be capable of operation at the high data rates employed in the FIER. Therefore the suggested implementation for the one or two required optical repeaters is a simplified version of the transmit and receive components that provide input to and output from the FIER. Figure 7.2 provides an illustration of the components of an optical repeater unit, showing where it is located in the FIER optical network, and how it resembles the I/O from the FIER to the PEs and MMs. The figure shows one high speed optical repeater unit; two repeater units are required for some larger system implementations (as derived above). Each of the two fiber bundles used in the optical repeater unit provides bidirectional I/O to a section of the FIER optical network. The high speed optical repeater unit uses independent transmit/receive modules (Tx/Rx modules) which may be located on separate component boards. Each independent Tx/Rx module contains individual locally controlled laser diodes and detectors that are identical to the Tx/Rx modules that are collocated with the PEs on the PE boards and the Tx/Rx modules that are collocated with the

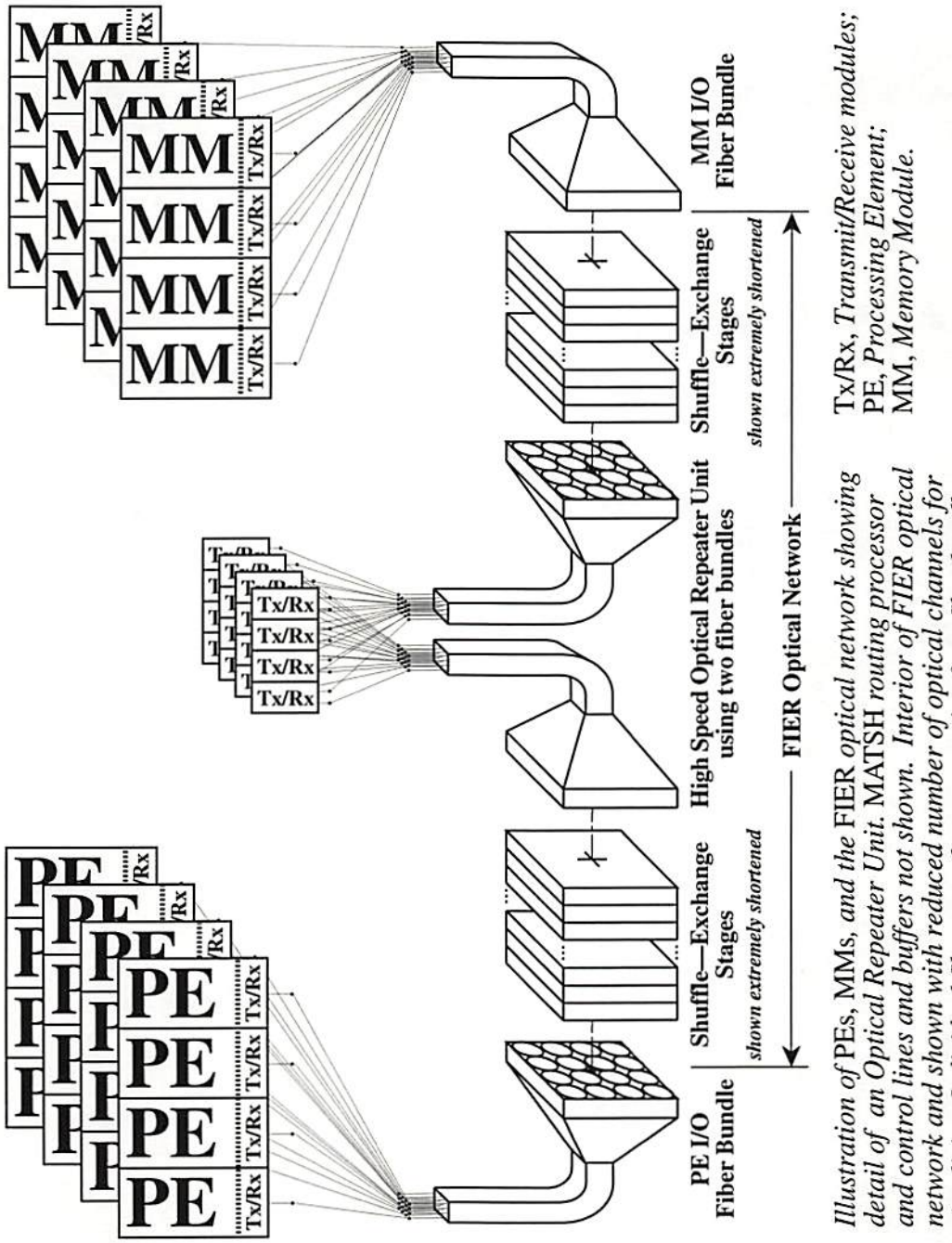


Figure 7.2: An optical repeater unit within the FIER optical network

MMs on the MM boards. Since identical technology is used to implement the optical repeater unit as is used for optical data generation and detection, the same high speeds may be maintained throughout the FIER optical network: at the PEs, through the one or two optical repeater units, and at the MMs. The high speed optical repeater unit design presented here differs from other optical repeater techniques, such as SLMs with gain, which are not capable of such high speed operation or as large an amount of gain.

7.1.2.7 Completed Specific System Implementations with Varied Sizes

The final nine system implementation recipes developed above are summarized in the following three tables. Table 7.1 lists the parameters that vary with system type, Table 7.2 lists the parameters that vary with system size, and Table 7.3 lists the parameters that vary both with system type and size.

Parameter	Computed by	System $\mathbf{I_C}$	System $\mathbf{I_N}$	System $\mathbf{I_F}$
$\tau_{\text{O-SW}}$	Hardware Parameter *Optical switching time*	50μs	10μs	3.0μs
r_{FIER}	Hardware Parameter *Data rate through FIER*	1 GHz	10 GHz	50 GHz
D	Restricted choice *Memory Module size (bits)*	16,384	131,072	524,288
T_{FIER}	$= D/r_{\text{FIER}}$ *Time to access entire MM thru FIER*	16μs	13μs	10μs
T_{MATSH}	$\tau_{\text{O-SW}} + 2T_{\text{FIER}}$ *Time to perform routing in the MATSH*	83μs	36μs	24μs
B	Restricted choice *Word size in MMs ("submodule" size)*	32	64	128
L	$= D/B$ *Number of words per MM*	512	2048	4096
C	Restricted choice *Number of wires or fibers per MATSH node*	8	8	8
K	$= L/C$ *Number of words per MATSH wire or fiber*	64	256	512
U	Software Parameter *Number of status bits for MATSH routing*	4	4	4
r_{W}	Hardware Parameter *Data rate between recirc. stage in MATSH*	300 MHz	1 GHz	5 GHz
$\overrightarrow{\tau}^{(\text{WRITE})}_{\text{XMIT}}$	$= (K+U)/r_{\text{W}}$	210ns	260ns	100ns
$\overleftarrow{\tau}^{(\text{WRITE})}_{\text{XMIT}}$	$= \overrightarrow{\tau}^{(\text{WRITE})}_{\text{XMIT}}$	210ns	260ns	100ns
$\overleftarrow{\tau}^{(\text{READ})}_{\text{XMIT}}$	$= U/r_{\text{W}}$	13ns	4.0ns	0.80ns

Table 7.1: SMOEC system implementation parameters for three system types

Parameter	Computed by	System Size		
		$n=10$	$n=15$	$n=20$
N	$= 2^n$	$2^{10} \approx 10^3$	$2^{15} \approx 10^{4.5}$	$2^{20} \approx 10^6$
	System size	=1024	=32,768	=1,048,576
F	from Richards' Formulae	16	16	32
	"Fan-out" and "Fan-in"			
f	$= \log_2 F$	4	4	5
S_S	from Richards' Formulae	14	26	33
	"Main section" stages in FIER			
S_F	$= S_S + 2f$	22	34	43
	Total number of stages in FIER			
E	$= n + f + S_S$	28	45	58
	Number of Path Vector Bits			
J	$= E + U$	32	49	62
	Number of Bits per READ			

Table 7.2: SMOEC system implementation parameters for three system sizes

Parameter	Computed by	System Size	System $\mathbf{I_C}$	System $\mathbf{I_N}$	System $\mathbf{I_F}$
$\overrightarrow{\tau}^{(\text{READ})}_{\text{XMIT}}$	$= J/r_{\text{W}}$	$n=10$	110ns	32ns	6.4ns
		$n=15$	160ns	49ns	9.8ns
		$n=20$	210ns	62ns	12ns
T_{XMIT}	$= S_F[2\overrightarrow{\tau}^{(\text{READ})}_{\text{XMIT}} + \overrightarrow{\tau}^{(\text{WRITE})}_{\text{XMIT}}$ $+ 2\overleftarrow{\tau}^{(\text{READ})}_{\text{XMIT}} + \overleftarrow{\tau}^{(\text{WRITE})}_{\text{XMIT}}]$ *Time devoted to data transmission in the MATSH*	$n=10$	15μs	13μs	4.8μs
		$n=15$	27μs	21μs	7.7μs
		$n=20$	37μs	28μs	9.9μs
T_{PROC}	$= T_{\text{MATSH}} - T_{\text{XMIT}}$ *Time devoted to routing processing in the MATSH*	$n=10$	68μs	23μs	19μs
		$n=15$	56μs	15μs	16μs
		$n=20$	46μs	8.5μs	14μs
$\tau^{(*)}_{\text{PROC}}$	$= T_{\text{PROC}}/4S_F$ *Available processing time in a MATSH node*	$n=10$	770ns	270ns	220ns
		$n=15$	410ns	110ns	120ns
		$n=20$	260ns	50ns	80ns
$\mathcal{R}$	(see Section 7.1.2.6) *Number of optical repeater units*	$n=10$	1	1	1
		$n=15$	1	2	2
		$n=20$	2	2	2

Table 7.3: SMOEC system implementation parameters for three system types and three system sizes

7.1.3 Conclusions From Implementation Examples

The SMOEC architecture is shown to be a feasible design, because sets of implementation parameters based on practical hardware characteristics illustrate the realizability of systems using current (as well as future) device technology. Practicality of the architecture is the fundamental objective of the design.

The development of these nine SMOEC implementation recipes illustrates the flexibility and adaptability of the SMOEC architecture. Throughout the specification of the preceding nine recipes, choices were available concerning which elements of the architecture to emphasize or give precedence to over other elements. Thus a SMOEC implementation may be tailored to reflect particular hardware capabilities or to optimize system behavior for particular intended applications.

The nine SMOEC implementation recipes also serve to demonstrate the scalability of the SMOEC architecture. The system parameters vary very slowly with system size; larger system sizes have quite reasonable hardware requirements and operating parameters.

Thus the SMOEC architecture is shown to exhibit practicality, flexibility, and scalability through the use of the nine implementation recipe examples.

7.2 Applications

The SMOEC computer is designed to function as a general-purpose parallel computer. However, some applications will naturally be more adaptable to the SMOEC architecture and be able to take advantage of its unique features.

One important feature of the SMOEC is its ability to take advantage of data that is organized into logical units within the MMs (as mentioned in Section 7.3). This means that the SMOEC is especially suitable for applications with data that are organized and accessed in an orderly manner. Examples of this type of application include digital simulation of physical phenomena such as aerodynamics and fluid flow, simulation of highway or air vehicle traffic patterns, analysis of data sets, and other structured scientific computing. Applications that depend on random data references are likely to be a poorer fit to this architecture. Further applications suitable for the SMOEC may also be found by selecting appropriate deterministic algorithms that are implemented on the RP3 [Darema 86], such as molecular dynamics and large 2-dimensional Fast Fourier Transforms (FFTs).

Another aspect of the SMOEC architecture with a large impact on applications is the result of large block data transfers. This aspect adds more weight to the preference for logically organized data. Thus structured data references to data blocks within the MMs will be very efficiently implemented in the SMOEC. Applications that work on structured data but which have unstructured, repeated random data requests will perform relatively slower. This preference for structured data references to logically organized data is a direct consequence of the reconfiguration rate of the FIER hardware (SLM switching speed) being much smaller than the bandwidth of data transfer through the passive FIER.

7.2.1 Simplified SMOEC Architectures For Specific Applications

For some types of applications, the algorithms may be developed to require only one-to-one connection patterns (permutations). A permutation-restricted SMOEC may be designed to run these types of applications, by replacing the FIER with a simplified permutation-restricted version which replaces the two TSLMs per stage with a single HWP SLM. This simplified FIER does not have the architecturally-required 50% loss per stage since no combine or broadcast operations are ever performed. In this case the MATSH would be identical to the MATSH for the general case SMOEC.

As a further special case, the SMOEC is also suitable for implementing sets of pre-computed fixed permutation patterns, like those that the GF11 [Beetem 85, Weingarten 90, Butler 93] was designed to implement. The GF11 was designed to do large-scale numerical solutions of problems in quantum chromodynamics (QCD). It stored 1024 fixed interconnection patterns for its rearrangeable network, which could be chosen as needed during run time. No other interconnection patterns could be directly implemented. The QCD algorithms operated on grids from 6×6×6×6 to 16×16×16×16. Numerical integration over from 72,576 to 4,670,016 dimensions are required for such calculations. It is likely that such QCD problems, or any similar orderly problems that work well on fixed sets of interconnection patterns, would also be very suitable for a simplified SMOEC.

A fixed-permutation-set SMOEC may be designed for such applications that require only specific sets of permutation interconnection patterns. For this case

the MATSH may be replaced by a control bit set storage bank. Since only one-to-one connection patterns will be implemented, the previously mentioned simplified FIER with single HWP SLMs replacing TSLM pairs would also be used. Restriction of the SMOEC to SIMD operation may also be appropriate for such applications.

7.3 Comparison with Other Shared Memory Computers

The SMOEC is, in several ways, similar to the electronic computer architecture known as the RP3 [Pfister 85a]. This computer is also based on a shared memory model (although it can also implement message passing — direct PE-to-PE communications). At the same time, these two architectures have some very interesting differences, which have implications on the method of operation of the SMOEC, the ability to simulate its performance, and the type of applications that are particularly suited to its strengths.

The RP3 uses two separate networks to do the processor-to-memory communication [Pfister 85a]. It has a simple "low latency" network that is used for simple communication tasks that are not expected to need combining, and a "combining" network that will combine such messages as F&A (Fetch-and-add, described in 6.3). The SMOEC really has four types of processor-memory communication, but they do not operate in separate networks. These communication types are: read phase, write phase, F&A phase, and data sort phase. Unlike the RP3, these phases

all incorporate message combining as an integral part of the procedure. The optical interconnection network is dedicated exclusively to one of these phases at at time.

The RP3 handles message conflicts in both networks by an elaborate set of queues that serve to hold messages until they can be properly routed or combined [Norton 85]. Both internal link conflicts and output port conflicts are buffered in these queues. This is in marked contrast to the SMOEC, which has no message queues whatsoever. All communication requests of a given type in the SMOEC are required to be satisfied in $\mathcal{O}[1]$ network passes. Conflicts are resolved during the progress of the routing algorithm rather than being stored for later processing. The SMOEC resolves internal link conflicts by using the routing algorithm within the Address Computer to effectively utilize the extra nodes provided by the EGS network design. Thus link conflicts which could arise in smaller networks (such as a simple Ω network) are avoided. Output port conflicts are resolved by incorporating message combining (of messages destined for the same MM) into the communication algorithms used in the SMOEC; thus in the SMOEC there are effectively no possible output port conflicts.

The lack of queues in the SMOEC has an interesting side benefit. Often a critical assumption in the simulation of the RP3 is that it has infinite queues, so that queue overflow will not have to be modelled. It is also assumed that memory requests are randomly distributed in an even manner. However, these assumptions may be invalid because in some cases hot spots can cause such queues to overflow in a tree-like pattern, slowing down or halting the entire network. Since the SMOEC

does not have queues, and resolves all requests in $\mathcal{O}[1]$ network passes, it may be simulated without requiring these limiting simplifications.

An especially important difference between these two architectures is the desired arrangement of information in the MMs. In the RP3, a hashing function is required to translate virtual addresses to physical addresses so that the probability of multiple (nonidentical) requests to the same MM is reduced [Gottlieb 83a]. This is because if a MM receives m independent requests, they must be handled by carrying out m serial memory accesses. On the other hand, since the SMOEC dumps the entire contents of a MM when data is requested from it, all requests are simultaneously satisfied. In fact, since the contents of a MM are delivered as a complete package, it is preferred that data be logically grouped together so that each MM contains logically related data. A hashing function would only serve to decrease the performance of the SMOEC.

One similarity between the SMOEC and the RP3 is that both are amenable to the same techniques of software data caching and ensuring cache coherence [Brantley 85]. A cache is a block of memory located within each PE that maintains private copies of some frequently used MM data. The purpose of caches is to reduce the number of relatively slower accesses to the shared memory by introducing another layer to the memory hierarchy between local memory and global shared memory. Cache coherence is the requirement that a value stored in the cache be identical to the last value stored at the corresponding shared MM location. This is accomplished in the RP3 by requiring compile-time coherence checks which tag a datum as either cacheable or noncacheable. Software data caching in this manner can easily be added to the SMOEC.

It is important to remember that for evaluating the relative merits of these two architectures, network metrics such as bandwidth and path count cannot be relied upon to give accurate measures. Only simulation of actual algorithms on these computers can give a reliable estimate of performance [Levitan 85]. Thus it is especially helpful that the SMOEC requires no inherent statistical assumptions to be made for simulation of its performance. Simulation of the operation of the SMOEC is an important area for future work.

Chapter 8

CONCLUSIONS

The objective of this research has been the design of a hybrid parallel digital computer system architecture which offers improved computational power over existing parallel computers, while maintaining a realizable hardware complexity level and utilizing only current and/or near future hardware devices. The anticipated improvement in performance stems from the use of an innovative shared memory architecture involving an integration of electronic processing elements with a novel passive optical interconnection network.

The formidable interconnection network requirements of the MIMD shared memory paradigm are in part satisfied through the incorporation of optics in several ways. The potentially large number and high bandwidth of the data lines through the passive optical interconnection network (FIER) provide massive data throughput. The capability of loading the FIER control signals in parallel permits a separate electronic routing procesor unit to perform routing control bit computation off-line and in parallel with data passing through the FIER. Furthermore, the need for simultaneous parallel access to individual shared memory locations also

warrants the incorporation of optics. Passive optical switching, with the enhanced combine and broadcast capabilities, provides powerful facilities for read and write conflict resolution. In addition, although the FIER was developed as the central subsystem for the SMOEC system, it is a self-contained subsystem that may be adapted for use in other computing and communication systems.

Essential routing and control aspects of interconnection network design are often not explored in sufficient detail to ensure that these aspects will not unduly limit system performance. Control issues for the FIER optical network are presented herein so that they can be solved during the design process. The FLAEM routing algorithm was designed to control a regular simplified combining (or permutation-restricted) EGS network in a circuit-switched manner in $\mathcal{O}[\log N]$ time. This algorithm is a useful new method for this class of EGS networks, which are particularly suited for passive optical implementation.

The philosophy employed in the SMOEC design process involved addressing architecture goals simultaneously with hardware concerns and control algorithm development. Simultaneous focus on these areas was crucial since each decision made in one area had a strong impact on the other two areas. This design philosophy is an important aspect of this effort. It is an uncommon philosophy in the field of optical and hybrid parallel computing, since control algorithms and their corresponding impact on computer architecture and optical hardware design have typically been given at most secondary consideration.

An example of the impact of the design philosophy's triple focus is the balancing of execution times of control bit computations, data communication phases, and optical interconnection network reconfiguration. The control algorithm for the

interconnection network in such a shared memory architecture is fairly complex, and takes significant computation time even when computed on a parallel machine (e.g. by the address computer). This provides a good match to the relatively slow (expected) switching time of the spatial light modulators (SLMs) in the optical interconnection network, while still providing rapid optical data transfer through the FIER (by using, for example, high-bandwidth laser diode sources and detectors at the FIER inlets and outlets). While SLM switching time will likely improve substantially with progress in the component optical technnologies, it is interesting that corresponding improvements in control and communication algorithms may also be needed for the faster switching times to be fully reflected as proportional increases in computer performance.

8.1 Contributions

Contributions of the research presented herein are summarized below.

- **The FIER optical network:** Designed a novel optical interconnection network (the Free-space Interconnection with Externally-controlled Routing) which incorporates the following features.
  - ▷ Passive switching allows high speed optical data rates.
  - ▷ No buffering within the network, so hot spots cannot form.
  - ▷ Circuit switching allows more powerful routing and arbitration techniques.
  - ▷ The network is nonblocking; all requests are satisfied during each communication cycle.
  - ▷ Homosyndetic (identical) optical shuffle–exchange stages are fully cascadable.
  - ▷ RS-EGS network topology is flexible and well-suited to optical implementation.
  - ▷ Broadcast and combine switch settings in the FIER allow combining of requests within the network.
  - ▷ Fiber bundle interfaces to and from the FIER provide a flexible method for conversion of signals between the optical and electronic domains.
  - ▷ The FIER is a self-contained subsystem that is applicable to other computer and communication systems (other than the SMOEC).

- **The FLAEM routing algorithm:** Developed and simulated an algorithm for routing in a circuit-switched combining RS-EGS network (the Flexible Localized Algorithm for EGS-network Management) which provides the following advantages.

  - ▷ Combining of requests is supported (and required, unless permutation-restricted operation is enforced).
  - ▷ Communication algorithms support multiple PEs writing to and/or reading from the same MM.
  - ▷ Routing decisions are based solely on local data available within a switch during routing.
  - ▷ The FLAEM simulation illustrates excellent performance; it requires only a small constant ($\approx$2) number of "tries" to complete a full pattern of N individual routing requests.
  - ▷ The FLAEM satisfies a full set of N connection requests in $\mathcal{O}[\log N]$ time.
  - ▷ The FLAEM method has the potential for application to other related interconnection networks.

- **The SMOEC system architecture:** Presented a detailed exposition of a hybrid parallel computer architecture (the Shared Memory Optical/Electronic Computer) which was designed to achieve the following properties.
  - ▷ **Hybrid system:** The SMOEC system incorporates electronic processing and optical communication, to best exploit the strengths and accomodate the weaknesses of the two technologies.
  - ▷ **System-level design focus:** Throughout the development of the SMOEC system each of the three design aspects, architecture, hardware, and control, were allowed to impact the design of the other two aspects, to produce a balanced system.
  - ▷ **System-level design advocacy and support:** The SMOEC system illustrates the results of a strong emphasis on system-level design. A system-level focus is explicitly and implicitly advocated throughout the presentation.
  - ▷ **Shared memory architecture:** Parallel access to memory is provided at the hardware level.
  - ▷ **Flexibility:** The SMOEC may be tailored for particular applications or to fit specific hardware requirements.
  - ▷ **Realizability:** The SMOEC is implementable using current technology or future technology.
  - ▷ **Scalability:** The SMOEC architecture provides viable system designs for a wide range of system sizes.

- **Sample system recipes:** Nine detailed system recipes exemplify the realizability, scalability, and flexibility of the SMOEC system.
- **General-purpose MIMD operation:** The SMOEC is designed to run as a synchronous MIMD computer (it may also be designed for SIMD operation). It is suitable for a large class of applications which have structured (not random) patterns of memory access, such as physical simulations and image processing.

Appendix A

WOLLASTON PRISM ANALYSIS

QUESTION: For what range of angles are the angular offsets produced by a Wollaston prism (relatively) constant?

Recall Snell's law for refraction at an interface between two indices of refraction, n_i and n_r, where the angle of incidence is ϑ_i and the angle of refraction is ϑ_r:

$$n_i \sin \vartheta_i = n_r \sin \vartheta_r. \tag{A.1}$$

Recall that the two eigenindices of a birefringent crystal are known as n_o for the "ordinary ray," and n_e for the "extraordinary ray." Figure A.1 shows the effect of the Wollaston prism on orthogonally polarized rays (notated $\odot$ and $\updownarrow$). Each ray undergoes refraction at three surfaces as it passes through the Wollaston prism, and is affected by differing indices of refraction.

At the first surface, the ray A is aligned to propagate undeflected regardless of polarization. For the ray B, Snell's law holds for $\odot$. (In this discussion it is

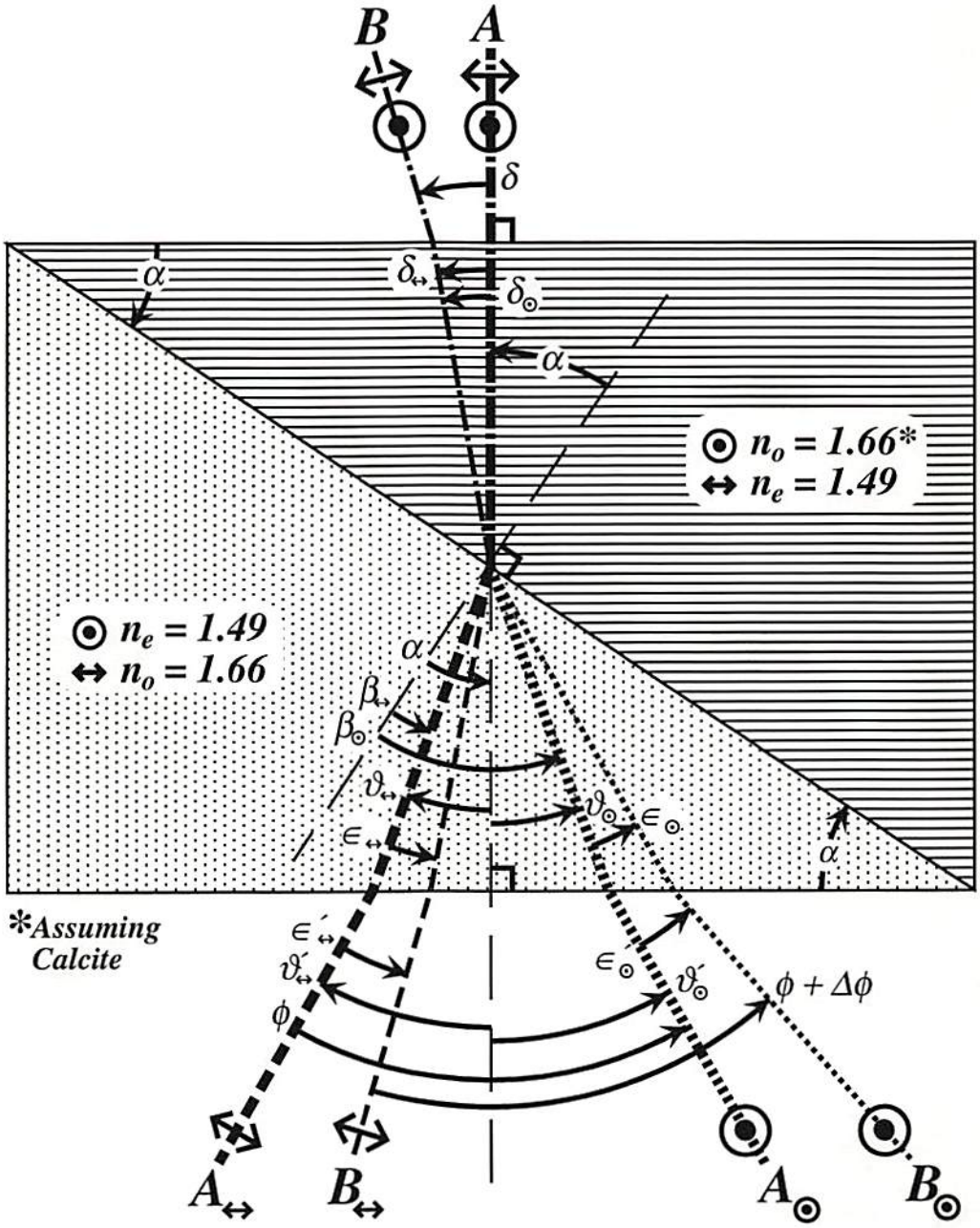


Figure A.1: The Wollaston Prism.

assumed that $n_o > n_e$, such as in calcite. An analogous calculation would hold for the opposite case.) For the case of $\updownarrow$, assume that Snell's law holds approximately for small angles δ, such that $n_r \approx n_e$. Then, for the incident angle δ, and the refracted angles $\delta_\odot$ and $\delta_\updownarrow$:

$$\begin{aligned} \odot: \quad & n_i = 1\,; \quad n_r = n_o \quad \Rightarrow \quad \sin\delta = n_o \sin\delta_\odot \\ \updownarrow: \quad & n_i = 1\,; \quad n_r \approx n_e \quad \Rightarrow \quad \sin\delta \approx n_e \sin\delta_\updownarrow \end{aligned} \tag{A.2}$$

For small δ, $\sin\delta \approx \delta$, therefore:

$$\begin{aligned} \odot: \quad & \delta \approx n_o\delta_\odot \quad \Rightarrow \quad \delta_\odot \approx \delta/n_o \\ \updownarrow: \quad & \delta \approx n_e\delta_\updownarrow \quad \Rightarrow \quad \delta_\updownarrow \approx \delta/n_e \end{aligned} \tag{A.3}$$

At the second surface in the Wollaston prism, the index difference is in opposite directions. Therefore, the refracted $\odot$ and $\updownarrow$ rays bend away from and toward the surface normal, respectively. This is what allows the Wollaston prism to accomplish the splitting of the two polarized rays. Due to the crystal axis orientation relative to this surface, the following index relations hold:

$$\begin{aligned} \odot: \quad & n_i = n_o\,; \quad n_r = n_e \\ \updownarrow: \quad & n_i \approx n_e\,; \quad n_r = n_o \end{aligned} \tag{A.4}$$

The second surface is at an angle α to the first surface. Therefore, ray A is incident to the second surface at angle α. For ray B, the incident angles to the second surface become $\alpha + \delta_\odot$ and $\alpha + \delta_\updownarrow$. So the relations at the second surface for the refracted rays A and B for both polarizations become (again using small angle approximations):

$$\begin{aligned} \odot: \quad & n_o\alpha \approx n_e\beta_\odot\,; \quad n_o(\alpha + \delta_\odot) \approx n_e(\beta_\odot + \epsilon_\odot) \\ \updownarrow: \quad & n_e\alpha \approx n_o\beta_\updownarrow\,; \quad n_e(\alpha + \delta_\updownarrow) \approx n_o(\beta_\updownarrow + \epsilon_\updownarrow) \end{aligned} \tag{A.5}$$

Combining equations A.3 and A.5 yields:

$$\begin{aligned} \odot: \quad & n_o\delta_\odot \approx n_e\epsilon_\odot; \quad \Rightarrow \quad \epsilon_\odot \approx \delta/n_e \\ \updownarrow: \quad & n_e\delta_\updownarrow \approx n_o\epsilon_\updownarrow; \quad \Rightarrow \quad \epsilon_\updownarrow \approx \delta/n_o \end{aligned} \tag{A.6}$$

At the third surface (the exit face) of the Wollaston prism, the rays exit into air. Thus the indices of refraction for all rays are now:

$$\begin{aligned} \odot: \quad & n_i = n_e; \quad n_r = 1 \\ \updownarrow: \quad & n_i = n_o; \quad n_r = 1 \end{aligned} \tag{A.7}$$

Define new (small) angles relative to the exit surface normal (note: all angles are treated as unsigned):

$$\begin{aligned} \odot: \quad & \vartheta_\odot \triangleq \alpha - \beta_\odot \\ \updownarrow: \quad & \vartheta_\updownarrow \triangleq \alpha - \beta_\updownarrow \end{aligned} \tag{A.8}$$

Using these angles, the relations for the refracted rays at the exit face become:

$$\begin{aligned} \odot: \quad & n_e\vartheta_\odot \approx \vartheta'_\odot; \quad n_e(\vartheta_\odot + \epsilon_\odot) \approx \vartheta'_\odot + \epsilon'_\odot \\ \updownarrow: \quad & n_o\vartheta_\updownarrow \approx \vartheta'_\updownarrow; \quad n_o(\vartheta_\updownarrow - \epsilon_\updownarrow) \approx \vartheta'_\updownarrow - \epsilon'_\updownarrow \end{aligned} \tag{A.9}$$

Combining equations A.6 and A.9 yields:

$$\begin{aligned} \odot: \quad & n_e\epsilon_\odot \approx \epsilon'_\odot \quad \Rightarrow \quad \epsilon'_\odot \approx \delta \\ \updownarrow: \quad & n_o\epsilon_\updownarrow \approx \epsilon'_\updownarrow \quad \Rightarrow \quad \epsilon'_\updownarrow \approx \delta \end{aligned} \tag{A.10}$$

Define ϕ to be the angle between the exit rays $A_\odot$ and $A_\updownarrow$. Further, define $\phi+\Delta\phi$ to be the angle between the exit rays $B_\odot$ and $B_\updownarrow$. Then $\Delta\phi$ is the difference in the angular offsets produced by the Wollaston prism. Note that $\Delta\phi = \epsilon'_\updownarrow - \epsilon'_\odot$. Since equation A.10 implies that $\epsilon'_\updownarrow \approx \epsilon'_\odot$, one can conclude that $\Delta\phi \approx 0$. Therefore the Wollaston prism will produce relatively constant offsets as long as the small angle approximation is valid for all angles involved.

Appendix B

The Sim-FLAEM PROGRAM

The C program implementation of the Basic-FLAEM algorithm is listed here. Notes about aspects of the program appear below and as comments within the program listing itself.

B.1 Sim-FLAEM C Program Notes

The following sim-FLAEM C program listing contains all the major routines to implement the Basic-FLAEM routing procedure. However, some simpler routines have not been included in the listing, and are summarized here. These functions are described briefly in Table B.1.

The state variables used in the C program below differ slightly from those explained elsewhere. They were changed for the text to add clarity to the algorithm presentation. Table B.2 lists the correspondence between the state variables as mentioned in the text and as used in the C program:

Function	*Purpose*
`init_utils()`	*Initialize various utility functions*
`set_EGS_params()`	*Assigns values to the fundamental EGS network parameters N, F, S_S, etc.*
`init_data_files()`	*Initialize the data files for output of routing traces and summary reports*
`clear_node()`	*Reinitialize a node, setting its state variable* (`EGS[][].cnd`) *to* `FREE`
`form_path_vector()`	*Compose a path vector from a given inlet index, path number, and outlet index*
`set_permuted_pri()`	*Assigns a permutation of the numbers from* 0 *to* $F-1$ *to the priorities of the F request copies during the "fan-out" into the W-wide EGS array*
`pull_link()`	*Extracts a link number from a path vector, given the path vector and the stage number of the link*
`propagate()`	*Copies all the parameters associated with a request copy at a link in a given stage to the next specified link in the next stage*
`path_surgery()`	*Alter the path vector to use an different path; used in a flexible state during modpath*
`exch_msb()`	*Invert the Most Significant Bit (MSB) of a link address*
`report_final_summary()`	*Produces a report of the number of patterns that took varying numbers of tries.*
`print_`*various*`_line()`	*Used to generate traces of the various routing passes for the purposes of program verification and illustration*

Table B.1: Summaries of FLAEM simulation C program routines.

Discussions in the text	`C program variables`
THRU	`NSAT`
FIXED	`SATS`
RUN	`PROG`
FREE	`FREE`
CONFLICT	`CONF`
COMBINED	`COMB`

Table B.2: State variable correspondences.

B.2 Sim-FLAEM Program Correctness

When using a computer simulation to evaluate the performance of an algorithm, it is essential that the program correctness be verified. The simulation must be verified for reasonable data sets so that the simulation results may be trusted. The Sim-FLAEM program was designed to produce data which trace the routing process from start to completion. Trace data were generated for minimal restricted-F RS-EGS networks for $n = 4$ through $n = 7$ ($N = 16$ through $N = 128$) and a non-minimal case for $n = 3$. Alghough the data traces were extremely large, it was necessary to product trace data for networks larger than $n = 5$ because the larger networks have more complexities involved in the routing procedure, such as multiple flexible stages. Therefore a full range of networks large enough to display sufficiently general routing characteristics were verified by hand. Sample (small) data output from the sim-FLAEM program are provided in the next section for the $n = 3$, $F = 4$, $S_S = 3$ example that was illustrated in Section 5.2.3. The `print_`*various*`_line()` routines mentioned in the preceding section were used to generate these output traces line by line. Data such as these were used to verify program correctness.

B.3 Sim-FLAEM Trace Data Sample Output

```
EGS Network:  (n=2) N=4  F=4  Ss=3 (St=5) -- Custom (non-minimal) Parameter set
Additionally: (p=3) P=8 (w=4) W=16  pv=7     nms = 1 = # of flexible stages
7-bit Path Vector: (xxffpyy)

st=Stage#, SRC=Source PE#, Link=Link index, pri=Priority, DEST=Destination PE#,
comb = the request copy was combined with another in the previous stage
next = link in next stage to be used by the request copy (from path vector).

m => modpath (randomly-generated path vector altered to avoid a conflict --
     used in flexible stages only)
# => conflict (see "with" line for PE# it conflicted with)
@ => combined (see "with" line for PE# it combined with)
V => combined in previous stage (used for reverse propagation bookkeeping)
! => marker indicating no special data in a request copy
| => satisfied (SATS) connection

=====================================================================
RANDOMLY-GENERATED CONNECTION PATTERN:
   00   01   02   03
   to   to   to   to
   02   03   01   03

>>>>> Try #1:
st=01_____________________________________________________________
Link 000 001 002 003 004 005 006 007 008 009 010 011 012 013 014 015
 SRC  00  00  00  00  01  01  01  01  02  02  02  02  03  03  03  03
 pri  2!  0!  1!  3!  2!  0!  1!  3!  0!  1!  3!  2!  2!  0@  1!  3!
with   !   !   !   !   !   !   !   !  m!   !   !   !   !  01   !   !
next 000 003 005 006 009 011 012 015 001 002 004 007 008 011 013 014
st=02_____________________________________________________________
Link 000 001 002 003 004 005 006 007 008 009 010 011 012 013 014 015
 SRC  00  02  02  00  02  00  00  02  03  01  ==  01  01  03  03  01
comb   !   !   !   !   !   !   !   !   !   !   .   V   !   !   !   !
 pri  2!  0!  1!  0!  3!  1!  3!  2!  2#  2!   .  0#  1!  1#  3#  3!
with   !   !   !   !   !   !   !   !  00   !   .  00   !  00  00   !
next 001 002 004 007 008 011 013 014 001 003   . 007 009 011 013 015
st=03_____________________________________________________________
Link 000 001 002 003 004 005 006 007 008 009 010 011 012 013 014 015
 SRC  ==  00  02  01  02  ==  ==  00  02  01  ==  00  ==  00  02  01
comb   .   !   !   !   !   .   .   !   !   !   .   !   .   !   !   !
 pri   .  2!  0!  2!  1!   .   .  0!  3!  1!   .  1!   .  3!  2!  3!
with   .   !   !   !   !   .   .   !   !   !   .   !   .   !   !   !
next   . 002 005 007 009   .   . 014 001 003   . 006   . 010 013 015
st=04_____________________________________________________________
Link 000 001 002 003 004 005 006 007 008 009 010 011 012 013 014 015
 SRC  ==  02  00  01  ==  02  00  01  ==  02  00  ==  ==  02  00  01
comb   .   !   !   !   .   !   !   !   .   !   !   .   .   !   !   !
 pri   .  3!  2!  1!   .  0!  1!  2!   .  1!  3!   .   .  2!  0!  3!
         !   !   !       !   !   !       !   !           !   !   !
DEST     [01][02][03]    [01][02][03]    [01][02]        [01][02][03]
```

```
>>>>> Try #2:
st=01_____________________________________________________________________
Link 000 001 002 003 004 005 006 007 008 009 010 011 012 013 014 015
 SRC  ==  ==  ==  00  ==  ==  ==  01  ==  ==  02  ==  03  03  03  03
 pri   .   .   .   |   .   .   .   |   .   .   |   .  1!  2!  0!  3@
with   .   .   .   |   .   .   .   |   .   .   |   .   !   !   !  01
next   .   .   . 006   .   .   . 015   .   . 004   . 009 011 012 015
st=02_____________________________________________________________________
Link 000 001 002 003 004 005 006 007 008 009 010 011 012 013 014 015
 SRC  ==  ==  ==  ==  02  ==  00  ==  ==  03  ==  03  03  ==  ==  01
comb   .   .   .   .   |   .   |   .   .   !   .   !   !   .   .   V
 pri   .   .   .   .   |   .   |   .   . 1!   . 2!  0!   .   .   |
with   .   .   .   .   |   .   |   .   .   !   .   !   !   .   .   |
next   .   .   .   . 008   . 013   .   . 003   . 007 009   .   . 015
st=03_____________________________________________________________________
Link 000 001 002 003 004 005 006 007 008 009 010 011 012 013 014 015
 SRC  ==  ==  ==  03  ==  ==  ==  03  02  03  ==  ==  ==  00  ==  01
comb   .   .   .   !   .   .   .   !   |   !   .   .   .   |   .   |
 pri   .   .   . 1!   .   .   . 2@   |  0!   .   .   .   |   .   |
with   .   .   .   !   .   .   .  01   |   !   .   .   .   |   .   |
next   .   .   . 007   .   .   . 015 001 003   .   .   . 010   . 015
st=04_____________________________________________________________________
Link 000 001 002 003 004 005 006 007 008 009 010 011 012 013 014 015
 SRC  ==  02  ==  03  ==  ==  ==  03  ==  ==  00  ==  ==  ==  ==  01
comb   .   |   .   !   .   .   .   !   .   .   |   .   .   .   .   V
 pri   .   |   .  0!   .   .   . 1!   .   .   |   .   .   .   .   |
           |       !                   !       |                   |
DEST      [01]    [03]                [03]    [02]                [03]

===== Pattern SATISFIED in 2 tries. =====

>>>>> FINAL VERSION of Pattern Settings
st=01_____________________________________________________________________
Link 000 001 002 003 004 005 006 007 008 009 010 011 012 013 014 015
 SRC  ==  ==  ==  00  ==  ==  ==  01  ==  ==  02  ==  ==  ==  ==  03
 pri   .   .   .   |   .   .   .   |   .   .   |   .   .   .   .   |
with   .   .   .   |   .   .   .   |   .   .   |   .   .   .   .   |
next   .   .   . 006   .   .   . 015   .   . 004   .   .   .   . 015
st=02_____________________________________________________________________
Link 000 001 002 003 004 005 006 007 008 009 010 011 012 013 014 015
 SRC  ==  ==  ==  ==  02  ==  00  ==  ==  ==  ==  ==  ==  ==  ==  01
comb   .   .   .   .   |   .   |   .   .   .   .   .   .   .   .   V
 pri   .   .   .   .   |   .   |   .   .   .   .   .   .   .   .   |
with   .   .   .   .   |   .   |   .   .   .   .   .   .   .   .   |
next   .   .   .   . 008   . 013   .   .   .   .   .   .   .   . 015
st=03_____________________________________________________________________
Link 000 001 002 003 004 005 006 007 008 009 010 011 012 013 014 015
 SRC  ==  ==  ==  ==  ==  ==  ==  ==  02  ==  ==  ==  ==  00  ==  01
comb   .   .   .   .   .   .   .   .   |   .   .   .   .   |   .   |
 pri   .   .   .   .   .   .   .   .   |   .   .   .   .   |   .   |
with   .   .   .   .   .   .   .   .   |   .   .   .   .   |   .   |
next   .   .   .   .   .   .   .   . 001   .   .   .   . 010   . 015
st=04_____________________________________________________________________
Link 000 001 002 003 004 005 006 007 008 009 010 011 012 013 014 015
 SRC  ==  02  ==  ==  ==  ==  ==  ==  ==  ==  00  ==  ==  ==  ==  01
comb   .   |   .   .   .   .   .   .   .   .   |   .   .   .   .   |
 pri   .   |   .   .   .   .   .   .   .   .   |   .   .   .   .   |
           |                                   |                   |
DEST      [01]                                [02]                [03]
```

B.4 Sim-FLAEM C Program Listing

```
#include "rt-hdr.h"

/* CONDITIONS: (used in PE[].cnd and EGS[][].cnd) */
#define NSAT      3 /* Newly Satisfied link, can use: [NSAT,SATS]≥SATS */
#define SATS      2 /* Link is complete, has a dest. (>0) */
#define PROG      1 /* Link is being routed, has a dest. (>0) */
#define FREE      0 /* Link is unused (Other fields may not be MT) */
#define CONF     -1 /* Link aborted due to conflict (<0) */
#define COMB     -2 /* Link has been combined, solo progress aborted. (<0) */
                    /* NOTE: aborted conditions above have flags<0. */
                    /* and conditions with a dest have flags>0. */

#define MIN_PRI -1 /* Minimum Unused Priority number (used in PE[].pri) */
#define MODDED  -1 /* Negative .wth field indicates modpath was used (or NO) */

#define SKIP     -1 /* For variable do_combs */
#define PERMS     0
#define COMBS     1

#define nmax                12
#define Nmax          (1<<nmax)  /* = 2^nmax */
#define STmax        (2*nmax-1)
#define Fmax                16
#define Wmax         (Nmax*Fmax)
#define MAX_TRIES           20
```

```
/* Fundamental set of EGS "constants" */
long  int N;    /* N = Number of PEs, MMs */
long  int F;    /* F = Fan-in, Fan-out */
long  int W;    /* W = N · F = Number of links in "Main Section" */
                /* of EGS network. */
short int n;    /* log2(N) = Number of bits in PE# (1...N) */
short int f;    /* log2(F) = Number of bits in Fans (1...F) */
short int w;    /* log2(W) = Number of bits in Link# (1...W) */
short int Ss;   /* What the richards_fan_table() uses. (#stgs w/o fans) */
short int St;   /* ST, where ST = SS + 2 */
                /* (fanin,fanout ⇒ 2 stages) */
short int Sf;   /* ST-1 = The final used "main" stage, */
                /* (final fanout stg = ST) */
long  int P;    /* P = F · (2^SS)/N = Number of paths */
short int p;    /* log2(P) = Number of bits in path only */
short int pv;   /* n + p + n = length (bits) of full xxxppyyy path vector */
short int nms;  /* n + p − w = # of stages after fan-in (=stg#1) that */
                /* contain modifiable path vectors; */
                /* i.e. whose desired link address ends in "p" instead */
                /* of "y" when extracted from the path vector. */
                /* Can show in addition, that nms = p − f (using w = n + f) */
                /* nms = Number of Modifiable Stages */

typedef char Logical;  /* use type char to hold Logical values */
typedef char Tiny;     /* use type char to hold tiny integer flag values */
Logical modpath;       /* Is path vector modification on-the-fly to be employed? */
Logical verbose;       /* 0 ⇒no routing display info, else verbose=1 */
Logical do_combs;      /* 1 ⇒combinations, 0 ⇒permutations */
int num_patts;         /* # of perms or combinations to be performed */

#include "rt-bitmanip.h"

struct PE_struct {
  Tiny      cnd;   /* CoNDition */
  short int dst;   /* DeSTintation MM# */
  short int pri;   /* PRIority — used to pick which successful conn is NSAT */
} PE[Nmax];
struct EGS_struct {
  Tiny      cnd;   /* CoNDition */
  Tiny      pri;   /* PRIority (lo value ⇒hi priority) */
  Logical   cmb;   /* Prev stg had CoMBined conn. */
  pv_type   pvc;   /* Path VeCtor */
  short int src;   /* SouRCe PE# (extra var. — for execution speed) */
  short int dst;   /* DeSTintation MM# (extra var. — for execution speed) */
  short int wth;   /* PE# Comb'd or Conf'd WiTH; or if PROG, YES ⇒modpath */
} EGS[STmax][Wmax];   /* NOTE: stg. st=0 is not used, nor is st=ST */
short int conn_tries[MAX_TRIES+1];
```

```
#include "rt-util.h"
#include "rt-setup-EGS.h"
#include "rt-fcns-EGS.h"

FILE *run,*out,*trc;
char trcfilename[30],runfilename[30],runcmd[50];
#include "rt-display.h"

void main(void)
{
long  int pn;          /* pn = PE# */
long  int st,sp,sn;    /* st = Stage#, sp = prev Stage#, sn = next Stage# */
long  int lnk,eln;     /* lnk = Link#, eln = exch(lnk), */
long  int pln,uln;     /* pln = Link# in prev stage, uln = Link# for undoing */
long  int nln;         /* nln = Link# in next stage */

short int conflicts_exist;   /* Number of conflicts encountered in a conn try */
short int conn_try,patt_num; /* Connection try index, Pattern attempt index */
short int mod_stgs_left;     /* Count modifiable stages; For example: */
                             /* mod_stgs_left==2 ⇒stages 2 & 3 are mod'able */
short int fo;                /* Counter: multiple Fan-Out copies to stage #1 */
long int path,rmn_path;
Logical  prev_wins;

  init_utils();
  set_EGS_params();
  init_data_files();

  for ( patt_num=1 ; patt_num<=num_patts ; patt_num++ ) {

    /* Choose a set of desired PE-MM connections to be performed */
    if      (do_combs==COMBS) assign_rnd_conns();
    else if (do_combs==PERMS) assign_rnd_perms();
    if (verbose) print_reqd_conns(trc);

    /* Initialize the EGS array info */
    for(st=1;st<St;st++) for(lnk=0;lnk<W;lnk++) clear_node(st,lnk);

    conn_try = 0; /* number of tries to make a particular set of connections */
    do {          /* LOOP (Counting: conn_try) until there are no conflicts */
      conn_try++;
      if (conn_try > MAX_TRIES) max_tries_err_exit(patt_num);
      if (verbose) fprintf(trc,"\n>>>>> Pattern #%d, Try #%d\n",
                           patt_num, conn_try);
```

```
/* Do the "fan-out" into the W-wide EGS array from the PE req's */
if (p>f) rmn_path = two_to(nms); /* using nms = p-f, where */
else     rmn_path = 0;           /* p-f = num of remaining unset path */
for( pn=0 ; pn<N ; pn++ ) {      /* bits after the input Fan-Out */
  if (PE[pn].cnd==SATS) continue;
  for( fo=0 ; fo<F ; fo++ ) {
    lnk = pn*F + fo;
    if (rmn_path>0) path = (fo<<nms) + rnd(rmn_path);
    else            path =  fo;
    EGS[1][lnk].src = pn;
    EGS[1][lnk].dst = PE[pn].dst;
    form_path_vector( &EGS[1][lnk].pvc , pn , path , PE[pn].dst );
    EGS[1][lnk].cnd = PROG;
    EGS[1][lnk].wth = NO;
    EGS[1][lnk].cmb = NO;
  }
  set_permuted_pri(pn);
}

if (verbose) {
  print_link_line(1);
  print_src_line(1);
}

/* MAIN ROUTING PROCESSING LOOPS */
for( st=2,sp=1,mod_stgs_left=nms ; st<St ; st++,sp++,mod_stgs_left-- ) {

  for( pln=0 ; pln<W ; pln++ ) {
    if (EGS[sp][pln].cnd==SATS) continue; /* continue to next for:pln */
    if (EGS[sp][pln].cnd==FREE) continue; /* continue to next for:pln */
    lnk = pull_link(EGS[sp][pln].pvc,st); /* lnk=link# req'd this stg. */

    if (EGS[st][lnk].cnd==FREE) {
      propagate(sp,pln,st,lnk);
      continue; /* continue to next for:pln */
    }

    /* Here, assume busy current node EGS[st][lnk] ==PROG or ==SATS */
    if (EGS[sp][pln].dst==EGS[st][lnk].dst) { /* COMBINE */
      EGS[sp][pln].cnd = COMB;
      EGS[sp][pln].wth = EGS[st][lnk].src;
      EGS[st][lnk].cmb = YES;
      EGS[st][lnk].pri = Min( EGS[sp][pln].pri , EGS[st][lnk].pri );
      continue; /* to next for:pln */
    } /*endif*/
```

```
/* Now, assume busy current node, and incompatible dests */
if ( modpath && (mod_stgs_left>0) ) { /* MOD PATH VECTORS */
  eln = exch(lnk);
  if (EGS[st][eln].cnd==FREE) {
    path_surgery(&EGS[sp][pln].pvc,st); /* To: */
                        /* change the pvc in prev stage sp to */
                        /* point to eln in current stage st. */
    propagate(sp,pln,st,eln);
    EGS[sp][pln].wth = MODDED;
    continue; /* continue to next for:pln */
  } else if (EGS[st][eln].dst==EGS[sp][pln].dst) {
                                /* COMBINE-and-MODPATH */
    path_surgery(&EGS[sp][pln].pvc,st); /* To: */
                        /* change the pvc in prev stage sp to */
                        /* point to eln in current stage st. */
    EGS[sp][pln].cnd = COMB;
    EGS[sp][pln].wth = -EGS[st][eln].src;  /* neg to show modpath */
    EGS[st][eln].cmb = YES;
    EGS[st][eln].pri = Min( EGS[sp][pln].pri , EGS[st][eln].pri );
    continue; /* to next for:pln */
  }
} /*endif*/

                                                    /* CONFLICT */
/* Now, must assume that there is an unavoidable conflict */
if              (EGS[st][lnk].cnd==SATS) prev_wins = NO;
else {/* Assume (EGS[sp][pln].cnd==PROG) */
  if (EGS[sp][pln].pri==EGS[st][lnk].pri) prev_wins = rnd(2);
  else prev_wins = (EGS[sp][pln].pri < EGS[st][lnk].pri);
}
if (prev_wins) {
  uln = pull_link(EGS[st][lnk].pvc,sp);/* get Link# for the conn */
                                       /* leading to current link */
  EGS[sp][uln].cnd = CONF;
  EGS[sp][uln].wth = EGS[sp][pln].src;
  propagate(sp,pln,st,lnk);
  continue; /* continue to next for:pln */
} else {  /* so current wins */
  EGS[sp][pln].cnd = CONF;
  EGS[sp][pln].wth = EGS[st][lnk].src;
  continue; /* continue to next for:pln */
}

} /*endfor:pln*/
```

```
  /*************************************************************************
  ** Now, if this is a modpath stage, step through current links to see
  ** if any requests should be split and copied (or even combined!)
  ** with the adjacent exch node, at a new priority. if verbose,
  ** label each split/copy in the output stage listings.
  **
  **                    NOT IMPLEMENTED
  **
  *************************************************************************
  */

  if (verbose) {     /*— DISPLAY DATA TRACE Section —*/
    print_pri_line(sp);     /* These functions print data from the */
    print_with_line(sp);    /* EGS[][] array, producing a trace of */
    print_next_line(sp);    /* the connection routing process. */
    print_link_line(st);
    print_src_line(st);
    print_comb_line(st);
  }

} /*endfor:st*/

if (verbose) { /*— FINAL DATA TRACE DISPLAY Section —*/
  print_pri_line(Sf);
  print_dest_line(Sf);
}

/* This is the backward trace, where satisfied conns are set to
** EGS[][].cnd=SATS; while untree-ing any combined successful paths
** and calling clear_node for any unsuccessful paths.
**
** Mark new PROG nodes in final stage as NSAT: for each new conn, mark:
** PE[].cnd=NSAT and EGS[][].cnd=NSAT.
** Go thru each stage (st) in reverse order, converting or clearing nodes
** link by link (lnk). Check EGS[][].cmb for add'l connections to be
** marked SATS. Mark PE[].cnd=NSAT for new COMB'd conns.
*/
for ( lnk=0 ; lnk<W ; lnk++ ) {
  if (EGS[Sf][lnk].cnd!=PROG) continue;
  pn = EGS[Sf][lnk].src;
  PE[pn].cnd       = NSAT;
  EGS[Sf][lnk].cnd = NSAT;
}
```

```
/* Now propagate satisfied conns backward thru EGS array */
/* noting that COMBs can cause new PE#s to be NSAT */
for ( st=Sf-1,sn=Sf ; st>=1 ; st--,sn-- ) {
  for ( lnk=0 ; lnk<W ; lnk++ ) {
    switch (EGS[st][lnk].cnd) {
      case SATS:
      case FREE:  break;
      case CONF:  clear_node(st,lnk);  break;
      case PROG:  /* Check if node in next stage is SATS or NSAT */
                  nln = pull_link(EGS[st][lnk].pvc,sn);
                  if (EGS[sn][nln].cnd<SATS) clear_node(st,lnk);
                  else                       EGS[st][lnk].cnd = NSAT;
                  break;
      case COMB:  /* Check if node in next stage is SATS or NSAT */
                  nln = pull_link(EGS[st][lnk].pvc,sn);
                  if (EGS[sn][nln].cnd<SATS) clear_node(st,lnk);
                  else {
                    pn = EGS[st][lnk].src;
                    PE[pn].cnd       = NSAT;
                    EGS[st][lnk].cnd = NSAT;
                  }
                  break;
      default:    printf("Bizarre error in backprop\n"); exit(1);
    } /*endswitch(EGS[][].cnd)*/
  } /*endfor:lnk*/
} /*endfor:st*/
```

```
/* (1) Loop across first stage st=1 and record MaxPri for EGS.cnd==NSAT
** nodes (skipping SATS and FREE links), saving it in PE[].pri
** (2) Loop across first stage to find the first node in which
** EGS.pri==PE.pri, and mark it with with: PE.cnd=SATS & EGS.cnd=SATS.
** So after Loops (1) & (2) have single "best" (MaxPri) successful
** links marked SATS each newly successful (PE[].cnd==NSAT) connection.
** (3) Loop across first stage to clear superfluous remaining NSAT
** marked nodes.
*/
for (pn=0;pn<N;pn++) if (PE[pn].cnd==NSAT) PE[pn].pri = MIN_PRI;
for ( lnk=0 ; lnk<W ; lnk++ ) {  /* Loop (1) */
  if (EGS[1][lnk].cnd!=NSAT) continue;
  pn = EGS[1][lnk].src;
  PE[pn].pri = Max( EGS[1][lnk].pri , PE[pn].pri );
  EGS[1][lnk].cnd = NSAT;
}
for ( lnk=0 ; lnk<W ; lnk++ ) {  /* Loop (2) */
  pn = EGS[1][lnk].src;
  if (PE[pn].cnd!=NSAT)      continue;
  if (EGS[1][lnk].cnd!=NSAT) continue;
  if (EGS[1][lnk].pri==PE[pn].pri) {  /* Only mark the first time that */
    PE[pn].cnd      = SATS;           /* equal .pri's are encountered. */
    EGS[1][lnk].cnd = SATS;
  }
}
for ( lnk=0 ; lnk<W ; lnk++ ) {  /* Loop (3) */
  if (EGS[1][lnk].cnd==NSAT) clear_node(1,lnk);
}
```

```
  conflicts_exist = 0;  /* Initialize to "no conflicts" */
  /* LOOP THROUGH PEs TO SEE IF ALL ARE PE[].cnd==SATS */
  for (pn=0;pn<N;pn++) if (PE[pn].cnd!=SATS) conflicts_exist++;
  /* NOTE(†): To get the link number for the two nodes that were combined
  ** into node "lnk", have: one is "pln" the usual way. The other is
  ** found by:
  ** > EGS[st][lnk].pvc = (xxxxffpyyy)
  ** > bcdefg  Link# at stage st=3
  ** > abcdef  Link# at stage sp=2
  ** If pln==abcdef then the other combined-in link in stage sp is:
  ** eln=(~a)bcdef
  */
  /* Clean up and resolve NSAT stuff */
  for( st=2,sp=1 ; st<St ; st++,sp++ ) {
    for( lnk=0 ; lnk<W ; lnk++ ) {
      if (EGS[st][lnk].cnd==NSAT) {
        pln=pull_link(EGS[st][lnk].pvc,sp); /* pln=link# req'd prev stg. */
        if (EGS[st][lnk].cmb) {  /* If Node was COMB'd from 2 prev nodes */
          eln = exch_msb(pln); /* See NOTE(†) above. */
          if ( (EGS[sp][pln].cnd==SATS) || (EGS[sp][eln].cnd==SATS) ) {
            EGS[st][lnk].cnd = SATS;
            EGS[st][lnk].cmb = ( (EGS[sp][pln].cnd==SATS)
                              && (EGS[sp][eln].cnd==SATS) );
            if (EGS[sp][eln].cnd==SATS) {
              EGS[st][lnk].src = EGS[sp][eln].src;
              EGS[st][lnk].pvc = EGS[sp][eln].pvc;
            } else {
              EGS[st][lnk].src = EGS[sp][pln].src;
              EGS[st][lnk].pvc = EGS[sp][pln].pvc;
            }
          } else clear_node(st,lnk);
        } else {
          if (EGS[sp][pln].cnd==SATS) {
            EGS[st][lnk].cnd = SATS;
            EGS[st][lnk].src = EGS[sp][pln].src; /* Ensure same src label*/
            EGS[st][lnk].pvc = EGS[sp][pln].pvc; /* and same pvc field*/
          } else clear_node(st,lnk);
        }
      }
    }
  }

  if (verbose) {
    if (conflicts_exist==0) fprintf(trc,"Pattern #%d SATISFIED in %d %s.\n",
                  patt_num, conn_try, ((conn_try==1) ? "try" : "tries") );
    fprintf(trc,"\f");
  }

} while (conflicts_exist>0);
```

```
    if (verbose) {
      fprintf(trc,"FINAL VERSION of Pattern #%d\n",patt_num);
      print_link_line(1);
      print_src_line(1);
      for( st=2,sp=1 ; st<St ; st++,sp++ ) {
        print_pri_line(sp);
        print_with_line(sp);
        print_next_line(sp);
        print_link_line(st);
        print_src_line(st);
        print_comb_line(st);
      }
      print_pri_line(Sf);
      print_dest_line(Sf);
    }

    conn_tries[conn_try]++;

  } /*endfor:patt_num*/

  report_final_summary();

} /*end-function-main*/
```

Bibliography

[Abdennadher 92] A. Abdennadher and T.-Y. Feng, "On rearrangeability of omega-omega networks," in *Proc. 1992 Int. Conf. on Parallel Processing*, pp. I159-I165, Aug. 1992.

[Barakat 87] R. Barakat and J. Reif, "Lower bounds on the computational efficiency of optical computing systems," Appl. Opt., Vol. 26, No. 6, pp. 1015-1018, Mar. 1987.

[Beetem 85] J. Beetem, M. Denneau, and D. Weingarten, "The GF11 supercomputer," in *Proc. 12th Ann. Int. Symp. on Computer Architecture* (IEEE Computer Society Press, Washington, DC), pp. 108-115, June 1985.

[Borodin 85] A. Borodin and J. E. Hopcroft, "Routing, merging, and sorting on parallel models of computation," J. Comput. and System Sciences, Vol. 30, pp. 130-145, 1985.

[Brantley 85] W. C. Brantley, K. P. McAuliffe, J. Weiss, "RP3 processor-memory element," in *Proc. 1985 Int. Conf. on Parallel Processing* (IEEE Computer Society Press, Washington, DC), pp. 782-789, Aug. 1985.

[Brenner 88] K.-H. Brenner and A. Huang, "Optical implementations of the perfect shuffle interconnection," Appl. Opt., Vol. 27, No. 1, pp. 135-137, Jan. 1988.

[Butler 93] F. Butler, H. CHen, J. Sexton, A. Vaccarino, and D. Weingarten, "Hadron mass predictions of the valence approximation to QCD," Phys. Rev. Lett., Vol. 70, No. 19, pp. 2849-2852, May 1993.

[Cheng 92] L. Cheng and A. A. Sawchuk, "Three-dimensional omega networks for optical implementation," Appl. Opt., Vol. 31, No. 26, pp. 5468-5479, Sept. 1992.

[Cloonan 92] T. J. Cloonan, G. W. Richards, A. L. Lentine, F. B. McCormick, and J. R. Erickson, "Architectural issues related to the optical implementation of an EGS network based on embedded control," Opt. and Quant. Elect. Vol. 24, No. 4, pp. S415-S412, Apr. 1992.

[Cloonan 94] T. J. Cloonan, G. W. Richards, R. L. Morrison, A. L. Lentine, J. M. Sasian, F. B. McCormick, S. J. Hinterlong, and H. S. Hinton, "Shuffle-equivalent interconnection topologies based on computer-generated binary-phase gratings," Appl. Opt., Vol. 33, No. 8, pp. 1405-1430, Mar. 1994.

[Darema 86] F. Darema-Rogers, G. F. Pfister, and K. So, "Memory access patterns of parallel scientific programs," IBM Research Report RC12086 (IBM T. J. Watson Research Center, Yorktown Heights, NY), July 1986.

[Feldman 88] M. R. Feldman, S. C. Esener, C. C. Guest, and S. H. Lee, "Comparison between optical and electrical interconnects based on power and speed considerations," Appl. Opt., Vol. 27, No. 9, pp. 1742-1751, May 1988.

[Feng 74] T.-Y. Feng, "Data manipulating functions in parallel processors and their implementations," IEEE Trans. Comput., Vol. C-23, pp. 309-318, Mar. 1974.

[Fortune 78] S. Fortune and J. Wyllie, "Parallelism in random access machines," in *Proc. 10th Ann. ACM Symposium on Theory of Computing* (Association for Computing Machinery, NY), pp. 114-118, 1978.

[Freeman 92] M. O. Freeman, T. A. Brown, D. M. Walba, "Quantized complex ferroelectric liquid crystal spatial light modulators," Appl. Opt., Vol. 31, No. 20, pp. 3917-3929, July 1992.

[Giles 86] C. L. Giles and B. K. Jenkins, "Complexity implications of optical parallel computing," in *Proc. Twentieth Annual Asilomar Conference on Signals, Systems, and Computers*, pp. 513-517, Nov. 1986.

[Goodman 84] J. W. Goodman, F. I. Leonberger, S.-Y. Kung, R. A. Athale, "Optical interconnections for VLSI systems," Proc. IEEE, Vol. 72, No. 7, pp. 850-866, July 1984.

[Gottlieb 83a] A. Gottlieb, R. Grishman, C. P. Kruskal, K. P. McAuliffe, L. Rudolph, and M. Snir, "The NYU Ultracomputer – designing an MIMD shared memory parallel computer," IEEE Trans. Comput., Vol. C-32, No. 2, pp. 179-183, Feb. 1983.

[Gottlieb 83b] A. Gottlieb, B. D. Lubachevsky, and L. Rudolph, "Basic techniques for the efficient coordination of very large numbers of cooperating sequential processors," ACM Trans. Prog. Lang. Syst., Vol. 5 No. 2, pp. 165-189, April 1983.

[Guha 90] A. Guha, J. Bristow, C. Sullivan, and A. Husain, "Optical interconnections for massively parallel architectures," Appl. Opt., Vol 29, No. 8, pp. 1077-1092, Mar. 1990.

[Handschy 87] M. A. Handschy, K. M. Johnson, W. T. Cathey, and L. A. Pagano-Stauffer, "Polarization-based optical parallel logic gate utlilizing ferroelectric liquid crystals," Opt. Lett., Vol. 12, No. 8, pp. 611-613, Aug. 1987.

[Hartman 86] D. H. Hartman, "Digital high speed interconnects: a study of the optical alternative," Opt. Eng., Vol. 25, No. 10, pp. 1086-1102, Oct. 1986.

[Hinton 93a] H. S. Hinton, J. R. Erickson, T. J. Cloonan, G. W. Richards, "Space-division switching," in *Photonics in Switching*, J. E. Midwinter, ed. (Academic Press, Boston MA) Vol. II, pp. 119-167, 1993.

[Hinton 93b] H.S. Hinton, J. R. Erickson, T. J. Cloonan, F. A. P. Tooley, F. B. McCormick, and A. L. Lentine, *An Introduction to Photonic Switching Fabrics*, (Plenum Press, New York NY) 1993.

[Hwang 93] K. Hwang, *Advanced Computer Architecture: Parallelism, Scalability, Programmability,* (McGraw-Hill, New York NY) 1993.

[Jahns 88] J. Jahns and M. J. Murdocca, "Crossover networks and their optical implementation," Appl. Opt., Vol. 27, No. 15, pp. 3155-3160, Aug. 1988.

[Jenkins 86] B. K. Jenkins and C. L. Giles, "Parallel processing paradigms and optical computing," in *Proc SPIE Vol. 625 Optical Computing*, pp. 22-29, Jan. 1986.

[Jenkins 88] B. K. Jenkins and C. L. Giles, "Superposition in optical computing," *Proc. SPIE Vol. 963, ICO Optical Computing*, pp. 407-413, Aug. 1988.

[Jenkins 92] B. K. Jenkins, S. H. Lee, and C. L. Giles, "Optical computing: introduction by the feature editors," Appl. Opt., Vol. 31, No. 26, pp. 5423-5425, Sept. 1992.

[Johnson 87] K. M. Johnson, M. A. Handschy, and L. A. Pagano-Stauffer, "Optical computing and image processing with ferroelectric liquid crystals," Opt. Eng., Vol. 26, No.5, pp. 385-392, May 1987.

[Johnson 88] K. M. Johnson, M. R. Surette, and J. Shamir, "Optical interconnection network using polarization-based ferroelectric liquid crystal gates," Appl. Opt., Vol. 27, No. 9, pp. 1727-1733, May 1988.

[Jutamulia 89] S. Jutamulia, and G. Storti, "Incoherent optical interconnects (perfect shuffle) based on shadow casting," Appl. Opt., Vol. 28, No. 20, pp. 4262-4263, Oct. 1989.

[Kompanets 95] I. N. Kompanets, "Spatial Light Modulators: Russian Research, Development, and Applications," *Optical Society of America Topical Meeting on Spatial Light Modulators and Applications, 1995 Technical Digest Series*, (Optical Society of America, Washington DC), pp. 144-145, Mar. 1995.

[Kumar 86] M. Kumar and G. F. Pfister, "The onset of hot spot contention," in *Proc. 1986 Int. Conf. on Parallel Processing* (IEEE Computer Society Press, Washington, DC), pp. 28-34, Aug. 1986.

[Lee 86] G. Lee, C. P. Kruskal, D. J. Kuck, "The effectiveness of combining in shared memory parallel computers in the presence of 'Hot Spots'," in *Proc. 1986 Int. Conf. on Parallel Processing* (IEEE Computer Society Press, Washington, DC), pp. 35-41, Aug. 1986.

[Lee 88] S. H. Lee and S. C. Esener, "Justifications for a hybrid approach to optical computing," *Proc. SPIE Vol. 881 Optical Computing and Nonlinear Materials*, pp. 177-178, Jan. 1988.

[Lev 81] G. F. Lev, N. Pippenger, and L. G. Valiant, "A fast parallel algorithm for routing in permutation networks," IEEE Trans. Comp., Vol. C-30, No. 2, pp. 93-100, Feb 1981.

[Levitan 85] S. P. Levitan, "Evaluation criteria for communication structures in parallel architectures," in *Proc. 1985 International Conf. on Parallel Processing* (IEEE Computer Society Press, Washington, DC), pp. 147-154, Aug. 1985.

[Li 92] K.-Y. Li, B. K. Jenkins, and C. Waterson, "Optical parallel-access read/write shared memory based on photorefractive crystals," *Optical Society of America Annual Meeting, 1992 Technical Digest Series*, (Optical Society of America, Washington DC), paper ThX1, p. 144, Sept. 1992.

[Li 95] K.-Y. Li, and B. K. Jenkins, "Optical parallel-access shared memory system: analysis and experimental demonstration," Appl. Opt., Vol. 34, No. 2, pp. 358-369, Jan. 1995.

[Lin 87] S.-H. Lin, T. F. Krile, and J. F. Walkup, "2-D optical multistage interconnection networks," *Proc. SPIE Vol. 752 Digital Optical Computing*, pp. 209-216, Jan. 1987.

[Lohmann 86a] A. W. Lohmann, W. Stork, and G. Stucke, "Optical perfect shuffle," Appl. Opt., Vol. 25, No. 10, pp. 1530-1531, May 1986.

[Lohmann 86b] A. W. Lohmann, "What classical optics can do for the digital optical computer," Appl. Opt., Vol. 25, No. 10, pp. 1543-1549, May 1986.

[Lohmann 95] A. W. Lohmann, "The history of optical computing: a personal perspective," *Optical Society of America Topical Meeting on Spatial Light Modulators and Applications, 1995 Technical Digest Series*, (Optical Society of America, Washington DC), pp. 252-254, Mar. 1995.

[McAdams 90] L. R. McAdams, R. N. McRuer, and J. W. Goodman, "Liquid crystal optical routing switch," Appl. Opt., Vol. 29, No. 9, pp. 1304-1307, Mar. 1990.

[McKnight 95] D. J. McKnight, S. A. Serati, K. M. Johnson, and M. H. Schuck, "Liquid crystal over silicon spatial light modulators: emphasis on design for applications," *Optical Society of America Topical Meeting on Spatial Light Modulators and Applications, 1995 Technical Digest Series*, (Optical Society of America, Washington DC), pp. 126-129, Mar. 1995.

[Miller 89] D. A. B. Miller, "Optics for low-energy communication inside digital processors: quantum detectors, sources, and modulators as efficient impedance converters," Opt. Lett., Vol. 14, No. 2, pp. 146-148, Jan. 1989.

[Nassimi 81] D. Nassimi and S. Sahni, "A self-routing Benes network and parallel permutation algorithms," IEEE Trans. Comp., Vol. C-30, No. 5, pp. 332-340, May 1981.

[Noguchi 91] K. Noguchi, T. Sakano, and T. Matsumoto, "A Rearrangeable multichannel free-space optical switch based on multistage network configuration," J. Lightwave Tech., Vol. 9, No. 12, pp. 1726-1732, Dec. 1991.

[Norton 85] A. Norton and G. F. Pfister, "A methodology for predicting multiprocessor performance," in *Proc. 1985 Int. Conf. on Parallel Processing*, pp. 772-781, Aug. 1985.

[Ostermayer 83] F. W. Ostermayer, Jr., P. A. Kohl, and R. H. Burton, "Photochemical etching of integral lenses on InGaAsP/InP light-emitting diodes," Appl. Phys. Lett., Vol. 43, No. 7, pp. 642-644, Oct. 1983.

[Parker 80] D. S. Parker, Jr., "Notes on shuffle/exchange-type networks," IEEE Trans. Comput., Vol. C-29, pp. 213-222, Mar. 1980.

[Patel 81] J. H. Patel, "Performance of processor-memory interconnections for multiprocessors," IEEE Trans. Comp., Vol. C-30, No. 10, pp. 771-780, Oct. 1981.

[Patel 87] J. S. Patel and J. W. Goodby, "Properties and applications of ferroelectric liquid crystals," Opt. Eng., Vol. 26, No. 5, pp. 373-384, May 1987.

[Pfister 85a] G. F. Pfister, W. C. Brantley, D. A. George, S. L. Harvey, W. J. Kleinfelder, K. P. McAuliffe, E. A. Melton, V. A. Norton, J. Weiss, "The IBM research parallel processor prototype (RP3): introduction and architecture," in *Proc. 1985 Int. Conf. on Parallel Processing* (IEEE Computer Society Press, Washington, DC), pp. 764-771, Aug. 1985.

[Pfister 85b] G. F. Pfister and V. A. Norton, " 'Hot spot' contention and combining in multistage interconnection networks," in *Proc. 1985 Int. Conf. on Parallel Processing* (IEEE Computer Society Press, Washington, DC), pp. 790-797, Aug. 1985.

[Pinkston 92] T. M. Pinkston, "The GLORI strategy for multiprocessors: integrating optics into the interconnect architecture," Ph.D. Dissertation, Stanford University, Stanford, CA, 1992.

[Psaltis 90] D. Psaltis, M. A. Neifeld, A. Yamamura, and S. Kobayashi, "Optical memory disks in optical information processing," Appl. Opt., Vol. 29, No. 14, pp. 2038-2057, May 1990.

[Richards 91a] G. W. Richards, "Concurrent multi-stage network control arrangement," U.S. Patent Number 4,991,168, Feb. 5, 1991.

[Richards 91b] G. W. Richards, "Network control arrangement for processing a plurality of connection requests," U.S. Patent Number 4,993,016, Feb. 12, 1991.

[Richards 93] G. W. Richards, "Theoretical aspects of multi-stage networks for broadband networks," *Tutorial Notes from IEEE Infocom '93*, San Francisco, CA, Mar. 28, 1993.

[Sawchuk 87] A. A. Sawchuk, "3-D optical interconnection networks," in *Proc. SPIE Vol 813 Proc. 14th Congress of the Int. Commission for Optics*, pp. 547-548, Aug. 1987.

[Sawchuk 88] A. A. Sawchuk and I. Glaser, "Geometries for optical implementations of the perfect shuffle," in *Proc. SPIE Vol. 963 Proc. Int. Optical Computing Conf.*, pp. 270-279, Aug. 1988.

[Schenfeld 95] E. Schenfeld, "Massively parallel processing with optical interconnections: what can be, should be and must not be done with optics," *Optical Society of America Topical Meeting on Spatial Light Modulators and Applications, 1995 Technical Digest Series*, (Optical Society of America, Washington DC), pp. 16-18, Mar. 1995.

[Schwartz 80] J. T. Schwartz, "Ultracomputers," ACM Trans. Prog. Lang. Syst., Vol. 2, No. 4, pp. 484-521, Oct. 1980.

[Sheng 89] Y. Sheng, "Light effective 2-D optical perfect shuffle using Fresnel mirrors," Appl. Opt., Vol. 28, No. 15, pp. 3290-3292, Aug. 1989.

[Stevens 95] A. J. Stevens, J. Gourlay, S. Samus, W. J. Hossack, D. G. Vass, and D. C. Burns, "Experimental Investigation of Free Space Optical Interconnects," *Optical Society of America Topical Meeting on Spatial Light Modulators and Applications, 1995 Technical Digest Series*, (Optical Society of America, Washington DC), pp. 38-41, Mar. 1995.

[Stirk 88] C. W. Stirk, R. A. Athale, and M. W. Haney, "Folded perfect shuffle optical processor," Appl. Opt., Vol. 27, No. 2, pp. 202-203, Jan. 1988.

[Stone 71] H. S. Stone, "Parallel processing with the perfect shuffle," IEEE Trans Comput., Vol. C-20, No. 2, pp. 153-161, Feb. 1971.

[Stone 73] H. S. Stone, "Problems of parallel computation," in *Complexity of Sequential and Parallel Numerical Algorithms*, J. F. Traub ed., Academic Press, New York, pp. 1-16, 1973.

[Thomas 86] R. H. Thomas, "Behavior of the Butterfly (tm) parallel processor in the presence of memory hot spots," in *Proc. 1986 Int. Conf. on Parallel Processing* (IEEE Computer Society Press, Washington, DC), pp. 46-50, Aug. 1986.

[Vishkin 83] U. Vishkin and A. Vidgerson, "Dynamic parallel memories," Information and Control, Vol. 56, pp. 174-182, 1983.

[Waterson 91] C. Waterson and B. K. Jenkins, "Shared Memory Optical/Electronic Computer: Architecture and Design," *Optical Society of America Topical Meeting on Optical Computing, 1991 Technical Digest Series*, (Optical Society of America, Washington DC), pp. 195-198, Mar. 1991.

[Waterson 94a] C. Waterson and B. K. Jenkins, "Shared Memory Optical/Electronic Computer: Architecture and Control," Appl. Opt., Vol. 33, No. 8, pp. 1559-1574, Mar. 1994.

[Waterson 94b] C. Waterson and B. K. Jenkins, "Passive optical interconnection network employing a shuffle–exchange topology," Appl. Opt., Vol. 33, No. 8, pp. 1575-1586, Mar. 1994.

[Waterson 95] C. Waterson and B. K. Jenkins, "Routing algorithm for a circuit-switched optical extended generalized shuffle network," *Optical Society of America Topical Meeting on Optical Computing, 1995 Technical Digest Series*, (Optical Society of America, Washington DC), pp. 29-31, Mar. 1995.

[Weingarten 90] D. Weignarten, "The status of GF11," Nuclear Physics B (Proc. Suppl.), Vol. 17, pp. 272-275, 1990.

[Wu 80] C.-L. Wu, T.-Y. Feng, "On a class of multistage interconnection networks," IEEE Trans. Comput., Vol. C-29, No. 8, pp. 694-702, Aug. 1980.

[Wu 81] C.-L. Wu, T.-Y. Feng, "The universality of the shuffle–exchange network," IEEE Trans. Comput., Vol. C-30, No. 5, pp. 324-332, May 1981.

[Yew 86] P.-C. Yew, N.-F. Tzeng, and J. H. Lawrie, "Distributing hot-spot addressing in large-scale multiprocessors," in *Proc. 1986 Int. Conf. on Parallel Processing* (IEEE Computer Society Press, Washington, DC), pp. 51-58, Aug. 1986.