Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a, user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use" that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select "Pages from: first page # to: last page #" on the print dialog screen

Order Number 9003133

Fault-tolerant interconnection networks for multiprocessor systems

Nassar, Hamed Mohamed, D.Eng.Sc.

New Jersey Institute of Technology, 1989

U·M·I
300 N. Zeeb Rd.
Ann Arbor, MI 48106

Fault-Tolerant Interconnection Networks for Multiprocessor Systems

by

Hamed Nassar

Dissertation submitted to the Faculty of the Graduate School
of the New Jersey Institute of Technology in partial fulfillment
of the requirements for the degree of
Doctor of Engineering Science
1989

Approval Sheet

Title of Thesis: Fault-Tolerant Interconnection Networks for Multiprocessor Systems

Name of Candidate: Hamed Nassar
Doctor of Engineering Science, 1989

Thesis and Abstract Approved:

Dr. John Carpinelli — Date
Assistant Professor
Department of Electrical Engineering

Dr. Raj Misra — Date

Dr. Peter Ng — Date

Dr. Anthony Robbi — Date

Abstract

Title of Thesis: Fault-Tolerant Interconnection Networks for Multiprocessor Systems

Hamed Nassar, Doctor of Engineering Science, 1989

Thesis directed by: Dr. John Carpinelli

Interconnection networks represent the backbone of multiprocessor systems. A failure in the network, therefore, could seriously degrade the system performance. For this reason, fault tolerance has been regarded as a major consideration in interconnection network design. This thesis presents two novel techniques to provide fault tolerance capabilities to three major networks: the Baseline network, the Beneš network and the Clos network.

First, the Simple Fault Tolerance Technique (SFT) is presented. The SFT technique is in fact the result of merging two widely known interconnection mechanisms: a normal interconnection network and a shared bus. This technique is most suitable for networks with small switches, such as the Baseline network and the Beneš network. For the Clos network, whose switches may be large for the SFT, another technique is developed to produce the Fault-Tolerant Clos (FTC) network. In the FTC, one switch is added to each stage. The two techniques are described and thoroughly analyzed.

VITA

Name: Hamed Mohamed Nassar.

Degree and date to be conferred: D.Eng.Sc., 1989.

Secondary education: Shebin El-Kanater Secondary School.

Collegiate institutions attended	Dates	Degree	Date of Degree
Ain Shams University, Egypt	1974-79	B.S.E.E.	May, 1979
New Jersey Institute of Technology	1983-85	M.S.E.E.	May, 1985
New Jersey Institute of Technology	1985-89	D.Eng.Sc.	May, 1989

Major: Electrical Engineering

Minor: Computer and I

To my parents, and to Manal and little Nancy.

Acknowledgements

Mere words are not enough to express my gratitude to my advisor, Dr. John Carpinelli. His mastery of the subject matter combined with a great deal of sincerity, enthusiasm and dynamism, have made working on this dissertation the most wonderful experience. Indeed, these qualities make him the kind of advisor every student wishes for, but seldom finds.

Many thanks are due to the distinguished members of my committee: Dr. Raj Misra, Dr. Peter Ng and Dr. Anthony Robbi. Their invaluable suggestions and stimulating discussions have considerably improved the quality of the dissertation.

I am really thankful to all the professors and staff of the Department of Electrical Engineering who have been helpful and supportive throughout the years I have spent at NJIT. Special thanks are due to Dr. Warren Ball, Dr. Joseph Strano, Dr. Edwin Cohen, Dr. Khalil Denno, and Ms. Brenda Walker.

I would also like to thank Dr. Roman Voronka, of the Math Department, who has made me love mathematics, and Mr. Steve Keeton, of the Computer Services Department, whose role was crucial in completing the computer work of the dissertation.

Last but not least, I would like to thank my immediate family. Many thanks go to my parents, for their continuous prayers and encouragement, and to my wife, Manal, and daughter, Nancy, for putting up with my study habits.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Parallel Processing

Fast computers are increasingly being required as sophisticated, computation-intensive applications continue to evolve. The search for a fast computer seems endless as more and more speed is always demanded. This need for speed was fulfilled in the early days of computers by advancements in technology. As tubes were replaced by transistors and other discrete solid state components, the computer became faster. Then came integrated circuits (IC) technology to make computers even faster. This race came to the point where the component technology could no longer catch up with the need for more speed. A drastic change in the architecture of the computer was the place where an answer could be found. The change was in the form of using *parallel processing.*

Early computers – the so-called von Neumann machines – used a single processor to *fetch* instructions from memory and *execute* them one at a time [57]. Parallel systems, however, are based on the principle that more than one task can be performed simultaneously. This concurrency can be realized either at the software level or at the hardware level [56].

At the software level, parallelism is obtained by *time-sharing* the computer re-

sources among different programs. Here, the operating system divides [30] the CPU time among the different programs so that no one program monopolizes the CPU for a long time while others are waiting. This technique has been used on computers with a single processor to achieve parallelism in the form of *multiprogramming*, *multitasking*, *multiuser* and *time-sharing* capabilities.

When parallelism is implemented at the hardware level, it can take place at the computer level, at the sub-processor level, or at the processor level. Parallelism can also be achieved by computers either having a single processor, *uniprocessors*, or having more than one processor, *multiprocessors*.

Distributed computing [26] is the name used when parallelism takes place at the computer level. Here the computation load is distributed among more than one computer. These computers, which are connected by a communications network, work totally independently and asynchronously. Communications between the different computers take place in the form of passing messages to obtain data or exchange results. The computers may exist in close proximity to each other, in which case they are connected by a Local Area Network (LAN), or they may be scattered over a wide geographic area, in which case they are connected by a Wide Area Network (WAN). Computers in a distributed computing system are said to be *loosely coupled*.

Computers can achieve parallelism at the sub-processor level in several ways. One way is by fetching an instruction while another is being executed. Another way is to overlap Central Processing Unit (CPU) and Input/Output (I/O) operations. Yet another way is by using *pipelining* [25]. In pipelined architectures, the idea of assembly lines is utilized. In an assembly line, the job is divided into many steps and each step is assigned to a specific worker along the line. This manufacturing technique has proven efficient, because all the elements of the line are

continuously busy. In a pipelined computer, a control unit divides the instruction [42] into a number of phases and assigns each phase to a subunit in the main processor. Each subunit performs its part and then sends the result to the next subunit along the way. This makes each subunit continuously busy, therefore increasing the throughput of the system [23,24]. At steady state, the flow of instructions into the pipeline is equal to the flow of instructions out of the pipeline.

Undoubtedly, more parallelism can be obtained by having more than one processor in the computer. In a *multiprocessor* computer [8], the different processors cooperate to execute the instructions of a program. The program is divided into different parts, each of which can be executed independently. The partial results from the different processors are exchanged and the overall result of the program can be obtained from them by a master processor or by a control unit. In a multiprocessor, all the processors access the same memory, which is usually divided into interleaved modules for greater efficiency [74]. An *array* computer [51] is similar to a multiprocessor computer except that the processors are replaced by Arithmetic and Logic Units (ALU) which work synchronously under the supervision of a common control unit. Moreover, in the array computer, each ALU is provided with a local memory to make up a *Processing Element* (PE).

It was once possible to classify a parallel computer as using one parallelism technique or another. Now, however, more than one technique may be used in the same computer, making categorizing a particular computer a difficult task. For example, some of the techniques used for parallelism in uniprocessors, such as pipelining, can be used for the individual processors in a multiprocessor computer, giving rise to greater execution speeds. Moreover, some software parallelism techniques can also be used with that computer, giving rise to even greater speeds, and so on.

1.2 Multiprocessors

Clearly, using many processors in the same system yields more speed than using one processor [36]. Recent advances in VLSI technology, coupled with the need for fast computers, have made large-scale multiprocessor systems economically feasible. In such systems, hundreds or even thousands of processors are used to carry out the computations of a program concurrently, thereby speeding up the execution of the program. Many applications can benefit from this enormous computing power. Typical applications include simulation programs, such as weather forecasting, and real-time programs, such as radar tracking.

The basic architecture of a multiprocessor system is shown in Figure 1.1. In this configuration, the N processors carry out computations on data stored in the M memory modules. For the interaction between the processors and memory, there must be a communications mechanism to enable any processor to access any memory module in the shortest possible time. This mechanism is of extreme importance, as the efficiency of the system depends mainly on its ability to establish the required paths between its two sides. Many such mechanisms have been proposed and explored in the literature.

At one extreme is the *shared bus* [46]. This is similar to the bus of the uniprocessor computer with a control unit to limit the access to the bus to one processor at a time. A multiprocessor system using a shared bus as its communications mechanism is shown in Figure 1.2. A processor requiring access to memory puts the address of the memory location it wants to access on the bus. The address is decoded and used to enable the memory module where the target location is. As simple and inexpensive as this mechanism is, it results in extremely poor performance when N is large [58]. That is because only one processor can use the bus at a time. As

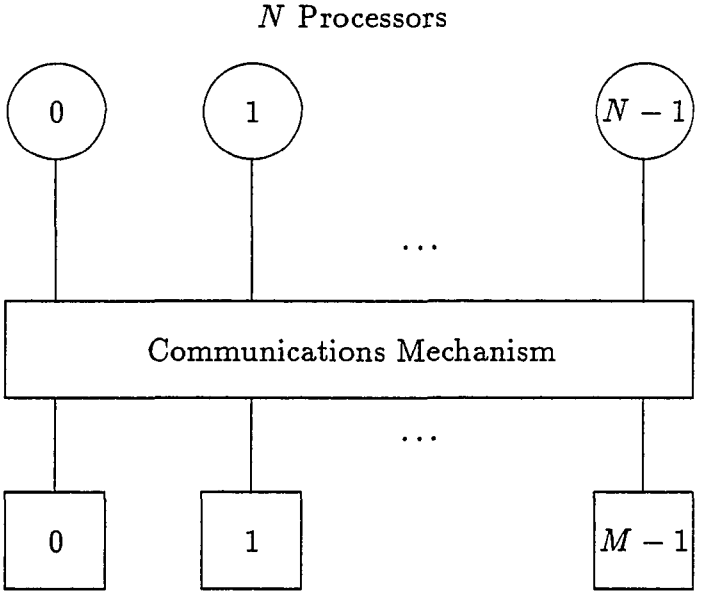


Figure 1.1: Basic multiprocessor architecture

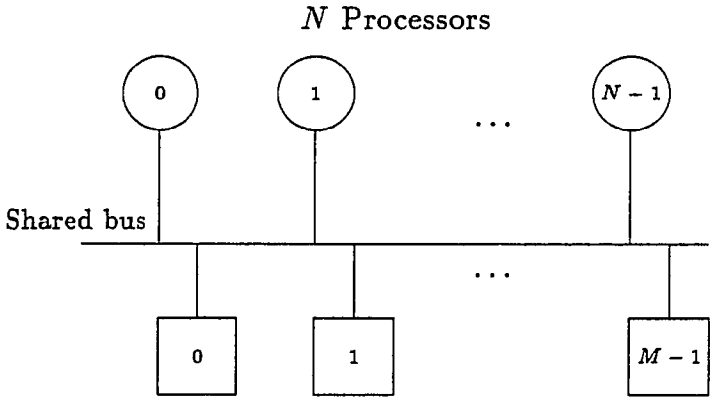


Figure 1.2: Multiprocessor system with a shared bus

a consequence, this mechanism has been ruled out for large-scale multiprocessor systems, those with possibly thousands of processors. However, if the number of processors is small enough, e.g. $N = 2$, the bus can be used as a simple, inexpensive communications mechanism. (Actually, this is the principle behind the Simple Fault Tolerance (SFT) technique presented in this thesis.) The possibility of having more than one bus in the system has been explored [14,60] for relatively large values of N. However, the work done in this direction indicates [29,50,83] that performance still degrades as the ratio N/B increases, where B is the number of buses.

At another extreme is the *crossbar switch*, such as the one shown in Figure 1.3. This is called an $N \times M$ switch because it has N inputs and M outputs. The crossbar switch can be thought of as two rows of conducting bars placed on top of each other without direct contact. Figure 1.3a shows this conceptual construction. There are N horizontal bars and M vertical bars. To establish a connection between, say,

horizontal bar 0 and vertical bar 1, one only has to connect the two bars at the point where they intersect (symbolized by the little circle in the figure.) Now if a signal is put at input 0, the same signal will appear at output 1. That is, a path has been created between input 0 and output 1. To remove this path, one only has to disconnect the two bars again at the point where they intersect. This connection and disconnection is implemented by a control unit attached to the switch. Moreover, the actual switch is an electronic circuit, usually an IC, where there are no actual bars. The crossbar switch is ideal in that it can be *set* to connect any x inputs, $x \leq N$, to any y outputs, $y \leq M$, simultaneously and in a one-to-one fashion. Its

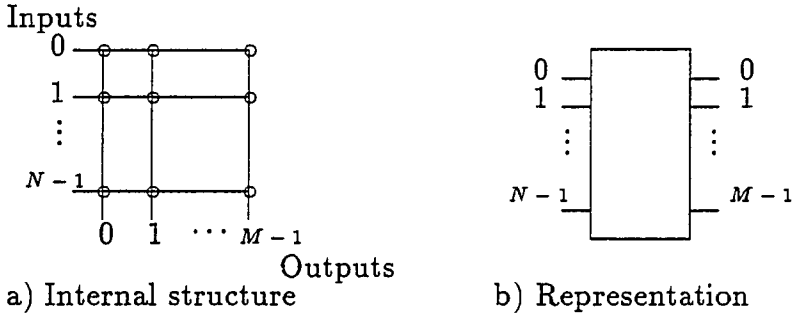


Figure 1.3: $N \times M$ crossbar switch

drawback, however, is that it becomes prohibitively expensive [15] for large values of N or M. Figure 1.3b shows the symbolic representation of the crossbar switch. This representation is used throughout the thesis to indicate crossbar switches.

Between the two extremes are *multistage interconnection networks* (MINs). These networks are built from small crossbar switches arranged in stages, with each stage being connected to the next stage through a set of *links*. The inputs to the network are called *inlets* and the outputs of the network are called *outlets*. The words *input* and *output* will be used only for the individual switches making up the network. A path can be established between an idle inlet and an idle outlet by setting the indi-

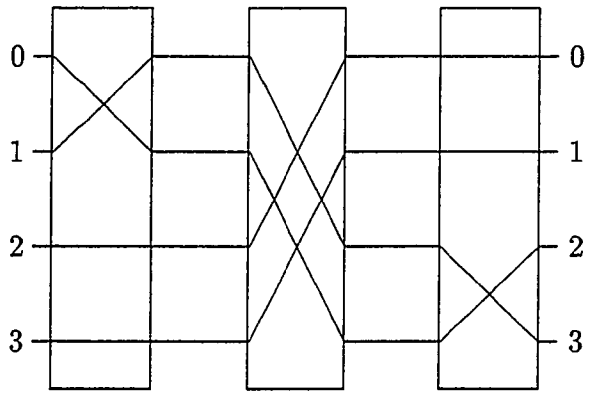
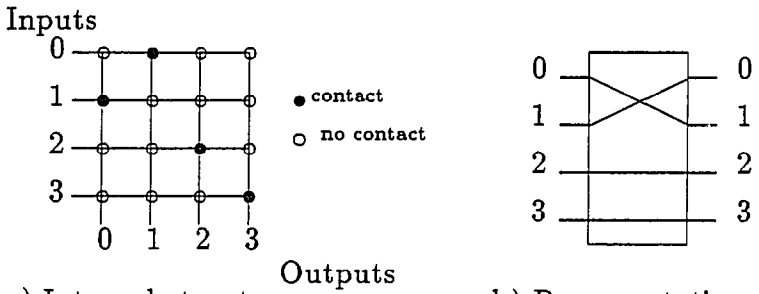


Figure 1.4: Basic configuration of interconnection networks

vidual switches. A 4×4 MIN is shown in Figure 1.4. There are three stages, each containing one switch, and the stages are interconnected by *interstage links*. The network has 4 inlets and 4 outlets, and it can be seen how paths can be established between inlets and outlets just by making appropriate connections of the inputs and outputs of the individual switches. It should be noted that the figure does not represent a real MIN, as each switch can perform all possible permutations of the inlets into the outlets. The figure is intended only to show how stages of switches may be linked to form a MIN. Figure 1.5 shows how a switch can be set to realize a given mapping. In fact, the switch is that shown in the first stage of the network of Figure 1.4.

1.3 Interconnection Networks and the Need for Fault Tolerance

MINs were developed originally for use in telephone systems, long before the idea of multiprocessor computer systems started to materialize. In telephony, MINs are used in switching offices to link different callers by connecting their respective lines



a) Internal structure b) Representation

Figure 1.5: 4 × 4 switch set to realize an arbitrary mapping

together. In multiprocessors, they are used for communications between processors and memory modules. *Setting* (or *routing*) a MIN is the process of setting the individual switches so that the network can *realize* a *mapping* from the inlets into the outlets. The network shown in Figure 1.4 is set to realize the mapping:

$$0 \rightarrow 2$$
$$1 \rightarrow 3$$
$$2 \rightarrow 0$$
$$3 \rightarrow 1$$

Throughout this thesis, MINs will be discussed in the context of their use in multiprocessors as defined in Section 1.2. In other words, no reference will be made to other computer architectures that use MINs. This is done because the focus in the thesis is on MINs, regardless of the operating environment or the architectural organization of the computer where they are used. However, it should be noted that MINs are also used in other computer architectures such as array computers, data flow computers and vector computers. In all these architectures, there are more than one processing unit connected together by a MIN.

Also in this thesis, the words *multiprocessor*, *multiprocessor computer*, *multiprocessor system* and *multiprocessor computer system* will be used interchangeably. It is worth noting that the processors in a multiprocessor system are said to be *tightly coupled.*

MINs may be classified according to the way they receive the mapping. Typically the processors connected to the inlets require access to the memory modules connected to the outlets by generating a *memory access request.* This request must contain the number of the particular outlet to which the memory module is connected. If all processors send their requests at the same time to the network, the network is said to work *synchronously.* A *memory access cycle* [69] is the time needed to establish the path, plus the time it takes to read or write a memory word. Thus in a synchronous environment, processors may send requests only at fixed intervals equal to the memory access cycle. If, on the other hand, the network is able to handle requests on an individual basis submitted at any time by the processors, then the network is said to work *asynchronously.* In any case, since the processors work independently, it is likely that more than one processor will seek access to the same memory module, causing a *memory conflict.* This problem is unrelated to the MIN operation, and a solution to it can be achieved in the design of the operating systems and memory management schemes of the multiprocessor system. As this thesis is concerned with MINs, only *network conflicts* will be considered.

It is worth mentioning that realizing *random permutations* is the ultimate figure of merit for testing the connectivity of any MIN. That is because if a MIN can realize any random permutation, it can realize any mapping. For this reason, MINs are normally designed and studied as *permutation networks.* MINs mentioned in this thesis are all permutation networks.

MINs may be classified according to the way they are routed [34]. Some MINs have a central routing unit. This unit receives the mapping and runs an algorithm to find how the individual switches should be set so that the mapping can be realized. The settings are then sent to the individual switches for implementation by local control units. This type of MIN is called *centrally routed.* By contrast, there are

self or distributed routed MINs. In a *self-routed* MIN a *routing tag* is placed on the inlet to establish the required path to the outlet. One or more bits in the routing tag are used to set the switch in the first stage, thereby allowing the remaining bits to go one stage towards the destination. Again, one or more bits are used to set the switch in the second stage, thereby allowing the remaining bits to travel one more stage towards the destination. This process continues until a path is created between the inlet and the target outlet. This thesis covers networks of both routing types, central and distributed.

MINs may be classified according to whether every possible mapping of the inlets into the outlets can be realized. If this is the case, the MIN is called *non-blocking.* An example of a non-blocking MIN is the Clos network [28]. If, on the other hand, one or more paths in the mapping cannot be realized, the network is called *blocking.* An example of a blocking MIN is Omega network [51]. Typically, self-routed MINs are blocking, and centrally routed MINs are not. This thesis covers MINs of both types.

Blocking MINs may be classified according to the way partially completed paths are treated [47]. Suppose a path gets blocked in a switch at a given stage between the inlet and the outlet. The *blocking switch*, the switch where the path was blocked, may send a signal to the preceding switches on the partially completed path to dismantle the path. The MIN in this case is called *unbuffered.* If, on the other hand, the partially completed path is kept, and the blocking switch stores the remaining routing bits in a queue for possible completion in the next memory cycle, then the MIN is called *buffered* [31]. This thesis deals only with unbuffered MINs.

Self-routed MINs may be classified as either *circuit switched* or *packet-switched.* In circuit-switched MINs, the path from the processor to the memory module is established before any data is sent by the processor. Moreover, the path is kept

for the entire duration of the memory cycle. Any "bare bones" MIN is circuit-switched; to make it packet-switched, additional hardware is required. In packet switched MINs [32], the processor sends a self-contained message, having its address and the address of the destination, to the switch connected to it in the first stage. The switch investigates the destination address of the packet and sends it to the next switch along the way. Once a switch sends a packet to the next switch, it can receive another packet and send it along. This operation mode allows for greater throughput, but at the expense of a complicated switch design. Packet-switched MINs are usually buffered for greater performance. The MINs studied in this thesis are all of the circuit-switched type.

Finally, MINs may be classified according to their *fault tolerance* capabilities. This is the topic of this thesis. Fault tolerance has been raised as an important issue in designing MINs for multiprocessor systems. The reason stems from the fact that these systems are likely to run important tasks where interruption could have damaging effects. Since the network is at the center of such systems, a failure in the network can seriously degrade the performance of the system. This is particularly true in networks where the paths between inlet/outlet pairs are unique, such as the shuffle networks [87]. In such networks, if a switch needed to establish a path is faulty, that path cannot be established until the fault is physically removed. Given the sequential and dependent nature of computer programs, a memory word can be vital to the completion of a program. If that word cannot be accessed, the program cannot be completed. In a multiprocessor system, therefore, it is necessary to maintain the ability of establishing communications paths between processors and memory modules at all times. This entails that the interconnection network be fault tolerant.

One cannot determine whether a network is fault-tolerant unless a fault tolerance

criterion is defined; a MIN can be fault-tolerant according to one criterion, but not so according to another. Furthermore, a network may tolerate a given number of faults of a specific type, but not a different number of faults or even the same number of faults when they are of a different type. Therefore, a fault-tolerant MIN design should specify a fault tolerance criterion, and the number and type of faults to be tolerated. These three items make up the *fault tolerance model.*

In this thesis, a novel technique is described to provide fault tolerance to virtually any MIN. The technique is demonstrated on one type of MIN, the Baseline network, where it can be used most efficiently. In the Clos network where the technique is less successful, another technique is suggested. Together, the two techniques should offer a comprehensive solution to the fault tolerance problem of an important class of MINs.

1.4 Outline

The rest of this thesis is organized as follows. Chapter 2 presents some mathematical background needed to understand the work that is either mentioned or developed in this thesis. In particular, a generalized model for multistage interconnection networks is developed for use later in the thesis. Furthermore, the basics of fault tolerance, combinatorics and reliability are presented. These basics are used quite extensively in this thesis. Chapter 3, shows some popular implementations of MINs. The construction, operation and routing of each MIN are discussed. Chapter 4 is a survey of some of the work that has already been done on fault tolerance for the class of MINs considered in this thesis. The advantages and shortcomings of these techniques are highlighted. In Chapter 5, the Simple Fault Tolerance (SFT) technique is introduced through implementation on a Baseline network. The construction, operation, and routing of the Simple Fault-Tolerant Baseline (SFTB)

network under both normal and faulty conditions are described in detail. The advantages and disadvantages of the SFT technique are discussed. It is shown why this technique is not suitable for networks with large switches such as Clos networks. Chapter 6 gives an alternative technique that is suitable for Clos networks. The last chapter contains some concluding remarks on the work done in the thesis.

Chapter 2

Basic Concepts and Notation

In this chapter, the basic concepts needed to understand the work in this thesis are introduced. These concepts include interconnection network modelling, fault tolerance principles, combinatorics, and reliability theory. They all are used either to explain work already done by others or to derive new results.

2.1 Interconnection Networks

Before, discussing the characteristics of MINs, some definitions will be introduced. The definitions are adapted mainly from [37].

Definition 2.1 *A function F from a set A into a set B is* one-to-one *if each element in B has* at most *one element in A mapped into it. In other words, F is one-to-one if $a_1F = a_2F$ implies $a_1 = a_2$. Further, a function F is* onto *B if every element in B has* at least *one element in A mapped into it. In other words, F is onto B if for each $b \in B$ there exists $a \in A$ such that $aF = b$. It should be noted that a function on A by definition is defined for each $a \in A$.*

Definition 2.2 *A permutation of a set A is a one-to-one function F mapping A onto itself. One may write this as*

$$F : A \rightarrow A. \tag{2.1}$$

If $aF = a$ for all $a \in A$, F is called the identity permutation.

To illustrate, consider the set $A = \{0, 1, 2, 3\}$. Suppose that a function F is given by the mapping:

$$\begin{aligned} 0 &\rightarrow 2 \\ 1 &\rightarrow 3 \\ 2 &\rightarrow 1 \\ 3 &\rightarrow 0 \end{aligned}$$

It is obvious that F is one-to-one and onto and, thus, defines a permutation. The permutation F can be written in a more standard notation as

$$F = \begin{pmatrix} 0 & 1 & 2 & 3 \\ 2 & 3 & 1 & 0 \end{pmatrix}$$

Definition 2.3 *Let φ and ψ be two functions. If φ maps element a into b, denoted as $a\varphi = b$, and ψ maps element b into c, denoted as $b\psi = c$, then $a(\varphi\psi) = (a\varphi)\psi = b\psi = c$. $\varphi\psi$ is called the* composition *of φ and ψ.*

Theorem 2.1 *If F_1 and F_2 are both permutations of the set A, then the composite function F_1F_2 is also a permutation of A.*

To illustrate Theorem 2.1, whose proof can be found in [37], consider the above set A and permutation F. Also consider the permutation G, given by

$$G = \begin{pmatrix} 0 & 1 & 2 & 3 \\ 1 & 3 & 2 & 0 \end{pmatrix}.$$

Then the composite function FG is given by

$$FG = \begin{pmatrix} 0 & 1 & 2 & 3 \\ 2 & 0 & 3 & 1 \end{pmatrix}$$

which is a permutation according to Definition 2.2. It should be noted that $FG \neq GF$.

Definition 2.4 *Let N be the cardinality (number of elements) of the set A. Then, the set of all possible permutations of A is called the* symmetric group, *denoted by Σ_N. The cardinality of Σ_N is $N!$.*

Figure 2.1 shows a generalized MIN. In this MIN there are N inlets, M outlets, ν stages, and s_j switches in each stage j, $0 \leq j \leq \nu - 1$. In addition, stage j derives *all* of its inputs from stage $j-1$, stage 0 derives *all* of its inputs from the inlets of the MIN, and the outlets are *all* derived from stage $\nu - 1$. All the MINs discussed in the thesis fit in this generalized model unless otherwise stated. The MINs that do not fit in the model will appear only at the end of Chapter 3 for the sake of completeness.

Let F_{l_j}, $1 \leq j \leq \nu - 1$ be the function realized by the set of links between stages $j-1$ and j, mapping the outputs of stage $j-1$ onto the inputs of stage j. Further, let F_{s_j}, $0 \leq j \leq \nu - 1$, be the function, realized by the set of switches in stage j, mapping the inputs of stage j onto its outputs. Finally, let F_I be the function, realized by the set of links connecting the inlets to the inputs of stage 0, mapping the inlets on the inputs of stage 0, and let F_O be the function, realized by the set of links connecting the outputs of stage $\nu - 1$ to the outlets, mapping the outputs of stage $\nu - 1$ on the outlets. Then a MIN is *completely* defined by the $(2\nu + 3)$-tuple

$$(I, O, F_I, F_O, F_{l_1}, F_{l_2}, \ldots, F_{l_{\nu-1}}, \mathcal{F}_{s_0}, \mathcal{F}_{s_1}, \ldots, \mathcal{F}_{s_{\nu-1}})$$

where $\mathcal{F}_{s_j}$ is the set of all mappings realizable by stage j of its inputs into its outputs, and I and O are the sets of inlets and outlets, respectively.

In any MIN, for all j, $1 \leq j \leq \nu - 1$, F_{l_j} is a *permanent* permutation of the outputs of stage $j-1$ into the inputs of stage j. However, for all j, $0 \leq j \leq \nu - 1$, F_{s_j} is a *variable* permutation that can be changed by setting the individual switches of stage j. Throughout this thesis, the word *mapping* will indicate a one-to-one mapping, unless otherwise specified.

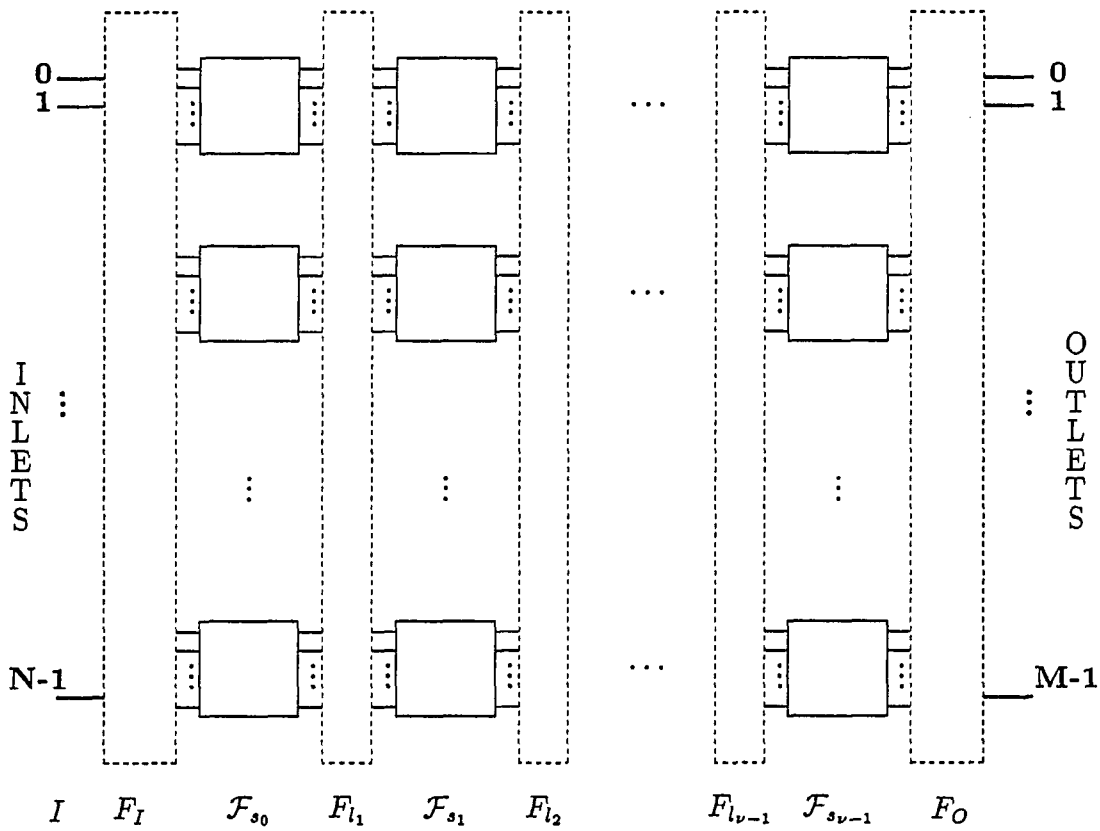


Figure 2.1: Generalized MIN

Let S_{NM} be the set of mappings of the N inlets into the M outlets realizable by the MIN defined above. Then, it can be seen that

$$S_{MN} = F_I \mathcal{F}_{s_0} F_{l_1} \mathcal{F}_{s_1} F_{l_2} \ldots F_{l_{\nu-1}} \mathcal{F}_{s_{\nu-1}} F_O. \tag{2.2}$$

A MIN may or may not have $N = M$. All MINs in this thesis have $M = N$. Such MINs are usually called permutation networks. An ideal permutation network is one where $S_{NN} = \Sigma_N$. This is the case for non-blocking networks. Blocking networks have $S_{NN} \subset \Sigma_N$.

2.2 Fault Tolerance for MINs

Before designing a fault-tolerant MIN, a *fault tolerance model* [4,49] must be defined. The fault tolerance model contains three elements: the *fault model*, the *fault tolerance criterion*, and the *fault tolerance size*.

The fault model is the type of faults that can occur in the network. Implicitly, the fault model specifies the type of faults that can be recovered from using the proposed fault tolerance design. Different designs specify different fault models. A good design, however, is one whose fault model includes as many fault types as possible. To illustrate, a typical fault model is as follows.

1. Any network component can fail: MINs are made up of two types of components: switches and links. This assumption then states that the switches and links are likely to fail, and that the proposed design is capable of recovering from any such fault. A link fails if it is open or short circuited. A switch fails due to some internal malfunction. More details on switch failures can be found in Chapter 5.

2. The extra hardware added to provide fault tolerance to the network cannot fail: This assumption is usually made for two reasons. First, if the extra

hardware added to the network to make it fault tolerant could be assumed to fail, then it would not be possible to propose any fault tolerance design. In addition, this assumption can be justified for MINs because these components usually remain idle under normal conditions. Thus they can be expected to have higher lifetime than the actively working components of the network.

The fault tolerance criterion is the condition that must be met in order for the system to be called fault tolerant. The fault tolerance criterion for the networks surveyed in this thesis is mainly *full-access retention.* That is, after a fault occurs, each processor must still be able to communicate with any memory module. However, the two fault tolerant designs introduced in this thesis can offer a higher criterion – full recovery. *Full recovery* is the ability of the network to regain its pre-fault connectivity after a fault occurs.

The fault tolerance size is the number of faults that the system can recover from. That is, the number of faults that the system can have and still meet the fault tolerance criterion imposed. *All* the fault-tolerant MINs in this thesis, whether surveyed or developed, are single fault-tolerant. This means that the fault-tolerance criterion can be met only if there is one fault anywhere in the network. If a fault-tolerant network can tolerate i specific faults, $i > 1$, but not any arbitrary i faults, it is called i-robust.

2.3 Combinatorics

This section discusses the counting techniques necessary to evaluate the probability of a given event. One is often faced with the question "In how many ways can this event occur?" The answer starts with the *Law of Composition* which states:

If event A_1 can happen in n_1 ways, event A_2 can happen in n_2 ways, ..., and event A_m can happen in n_m ways, then the compound event (A_1 AND A_2 AND ... AND A_m) can happen in $\prod_{i=1}^{m} n_i$ ways.

Example 2.1 *How many 3-digit numbers can be written in the three slots below, given that each slot can be filled with a decimal digit* $i \in \{0, 1, \ldots, 9\}$.

\- - -

The solution is obtained by applying the Law of composition. Label the slots from left to right as 1, 2, and 3. Slot 1 can be filled with any one of the ten digits, and so can slots 2 and 3. That is, there are ten ways to fill in each slot, for a total of $10 \times 10 \times 10 = 1000$ ways. This answer is correct because in three positions, one can have any number from 000 to 999, which are 1000 numbers in total.

Now there are two additional counting problems that are often encountered: permutations and combinations.

2.3.1 Permutations (arrangements)

Here the number of ways k elements can be arranged is required. It should be noted that order *matters* in permutations, i.e. $AB \neq BA$.

Example 2.2 *How many 3-letter words can be formed from the 5 letters A, B, C, D, and E, without repeating any letters?*

This example is solved again by drawing three slots as shown below.

\- - -

Label the slots as in Example 2.1. In slot 1, one can put any one of the five letters, leaving four letters for the remaining two slots. Slot 2 can be filled with one of four letters, for each choice made for slot 1, leaving three letters to be used with

slot 3. Thus, the number of ways one can fill in the three slots, thereby creating 3-letter words, is $5 \times 4 \times 3 = 60$ words. The number 60 is the number of *permutations* of 5 things taken 3 at a time. In general the number of permutations of n things taken k at a time, denoted as $P(n,k)$, is

$$P(n,k) = n \cdot (n-1) \cdots (n-k+1) = \frac{n!}{(n-k)!}. \tag{2.3}$$

In fact, this is the first k factors of $n!$.

In the above example, repetition was not allowed. (Implicitly it was assumed that one copy of each letter was available.) In other words, words like AAA, AAB, ... etc. were not included in the number given. However, if repetition is allowed, the equation for $P(n,k)$ becomes

$$P(n,k) = n^k. \tag{2.4}$$

Thus in the example above, if repetition was allowed, the total number of 3-letter words would be $5^3 = 125$, which can be verified by the slots method. It should be noted that in Example 2.1 repetition was allowed. In general, one can tell easily whether repetition is allowed or not from the context of the problem.

Of interest is the permutations of n things taken n at a time, where the n things are not distinct. Suppose that of the n things, n_1 are similar, n_2 are similar, $\cdots$ etc., such that $n_1 + n_2 + \ldots + n_k = n$. Then the number of *visible* permutations is

$$P(n,n) = \frac{n!}{n_1! n_2! \cdots n_k!} \tag{2.5}$$

This formula is very useful in dealing with binary numbers, as illustrated by Example 2.3.

Example 2.3 *How many 8-bit binary numbers can be formed from three 0's and five 1's?*

Applying Equation 2.7, one finds that the number of the 8-bit numbers is

$$P(8,8) = \frac{8!}{(3!)(5!)} = 56.$$

2.3.2 Combinations (selections)

Here the number of ways k elements can be selected is required. It should be noted that order does *not* matter in combinations, i.e., $AB = BA$. This would be the situation for example in forming a team from a pool of players. Suppose that it is required to form a team of three players from a pool of 10 players. Suppose that a selection was made in which John was chosen first, Jack second, and Bill last. One cannot say that another team can be formed by choosing Bill first, John second and Jack last; it is the same team. This analogy should serve as a reminder that the approach needed to tackle a combinatorial problem depends on the problem itself.

Since order does not matter in combinations, it is expected that the formula for the combinations of n things taken k at a time, $C(n,k)$, will be the same as that of the permutations of n things taken k at a time, $P(n,k)$, after eliminating the "permutations" within each selection. Since each k-item selection can be arranged $k!$ times, the formula for the combinations is obtained by dividing the permutations formula by $k!$ to get

$$C(n,k) = \frac{n!}{(n-k)! \cdot k!} \tag{2.6}$$

In fact, this is the first k factors of $n!$ divided by $k!$. Sometimes $C(n,k)$ is written as $\binom{n}{k}$. The latter notation is the one adopted in the thesis for indicating combinations.

Example 2.4 *How many 11-player teams can be formed out of 20 players?*

The number of 11-player teams is

$$\binom{20}{11} = 167960.$$

2.4 Fundamentals of Reliability

The *reliability* of a system is defined as [68,82] the probability that the system will perform a required function under stated conditions for a stated period of time t. Mathematically, the reliability, R, of a system is

$$R = e^{-\lambda t}, \tag{2.7}$$

where λ is a constant representing the failure rate (per unit time). To simplify the analysis in this thesis, the time factor will be only implicit. In other words, when it is said that the reliability of a switch is r, it will mean the reliability of the switch over a given period of time t. This is done, because the focus will be on comparing reliabilities, rather than obtaining the absolute reliability value. In comparing two networks, for instance, the two networks should be under the same circumstances, including the period of time, t, hence the omission of the time factor.

Predicting reliabilities usually involves dealing with probabilities. It stands to reason then that an overview of probability theory be given before discussing the fundamentals of reliability.

2.4.1 Probability of a simple event

In an experiment of n equally likely outcomes, the probability that one event will occur is $1/n$.

Example 2.5 *What is the probability of obtaining 6 on a fair die?*

Since the die is fair, it is equally likely that any one of its 6 sides will appear if the die is thrown. The six events are {1,2,3,4,5,6}. Thus the probability of a 6 appearing, denoted as $Pr(6)$, is

$$Pr(6) = 1/6.$$

2.4.2 Probability of a compound event

A compound event is a composition of simple events using two rules: AND and OR. For example the event that one gets either a 6 OR a 4 on a die if thrown is a compound event made up of the simple events 6 and 4 and the rule "OR". Similarly the event of getting 4 AND 6 on two different dies is a compound event made up of the simple events 4 and 6 and the rule "AND".

Let A_1 and A_2 be two simple events. Then the two rules are defined as follows.

1. The probability that A_1 OR A_2 will occur, denoted as $Pr(A_1 \cup A_2)$, is

$$Pr(A_1 \cup A_2) = Pr(A_1) + Pr(A_2) - Pr(A_1 \cap A_2) \tag{2.8}$$

   If the two events are *mutually exclusive*, the last term vanishes and the probability of the compound event becomes

$$Pr(A_1 \cup A_2) = Pr(A_1) + Pr(A_2) \tag{2.9}$$

   In general, for any k mutually exclusive events, $A_1, A_2, \ldots, A_k$, the probability of the compound event A_1 OR A_2 OR ... OR A_k is

$$Pr(A_1 \cup A_2 \cup \ldots \cup A_k) = \sum_{i=1}^{k} A_i. \tag{2.10}$$

   The OR rule is used usually when words like "at least" or "either" are mentioned.

2. The probability that A_1 AND A_2 will occur, denoted as $Pr(A_1 \cap A_2)$, is

$$Pr(A_1 \cap A_2) = Pr(A_1) \cdot Pr(A_2|A_1), \tag{2.11}$$

   where $Pr(A_2|A_1)$ indicates the probability that A_2 occurs given that A_1 has occurred. If the two events are *independent*, the last term becomes $Pr(A_2)$

and the probability of the compound event becomes

$$Pr(A_1 \cap A_2) = Pr(A_1) \cdot Pr(A_2) \tag{2.12}$$

In general, for any k independent events, $A_1, A_2, \ldots, A_k$, the probability of the compound event A_1 AND A_2 AND ... AND A_k is

$$Pr(A_1 \cap A_2 \cap \ldots \cap A_k) = \prod_{i=1}^{k} A_i. \tag{2.13}$$

The AND rule is usually used when words like "all" or "both" are mentioned.

Simple as they are, these two rules, OR and AND, can be used to solve complex probability problems by using them *systematically.*

Example 2.6 *What is the probability of having 2, 4, or 6 when throwing a die?*

This compound event can be expressed as a composition of the simple events it contains. Let getting a 2 on the die be denoted as A_1, and similarly let the two events of getting 4 and 6 be denoted as A_2 and A_3, respectively. Then what is required is $Pr(A_1$ OR A_2 OR $A_3)$. Since $Pr(2) = Pr(4) = Pr(6) = 1/6$, and since A_1, A_2 and A_3 are mutually exclusive, then, by Equation 2.12, the probability of getting 2, 4 or 6 is

$$Pr(A_1 \cup A_2 \cup A_3) = 1/6 + 1/6 + 1/6 = 1/2.$$

Example 2.7 *What is the probability of getting a 4 or an even number when throwing a die?*

This is again a compound event involving the OR rule. Let the event of getting a 4 be denoted as A_1 and the event of getting an even number be denoted as A_2.

Clearly the two events are *not* mutually exclusive and therefore, Equation 2.10 must be applied.

From Examples 2.5 and 2.61, $Pr(A_1) = 1/6$ and $Pr(A_2) = 1/2$. The last term in the equation denotes the intersection of the two events. Clearly, the two events intersect in (have in common) the number 4 (which is again A_1) whose probability is 1/6. Substituting in Equation 2.10 the probability of getting a 4 or an even number is

$$Pr(A_1 \cup A_2) = 1/6 + 1/2 - 1/6 = 1/2.$$

Example 2.8 *What is the probability of getting at least 2 on a die?*

Here we need to evaluate $Pr(2$ OR 3 OR 4 OR 5 OR $6)$. These events are all mutually exclusive, and therefore the solution is

$$Pr(2 \cup 3 \cup 4 \cup 5 \cup 6) = 1/6 + 1/6 + 1/6 + 1/6 + 1/6 = 5/6.$$

The same result could have been obtained by evaluating $1 - Pr(1)$, which is obviously equal to 5/6. In general, problems involving *at least* can be best solved by subtracting the probability of the complementary event (getting a 1, in Example 2.8), from unity.

Example 2.9 *Assuming that the probability of a child in a particular family being male is 0.53, find the probability that in a family of 5 children,*
1. the 3 oldest will be boys and the 2 youngest will be girls, and
2. there are three boys in the family and 2 girls.

Let the event that a child will be a boy be denoted by b and the event that a child will be a girl be denoted by g. Then,

$$Pr(g) = 1 - Pr(b) = 1 - 0.53 = 0.47.$$

1. Let the probability that the three oldest will be boys and the two youngest will be girls be denoted as $Pr(1)$. Then $Pr(1)$ can be written as (not the order of the events)

$$Pr(1) = Pr(b \cap b \cap b \cap g \cap g).$$

Clearly, the sex of the new born child is *independent* from the sex of the previous child. Thus one can write

$$Pr(1) = Pr(b) \cdot Pr(b) \cdot Pr(b) \cdot Pr(g) \cdot Pr(g) = (0.53)^3(0.47)^2 = 0.033.$$

2. Let the probability that there are 3 boys and 2 girls in the family be denoted as $Pr(2)$. Then,

$$Pr(2) = Pr([b \cap b \cap b \cap g \cap g] \cup [b \cap b \cap g \cap g \cap b] \cup \ldots)$$

The probabilities of the compound events inside the brackets are all equal, namely, they are equal to $Pr(1)$ obtained in Part 1 of this example. Thus the question is: In how many ways can one arrange 5 items (children) three of which are similar (boys) and the remaining two are also similar (girls). This is the permutation problem discussed in Section 2.3.1, and the answer to the question is $\frac{5!}{3!2!}$. Although this answer is obtained from using permutations, it is the same as the number of combinations of 5 items taken 3 (or 2) at a time. Thus,

$$Pr(2) = \binom{5}{3} Pr(1) = \frac{5!}{2!3!} \cdot 0.033 = 0.33$$

This example illustrates the *binomial distribution* or *Bernoulli trials*. It is any experiment whose outcomes must be one of two things (e.g. success or failure), and whose next outcome is independent of the present one. Tossing a coin, for example, is a Bernoulli trial, because the outcome is either *heads* or *tails*, and the outcome

this time does not affect the outcome next time. The binomial distribution is used in this thesis in obtaining the reliability of the fault tolerant Clos network.

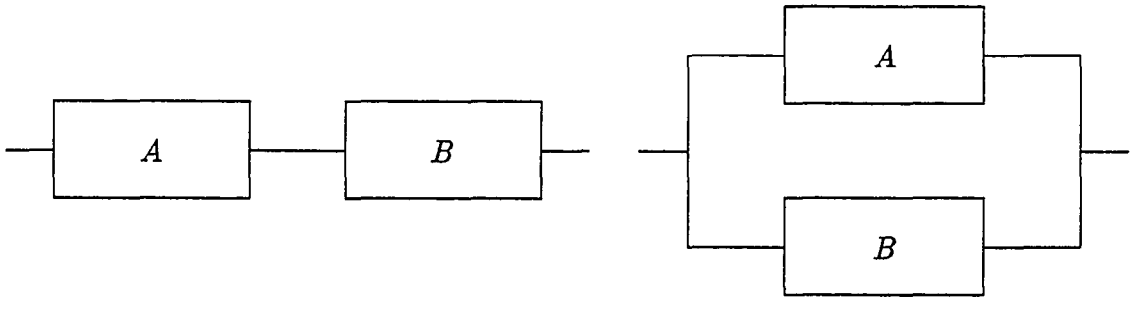
2.4.3 Reliability models

Once again, it will be shown that the AND/OR rule is useful in evaluating the reliability of a system. A system can be broken down from the reliability standpoint into isolated blocks. For simplicity, suppose that a system can be broken down into two blocks A and B. Then there are two situations possible.

If the system fails when *either* block fails, then the blocks are said to be in series and the combined system reliability is the probability that both A AND B are operational. This situation is depicted in Figure 2.2a. If the two blocks are statistically independent, i.e. the failure of one is independent from the failure of the other, then

$$R = R_1 R_2, \tag{2.14}$$

where R is the reliability of the system, R_1 is the reliability of block A and R_2 is the



a) Series system b) Parallel system

Figure 2.2: Basic reliability models

reliability of block B. In general, for a series system of n statistically independent blocks, the combined reliability of the system is

$$R = \prod_{i=1}^{n} R_i \tag{2.15}$$

On the other hand, if the system fails if *both* blocks fail, then the blocks are said to be in parallel and the combined system reliability is the probability that either A OR B are operational. This situation is depicted in Figure 2.2b. If the two blocks are statistically independent, i.e. the failure of one is independent from the failure of the other, then

$$R = R_1 + R_2 - R_1 R_2. \tag{2.16}$$

This expression can also be written as

$$R = 1 - (1 - R_1)(1 - R_2). \tag{2.17}$$

Recall that this form could have been obtained directly by considering the fact that in a parallel system, the reliability of the system is the probability that *at least* one block is operational (which is obtained by subtracting the probability that both A AND B are inoperational from unity. In general, for a parallel system of n statistically independent blocks, the combined reliability of the system is

$$R = 1 - \prod_{i=1}^{n} (1 - R_i) \tag{2.18}$$

Just as the AND and OR rules can be used to solve complex probability problems, the series and parallel reliability decompositions can be used *systematically* to find the reliability of complex systems.

Sometimes, a system has n parallel blocks, but needs at least m of them to remain operational. This problem is a binomial distribution. The reliability of the system in this case can better be expressed as unity minus the probability of the

complementary event (that is, failure occurring from having between 0 and $m-1$ operational blocks). The operational blocks are indistinguishable from each other, and so are the non-operational blocks. Recall that the way to count the number of ways these blocks can be arranged together is a combination problem (although it is originally a permutation problem.) Thus the reliability of the system is

$$R = 1 - \sum_{i=0}^{m-1} \binom{n}{i} R^i (1-R)^{n-i} \tag{2.19}$$

This formula is used in the thesis to obtain the reliability of the fault-tolerant Clos network in Chapter 6.

2.5 Notation

The following notation is used throughout the thesis.

i: general index, switch number in a network stage

j: general index, stage number in a network

$X(i,j)$: (crossbar) switch number i in stage number j

N: network size, number of inlets or outlets of a network

I: set of all inlets of a network

O: set of all outlets of a network

m: in a Clos network, number of inputs to a first-stage switch, or number of outputs to a third-stage switch

n: in a Clos network, number of outputs of a first-stage switch or number of inputs to a third-stage switch

k: in a Clos network, the number of inputs or outputs of a middle-stage switch

lg: $\log_2$, logarithm to the base 2

$\lfloor x \rfloor$: integer value less than or equal to the real number x, also called the floor of x

$\lceil x \rceil$: integer value grater than or equal to the real number x, also called the ceiling of x

S: source inlet (integer value)

D: destination outlet (integer value)

$(D)_b^\nu$: ν-bit binary representation of the integer D

a mod b: remainder after dividing the integer a by the integer b

ν: number of stages in a MIN

r: reliability of a switch over a given period of time

R: reliability of a network over a given period of time

P: permutation

$\mathcal{P}$: set or group of permutations

e: identity permutation

$\mathcal{P}_N^{(\xi)}$: set of all permutations blocked in exactly ξ paths when realized on a network of size N

$\sum_N$: symmetric group, the set of all permutations of size N

$|A|$: cardinality of A, the number of elements in the set A

ϕ: empty set

Chapter 3

MIN Implementations

Over the past three decades, a large number of MINs have been proposed. The MINs are mostly an implementation of the generalized MIN model defined in Chapter 2. Since this thesis centers on MINs, a thorough understanding of their construction, operation and routing is necessary before attempting to enhance them with fault tolerance capabilities. However, due to the large number of MIN implementations, it is difficult to cover all of them in this thesis. Instead, only the most popular implementations will be discussed.

In this chapter, the set of MINs discussed in the thesis will be presented. The generalized MIN model given in Chapter 2 will be used *systematically* to introduce and rigorously define these MINs. Moreover, the model will serve as a tool to highlight the differences among the different MIN implementations. The model is repeated here, as Equation 3.1, for convenience.

$$MIN = (I, O, F_I, F_O, F_{l_1}, F_{l_2}, \ldots, F_{l_{\nu-1}}, \mathcal{F}_{s_0}, \mathcal{F}_{s_1}, \ldots, \mathcal{F}_{s_{\nu-1}}) \tag{3.1}$$

where

- I and O are the sets of inlets and outlets, respectively,

- F_I is the permutation realized by the set of links between the inlets and the inputs of the first stage,
- F_O is the permutation realized by the set of links between the outputs of the last stage and the outlets,
- F_{l_j} is the permutation realized by the set of links between stage $j-1$ and j, $1 \leq j \leq \nu - 1$, and
- $\mathcal{F}_{s_j}$ is the set of all mappings F_{s_j} realizable by stage j, $0 \leq j \leq \nu - 1$.

Recall that the model applies only to MINs where all the inputs of stage j, $1 \leq j \leq \nu - 1$, are derived from the outputs of stage $j-1$, all the inputs of stage 0 are derived from the inlets and all the outlets are derived from the outputs of stage $\nu - 1$. Incidentally, not all MINs have this characteristic; at the end of this chapter, some of those MINs will be shown.

According to the generalized model, *all* MINs mentioned in this thesis, unless otherwise specified, have

1. $I = O$ and $|I| = |O| = N$,
2. $\nu \geq 2$,
3. switches in each stage are all of the same size,
4. for all j, $0 \leq j \leq \nu - 1$, F_{s_j} is a permutation (the implication here is that the number of inputs of any stage is equal to the number of its outputs), and
5. for all j, $0 \leq j \leq \nu - 1$, $\mathcal{F}_{s_j} \subset \Sigma_N$, where N is the MIN size (the implication here is that there is more than one switch in any stage of the MIN).

From the first point above, it is no longer necessary to refer to the size of a network as $N \times M$, where N is the number of inlets and M is the number of outlets. Since both numbers are now equal, a network of size $N \times N$, will be referred to simply as a network of size N.

3.1 The Baseline Network

The Baseline network shown in Figure 3.1 is chosen in this thesis as representative of a family of networks, called the *shuffle* family [80,87]. This family is characterized by using the same switch structure and layout. More specifically, a network of size N in this family must have $\nu = \lg N$ stages, with each stage having $N/2$ switches, each 2×2. Moreover, the switches of any stage $j+1$ can be interchanged so that the links between stages j and $j+1$, $0 \leq j \leq \nu - 1$, form a 2-shuffle of the terminals of one stage into the terminals of the other.

The c-shuffle of cq objects, where c and q are two positive integers, is formed as follows [41]. Think of the cq objects as cards in a deck. Divide the cards into c piles of q cards each. Put the piles in a row, in any arbitrary order. Pick up the top card of the first pile and put it as the first card of a new pile. Pick up the top card of the second pile and put it on top of the first card of the new pile. Repeat this process until the top card of each of the c piles is picked up. Now, visit the c piles again in the same order picking up the top card of each pile and putting it in the order it was picked up on top of the new pile. Every time the c piles are all visited, c cards are added to the new pile. Repeat this process in a circular fashion until the cards in the c piles are all picked up. Clearly, the new pile now has all cq cards. The ordered cards of the new pile represent the c-shuffle of the original deck. Figure 3.2a shows the 4-shuffle of 8 objects. The column on the left represents the original 8 objects before shuffling. These objects are divided from top to bottom

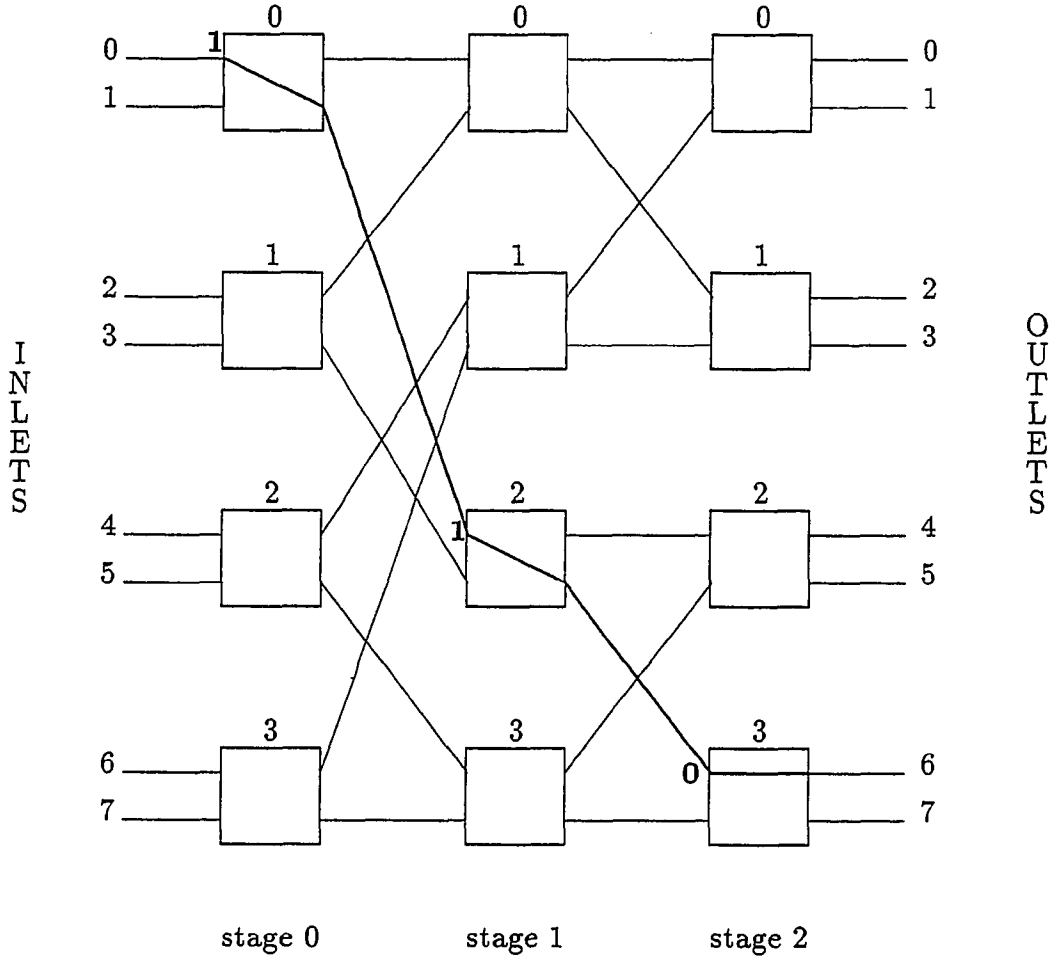
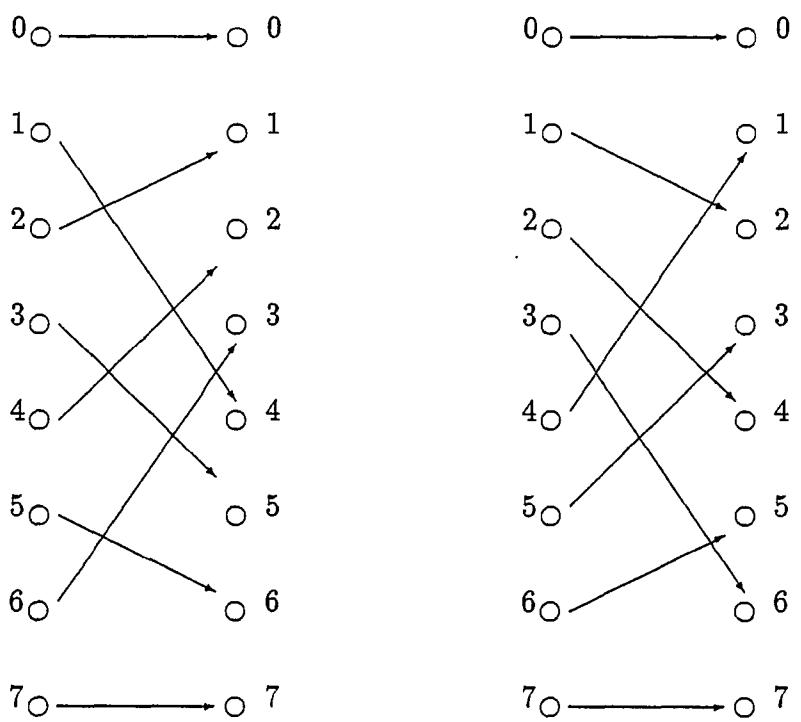


Figure 3.1: 8×8 Baseline network with routing example

into 4 sections. The arrows in the figure signify how the objects of each section were *interleaved* in the manner described above to form the 4-shuffle. The column on the right then is the 4-shuffle of the column on the left.



a) 4-shuffle of 8 elements

b) 2-shuffle of 8 elements

Figure 3.2: Shuffling 8 objects

If $c = 2$, the shuffle is called *perfect*. A perfect shuffle of 8 objects is shown in Figure 3.2b. It should be noted that a c-shuffle is the inverse of a q-shuffle. That is, if a c-shuffle is performed on N objects, the order of the N objects can be restored

by performing a q-shuffle, where $cq = N$, on the N objects. In other words,

$$(c - \text{shuffle}) \times (q - \text{shuffle}) = e.$$

This identity is demonstrated in Figure 3.2.

The 8×8 Baseline network shown in Figure 3.1 consists of three stages, each having four 2×2 switches. For the purpose of this study, the stages will be labelled from left to right as 0,1,2, and the switches in each stage will be labelled from top to bottom as 0,1,2,3. A similar numbering scheme will be assumed for all MINs in this thesis unless otherwise specified. Note also that for all MINs in this thesis, the inlets will be on the left of the MIN, whereas the outlets will be on the right. In general, a Baseline network of size N must have $N/2$ switches, each 2×2, in each stage, and the number of stages must be $\nu = \lg N$.

In reference to Equation 3.1, the Baseline network fits in the model as follows.

1. $\nu = \lg N$, where N is the network size
2. $F_I = F_O = e$, where e is the identity permutation
3. Within any stage, the switches can be rearranged so that permutations $F_{l_1}, \ldots, F_{l_{\nu-1}}$ represent either a 2-shuffle or an $N/2$-shuffle
4. $\mathcal{F}_{s_0} = \mathcal{F}_{s_1} = \ldots = \mathcal{F}_{s_{\nu-1}}$

All networks in the shuffle family can be constructed from one another [86]. For example, consider the Baseline network of Figure 3.1. It can be seen that the links between stages 0 and 1 form a perfect shuffle of the inputs of stage 1 into the outputs of stage 0. Now, interchange switches 1 and 2 of stage 2. The result will be a perfect shuffle from the inputs of stage 2 into the outputs of stage 1. If the links between the outlets and the outputs of stage 2 can now be rearranged to form

a perfect shuffle from the outlets into the outputs of stage 2, and if the inlets are used as outlets and the outlets are used as inlets, then the resulting network, shown in Figure 3.3, is called the *Omega* network [51].

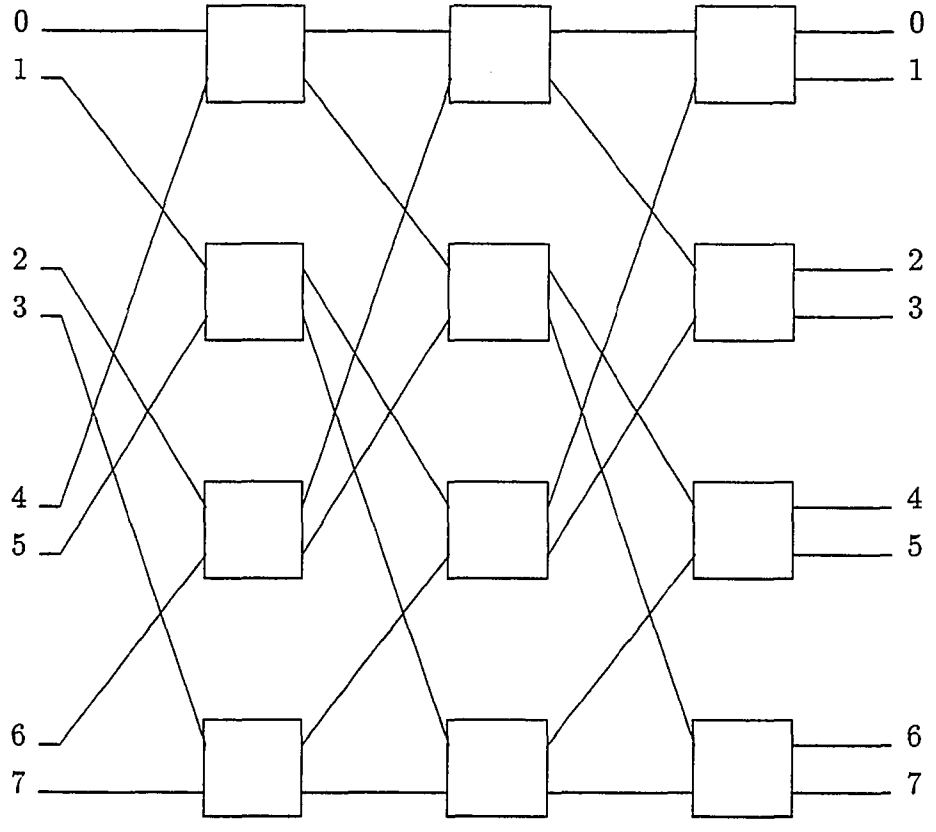


Figure 3.3: 8×8 Omega network

In reference to Equation 3.1, the Omega network fits in the model as follows.

1. $\nu = \lg N$,

2. F_I = 2-shuffle and $F_O = e$,

3. $F_{l_1} = F_{l_2} = \cdots = F_{l_{\nu-1}}$= 2-shuffle,

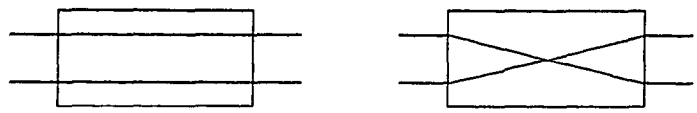
4. $\mathcal{F}_{s_0} = \mathcal{F}_{s_1} = \cdots = \mathcal{F}_{s_{\nu-1}}$.

The fact that many networks can be developed from one shuffle network by changing the link maps between the stages was recognized after many networks in the shuffle family had been proposed. Examples of such networks are the Baseline [85], Generalized Cube [77], Indirect Binary n-cube [71], Omega [51], Shuffle-exchange [81], STARAN™ flip [11] and SW-banyan (S=F=2) [40] networks. These networks seemed different at the beginning, but later they were proven to be topologically equivalent [77,78,86]. Therefore, when one needs to speak about the shuffle family, it is sufficient to speak about only one member of the family. As mentioned earlier, the member selected in this thesis is the Baseline network.

It should be noted that the Baseline network is a special case of the *delta network* [69], with $a = b = 2$ and 2-shuffle for the link maps between the stages. The delta network is a generalization of the shuffle family. Its basic components are delta elements, which are $a \times b$ crossbar switches, with c-shuffles between the stages.

3.1.1 Routing the Baseline network

The Baseline network is built from 2×2 crossbar switches. Other names for 2×2 switches are beta elements, binary cells, binary switches, binary modules, interchange boxes, and exchange boxes. The two names used in this thesis are the 2×2 switch and binary switch. A binary switch can assume one of two legal states, shown in Figure 3.4. In the *straight* state, the upper input is connected to the upper output and the lower input is connected to the lower output. In the *cross* state, the upper input is connected to the lower output and the lower input is connected to the upper output. A switch assumes one of its legal states based on the *routing*



a) straight state b) cross state

Figure 3.4: Legal states of the binary switch

bits on its inputs.

An input is connected to one of the two outputs depending upon whether the routing bit is 1 or 0. Normally, if the bit is 0, the input is connected to the upper output; and if the bit is 1, the input is connected to the lower output. Therefore, the upper output of a binary switch is sometimes referred to as the 0 output while the lower output is referred to as the 1 output. A problem arises if the two inputs of a switch have identical routing bits. In such a case the two inputs would be in effect asking for the same output, giving rise to a *conflict.* Put differently, a conflict occurs when trying to put the switch in *both* of its legal states simultaneously. Since only one input can be connected to a given output, one of the two inputs will not be connected to any output; it will be *blocked.* It is obvious then that the Baseline network is a *blocking* network; not all sets of paths can be established between the inlets and outlets. One can also easily verify from Figure 3.1 that the path between any given inlet and any given outlet is *unique.*

Routing the Baseline network is carried out in a distributed fashion. To establish a path between inlet S and outlet D, one only has to send the ν-bit binary representation of D on inlet S and let the individual bits control the switches they

traverse as they pass from one stage to another. Let $(D)_b^\nu$ be the ν-bit binary representation of the integer D. That is,

$$(D)_b^\nu = d_{\nu-1}, d_{\nu-2}, \ldots, d_0.$$

Then bit d_i will be used to control a switch in stage $\nu - 1 - i$, $0 \leq i \leq \nu - 1$. This is called *distributed* routing, and is the main advantage of the Baseline network. A demonstration of this routing technique is shown in Figure 3.1. The thick line represents a path established between inlet 0 and outlet 6 by putting the 3-bit binary representation of 6, 110, on inlet 0. Each bit in this *routing tag* is shown next to the switch it controls. The time complexity of the distributed routing scheme of the Baseline network is $O(\lg N)$.

The disadvantage of the distributed routing, and hence the disadvantage of the Baseline network, is that it cannot realize every permutation in the symmetric group, Σ_N; only a family of permutations can be realized. This family has already been identified and found [1] to include many of the permutations often needed [53] in parallel processing.

3.2 The Clos Network

A Clos network of size 8 is shown in Figure 3.5. It has three stages, which can be numbered 0, 1, and 2 from the input side to the output side, respectively. Stage 0 has four switches, each 2×2, stage 1 has two switches, each 4×4, and stage 2 has four switches, each 2×2.

Clos networks in general have three stages. A Clos network of size N must have $k = N/m$ switches, each $m \times n$, in stage 0, and $k = N/m$ switches, each $n \times m$, in stage 2. All three stages are connected by inter-stage links in such a way that a switch in a given stage has access to all the switches in the next stage. Since there

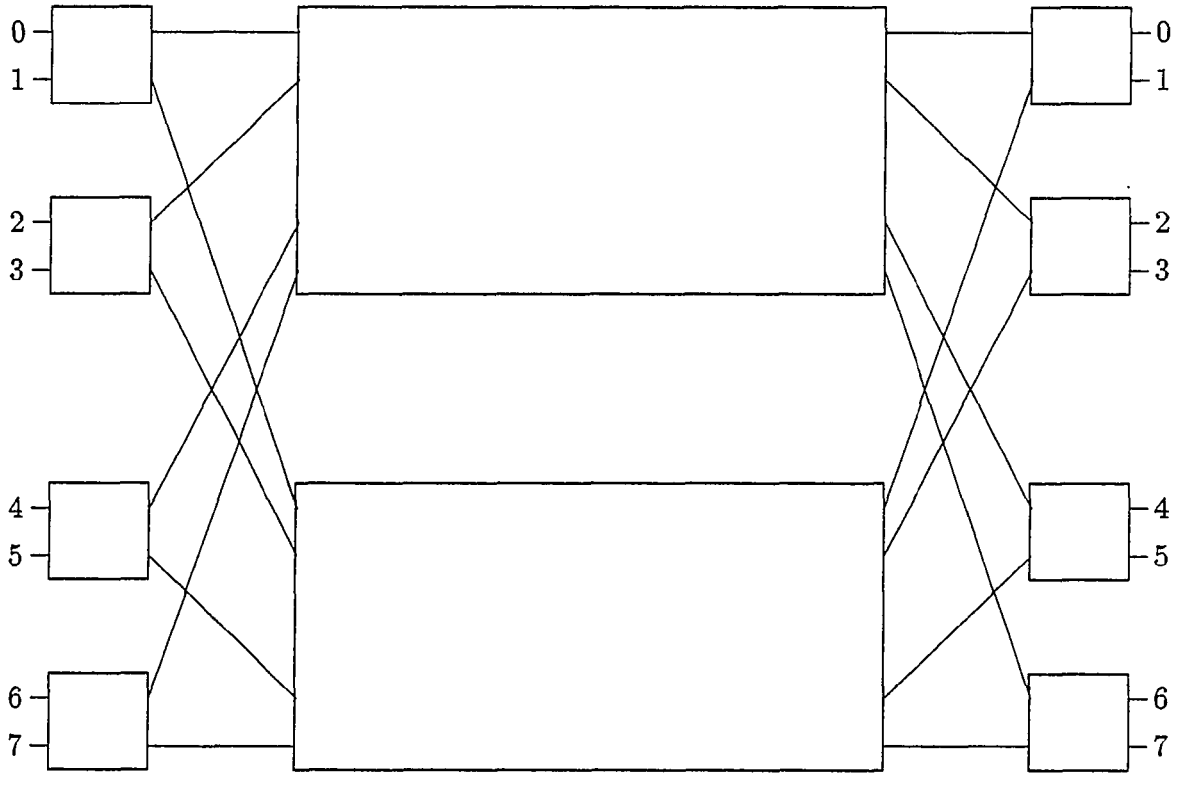


Figure 3.5: 8×8 ordinary Clos network

is a link from each switch in stage 0 or stage 2 to every switch in stage 1, there are exactly n switches, each $k \times k$, in stage 1. It should be noted that in Clos networks, $n \geq m$. In this thesis, the term *ordinary* Clos network, or just the Clos network, will refer to the case where $n = m$. When $n > m$, some degree of fault tolerance is obtained, a fact utilized by the work of this thesis.

In reference to Equation 3.1, the Clos network fits in the model as follows.

1. $\nu = 3$
2. $F_I = F_O = e$
3. $F_{l_1} F_{l_2} = e$
4. $\mathcal{F}_{s_0} = \mathcal{F}_{s_2}$
5. $\mathcal{F}_{s_0} F_{l_1} \mathcal{F}_{s_1} F_{l_2} \mathcal{F}_{s_2} = \Sigma_N$

3.2.1 Routing the Clos network

In the Clos network, there is a central routing unit whose function is to receive a mapping, usually a permutation, and to find the proper setting for each individual switch to realize that permutation. This routing task turns out to be the main drawback of Clos networks. Setting the switches of stages 0 and 2 first and then trying to set the switches of stage 1 is not the right procedure, as conflicts will arise in stage 1. The proper procedure is to start by setting the switches of stage 1, then the switches of stages 0 and 2 can be set accordingly. However, finding the right settings for the switches of stage 1 such that no conflict occurs is not a trivial task. Three approaches have been explored in the literature for solving this problem: the group theoretic approach [66], the direct matrix decomposition approach [22], and the graph theoretic approach [54]. The group theoretic approach has been found

[17] unfeasible and therefore will not be discussed here. The other two approaches will be illustrated by way of an example.

Suppose it is required to realize on the Clos network of Figure 3.5 the following permutation

$$P = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 4 & 3 & 2 & 1 & 5 & 0 & 7 & 6 \end{pmatrix}.$$

The direct matrix decomposition approach starts by constructing an $N \times N$ matrix, I, from the permutation to be realized above as follows.

$$I[i,j] = \begin{cases} 1 & \text{if inlet } i \text{ is to be routed to outlet } j \\ 0 & \text{otherwise} \end{cases}$$

where $I[i,j]$ is the element of matrix I in row i and column j, $0 \leq i,j \leq N-1$. Thus for permutation P above,

$$I = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}.$$

The next step is to partition I into $k \times k$ quadrants, $m \times m$ each, and then construct a $k \times k$ matrix, H_m, from I as follows.

$$H_m[i,j] = \text{ sum of the } m \times m \text{ entries in quadrant } i,j \text{ of } I.$$

Thus, for the example under consideration,

$$H_2 = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix}.$$

Decomposing the H_m matrix into m matrices, $k \times k$ each, such that each row or column in any of the m matrices has only one 1 and all the other entries are 0's,

gives the proper settings for the switches of stage 1. For the example above, since the matrix is so small, this decomposition can be performed by inspection to yield

$$H_2 = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} + \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

The two matrices to the right represent the settings of the two switches in stage 1 of the Clos network of Figure 3.5. A 1 in row i and column j, indicates that input i of the switch is to be connected to output j of the same switch. The settings resulting from decomposing H_m insure that no conflict will occur in stage 1 and that all required paths specified by the permutation will be accommodated. Once the switches of stage 1 have been set, the switches of stages 0 and 1 can be easily set to complete realization of the permutation. Many algorithms have been proposed to decompose H_m in the general case.

One such algorithm is Neiman's algorithm [64]. Neiman's algorithm consists of the following two steps.

1. Starting with the leftmost column, mark in each column of H_m a non-zero element, such that no two non-zero elements are marked in the same row. If a column is encountered where no such an element can be marked proceed to the next column. If k elements are marked this way, then the algorithm is done; otherwise the second step below must be performed.

2. Assume there are x marked elements, $x < k$, from previous marking operations. Mark a non-zero element in the column where no element could be marked before and unmark the previously marked element in that row. Visit the column where an element has just been unmarked and mark an element in that column, in a row where there is no element marked. Keep this process of marking and unmarking, following the rule that there cannot be two marked

elements in the same row or the same column and that rows and columns cannot be revisited, until $x + 1$ elements are successfully marked.

It should be clear then that every time Step 2 is performed, one more element is marked. Hence, Step 2 must be repeated until k elements are successfully marked.

The k marked elements now represent the setting for one of the m switches of stage 1. A marked element in row i and column j means that input i of the switch is to be connected to output j of the same switch. Each marked element in H_m is then decremented by one to obtain H_{m-1}. Then the algorithm is applied to H_{m-1} to obtain the setting for another switch and also H_{m-2} as before. This process is repeated until H_1 is obtained. H_1 itself will represent the setting for a switch in stage 1. Clearly, to obtain the settings for all the m switches of stage 1, the above algorithm is performed $m - 1$ times (notice that the mth time is not needed). An analysis of Neiman's algorithm shows that it runs in $O(mk^4)$ time [44]. This time complexity is rather large, especially for large k.

Two other algorithms have been proposed to decompose H_m: Ramanujam's algorithms [73] and Jajszczyk's algorithm [44]. However, both of them were later proven wrong [16,48]. The reason why the two algorithms fail has been identified and a solution to make them succeed has been proposed [17].

On the other hand, the graph theoretic approach to finding the settings of the switches of stage 1 starts by treating each switch in stages 0 and 2 as a *vertex* in a multigraph G. Let the set of switches of stage 0 be denoted as V_0 and the set of switches of stage 2 be denoted as V_2. Then, given a permutation, P, an *edge* is stretched between vertex i and vertex j if an inlet attached to switch i of stage 0 is to be routed to an outlet attached to switch j of stage 2. The result of this is

the bipartite multigraph $G = (V_0, V_2, E)$, where E is the set of edges between V_0 and V_2. G is a multigraph since multiple edges between vertices are allowed, and is bipartite since each edge in G is incident on two vertices, one in V_0 and the other in V_2. The degree of G (the number of edges incident on any vertex) is clearly m. The graph theoretic approach then decomposes G into m subgraphs of degree 1 each. Each such subgraph will represent the setting of one of the m switches of stage 1.

For the example above, the graph representing permutation P is shown in Figure 3.6 with its two equivalent subgraphs. The two subgraphs in the figure represent the settings for the two switches of stage 1 of the Clos network of Figure 3.5. An edge in a subgraph between vertex i, $i \in V_0$ and vertex j, $j \in V_2$, indicates that input i of the switch is to be connected to output j of the same switch. These settings insure that no conflict will occur in stage 1 and that all required paths specified by the permutation will be accommodated. Many algorithms have been proposed to decompose G in the general case.

One such algorithm uses *matching* [43] to extract the individual subgraphs. A matching is a set of edges in G such that no two are incident on the same vertex. This algorithm runs in $O(k^{2.5})$ time. Other algorithms also exist where techniques such as *edge coloring* [18,84] and *Euler partitioning* [38] are used. The graph-based algorithms are outside the scope of this thesis.

The two routing approaches mentioned above have been discussed extensively in the literature and the graph theoretic techniques have always been described as more efficient. Recently, however, it has been found that both edge coloring and direct matrix decomposition approaches are equivalent [19]. This finding may well lead to a new, unified routing algorithm that makes the Clos network particularly suitable for processor interconnection in large-scale multiprocessor systems.

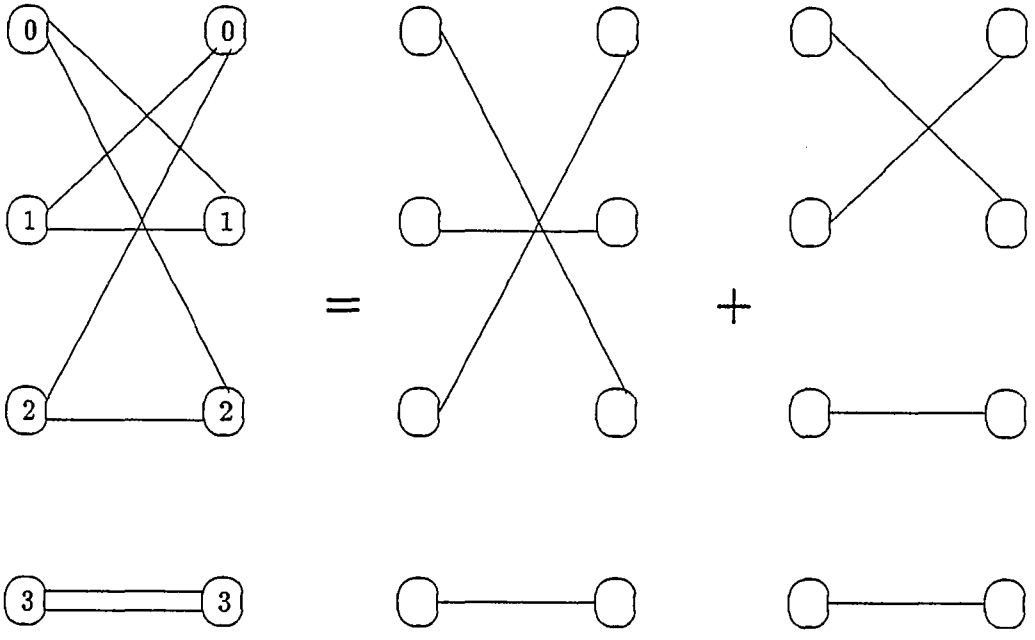


Figure 3.6: Graph representation of permutation P

3.3 The Beneš Network

The Beneš network can be developed from a Clos network with $n = m = 2$ and $k = 2^t$, for some positive integer $t > 1$, by recursively decomposing the middle stage into a 3-stage Clos subnetwork whose outer stages contain only 2×2 switches. If this decomposition is continued until every switch in the network is of size 2×2, the resulting network is called the Beneš network [12,13].

Consider for example the Clos network shown in Figure 3.5. Replace each of the two switches in stage 1 by a 3-stage Clos subnetwork. Then the resulting network, shown in Figure 3.7, is called the Beneš network. There are 5 stages in the network, each containing 4 switches. In general, a Beneš network of size N must

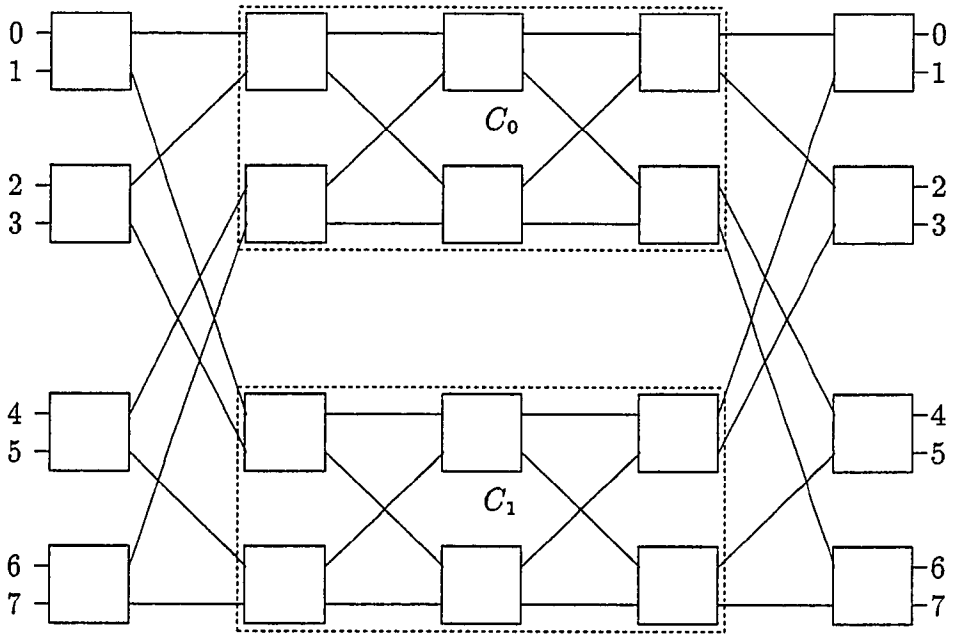


Figure 3.7: 8×8 Beneš network

have $\nu = 2(\lg N) - 1$ stages, with each stage having $N/2$ switches, each 2×2.

In reference to Equation 3.1, the Beneš network fits in the model as follows.

1. $\nu = 2(\lg N) - 1$

2. $F_I = F_O = e$

3. $F_{l_j} F_{l_{\nu-j}} = e$, for all j, $1 \leq j \leq (\nu - 1)/2$

4. $\mathcal{F}_{s_0} = \mathcal{F}_{s_1} = \ldots = \mathcal{F}_{s_{\nu-1}}$

It is interesting to note that the Beneš network can be viewed as two back-to-back Baseline networks, with one of the two middle stages eliminated. At the beginning of this section, the relationship between the Clos network and the Beneš network was mentioned. It should, therefore, be obvious that the three networks used in this thesis, the Baseline network, the Clos network and the Beneš network, are closely related.

3.3.1 Routing the Beneš network

Many algorithms have been proposed for routing the Beneš network. In general two methods can be applied: central routing or distributed routing.

The best known central routing algorithm is the *looping* algorithm [7,65]. It uses the fact that at any stage j, $0 \leq j \leq (\nu - 1)/2$, the stages between j and $2(\lg N) - j - 2$ form two Beneš subnetworks of size $N/2^j$ each. The two subnetworks between stages 0 and 4 of the Beneš network of Figure 3.7 are enclosed in a dashed box and denoted C_0 and C_1 for easy reference. If the proper paths are routed through the proper subnetwork, no blocking will occur and random permutations can be realized conflict free. To understand how the looping algorithm works, the *dual* of a number is first defined. Given two integers a and b, a is said to be the dual of b, denoted as $a = \overline{b}$, if $\lfloor a \rfloor = \lfloor b \rfloor$. For instance, 0 is the dual of 1, and 7 is the dual of 6, and so on. The looping algorithm says that if no two dual inlets or dual outlets are routed through the same subnetwork, then any permutation can be

realized on the network conflict-free. The algorithm starts by assigning an element from the permutation arbitrarily to either subnetwork, and then it divides all other elements between the two subnetworks on the condition that no dual inlets or dual outlets go through the same subnetwork. This will be illustrated by an example.

Suppose it is required to realize the same permutation P as before. P is repeated here for convenience.

$$P = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 4 & 3 & 2 & 1 & 5 & 0 & 7 & 6 \end{pmatrix}.$$

First, arbitrarily assign $\binom{0}{4}$ to C_0. Since outlet 4 goes now through C_0, its dual, outlet 5, must go through C_1. Thus $\binom{4}{5}$ goes to C_1. Now, since inlet 4 goes through C_1, its dual, inlet 5, must go through C_0. As a result, $\binom{5}{0}$ is assigned to C_1. This procedure is repeated until all elements of the permutation are processed. For the example under consideration the elements of P will be divided between the two subnetworks as follows.

$$C_0 = \begin{pmatrix} 0 & 2 & 5 & 6 \\ 4 & 2 & 0 & 7 \end{pmatrix} \text{ and } C_1 = \begin{pmatrix} 4 & 3 & 1 & 7 \\ 5 & 1 & 3 & 6 \end{pmatrix}.$$

The two outer stages, stages 0 and 4, will be set according to the assignments made to C_0 and C_1. This procedure is repeated until all the switches are set. The time complexity of the looping algorithm is $O(N \lg N)$.

The Beneš network can also be routed in a distributed fashion [63] in much the same way as the Baseline is routed. The time complexity of the distributed routing algorithm is $O(\lg N)$. The tradeoff, however, is that not every permutation in Σ_N can be realized on a network of size N. That is, the Beneš network becomes blocking if routed in a distributed fashion. Recall that the same network is non-blocking if routed centrally. It should also be noted that a group theoretic approach has been suggested [20] to route the Beneš network. The results are promising and

can eventually lead to a linear-time set up algorithm. If routing is performed on a multiprocessor, as opposed to a uniprocessor as has been the case in the above algorithms, a great deal of routing time can be saved [21].

3.4 Other MIN Implementations

Thus far three MIN implementations have been discussed in some detail: the Baseline network, the Clos network and the Beneš network. These three MINs will be the focus of the thesis because of their popularity. A great deal of research has been done to investigate their properties. Furthermore, these three MINs have well-established routing algorithms and their area complexity is highly acceptable. There are, however, many other types of MINs that can be found in the literature. One such MIN is the BBC network [9].

A BBC network of size 6 is shown in Figure 3.8. In general, a BBC network of size N must have $N-1$ stages of one switch each. Each stage produces exactly one outlet, except the last stage which produces two outlets. For this reason, the BBC does not fit in the generalized MIN model which requires that all outlets be derived from the last stage. The BBC realizes a permutation P of size N one element at a time throughout the network except at the last stage where two elements are realized simultaneously. Consider for example the BBC of Figure 3.8 and consider a size 6 permutation. The first stage connects only one of the 6 inlets to outlet 0 and passes the rest of the permutation to the second stage. The second stage connects one of its inputs, according to the permutation to outlet 1. This process is continued until the permutation is fully realized.

More details on the BBC and other MINs with regular geometries can be found in [17].

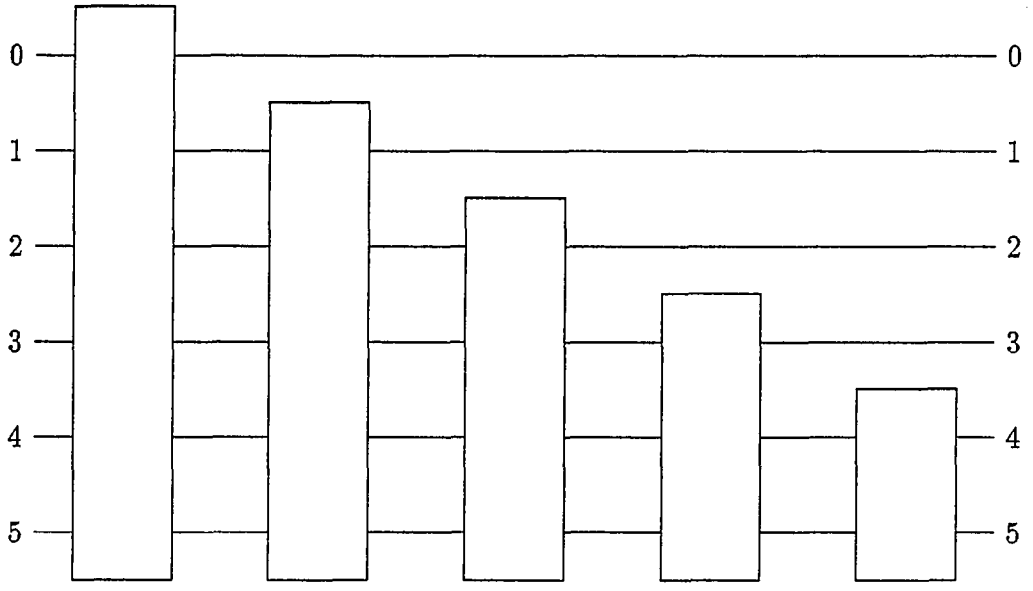


Figure 3.8: 6×6 BBC network

3.5 The Crossbar Switch

Evidently, the building block of any MIN is the crossbar switch. This switch has been shown so far as a simple box. However, the switch is not as simple as it looks – even for the smallest size, 2×2. The complexity of the switch is due primarily to the need for a control unit attached to the switch which can "talk" either to a central routing unit, in the case of a centrally-routed MIN, or to switches in other stages, in the case of a self-routed MIN. For this reason, most of the switch implementations proposed in the literature have been for binary switches.

One such implementation [69] is shown in Figure 3.9. It consists of two blocks, the DATA block and the CONTROL block. The bidirectional arrows connected to the DATA block represent the data bus and Read/Write control lines. It should be noted that in distributed routing the data bus is used at the beginning of each

memory cycle to carry the routing tag of the destination. The DATA block is the main element of the switch; this is the place where the data passes from the input of the switch to its output. The inputs of the switch are connected to the outputs, either in the straight or the cross state as explained earlier, based on information received from the CONTROL block.

Connected to the CONTROL block are two sets of 1-bit control lines corresponding to the two sets of data lines of the DATA block. These lines act as signaling lines between neighboring stages to help set up a new path. There are three such

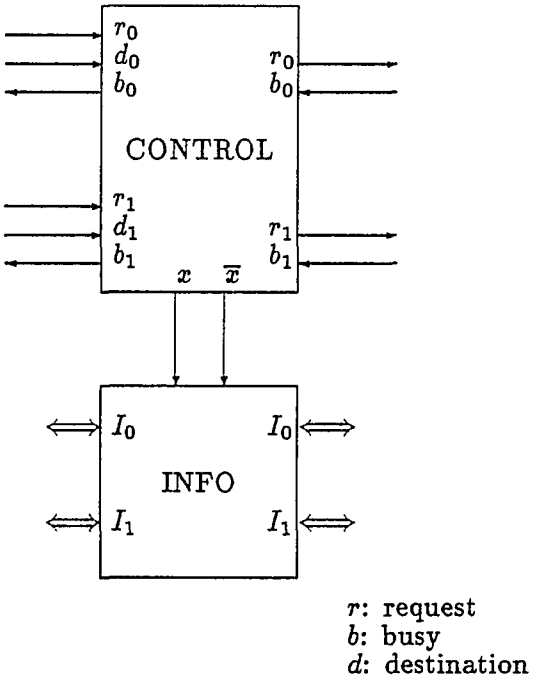


Figure 3.9: Implementation of a binary switch

signaling lines: request (r), busy (b) and destination (d). Lines of the same type on each switch are connected to their counterparts in the previous and next stages. The signaling lines on the input side of the switches of the input stage are connected

to the processors, and the signaling lines on the output side of the switches of the last stage are connected to the memory modules. The operation of these signaling lines is described below.

The binary switch described above operates in a network of ν stages as follows. All processors requiring memory access place a logic 1 on the *request* lines of stage 0 and place the address of the destination on the data lines. The request propagates from one stage to another from the input side of the network to the output side. Once the request signal reaches a switch, the switch investigates its destination line. The destination line of a switch in stage j is connected to data line $\nu - j - 1$ of the data bus input to the switch. The switch is put in one of its legal states, straight or cross, based on one of the two destination lines d_0 and d_1 (arbitration is used if there is a conflict). Lines x and $\overline{x}$ carry the connection decision from the CONTROL block to the DATA block. They are two lines, and not one, for hardware considerations.

The busy line goes high if the connection requested for a data bus is denied by the switch due to a conflict. This busy signal propagates backwards towards the input of the network and finally to the processor requesting the connection. After 8ν gate delays, the *busy* line attached to the processor is valid and the processor can investigate it. If the busy line is 0, it means the route has been successfully set up. On the other hand, if the busy line is 1, the processor must try again later, as a conflict has happened somewhere on the way to the destination.

Another implementation of a binary switch is reported in [10], while a fault tolerant implementation is given in [55]. In the latter implementation, a built-in fault detection capability through data bits checking is impeded at each switch. In this implementation the switch ends up having 88 pins for a data bus of 16 bits, because it uses more control lines than the one shown in Figure 3.9.

Chapter 4

Fault Tolerant MINs

In this chapter, an overview of some fault tolerance techniques that have appeared in the literature for MINs will be presented. The advantages and shortcomings of each design will be highlighted. This will help explain the problem of fault tolerance, and thus will facilitate its solution.

In general, MINs can be made fault tolerant by adding extra hardware. An obvious approach then is to fully duplicate the MIN (100% redundancy). Here two MINs are put in parallel, with one being active and the other being standby. If a fault occurs, the standby MIN is switched in and the faulty MIN is switched out, and normal operation resumes. The advantage of this approach is that performance remains the same under faulty conditions as under normal (no faults) conditions. The disadvantage of duplication is the increased cost and size of the system. To keep the cost and size of the system at a minimum, one must search for a solution other than duplication. Adding extensive hardware usually decreases performance degradation under faulty condition, but increases the cost and size. Adding little hardware, on the other hand, increases performance degradation under faulty conditions but keeps the cost and size down. As a consequence, a compromise must be made where the tradeoffs are weighed carefully and the best design is reached. A good fault tolerance technique is one that needs minimal hardware and causes

minimal performance degradation under faulty conditions. Needless to say, any fault tolerance technique should cause no performance degradation under normal conditions.

Recognizing the need for fault tolerance in multiprocessor systems, a number of fault tolerant MINs have recently been suggested. The details of these techniques depend mainly on the type of network and the fault tolerance model used. For example, fault tolerance has been provided for the shuffle family by adding an extra stage [2,3,33,88], by adding extra interstage links and using non-binary switches as a result [27,45,67], or by adding intrastage links and modifying the switch design [49]. Fault tolerance has also been provided for some other network architectures [4,52,39,59,72,75,76] through various approaches. All these techniques offer some level of fault tolerance to the MIN by avoiding the costly approach – duplication.

Unfortunately, most of the fault tolerant techniques cited above are MIN-specific. Moreover, most of these techniques are suggested only for the shuffle family. Despite an extensive search in the literature, it has not been possible to find any fault tolerant work done for either the Clos or the Beneš networks. If there is really no such work, this thesis may well be the first attempt in that direction. In this thesis, a fault tolerance technique that is not MIN-specific is developed. As such, it can be used with either Beneš networks or Clos networks, or any network that can fit in the generalized MIN model. The generalized technique, however, is most efficient for MINs with small switch sizes, e.g. binary switches. Since the Clos network is characterized by using large switches, the generalized technique becomes less efficient. It is for this reason that another technique will be presented in Chapter 6 to provide fault tolerance for the Clos network. Together, the two techniques should offer a reasonably comprehensive solution to the fault tolerance problem in a great number of MINs.

With this in mind, two fault tolerant MINs that have recently been suggested will now be presented. The construction of each MIN will be described as well as the fault tolerance model used in its design and the recovery method. The advantages and disadvantage of each MIN will also be examined.

4.1 The Extra Stage Cube

The Extra Stage Cube (ESC) has been suggested [2] for the shuffle family. It is demonstrated on a variation of that family, the Generalized Cube network, in Figure 4.1. Stage 3 in the figure is the extra stage. It should be noted here that the stage numbering scheme in this figure is from right to left, opposite to the scheme adopted everywhere else in this thesis. The position of the inlets and outlets, however, conforms with the convention adopted in this thesis, that is, to the left and right of the MIN, respectively. The inlets are connected to 1×2 demultiplexers (shown as little boxes). One of the two outputs of each demultiplexer is connected to a switch in stage 3, and the other output is connected to a multiplexer on the other side of the switch. The use of multiplexers and demultiplexers in fault tolerant MINs seems inevitable. They can be looked upon as switches choosing one of many target lines on one side at a time and connecting it to a single line on the other side. They are usually provided with *selection* (or control) lines to select a specific target line. Thus in the ESC, stage 3 as a whole can be bypassed if the target lines chosen by the demultiplexers and multiplexers are not those attached to the switches. This is the idea behind the ESC – that a stage can be switched in or out at will.

4.1.1 Operation and fault tolerance model

It should be noted that the generalized cube generates its routing tag, T, as the bit-wise exclusive-or of the two integers representing the source and destination.

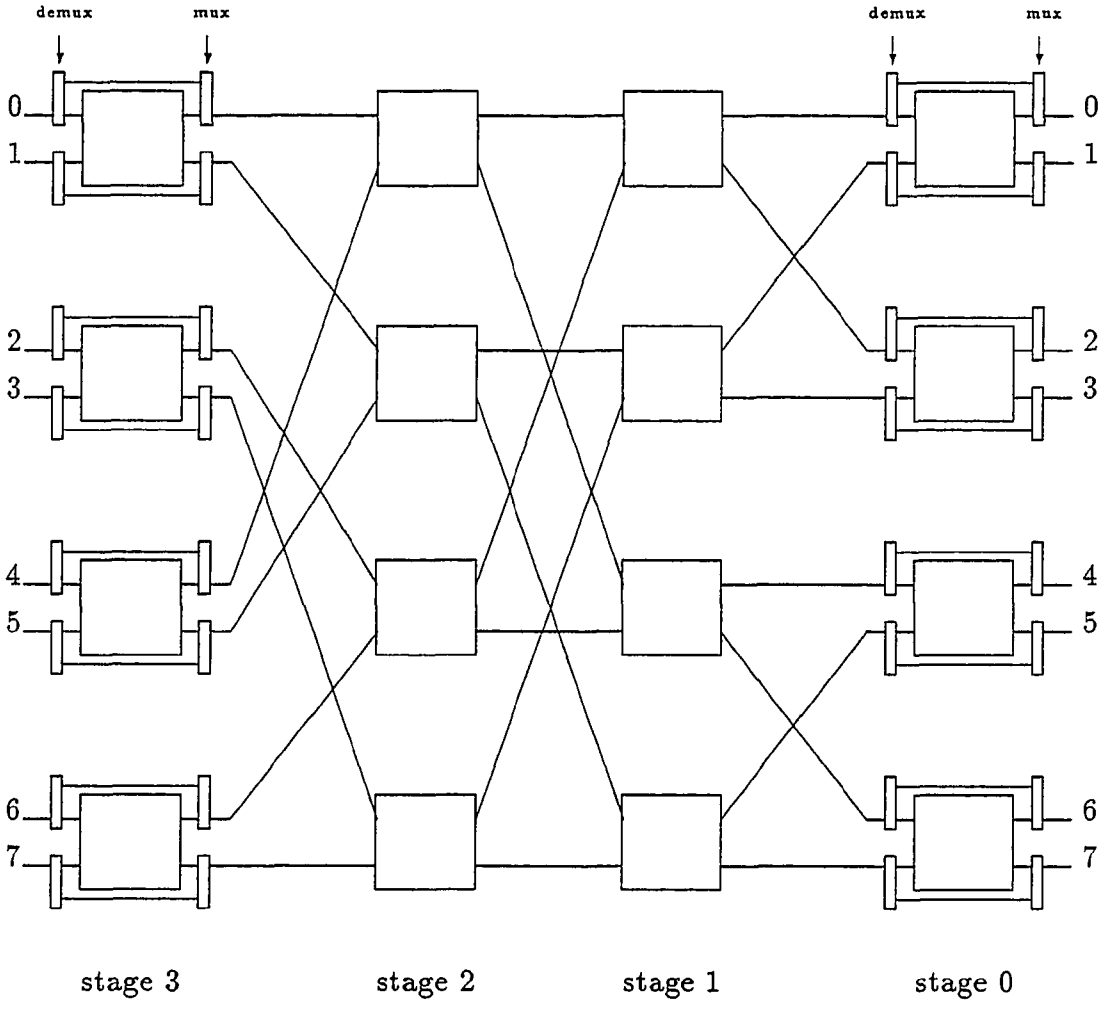


Figure 4.1: The Extra Stage Cube (ESC)

Suppose it is required to establish a path between inlet S and outlet D. Then the routing tag will be

$$T = S \oplus D = t_{\nu-1} \ldots t_1 t_0,$$

where $\nu = \lg N$ is the number of stages in the generalized cube. A switch in stage i needs only examine t_i. If $t_i = 0$, the straight state of the switch is assumed. If $t_i = 1$, the cross state is assumed. As is the case with any network in the shuffle family, if the two inputs of the switch have routing bits that try to put the switch in the two states simultaneously, only one will be given priority and the other will be blocked.

The ESC, on the other hand, is normally set with stage ν disabled (bypassed) and stage 0 enabled. If a fault occurs in stage 0, stage 0 is disabled and stage ν is enabled. If a fault occurs in stage x, $0 < x < \nu$, then both stages 0 and ν are enabled. It should be noted that stage 0 can also be switched in or out of the MIN by adjusting the multiplexers and demultiplexers around that stage. Having an extra stage in the network offers two paths between any inlet/outlet pair. One path, the *primary* path, corresponds to the normal path that would otherwise be established on the normal cube, and the other path, the *secondary* path is the one used when stage ν is enabled (in case a fault occurs in the network).

Since each bit in the routing tag controls a switch in the network, and since the switches to be controlled change according to the location of the fault, a dynamic $(\nu + 1)$-bit routing tag, $\acute{T}$, other than the normal T must be generated by the processors for each path desired. Table 4.1 shows these routing tags for all possible fault locations, where $\acute{t}_\nu$ is a dummy bit that can be assigned any arbitrary value.

It is evident that a processor must know the exact location (stage) of the fault so that it can generate the proper routing tag. Thus once a fault is detected and located, after running some test, all processors must be notified of the location of

Fault location	Routing tag $\acute{T}$
No Fault	$\acute{T} = \acute{t}_\nu t_{\nu-1} \ldots t_1 t_0$
Stage 0	$\acute{T} = t_0 t_{\nu-1} \ldots t_1 t_0$
Stage x, $0 < x < \nu$	$\acute{T} = \begin{cases} 0 t_{\nu-1} \ldots t_1 t_0 & \text{If primary path is fault free} \\ 1 t_{\nu-1} \ldots t_1 \overline{t_0} & \text{If primary path is faulty} \end{cases}$

Table 4.1: Routing Tags for the ESC

the fault. It is assumed that there is an external hardware unit that will enable stage ν if there is a fault in the network.

The fault model in the ESC is as follows.

1. Any network component can fail

2. Faulty components are unusable

3. Faults occur independently

4. Multiplexers and demultiplexers as well as links attached to them cannot fail

The fault size of the ESC is 1, and the fault tolerance criterion is full access retention, that is, any inlet must remain capable of accessing any outlet after the ESC recovers from a fault. The ESC is robust in the presence of multiple faults.

The main advantage of the ESC is ease of operation; the multiplexers and demultiplexers have to be adjusted only once after a fault occurs. Another advantage is that the ESC does not need specially designed switches; the normal binary switches are still used. Other than links associated with the multiplexers and demultiplexers, no additional links, interstage or intrastage, are needed.

The shortcomings of the ESC, however, can be summarized as follows. First, for a MIN of size N, $N/2$ extra switches are needed in addition to $2N$ multiplexers and $2N$ demultiplexers. It should be mentioned that the ESC was later modified [3] where the demultiplexers at the input of the network were replaced by dual

input ports and the multiplexers at the output of the network were replaced by dual output ports. Second, to enable stage ν after a fault occurs, there must be an external hardware unit to adjust all the multiplexers and demultiplexers around the stage so that data is routed through stage ν rather than around it. Third, the ESC cannot realize a permutation after it recovers from a fault in any stage except stage 0. For example, if a switch in stage x fails, $0 < x \leq \nu - 1$, the maximum number of paths that can be realized simultaneously will be $N - 2$, where N is the size of the network. Fourth, the Extra Stage technique is relatively MIN-specific; it works only with networks of the shuffle family, or any network where the link maps between the stages are the same or can be made the same by rearranging the switches within the same stage. Fifth, after recovering from a fault, time is needed, before generating a new routing tag, to find if the fault lies on the primary path or not. This time constitutes performance degradation, as it slows down the system.

As a whole, the Extra Stage technique is one of the best fault tolerant MINs published. It has been adapted for two other networks: the delta network (introduced in Chapter 3) [33] and the gamma network [88] which is also a shuffle network.

4.2 Augmented Shuffle-Exchange MIN

Before discussing this design, a word on the structural layout of shuffle networks is in order. Shuffle networks are characterized by having switches in each stage forming groups called *conjugate subsets*. Each conjugate subset of a stage leads to a unique subset of outlets. The two outlet subsets reachable from two conjugate subsets of a given stage are disjoint. To illustrate this, consider the Baseline network of Figure 4.2. Outlets 0,1,2 and 3 can be reached from either switch $X(0,1)$ or switch $X(1,1)$ in stage 1. Therefore, switches $X(0,1)$ and $X(1,1)$ belong to the same conjugate subset. On the other hand, outlets 4,5,6 and 7 can be reached from

either switch $X(2,1)$ or switch $X(3,1)$ in stage 1. Therefore, switches $X(2,1)$ and $X(3,1)$ belong to the same conjugate subset. It is clear then that stage 2 is made up of two conjugate subsets. Notice that the two subsets of outlets reachable from these two conjugate subsets of switches are disjoint. Consider now stages 0 and 1. Since all the outlets are reachable from any switch in stage 0, then all switches of stage 0 belong to the same conjugate subset. On the other hand, stage 2 can be easily seen to have 4 conjugate subsets, one switch each. It should be noted that a conjugate subset in stage j, $0 \leq j \leq \nu - 2$ has access to exactly two conjugate subsets in stage $j+1$. More about conjugate subsets can be found in Chapter 5.

The Augmented Shuffle Exchange Network (ASEN) is another MIN-specific fault tolerance scheme proposed [49] for the shuffle family of network. It is demonstrated in Figure 4.3 on a network close to the generalized cube discussed in Section 1. Although the ASEN published does not show explicitly the demultiplexers connected to the inlets and the multiplexers connected to the outlets, they are implicitly present. Therefore, they are explicitly shown in Figure 4.3. This will facilitate performing comparisons between different design shown in this thesis.

The ASEN replaces the binary switches of a shuffle network by 3×3 switches, which are similar in operation to the binary switch but with an auxiliary input and an auxiliary output. The switches of any conjugate subset can always be recognized as two groups, with each group having access to all the switches in the two conjugate subsets in the next stage. The ASEN links together such a group using the auxiliary terminals to form a loop as shown in Figure 4.3. For instance, switches $X(0,0)$ and $X(1,0)$ of the only conjugate subset in stage 0 have access to the two conjugate subsets of stage 1. Thus, these two switches are looped together as shown. In stage 1, since each subset contains only two switches, each group will have only one switch, and therefore there is no point in forming a loop around the same switch.

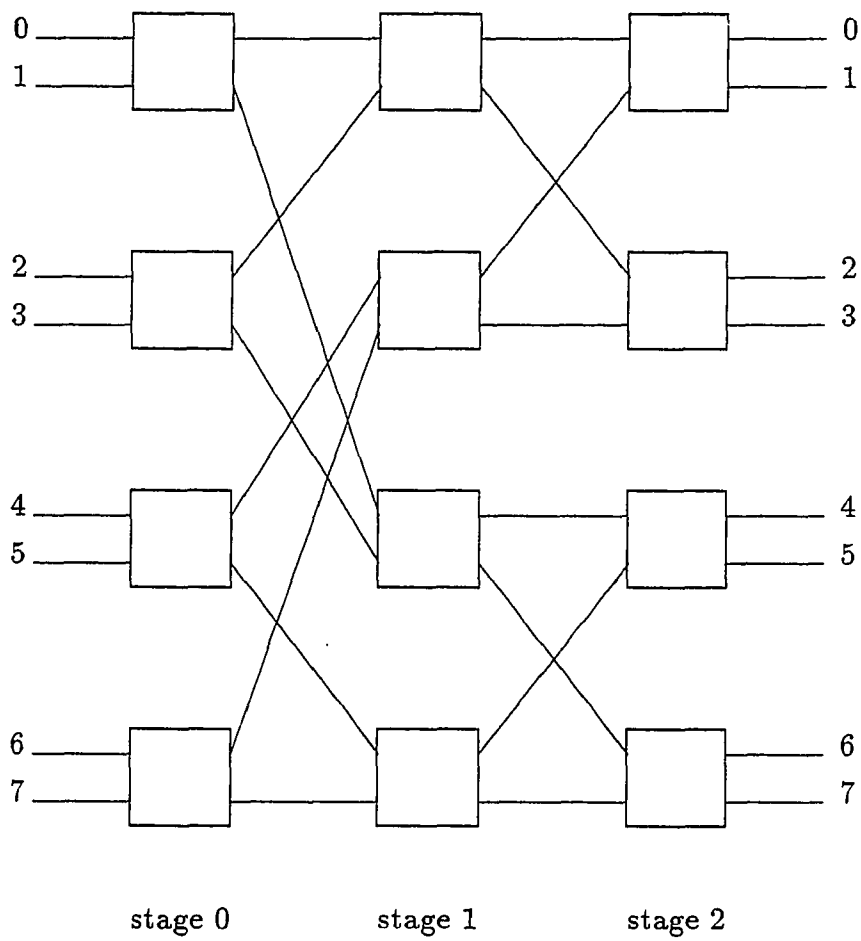


Figure 4.2: 8 × 8 Baseline network

The idea behind these loops is that if one switch fails in stage 1, say, then a route to it from stage 0 can be sent through the loop to another switch in the group where it can access again the original outlet. In reference to the ASEN of Figure 4.3, suppose that switch $X(1,0)$ wants to establish a route to outlet 3, but finds that switch $X(1,1)$ is defective. Then switch $X(1,0)$ can send the route through its auxiliary output to switch $X(0,0)$. As can be seen switch $X(0,0)$ has access to outlet 3, and thus the route can still be establish despite the existence of a fault along the original path. Routing the ASEN is described in more detail below.

The input and output stages, stages 0 and $\nu - 1$, respectively, are made fault tolerant as usual by using multiplexers and demultiplexers. Each inlet has access to two loops so that if one loop is defective, the inlet can route its connections through the other. Similarly, each outlet is reachable from two distinct switches so that if a switch fails, the outlet can be reached from the other switch. It should be noted that this arrangement at the output side of the network eliminates the last stage of switches, stage ν. It is also interesting to note that the number of loops in stage j is 2^{j+1}. This means that the number of loops in stage $\nu - 2$ is equal to the number of switches of the stage $(2^{\nu-1})$. This eliminates the need for having loops in stage $\nu - 2$, as it would be meaningless to form a loop around the same switch.

4.2.1 Operation and fault tolerance model

Given its construction, the ASEN works as follows. A processor requests a path by putting the routing tag for the destination on the inlet. For each switch j, $0 \leq j < n - 2$, the request may arrive on any of the three inputs. The switch must use the proper routing bit in the routing tag to extend the path to the next stage on one of its two normal inputs. If a switch cannot use one of its two normal outputs because it receives a signal from the next switch indicating that it is busy

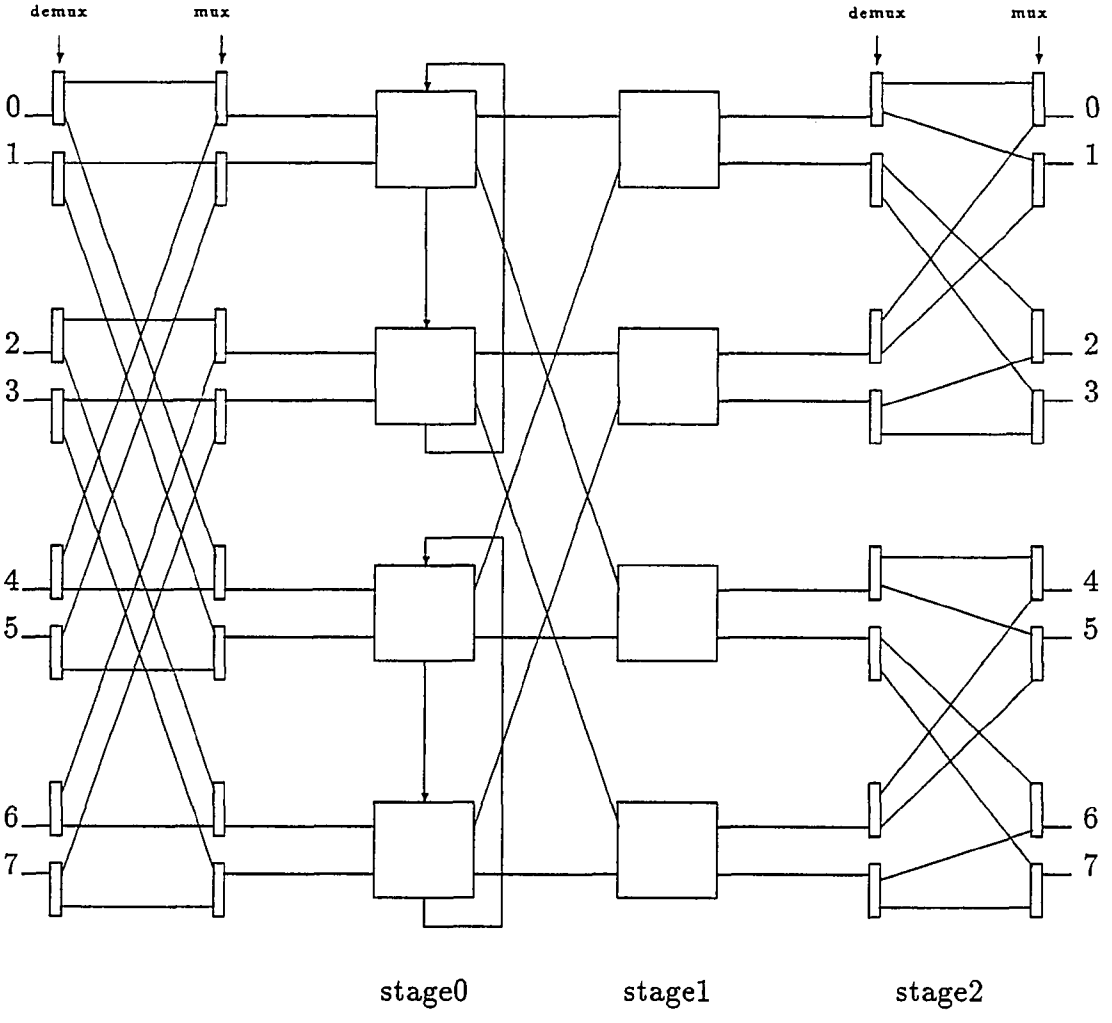


Figure 4.3: The Augmented Shuffle-Exchange Network (ASEN)

or faulty, the switch routes the path on its auxiliary output. (The operation of the ASEN depends on having switches capable of detecting faults in switches one stage ahead.) This process continues until stage $\nu - 3$ is reached. As mentioned earlier, the switches of stage $\nu - 2$ are normal binary switches and behave as such. However, if a switch in that stage finds that the demultiplexer it wants to use is defective the route cannot continue. Clearly, the demultiplexers should be able to send a busy signal back to stage $\nu - 2$ which can then be relayed back to the processor requesting the path.

To illustrate with an example, consider the ASEN of Figure 4.3. Suppose now that switch $X(1,1)$ is faulty and it is required to establish a route from inlet 2 to outlet 0. Under normal circumstances, this route would traverse switches $X(1,0)$, $X(1,1)$ and then through the demultiplexer connected to the upper output of switch $X(1,1)$ to the destination. But now since $X(1,1)$ is faulty, switch $X(1,0)$ will use its auxiliary output to route the request to switch $X(0,0)$. Now if switch $X(0,0)$ has its upper output vacant, it can route the request to switch $X(0,1)$ and from there it can go through the upper output to the destination.

The fault model in the ASEN is identical to that of the ESC. That is,

1. Any network component can fail
2. Faulty components are unusable
3. Faults occur independently
4. Demultiplexers cannot fail

The fault size of the ASEN is 1, and the fault tolerance criterion is full access retention, that is, any inlet must remain capable of accessing any outlet after the ESC recovers from a fault. The ESC is robust in the presence of multiple faults.

The advantages of the ASEN are as follows. First, no external hardware unit is needed to control the network; every routing step remains distributed as in an ordinary shuffle network. Second, the number of multiplexers and demultiplexers are half that used by the ESC. Third, one stage of switches, stage $\nu-1$ is eliminated.

The shortcomings of the ASEN can be summarized as follows. First, specially designed switches are needed to construct the ASEN. Unlike the ESC, the ASEN requires 3×3 switches with intelligence built in each switch so that it can make a decision as to which output it will route a request. Adding only one input and one output to a binary switch, adds at least 50% to its hardware complexity. Second, in the ASEN intrastage links are needed to form the loops, adding to the complexity of the network. Recall that these links are not single lines; they are complete buses incorporating data and signal lines as mentioned at the end of Chapter 3. Third, the number of links connecting the network to the inlets and outlets is double that in a normal shuffle network.

It is not clear in the ASEN what an outlet would do if it receives two requests. This is particularly perplexing in a multiprocessor environment where the outlets are connected to memory modules which normally have no intelligence. Also the question is open as to how the multiplexers at the input side would resolve contention in case two requests were received simultaneously. Clearly, these multiplexers are not the same as the ones mentioned in the ESC. More likely they are intelligent components which can make an autonomous decision. In the work developed in Chapter 5, intelligent multiplexers will also be needed.

From the above overview of two fault tolerance techniques for MINs, it should be obvious that the problem is not simple. It is difficult to present a design without having some drawbacks. A good technique is one that tries to minimize those drawbacks, rather than eliminate them. These guidelines will be utilized in developing

two fault tolerance techniques in the next two chapters.

4.3 Fault Detection and Location

The work of any fault tolerant MIN depends on two things: fault detection and fault location. Two techniques have been proposed in the literature for fault detection and location. First, fault detection and location can be performed off-line through applying prescribed test patterns to the inlets and comparing the output at the outlets with the expected values [5,35]. Second, faults can be detected and located dynamically online through either parity checking [79] or data bits checking [55]. As good as the online techniques may sound, they require a special switch design with built-in hardware to carry out the dynamic checking. This online fault detection and location technique is the mechanism assumed by the ASEN. However, the ESC does not require any particular mechanism; rather it requires only that the processors be notified of the location of the fault, if any. For the work done in this thesis, it is assumed that there is some mechanism to detect and locate faults and notify the processors of the location of the fault.

Chapter 5

The Simple Fault Tolerant Baseline network

As has been mentioned, fault tolerance has been provided for MINs by adding extra hardware in the form of extra switches, extra interstage links, extra intrastage links, or a combination of these components. In this chapter, a fundamentally different approach to fault tolerance of MINs will be introduced: the Simple Fault Tolerance (SFT) technique. The primary advantage of this technique is that it is not MIN-specific. In fact, it can be used with any MIN that fits into the MIN model of Chapter 2. The SFT will be demonstrated below on a Baseline network to construct the Simple Fault Tolerant Baseline (SFTB) network. The 8×8 Baseline network of Figure 3.1 will be chosen for this demonstration, and is repeated here as Figure 5.1 for convenience.

As its name implies, the idea behind the SFT technique is simple. In Chapter 1, it was mentioned that the interconnection mechanism in a multiprocessor system can be either a single bus or a MIN. The SFT technique combines these two mechanisms in one, with the MIN being the primary mechanism and the bus being used only after a fault occurs, and only by the processors affected by the fault. The resulting network thus combines all the characteristics of the original MIN. In addition, the

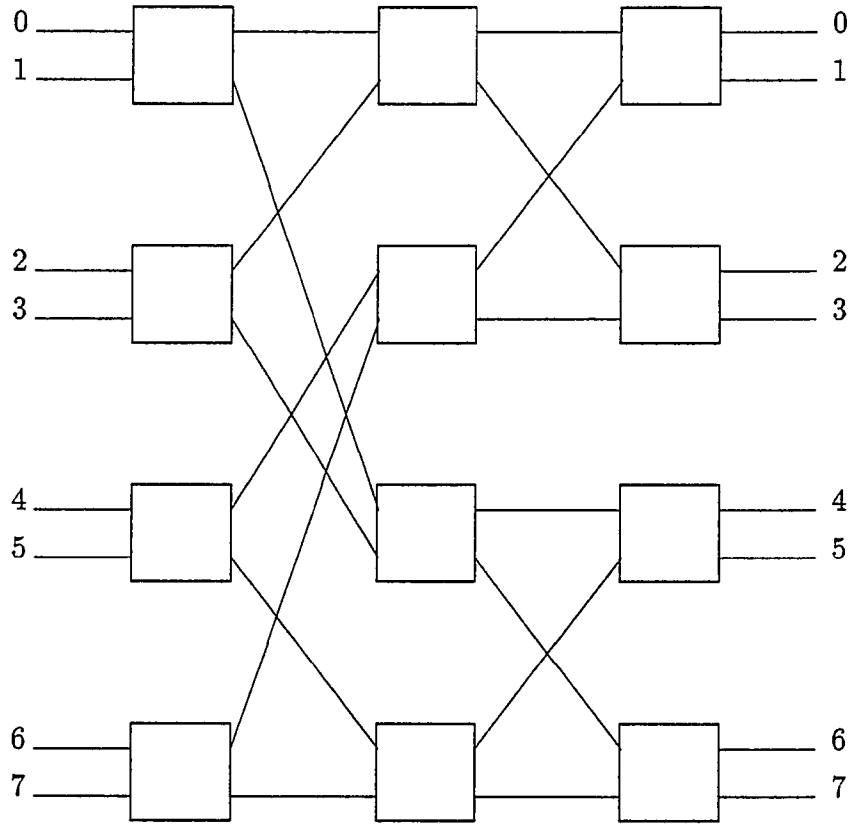


Figure 5.1: Baseline network of size 8

resulting network has the fault tolerance capability at a low cost.

The philosophy behind the SFT technique is that faults exist only temporarily. Therefore, drastic changes in the design of a MIN to make it fault tolerant are not warranted. On the one hand, those changes normally increase the cost of the MIN. On the other, in some cases the changes tend to have negative impacts on MIN operation under normal conditions. The latter point may not be apparent to the designer, but it can be made clear as follows. If the fault tolerance capability is impeded in the switches, the switches will be more complex. The direct consequence of this complexity is that the propagation delay of the switch will increase. This increase is of course unwanted, as it will decrease the throughput of the system. The SFT technique, by using an external bus in parallel with the MIN, does not change anything in the original MIN. The technique cannot cause negative impacts on the operation of the MIN under normal conditions, as the bus is totally invisible under those conditions.

5.1 Design of the SFTB

For the SFTB, the fault model is defined as follows.

1. Any switch can fail: A switch can fail in several ways. For instance, the switch can be stuck in one of its legal states, giving a proper connection only if that happened to be the desired state. Also, a switch can be stuck in a partially legal state, such as connecting only one input to one output. This permanent connection again may happen to be desired to establish a path. A switch can be stuck in an illegal state, such as connecting the two inlets to each other and the two outlets to each other, making it totally useless. In addition, a switch may be responsive to its control unit but give sometimes or always the

wrong state. All these cases will be lumped as a switch failure, for which the switch is totally useless and must be avoided.

2. Any link can fail: A link can fail if it is disconnected from a switch to which it should be connected. In fact, an open circuit in a link disconnects communications between two switches. Although the SFTB, as will be shown, can handle link failures, the discussion will mainly focus on switch failures for two reasons. The first reason is brevity and clarity, as analysis of link faults will only clutter the work without adding any new substance. The second reason is that switch failures are more difficult to recover from and once a network is able to recover from switch failures, it is trivial to adapt the results to link faults.

3. The standby bus, demultiplexers, multiplexers, and external links cannot fail. These items are the hardware that is supposed to provide fault tolerance to the system. If it could be assumed to fail, then it would not be possible to propose any fault tolerance design. In addition, this assumption can be justified here because these components remain idle under normal conditions. Thus they can be expected to have higher reliability than the actively working switches.

It should be mentioned that faults are assumed to occur independently, and that faulty components are unusable.

The fault tolerance criterion for the SFTB is *full-access retention.* That is, after a fault occurs, each processor must still be able to communicate with any memory module. It is worth noting that in the enhanced SFTB, to be presented later, a higher level of fault tolerance criterion can be achieved – full recovery. *Full recovery* is the ability of the network to regain its pre-fault connectivity after a fault occurs.

The fault tolerance size is the number of faults that the system can recover from. The SFTB can tolerate as many faults as possible (all switches and all links fail). In other words, if every component of the Baseline network fails, the communications between the processors and memory can still be maintained. One caveat, however, is that the more faults that exist, the worse the performance of the SFTB will be. For this reason, and to keep performance at almost the pre-fault level, the SFTB network will be analyzed as a single fault tolerant network.

With that in mind, the SFTB can now be described. Starting with a Baseline network, an external bus bypassing the network and connecting the processors to memory is added. Each processor is connected to both the network and the bus through a demultiplexer. Similarly, each memory module is connected to both the network and the bus through a multiplexer. Under normal conditions, the processors and memory will have connections only to the network. Since the bus does not become active until a fault occurs, it is called a *standby bus.*

This technique is applied to the Baseline network of Figure 5.1 to create the SFTB, shown in Figure 5.2. As can be seen, each inlet is connected both to the ordinary network and to the standby bus through a 1×2 demultiplexer. The demultiplexer can be looked upon as a switch connecting its input to only one of its two outputs at a time, based on a control (selection) bit. If the demultiplexer is set to connect the inlet to the network, it is said to be in the 0 position. Putting the demultiplexer in the 1 position connects the inlet to the bus. Similarly, each outlet is connected to both the network and the standby bus through a 2×1 multiplexer. Here again, the multiplexer can be either in the 0 position, thereby connecting the network to the outlet, or in the 1 position, connecting the bus to the outlet.

The SFTB operates as follows. Under normal condition, the multiplexers and demultiplexers should be in the 0 position. This makes the SFTB functionally

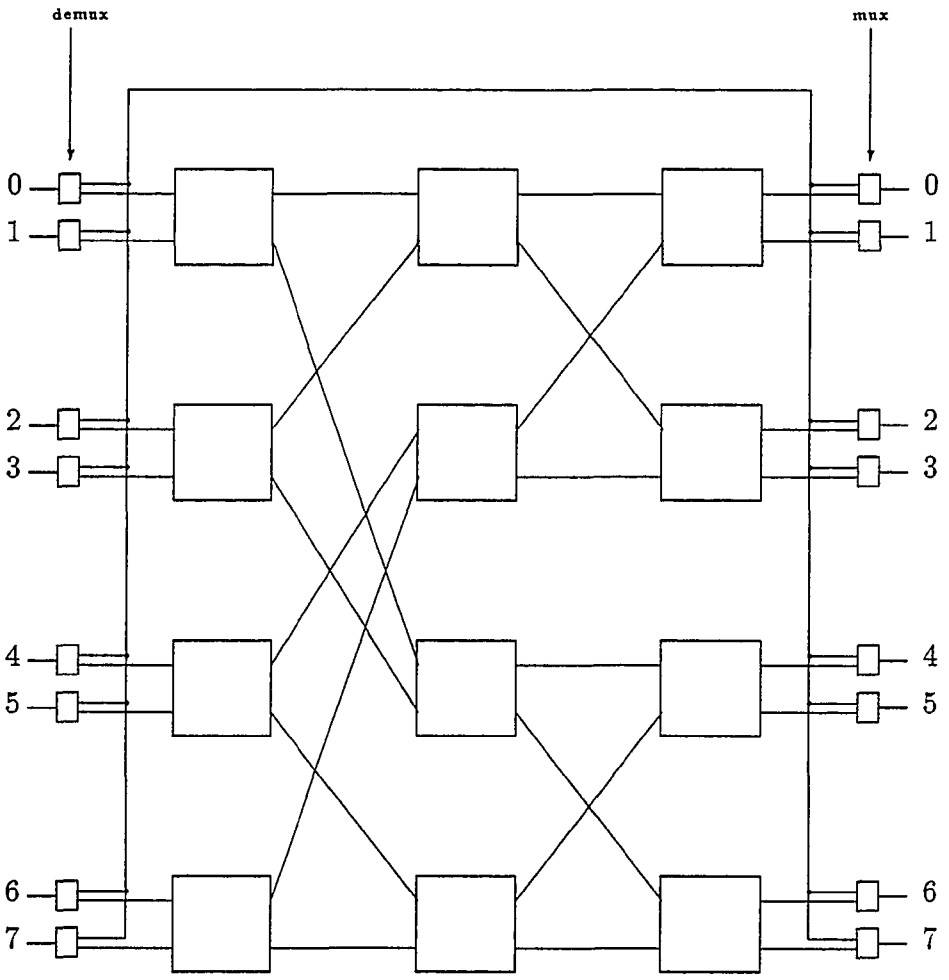


Figure 5.2: The SFT equivalent of the Baseline network of Figure 5.1

identical to the ordinary Baseline. It follows that under normal conditions, the SFTB will use the routing algorithm of the ordinary network and therefore the addition of the standby bus will not have any negative impact on the operation of the network.

5.2 Routing the SFTB Under Faulty Conditions

Upon the occurrence of a fault, the SFTB must be reconfigured to cope with the fault. Thus, a mechanism for detecting and locating faults must be used to invoke this configuration process.

As indicated earlier, the discussion will deal only with switch faults to avoid cluttering the discussion unnecessarily. Assuming now that a switch has been identified as being faulty, the normal sequence of establishing paths will be modified as follows.

1. At the beginning of each memory cycle, each processor requiring memory access must find first if the defective switch is along the path it wants to establish.

2. If the defective switch is along the path, the processor will have to access memory using the standby bus instead of the network.

3. If the defective switch is not along the path, the processor starts the memory cycle as it normally would under normal condition.

Step 1 above results in some performance degradation due to the time taken by the processor to find if the defective switch lies along the path. The details of this will be discussed later. Step 2 requires accessing the bus. Since more than one processor, at most two, may try to access the bus simultaneously, a contention

problem may arise. This is particularly true in *synchronous* operation, where all processors start their memory cycle at the same time. Some solutions to this problem are presented in Section 5.2.2. Step 3, establishing the path over the network, is the ordinary Baseline procedure used under normal conditions, and therefore needs no further explanation. After a fault occurs, all the processors except two will be able to still use the same routing scheme they do under normal condition. This is a principal advantage of the SFT technique.

5.2.1 Performance degradation under faulty conditions

As has been shown, using the SFT technique under normal conditions is completely transparent; the performance of the SFTB is identical to that of the ordinary Baseline network. However, when the network is faulty, some performance degradation occurs due to the fact that at the beginning of each memory cycle, a processor requiring access to memory must find out if the faulty switch lies on the path to the destination. Suppose that processor S, $0 \leq S \leq N-1$, needs a path to destination $D, 0 \leq D \leq N-1$, whose binary representation is $d_{\nu-1}, d_{\nu-2}, \ldots, d_0$. Recall that the binary representation of D represents the routing tag needed to establish the path from S to D. On a Baseline network, given the current switch $X(i,j)$, and the routing bit $d_{\nu-1-j}$, one can find the next switch along the path $X(\acute{i}, j+1), 0 \leq \acute{i} \leq (N/2)-1$, as follows.

$$\acute{i} = \begin{cases} \lfloor i/2 \rfloor + (\alpha/2)\lfloor i/\alpha \rfloor & \text{if } d_{\nu-1-j} = 0 \\ \lfloor i/2 \rfloor + (\alpha/2)\lfloor i/\alpha \rfloor + \alpha/2 & \text{if } d_{\nu-1-j} = 1 \end{cases}$$

where $\alpha = 2^{\nu-j-1}$ and $\lfloor x \rfloor$ denotes the integer part in the real number x. Assume now that switch $X(a,b)$ is defective. Processor S can apply the above formula recursively to find if $X(a,b)$ is along the path to memory module D. Procedure AVOID_NETWORK below utilizes this formula and must be used at the beginning

of the memory cycle by each processor requiring memory access. The procedure adjusts a binary flag *avoid*. If *avoid* is set, it means that the defective switch is along the path and therefore the processor must use the bus to establish the required connection. If, on the other hand, *avoid* is reset, it means that the processor can use the network as if there was no fault.

```
PROCEDURE AVOID_NETWORK (ν, a, b, D)

BEGIN
IF ((b = 0 AND ⌊S/2⌋ = a) OR (b = ν − 1 AND ⌊D/2⌋ = a)) THEN avoid ← 1
ELSE
  BEGIN
  avoid ← 0, j ← 0, i ← ⌊S/2⌋
  WHILE j < b DO
   BEGIN
   α ← 2^(ν−j−1)
   IF d_(ν−1−j) = 0 THEN i' ← ⌊i/2⌋ + α/2⌊i/α⌋
   ELSE i' ← ⌊i/2⌋ + α/2⌊i/α⌋ + α/2
   j ← j + 1, i ← i'
   END {while}
  IF i' = a THEN avoid ← 1
  END {else}
RETURN (avoid)
END {AVOID_NETWORK}
```

This procedure represents the difference between the operation of the SFTB under normal and faulty conditions. As a result, the time it takes to execute the procedure is the measure of performance degradation under faulty conditions. To estimate this time, notice first that if the network has two stages, $\nu = 2$, the procedure will not enter the WHILE loop. For $\nu \geq 2$, the statements outside the loop will be executed only once. Assume the time it takes to execute these statements is T_o, which is $O(1)$. As for the loop, in the worst case it will be executed $\nu - 2$ times. It is obvious that the worst case is if the faulty switch happens to be in stage $\nu - 2$. If the loop execution time is T_l, then the worst case run time for the

procedure, T_p, is

$$T_p = T_o + (\nu - 2)T_l.$$

Note that $T_l = O(1)$, hence $T_p = O(\nu)$. This is an acceptable value, in view of the fact that the alternative is to use hardware means to notify the processor that the path cannot be established due to a faulty switch. That would be in the form of a signal from the faulty switch back to the processor. Taking the worst case, a fault at stage $\nu - 2$, it would take the signal twice the propagation delay time from the processor to the faulty switch. That is, it takes $2(\nu - 1)T_s$, where T_s is the switch propagation delay which is estimated [69] to be 8 gate delays.

In either case, software or hardware, the complexity of the delay time under faulty conditions is $O(\nu)$, but the software procedure has the advantage of not requiring any specially designed switches or extra interstage signaling lines.

5.2.2 Accessing the bus

After finding that a path cannot be established on the network because of a fault, the processor must use the bus to establish that path. If the fault is in a link, the single processor affected can start using the bus with no problems. But if the fault is in a switch, two processors will need to use the bus. If the two start using the bus simultaneously, a conflict occurs. To avoid the contention, the bus is provided with an extra line indicating whether the bus is currently available, the *bus-busy* line. A processor wanting to use the bus must first sense the bus-busy line. If it is low, the processor asserts it and starts using the bus. If the line is high, the processor waits in a loop until it is low and then seizes it. A problem, however, can occur if the two processors simultaneously test the line, find it low and try to seize it at the same time. This will most likely be the case in a synchronous environment, where all processors start accessing memory at the same time. For this situation,

a more elaborate mechanism must be devised. Below, two such mechanisms are suggested and discussed: one depends on a Central Control Unit (CCU) and the other is dynamic.

The CCU is a hardware unit that can be accessed by all processors. Its function is to receive requests for the bus and grant access to one of the processors at a time. A processor wanting the bus sends a request to the CCU giving its number and the memory module number. The CCU identifies the multiplexer in question, puts it in the 1 position, and sends an acknowledgement to the processor upon which it can safely start using the bus. Notice that the demultiplexers can always be set by the processors without the help of any hardware unit. The CCU will grant the bus immediately to the processor that asks for it. But if the two processors submit their requests at exactly the same time, a built in arbiter [70] must decide to which processor it will grant the bus. The arbitration can be either random or prioritized. At the end of the memory cycle, the CCU must put the multiplexer back in the 0 position, preparing for the next memory cycle.

Instead of relying on the services of a central control unit, the processors can perform the services themselves dynamically. A processor wanting to communicate over the bus can put its own demultiplexer in the 1 position. However, two problems can arise: competing for the bus and setting the multiplexers on the other side of the network. The bus contention problem can be solved using the bus-busy line mentioned above in two ways together with a round robin scheme for the processors to test that line. In the round robin scheme, a processor wanting to use the bus counts a number of clock cycles equal to its number in the system before it tests the bus-busy line. If the line is low, the processor asserts it and start using it on the next clock pulse. If the line is high the processor keeps testing it continuously until it goes low, when it asserts it and starts using the bus on the next clock pulse.

If the clock period is T, then the minimum wait time before seizing the bus is T, for processor 0, and NT, for processor $N-1$, with the average waiting time being $NT/2$. It should be noted here that T must be greater than $\tau_{\max}$, the propagation delay between processors 0 and $N-1$. Otherwise, processor i, $0 \leq i \leq N-1$ will have to wait $(i+1)\tau_{\max}$ instead of $(i+1)T$, before it tests the bus-busy line.

Assuming now that the processor has seized the bus dynamically, it puts on the bus the number of the memory module it wants to communicate with. This number will be decoded and used on the other side of the network to both put the multiplexer in the 1 position and enable the memory module. This procedure is the mechanism adopted in uniprocessor systems with one bus. The processor can proceed then talking to the memory module as in a uniprocessor. Once it finishes communicating with the memory module, the processor should put the multiplexer and demultiplexer back in the default, 0, position.

5.3 Design of the Enhanced SFTB

In some applications, full access retention *only* may not be enough as a fault tolerance criterion. Note that with a full-access retention only capability, the network cannot realize any permutation. If the network is used mainly to realize permutations, then a higher fault tolerance criterion must be imposed, *full recovery*. This allows the network to be able to realize after recovery any connection pattern it was capable of before the fault. Such a criterion can be easily met for any network using binary switches with the help of an enhanced version of the SFT technique. The idea is to add two standby buses to the network instead of one. Recall that the worst single fault in a network with only binary switches, such as the Baseline network or the Beneš network, prevents the realization of exactly two paths. With two standby buses, these two paths can be implemented resulting in full recovery

Selection word	Selection
00	inputs of network
01	standby bus No. 1
10	standby bus No. 2
11	unused

Table 5.1: Multiplexer and demultiplexer operation modes

of the system. The enhanced technique will be demonstrated below on a Baseline network, where the the advantages of this enhancement both under normal and under faulty conditions will be illustrated.

The enhanced SFT equivalent of the Baseline network of Figure 5.1 is shown in Figure 5.3. Using two buses, entails the use of demultiplexers and multiplexers of size 1×4 and 4×1, respectively. Each of the multiplexers or the demultiplexers has two selection lines controlling the operation mode of their respective units as shown in Table 5.1.

Some work on multiple bus systems has already been performed [61,62,83], where solutions to the bus access problem are suggested. Basically, the techniques used for accessing a bus in a multibus system are similar to those discussed for the SFTB, with CCUs equipped with arbiters being the most recommended.

5.3.1 Permutation realization capabilities of the enhanced SFTB

As indicated earlier, an $N \times N$ Baseline network cannot realize every permutation in the symmetric group, Σ_N [37]. Only a class of permutations can be realized, and this class has been already identified [1]. As will be proven later, if there is blocking while a permutation is being realized on a Baseline network, the minimum number of blocked paths is 2. If a permutation is blocked in exactly 2 paths, it can be realized blocking-free on the SFTB by realizing the two blocked paths on the two

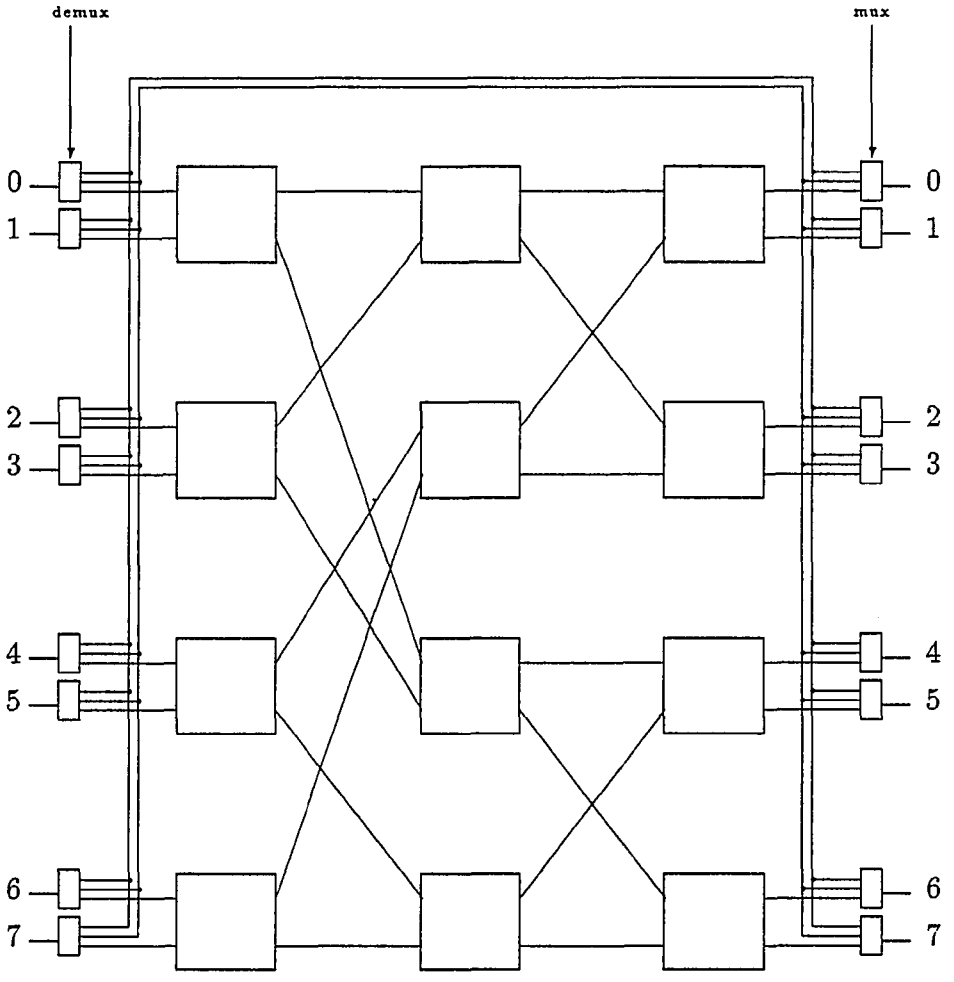


Figure 5.3: The enhanced SFT equivalent of the Baseline network of Figure 5.1

standby buses. A permutation, P, of a set of elements A, $A = \{0, 1, \ldots, N-1\}$, is a one-to-one function mapping A onto itself, such that

$$P = \{P_i : i, P_i \in A\}$$

where $P_i = P_j$ if and only if $i = j$, for all $i, j \in A$. Realizing permutation P on a MIN means connecting inlet i to outlet P_i for all $i, P_i \in A$, where A in this case is the set of all inlets (or outlets). A permutation is also sometimes called a *bijection* [37]. The size of a permutation is the cardinality of the set it acts on. A size-N permutation is one that maps a set of cardinality N. In this section, a permutation with ξ, $0 \leq \xi \leq N$, unrealizable elements will be called a permutation with ξ blocked paths. The switch where a conflict occurs is called a *blocking switch.*

The enumeration of permutations of different numbers of blocked paths on the Baseline network will be performed by counting the connection patterns on the network which should be unique for distinct permutations. A connection pattern can be looked upon as a "snap shot" of the settings of the switches and the location and type of the blocking. It turns out, however, that with the same permutation, one can see more than one connection pattern unless some consistency is adopted. Specifically, a fixed arbitration policy must be maintained for all switches in case a conflict occurs. A conflict occurs if the two inputs of a switch ask for the same output. The switch may give priority to the upper input, in which case it is said to adopt a Priority To Upper (PTU) Policy, or it may give priority to the lower input, in which case it is said to adopt a Priority To Lower (PTL) policy. Assigning a fixed priority policy to each switch throughout the counting process is necessary to achieve a one-to-one relationship between each permutation and the connection pattern resulting from its realization on the network. It should be noted, however, that in practice a switch may change its priority policy adaptively. Once this

consistency is achieved, one can easily count the number of permutations of every blocking class, taking one pattern to represent one unique permutation. To show that changing the arbitration policy of a switch can create more than one connection pattern for the same permutation, consider the permutation

$$P = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 2 & 1 & 4 & 6 & 7 & 3 & 0 & 5 \end{pmatrix}.$$

Realizing this permutation on the 8×8 network of Figure 5.1, with *all* switches having a PTU policy, results in two blocked paths and 6 conflict-free paths, as shown in Figure 5.4.

The two blocked paths are $\binom{1}{1}$ and $\binom{3}{6}$. The two inlets associated with these two paths are marked in the figure. Shown also are the two blocking switches with arrows inside. The arrow shows the original direction of the blocked path and how the path was blocked. The goal now is to take a "snap shot" of the pattern in Figure 5.4, including the blocking switches and the arrows inside, and consider it one permutation, namely P above.

Now, consider permutation P again. Only this time assign switch $X(1,0)$ a PTL policy. The resulting connection pattern is shown in Figure 5.5. It can be seen that the pattern is different from that of Figure 5.4. It can also be seen that although this is still a permutation with two blocked path, the two blocked paths are different from the previous realization; in Figure 5.5, the two blocked paths are $\binom{1}{1}$ and $\binom{2}{4}$.

Besides having only one pattern for each permutation, to be able to calculate the permutations with ξ blocked paths, the converse should also be true. That is, there should exist only one permutation for each connection pattern. This last requirement is evidently true.

Although the focus here is on permutations with exactly two blocked paths, some interesting results on the capability of the Baseline network to realize random

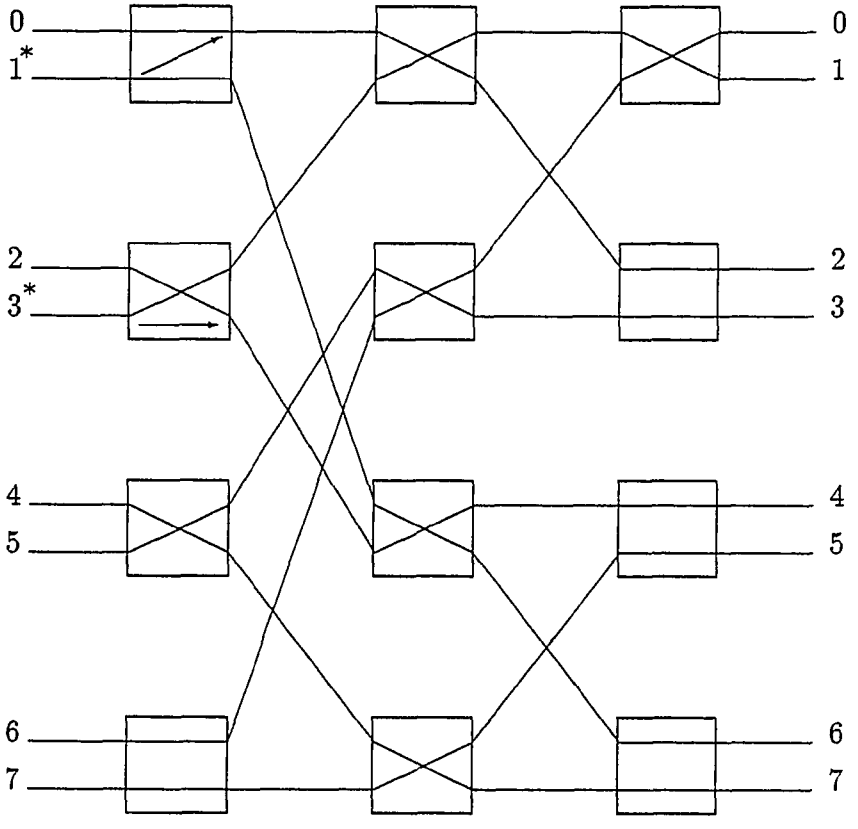


Figure 5.4: Permutation P realized with a PTU policy at all switches

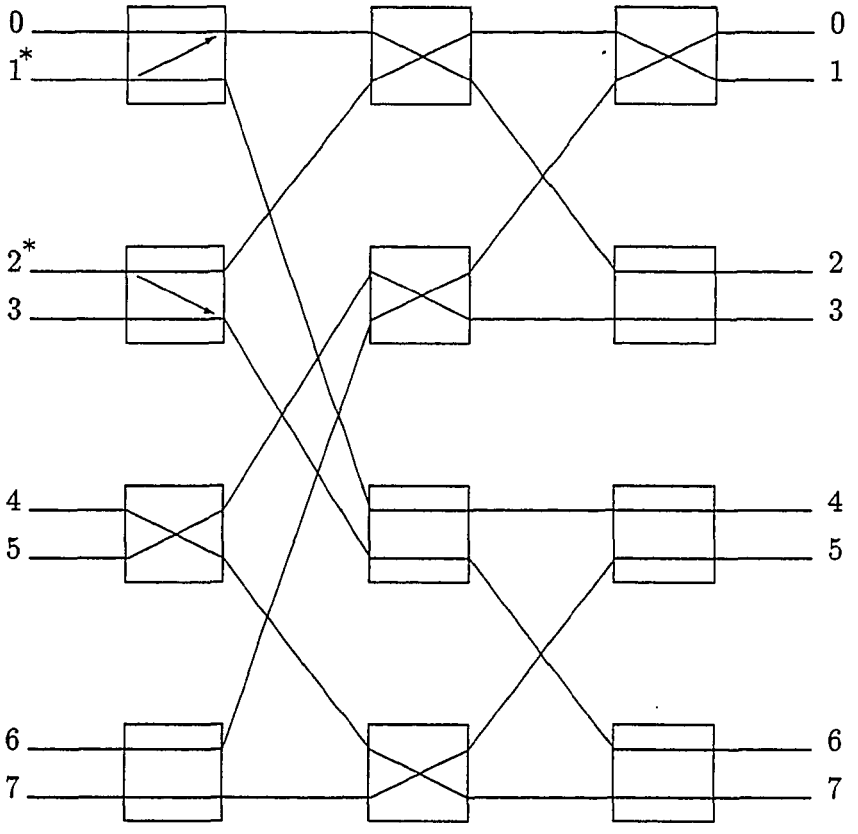


Figure 5.5: Same permutation of Figure 5.4, realized with a PTU policy at all switches except $X(1,0)$

permutations will be presented.

Let $\mathcal{P}_N^{(\xi)}$ be the set of all permutations of size N that, when realized on an $N \times N$ Baseline network, have exactly ξ blocked paths, $0 \leq \xi \leq N$, and $N - \xi$ conflict-free paths. Then, it is obvious that

$$\sum_{\xi=0}^{N} \left|\mathcal{P}_N^{(\xi)}\right| = N!,$$

where $\left|\mathcal{P}_N^{(\xi)}\right|$ denotes the cardinality of the set $\mathcal{P}_N^{(\xi)}$.

Theorem 5.1 *In a permutation $P \in \mathcal{P}_N^{(\xi)}$, let the number of blocking switches be donated as ψ. Then $\psi = \xi$.*

Proof: The theorem will be proven in two steps. First, $\psi \not< \xi$ is proven by contradiction. Assume that $\psi < \xi$. This means that there are k switches, $1 \leq k \leq \xi/2$, each blocking two paths. This implies that a switch can be in a state where neither of its two inputs can get through. Obviously, this is not true because, from the way the switches operate, at least one input must get through. The second part is to prove that $\psi \not> \xi$. This again will be proven by contradiction. Assume that $\psi > \xi$. This entails that a path can be blocked in more than one switch. But by definition, if a path is blocked at a switch, it stops there; it does not go to another switch where it may be blocked again. Thus the second assumption is also wrong and the theorem is proven. □

Theorem 5.2 *For any $N \times N$ Baseline network, $\left|\mathcal{P}_N^{(0)}\right| = 2^{\nu 2^{\nu-1}}$.*

Proof: From the uniqueness of path property of the Baseline network, realizing a permutation should result in a unique connection pattern. Arbitrarily put each switch in the Baseline network in one of its two legal states, shown in Figure 3.4. The pattern resulting will represent one size-N permutation. By changing the settings of

all the switches, one at a time, different patterns will appear, with each representing a conflict-free permutation. In any Baseline network the number of switches is $nN/2 = \nu 2^{\nu-1}$. Since each switch can have one of two states, the total number of patterns that can be created by changing the states of the switches is $\left|\mathcal{P}_N^{(0)}\right| = 2^{\nu 2^{\nu-1}}$. □

Theorem 5.3 *For any $N \times N$ Baseline network, $\left|\mathcal{P}_N^{(1)}\right| = 0$.*

Proof: To prove this theorem, it is required to prove that once there is a blocked path, there is *at least* one other blocked path. Consider that the path between inlet i and outlet o is blocked. Then there is a path between i and the wrong outlet $\acute{o}$. This means that i is missing its right outlet, i.e. a missing path. But a permutation by definition, is a one-to-one and onto mapping of N elements onto themselves. Thus, outlet $\acute{o}$ must have an inlet in the permutation to which it should be connected. This means that $\acute{o}$ is missing its right inlet, i.e. another missing path. That is, one cannot have a permutation blocked in exactly one path, or $\left|\mathcal{P}_N^{(1)}\right| = 0$. □

It should not be inferred, however, that blocked paths occur in pairs, because overlapping is possible for more than two blocked paths. In other words, it is possible to see a pair of blocked paths satisfying the above theorem, and another pair also according to the theorem, with one path in common with the two pairs, resulting in a total of three blocked paths. The theorem is demonstrated in Figures 5.4 and 5.5, where the two blocked paths are shown. It can be seen that when a path is blocked, the inlet can still reach an outlet, but the wrong one.

It has been proven [6] that the minimum number of conflict-free paths is $2^{\lceil \nu/2 \rceil}$. This translates in the notation developed here as

$$\left|\mathcal{P}_N^{(\xi)}\right| = 0$$

for all ξ, $N - 2^{\lceil \nu/2 \rceil} < \xi \leq N$.

From the Baseline network of Figure 5.1, it can be seen that the set of outlets accessible to switch $X(i,j)$ depends on both i and j. For example, any switch in stage 0, can access any outlet. A switch in stage 1 has access only to half the outlets. For instance, switch $X(2,1)$ has access only to outlets 4 through 7. Let Ω be the set of all outlets, and let S_j be the set of all switches of stage j. Also, let $S_{u,j} \subseteq S_j$ be the subset of all switches $X(i,j)$ that have access to the same subset of outlets $\Omega_{u,j} \subseteq \Omega$. Then, $\Omega_{u,j} = \Omega_{\acute{u},j}$ if and only if $u = \acute{u}$.

$S(u,j)$ is called a *conjugate subset* of switches [49]. It can be seen that each S_j is divided into γ_j disjoint subsets $S_{u,j}, 0 \leq u \leq \gamma_j - 1$, such that

$$\bigcup_{u=0}^{\gamma_j - 1} S_{u,j} = S_j,$$

and

$$S_{u,j} \cap S_{\acute{u},j} = \begin{cases} S_{u,j} & \text{if } \acute{u} = u \\ \phi & \text{otherwise} \end{cases}$$

where ϕ denotes the empty set. It can be easily verified that for all j, $0 \leq j \leq \nu - 1$,

$$\gamma_j = 2^j,$$

$$|S(u,j)| = 2^{\nu - j - 1},$$

where $0 \leq u \leq \gamma_j - 1$, and

$$\sum_{u=0}^{\gamma_j - 1} |S_{u,j}| = |S_j| = N/2.$$

Considering the subset structure of the Baseline network, a new representation for the network can be arrived at, the *subset* representation. For the Baseline network of Figure 5.1, the subset representation is shown in Figure 5.6. Subset $S_{u,j}$ at stage j, $0 \leq j \leq \nu - 2$ has access to exactly 2 subsets in stage $j+1$, namely, $S_{2u,j+1}$ and $S_{2u+1,j+1}$. These two subsets are complementary in the sense that if an

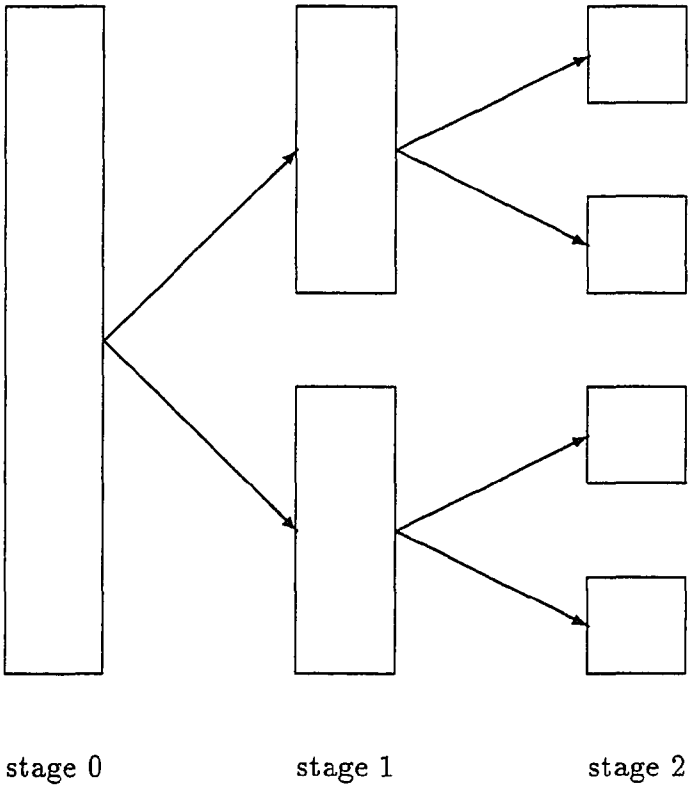


Figure 5.6: Subset structure of the Baseline network of Figure 5.1

input of a switch in subset $S_{u,j}$ cannot reach one of them because of a conflict, it will by default reach the other. This observation will be utilized in the next theorem.

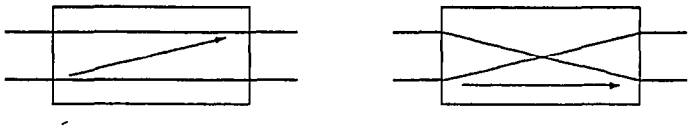
Theorem 5.4 *In a permutation $P \in \mathcal{P}_N^{(2)}$ let the two blocking-switches be $X(i,j)$ and $X(\dot{i},\dot{j})$. Then $\dot{j} = j$, and $X(i,j), X(\dot{i},j) \in S_{u,j}$, where $u = \lfloor \dot{i}/\gamma_j \rfloor = \lfloor i/\gamma_j \rfloor$.*

Proof: From the subset structure of the Baseline network, a switch at subset $S_{u,j}, 0 \leq j \leq \nu - 2$, has access only to the two subsets $S_{2u,j+1}, S_{2u+1,j+1}$. Suppose one of the two inputs of switch $X(i,j) \in S_{u,j}$ wants to go to subset $S_{2u,j+1}$ but cannot enter because of a conflict. Then that input will be connected incorrectly to $S_{2u+1,j+1}$. This wrong connection will be of course at the expense of a right connection which will incorrectly go to $S_{2u+1,j+1}$ from another switch. That is, the two switches, from which wrong connections emanate, have access to the two subsets $S_{2u,j+1}$ and $S_{2u+1,j+1}$, and both are one stage before $j+1$. These two switches can only exist in one unique subset in the network, $S_{u,j}$. □.

By looking at Figures 5.4 or 5.5, one can recognize two types of blocking: the first is when the Two inputs Request the Upper (TRU) output and the second when the Two inputs Request the Lower (TRL) output. These two types of blocking are shown in Figure 5.7. The arrow indicates the direction in which the lower input wanted to go but could not because of a conflict with the upper input which has a higher priority.

Theorem 5.5 *In a permutation $P \in \mathcal{P}_N^{(2)}$, if one of the two blocking switches has a TRL-type blocking, the other must have TRU type of blocking, and vice versa.*

Proof: The proof follows again from the subset structure of the network and from the definition of a permutation. The requests of a given subset must be divided exactly in half between the next two subsets. If two inputs of a switch ask for the



a) Two Request Upper (TRU) b) Two Request Lower (TRL)

Figure 5.7: The two types of blocking, assuming higher priority for the upper input

lower subset (TRL), then the two inputs of the other switch must be asking for the upper subset (TRU). By the same token, the converse is also true. □

The above theorems and discussion will be used to find the number of permutations with exactly two blocked paths, $\left|\mathcal{P}_N^{(2)}\right|$. First, from Theorem 5.1, there are exactly two blocking switches. For each position assumed by the two blocking switches, one can arbitrarily set the remaining switches, with each setting representing one permutation with exactly two blocked path. Let B be the number of permutations $P \in \mathcal{P}_N^{(2)}$ caused by blocking in two specific switches having two specific blocking types. It is obvious that there is a permutation for each time the state of one of the remaining $\nu 2^{\nu-1} - 2$ switches is changed. Thus,

$$B = 2^{\nu 2^{\nu-1}-2} \tag{5.1}$$

Now, let C be the number of ways in which the two blocking switches can appear in the network. Then the total number of permutations with exactly two blocked path can be found as

$$\left|\mathcal{P}_N^{(2)}\right| = BC \tag{5.2}$$

Recall from Theorem 5.5 that the two blocking switches always have opposite (or

complementary) types of blocking. Therefore the two switches are distinguishable and in finding C, permutations, rather than combinations, must be used. Recall further from Theorem 5.4, that the two switches cannot assume any arbitrary locations in the network, but they must be in the same subset. Let C_s and C_j be the number of ways the two switches can appear in subset $S_{u,j}$ and stage j, respectively. Then,

$$C_s = p\left(|S_{u,j}|, 2\right) = p\left(2^{\nu-j-1}, 2\right) \tag{5.3}$$

where $p\left(2^{\nu-j-1}, 2\right) = \frac{(2^{\nu-j-1})!}{(2^{\nu-j-1}-2)!}$ is the permutations of $2^{\nu-j-1}$ things taken 2 at a time.

Since there are γ_j subsets in stage j, then

$$\begin{aligned} C_j &= \gamma_j C_s \\ &= \gamma_j p\left(2^{\nu-j-1}, 2\right) \end{aligned} \tag{5.4}$$

Substituting 2^j for γ_j and writing $p\left(2^{\nu-j-1}, 2\right)$ explicitly in the above equation gives

$$\begin{aligned} C_j &= 2^j 2^{\nu-j-1}\left(2^{\nu-j-1} - 1\right) \\ &= 2^{\nu-1}\left(2^{\nu-j-1} - 1\right) \end{aligned} \tag{5.5}$$

The number of permutations with exactly two blocked paths associated with blocking in stage j is obviously BC_j. Since permutation blocking can occur in any stage but stage $\nu - 1$, the total number of permutations with exactly two blocked paths in any Baseline network is

$$\begin{aligned} \left|\mathcal{P}_N^{(2)}\right| &= \sum_{j=0}^{\nu-2} BC_j \\ &= \sum_{j=0}^{\nu-2} 2^{\nu 2^{\nu-1}-2} 2^{\nu-1}\left(2^{\nu-j-1} - 1\right) \\ &= 2^{\nu 2^{\nu-1}+\nu-3} \sum_{j=0}^{\nu-2}\left(2^{\nu-j-1} - 1\right) \end{aligned}$$

N	ν	$\lvert\mathcal{P}_N^{(0)}\rvert$	$\lvert\mathcal{P}_N^{(2)}\rvert$	$\lvert\mathcal{P}_N^{(2)}\rvert/\lvert\mathcal{P}_N^{(0)}\rvert$
4	2	16	8	0.5
8	3	4096	16384	4
16	4	4.23×10^9	9.49×10^{10}	22
32	5	1.2×10^{24}	1.26×10^{26}	104
64	6	6.28×10^{57}	2.86×10^{60}	456

Table 5.2: Values of $\left|\mathcal{P}_N^{(2)}\right|$ and $\left|\mathcal{P}_N^{(0)}\right|$ and their ratio for some values of ν

$$= 2^{\nu 2^{\nu-1}+\nu-3}\left[2^{\nu-1}\left(\sum_{j=0}^{\nu-2}2^{-j}\right)-(\nu-1)\right] \tag{5.6}$$

The factor $\sum_{j=0}^{\nu-2}2^{-j}$ is a finite geometric series whose sum is $2-\frac{1}{2^{\nu-2}}$. Substituting in Equation 5.6, and with some algebraic manipulation, one can find that

$$\left|\mathcal{P}_N^{(2)}\right| = 2^{\nu 2^{\nu-1}+\nu-3}\left(2^{\nu}-\nu-1\right) \tag{5.7}$$

Equation 5.7 represents the number of all the size-N permutations which when realized on a Baseline network, result in exactly two blocked paths. With an enhanced SFTB, if these two paths are realized on the two standby buses under normal conditions, then the equation gives the number of extra permutations that the enhanced SFTB can realize. Table 5.2 summarizes these results for some values of N.

5.4 Use of the Standby Bus Under Normal Conditions

It has been shown that the two buses of the enhance SFTB can be used under normal conditions to enhance the network capability of realizing permutations. In addition, there are two more functions that both the SFTB and its enhanced version can perform under normal conditions. These two functions are broadcasting

and establishing critical connections that cannot otherwise be established over the Baseline.

Broadcasting has been suggested [79] as a useful operation in multiprocessor systems. A processor may want to access all the memory modules at once and write to them simultaneously. Providing this capability on the ordinary Baseline network requires specially designed switches that can connect one of their inputs to the two outputs simultaneously. It turns out that providing this capability adds considerably to the complexity of the switch. However, with a standby bus having access to all memory modules, the broadcast function is readily available. With a CCU, a processor can ask it to put all the multiplexers in the one position. With the dynamic access situation, the multiplexers can be made to understand a broadcast "bit" which throws them immediately in the 1 position. One more advantage of implementing broadcasting on the bus, rather than on the network, is speed. The time needed to establish a broadcast connection on the bus, T_B, is the time needed to access the bus added to the time needed to put the multiplexer in the 1 position (assuming that the propagation delay on the bus is negligible). Clearly, T_B is at most as large as the propagation delay of one switch, T_S, which is estimated [69] to be 8 gate delays. Thus broadcasting on the bus is at least ν times as fast as it would be on the network.

The other function that the bus can perform under normal conditions is establishing connections that are needed urgently but cannot be established over the Baseline network. Suppose that a processor critically needs to establish a path to a given memory module, but the path is blocked. With the SFTB, one such connection can be established at a time, while with the enhanced version two can be established at the same time.

5.5 Using the SFT technique in MINs with large switches

In this section, the performance of an SFT network under faulty conditions will be examined. In the analysis below, only the ordinary SFT technique (that is, only one standby bus) will be considered; the results can be easily adapted to other variations of the technique.

It should be evident that the main problem in the SFT technique is that only one processor can access the bus at a time. Two approaches were specified to resolve contention for the bus: the CCU approach and the dynamic approach. In both approaches, a processor waits at most a period of time, $T_{\text{maximum wait}}$, before it takes control of the bus. Clearly,

$$T_{\text{maximum wait}} = (\mu - 1)T_c + T_{\text{misc}} \tag{5.8}$$

where

1. μ is the number of processors competing for the bus,
2. T_c is the memory cycle duration, and
3. T_{misc} is time waited by the processor due mainly to propagation delays. In the CCU scheme, for example, T_{misc} is the time taken in communicating with the CCU and the arbitration time. In the dynamic access scheme, T_{misc} is the time taken in counting before testing the busy line. In both schemes, T_{misc} includes the time needed to set one multiplexer and one demultiplexer.

The second term of $T_{\text{maximum wait}}$, T_{misc}, may seem independent of μ but careful examination reveals that it is indeed dependent on μ, regardless of which access scheme is used. Moreover, the first term of $T_{\text{maximum wait}}$ depends linearly

on μ. It is therefore the value of μ that determines the efficiency of the SFT scheme for any network. The larger the value of μ is the less efficient will be the network.

The value of μ is determined by the number of inputs or outputs of the largest switch used in the MIN. For instance, if a network has x switches, one of which of size 8×8 and the remaining $x - 1$ switches of size 2×2, that network will have $\mu = 8$. In such network, the efficiency of the SFT technique will be much less than with a binary network, such as the Baseline network, where $\mu = 2$. Therefore, binary networks are the best candidates for the SFT techniques. Other techniques must be devised for networks with large switch sizes. Such technique is introduced in Chapter 6 for the Clos network.

5.6 Discussion

In this chapter a novel technique to add fault tolerance capabilities to MINs has been introduced, the Simple Fault Tolerance (SFT) technique. In this technique, an external bus is used to offer an "emergency link" between the inlets and outlets of the network, in case a fault occurs. Under normal conditions, the processors use the network as normal, with the bus totally invisible. Under faulty conditions, the processors affected by the fault use the bus, while the unaffected ones continue using the network. The SFT network, thus, incorporates two interconnection mechanisms, the original network and the bus, both of which have been thoroughly analyzed in the literature for use in multiprocessor systems. The advantages and shortcomings of each have been pointed out: the bus is simple but cannot support a large number of processors, while the network can support a large number of processors but its hardware is complex. Thus if the bus can be guaranteed to serve a small number of processors, it can give both simplicity and good performance. This is the principle behind the SFT technique, as the number of processors affected by a fault in a MIN

is much less than the total number of system processors.

Although this technique can be applied to virtually any MIN, it works best with binary networks, those with 2×2 switches such as the Beneš and Baseline networks. In these networks, a single fault affects at most only two processors. In such a case the performance of the bus will be nearly as high as that of a bus in a uniprocessor system. In this chapter, the technique is applied to design the Simple Fault Tolerant Baseline (SFTB) network. The design is described and the performance is examined. The SFTB is shown to have five advantages. First, there is no performance degradation under normal conditions. Second, it allows immediate full-access retention to affected processors under faulty conditions. Third, it uses the same binary switches of the ordinary Baseline; no specially designed switches are required. Fourth, it uses the same number of switches and stages as the ordinary Baseline. Finally, the SFTB uses the same distributed routing algorithm as the ordinary Baseline *both* under normal and faulty conditions. Only those processors affected by the fault (at most 2) use a different algorithm. A by-product feature of the SFTB is that it can implement quickly and easily broadcast connections on the bus under normal conditions, thus increasing the system efficiency. The control of the SFTB is extremely easy. Several control schemes are suggested and examined.

An enhanced SFTB can be developed by adding another bus. With this addition, an ultimate criterion of fault tolerance is achieved, complete recovery. Moreover, the two extra buses can be used to relieve blockage under normal conditions. Here, if all the switches are operational but a path cannot be realized due to a conflict, that path can be established on the bus, thereby improving the throughput of the system. The number of the extra permutations realizable in this manner is calculated. In the context of this calculation, some new and interesting results on blocking in the Baseline network are presented.

It has been seen that the number of processors affected by the worst case failure event, determines the amount of performance degradation under faulty conditions of networks using the SFT technique. In any MIN, the worst case failure event is switch failure. It is clear then that as the size of the network switches increases, the performance of the SFT technique will decrease.

Clos networks are characterized by using non-binary switches. In fact, there is no limit on the size of a switch in the Clos network. Thus, a fault tolerance technique suitable for the Clos network is worth developing. This point, coupled with the fact that the literature does not seem to have any fault tolerant design for Clos networks, have been the motivation behind a fault tolerant Clos network presented in the next chapter.

Chapter 6

The Fault-Tolerant Clos Network

Clos networks inherently have the full access retention property if the fault is in a middle stage switch. This is due to the fact that each outer stage switch is connected to all middle stage switches. But with this inherent fault tolerance, one cannot realize a permutation that was realizable before the failure of the middle-stage switch. Moreover, a fault in one of the two outer stages cannot be tolerated by the network. In either case, the ability of the network to realize permutations will be impaired until the fault is physically removed. The design of a Fault-Tolerant Clos (FTC) network presented in this chapter offers the complete recovery capability which of course includes full access retention.

6.1 Design of the FTC

For the FTC, the fault model is defined as follows.

1. Any switch can fail.

2. Any interstage link can fail.

3. External links and multiplexers/demultiplexers cannot fail.

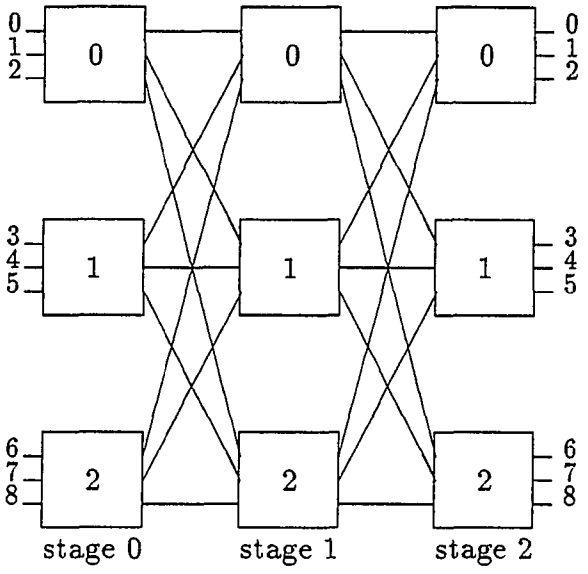


Figure 6.1: 9×9 ordinary Clos network

It should be mentioned that faults are assumed to occur independently, and that faulty components are unusable.

The fault-tolerance criterion of the FTC is complete recovery, that is, regaining pre-fault connectivity after a fault occurs. The fault tolerance size of the FTC is 1. Since the FTC is single-fault tolerant, complete recovery is possible if only one fault occurs. In the FTC, one switch in each stage can fail with the network remaining fully functional; therefore it can be called 3-fault robust.

A Clos network of size 9 is shown in Figure 6.1. In stage 0, each crossbar switch has 3 inputs and 3 outputs, hence its size is 3×3.

Recall form Chapter 3 that a Clos network of size N, must have $k = N/m$ switches of size $m \times n$ in stage 0, and $k = N/m$ switches of size $n \times m$ in stage 2. The switches of stage 1 must be of size $k \times k$. Furthermore, there are exactly n switches in stage 1. It should be noted that in Clos networks, $n \geq m$. An *ordinary* Clos network has $n = m$. When $n > m$, some degree of fault tolerance is obtained,

a fact utilized in the design of the FTC.

An FTC of size N is formed from an ordinary Clos of size N as follows. First, use switches with $n = m + 1$ in the outer stages. Second, add one extra switch to each of the three network stages. Each switch must be of the same size as the switches of the stage to which it is added. Third, connect the network inlets to the inputs of the first stage switches via 1×2 demultiplexers, and the network outlets to the outputs of the third stage switches via 2×1 multiplexers. As an example, the FTC equivalent of the network of Figure 6.1 is shown in Figure 6.2. It should be noted that using switches with $n = m + 1$ in the outer stages automatically adds an extra switch to the middle stage. As will be seen later, this provides fault tolerance to the middle stage. What remains then is to make the outer stages also fault-tolerant; that is why one extra switch is added to each of these two stages as shown. In the FTC, each inlet is connected by a demultiplexer to two distinct switches in stage 0. Also, each outlet is connected by a multiplexer to two distinct switches in stage 2. These multiplexers and demultiplexers serve as a fault recovery system in the case of a fault in either of the two outer stages. This type of fault as well as faults in the middle stage, stage 1, will be described later.

6.2 Reconfiguration of the FTC

The major feature of the FTC is its ability to be reconfigured such that pre-fault connectivity is totally regained. At any given time, there are three unused switches in the FTC, one per stage. Let these three switches be $X(f_0, 0)$, $X(f_1, 1)$ and $X(f_2, 2)$, where f_0, f_1 and f_2 are the unused switch numbers for the first, second and third stage, respectively. The configuration of the FTC at any time is a function of the present values of f_0, f_1 and f_2.

In general, the reconfiguration of the FTC can be performed through one or

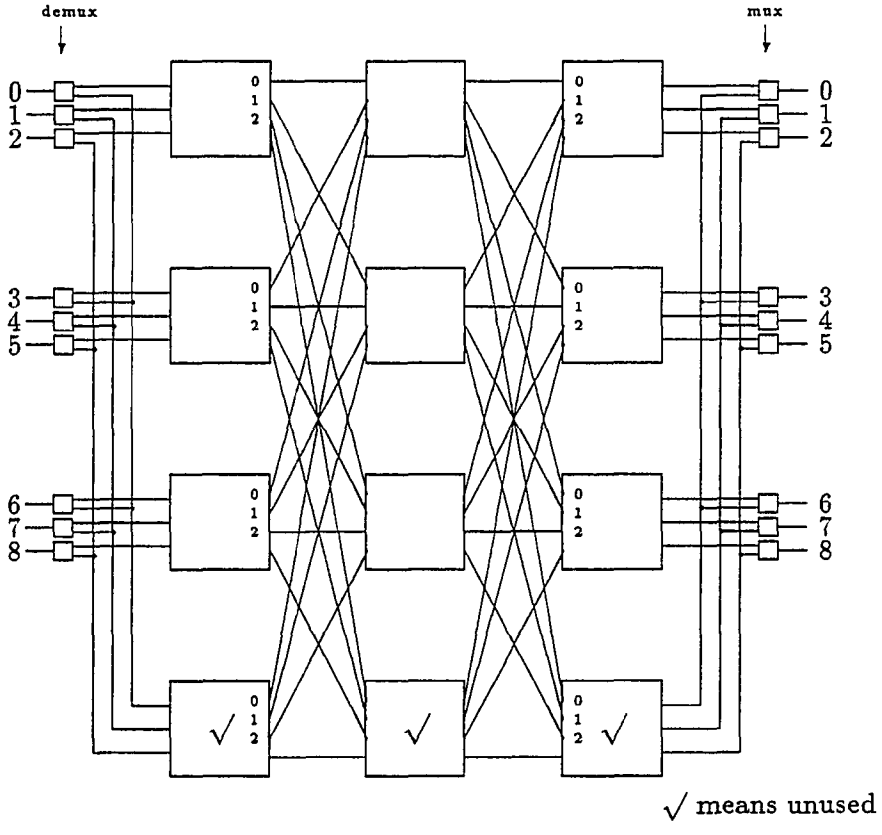


Figure 6.2: The equivalent FTC of the network of Figure 6.1

more of the following operations:

1. Changing the state of the multiplexers and demultiplexers

2. Terminal relabelling

3. Permutation translation

As will be seen below, the value of f_1 affects operation 2, while the values of f_0 and f_2 affect operations 1 and 3.

The multiplexer/demultiplexer state change operation is performed if an outer stage switch fails. When the FTC is not faulty, one switch in each stage will be unused. This unused switch can theoretically be any switch, but for convenience it will be assumed to be the last switch in each stage, i.e. $X(k,0)$, $X(n-1,1)$, and $X(k,2)$. This choice is convenient because it makes the multiplexers and demultiplexers remain in state 0 under normal conditions until a fault occurs; then they switch to state 1, thereby avoiding the defective switch. Suppose for example that $X(1,0)$ in Figure 6.2 fails during normal operation. Then the demultiplexers attached to that switch will change their state to state 1. This gives the resources attached to $X(1,0)$ access to the network through $X(3,0)$ instead. Realize now that $X(1,0)$ is the unused switch in stage 0, which confirms the fact that at all times there is one unused switch in each stage.

Permutation translation is also performed if an outer stage switch fails. Let $P = \{P_0, P_1, \ldots, P_{N-1}\}$ be an arbitrary permutation of $\{0, 1, \ldots, N-1\}$. In the actual network, P_i is the outlet to which inlet i is to be connected. In an ordinary Clos network, P goes directly to the central routing unit where the settings of the individual switches are extracted and delivered to the switches for implementation. In the FTC, the same steps are to be taken with the exception that permutation P is translated before it goes to the central routing unit.

Terminal relabelling is performed if a middle-stage switch fails. As mentioned above, f_1 affects the labelling of the outputs of switches $X(z,0)$, and the inputs of switches $X(z,2)$, $0 \leq z < k+1$. Let these outputs and inputs be referred to as the *inward terminals* of the outer stages or just the inward terminals. In each of these switches, only m out of the n inward terminals will be used, and will be referred to as the *active terminals*. Each active terminal will have two labels: a local one, to be used by the switch's control unit, and a global one, to be used by the central routing unit. The local label is an integer z, $0 \leq z < m$, and the global label is also an integer Z, $0 \leq Z < m(k+1)$. The active terminals will be labeled from top to bottom locally, with respect to the switch, as the sequence $0, 1, \ldots, m-1$. Globally, the active terminals that were labelled locally will be labelled from top to bottom, with respect to the stage, as $0, 1, \ldots, m(k+1)-1$. This labelling is shown in Figure 6.2, for $f_1 = 3$. In Figure 6.3, $X(2,1)$ is faulty, i.e. $f_1 = 2$. Therefore, the active terminals have been changed to exclude the third inward terminal in each switch. The new labels are shown in the figure.

The labels are updated always after a fault occurs, and the current labels are the ones used to implement the routing information received from the control unit. Realize that leaving out one inward terminal in each switch in stages 0 and 2 sums up to leaving out $2k$ inward terminals. If these 2k terminals in each of the two stages are chosen to have the same position with respect to the individual switches, then one middle stage switch will be left out. This switch is the unused switch under normal conditions and the defective switch under fault conditions.

6.3 Routing the FTC

The FTC is still required to perform the same function as the ordinary network – realization of permutations. For an ordinary Clos network, a permutation is sent

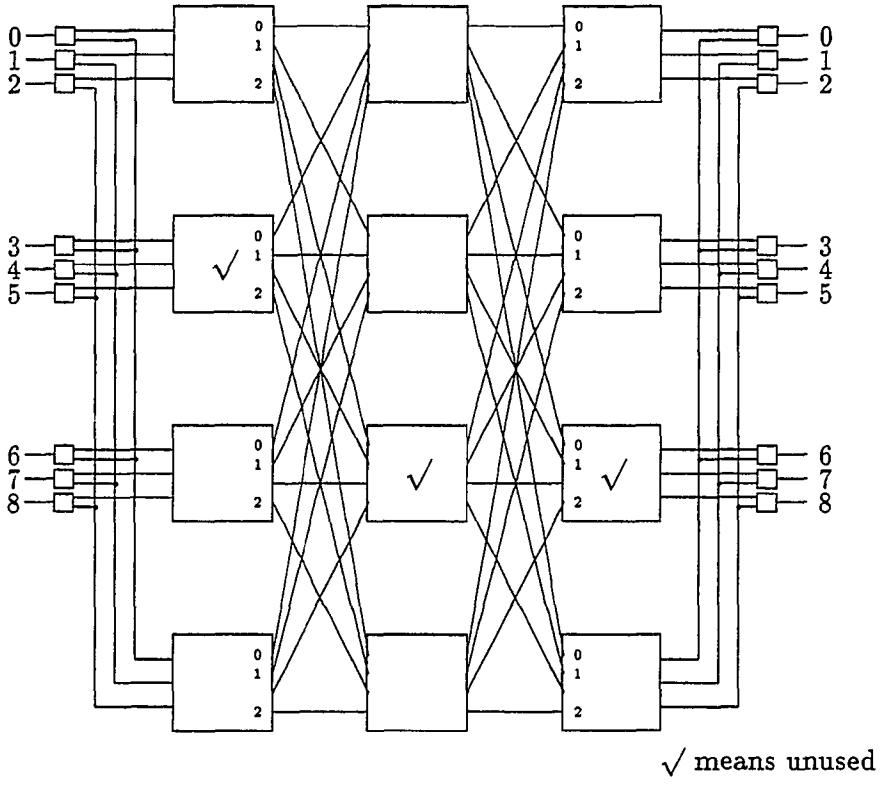


Figure 6.3: FTC reconfigured to accommodate faults in $X(1,0)$, $X(2,1)$ and $X(2,2)$

directly to the routing algorithm where the proper switch settings to realize the permutation are found. However, for the FTC the permutation cannot be submitted to the routing algorithm as is, for $n \neq m$. Instead, a new permutation Q is generated from P as described below. This permutation translation is a flexible routine that can easily accommodate faults in the switches of the input or output stages. Together with adjusting the right demultiplexers and/or multiplexers, this routine can keep the network running after the occurrence of a fault in either or both of the outer stages.

Given permutation $P = \{P_0, P_1, \ldots, P_{N-1}\}$, permutation $Q = \{Q_0, Q_1, \ldots, Q_{N+m-1}\}$ can be created as follows.

If $\lfloor i/m \rfloor \neq f_0$ and $\lfloor P_i/m \rfloor \neq f_2$ then $Q_i = P_i$.

If $\lfloor i/m \rfloor \neq f_0$ and $\lfloor P_i/m \rfloor = f_2$ then $Q_i = N + (P_i \bmod m)$.

If $\lfloor i/m \rfloor = f_0$ and $\lfloor P_i/m \rfloor \neq f_2$ then $Q_{N+(i \bmod m)} = P_i$.

If $\lfloor i/m \rfloor = f_0$ and $\lfloor P_i/m \rfloor = f_2$ then $Q_{N+(i \bmod m)} = N + (P_i \bmod m)$.

The m remaining elements in Q are formed by arbitrarily mapping the m labeled outputs of $X(f_0, 0)$ onto the m labeled inputs of $X(f_2, 2)$.

To illustrate with an example on the FTC shown in Figure 6.2, consider the permutation

$$P = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 3 & 4 & 8 & 7 & 6 & 1 & 2 & 5 & 0 \end{pmatrix}.$$

The realization of the element $\binom{i}{P_i}$ means that inlet i is to be connected to outlet P_i on the actual network.

Initially, let the unused switches be $X(3,0)$, $X(3,1)$ and $X(3,2)$ in the three network stages, as shown in Figure 6.2. Recall that this is the configuration suggested to be used under normal conditions. Then permutation Q, according to the

rules set forth above will be

$$Q = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\ 3 & 4 & 8 & 7 & 6 & 1 & 2 & 5 & 0 & x & x & x \end{pmatrix}.$$

Where $x \in \{9, 10, 11\}$ and the mapping is one-to-one and onto. The condensed matrix representation of P is

$$H_3 = \begin{bmatrix} 0 & 2 & 1 \\ 1 & 0 & 2 \\ 2 & 1 & 0 \end{bmatrix}.$$

On the other hand, the condensed matrix representation of Q is

$$H_3 = \begin{bmatrix} 0 & 2 & 1 & 0 \\ 1 & 0 & 2 & 0 \\ 2 & 1 & 0 & 0 \\ 0 & 0 & 0 & 3 \end{bmatrix}.$$

It is obvious that the size of the matrix increases by exactly one row and one column. This is because of the two unused switches $X(3,0)$ and $X(3,2)$.

Using Neiman's algorithm for routing, it can be seen that the new permutation does not complicate decomposing the matrix. That is because the algorithm uses the condensed matrix representation of the network and proceeds by selecting a non-zero element from a row (or a column) at a time. Problems arise in this algorithm if there are rows or columns with more than one non-zero element. But since the condensed matrix of the FTC introduces a row and a column with each having only one non-zero element, the algorithm will be forced to choose this element every pass of the decomposition process. It is obvious that the new condensed matrix is not easier to decompose than the old one either.

To give another translation example, assume that switches $X(1,0)$, $X(2,1)$ and $X(2,2)$ of Figure 6.2 suddenly failed. This situation is depicted in Figure 6.3. Then, the new values for the unused switches will be $f_0 = 1$, $f_1 = 2$ and $f_2 = 2$. Due to

the failure of $X(2,1)$, the inward terminals of stages 0 and 1 should be relabelled. Specifically, inward terminal number 2 of each switch should be left out in assigning the numbers. The failure of $X(1,0)$ and $X(2,2)$ affects the permutation translation. Permutation P given before is translated according to the rules laid down above to

$$Q = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\ 3 & 4 & 11 & x & x & x & 2 & 5 & 0 & 10 & 9 & 1 \end{pmatrix},$$

where, $x \in \{6,7,8\}$ and the mapping is one-to-one and onto. The routing result will be implemented by all the switches except the ones that are defective, namely, $X(1,0)$, $X(2,1)$ and $X(2,2)$. The matrix representation of permutation Q above is

$$H_3 = \begin{bmatrix} 0 & 2 & 0 & 1 \\ 0 & 0 & 3 & 0 \\ 2 & 1 & 0 & 0 \\ 1 & 0 & 0 & 2 \end{bmatrix}.$$

Again, if Neiman's algorithm is used, the new element in H_3, namely $H_3(2,3)$, will neither complicate nor facilitate the algorithm. Notice that the element added to H_m is always $H_m(f_0+1, f_2+1) = m$. The time complexity of routing for an interconnection network is an important measure of the efficiency of the network. It is shown below that if Neiman's algorithm is used, the time complexity of routing an FTC is equal to that of routing the ordinary Clos network.

The time complexity of Neiman's algorithm is

$$O(T) = O(mk^4).$$

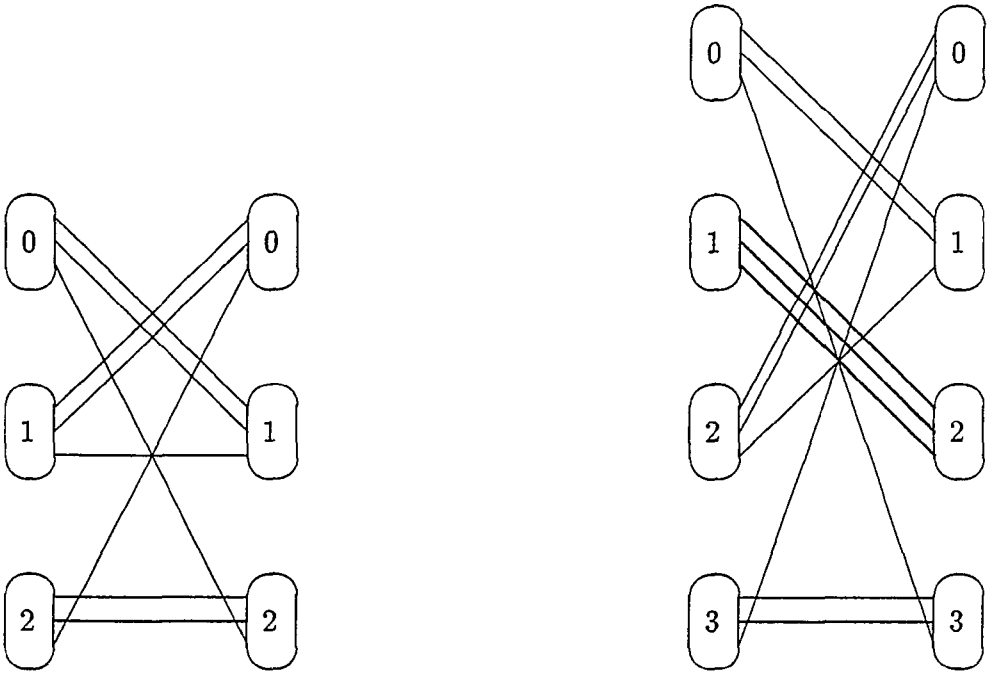
The FTC has one extra switch in the outer stages, i.e, $k+1$ switches in each outer stage. So if $k+1$ is substituted in the above expression for k, then

$$\begin{aligned} O(T) &= O(m \times (k+1)^4) \\ &= O(m(k^4 + 4k^3 + 6k^2 + 4k + 1)) \\ &= O(mk^4) \end{aligned}$$

Likewise, if a graph-based algorithm is used, it can be shown that the time complexity for routing remains the same.

Using a graph-based algorithm, the new row and column added to the condensed matrix represent two vertices in the bipartite graph. These two vertices are connected by m edges, where m is as defined above. Figure 6.4 shows the graph representation of both the ordinary Clos network of Figure 6.1 and the FTC of Figure 6.3 as they realize permutation $P = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 3 & 4 & 8 & 7 & 6 & 1 & 2 & 5 & 0 \end{pmatrix}$.

Since $m = 3$, three edges, shown as dark lines in Figure 6.4b, will be stretched between the two extra vertices. It can be seen that edge-coloring the new graph is



a) Ordinary network b) FTC

Figure 6.4: The graph representation of both the ordinary network of Figure 6.1 and the FTC of Figure 6.3 as they realize permutation P

neither easier nor more difficult than for the original graph. That is because three

colors will be chosen and assigned to the edges such that no two edges incident on the same vertex have the same color. It is easy to see that in the new graph, each of the three added edges will be assigned a color in a straightforward manner. In fact, any algorithm with a polynomial time complexity will have the same time complexity on the FTC as on the ordinary Clos network.

The discussion so far has dealt only with switch faults. Link faults can be easily handled as follows. If a link between switches $X(i,j)$ and $X(a,j+1)$, $0 \leq j \leq 1$, fails, the case is treated as if the two switches have failed, and the procedures discussed above are applied. Recall that the FTC is capable of tolerating more than one simultaneously faulty switch provided that there is only one such switch per stage. This solution has the advantage of keeping the reconfiguration process as simple as possible. More elaborate solutions can be designed but will complicate the ability of the network to reconfigure itself easily.

6.4 Reliability Analysis

Here the reliability [82] of both the ordinary Clos network and the FTC are examined. First, define the reliability, r, of a single switch as the probability that the switch does not fail over a period of time τ. Then, $f = (1 - r)$ is the probability that the switch fails in the same period τ. Similarly, define the reliability R of the network, ordinary or FTC, as the probability that the network does not fail over a period of time τ. Then $F = (1 - R)$ is the probability that the network fails in the same period τ. A switch fails if it cannot realize, partially or completely, a mapping of its inputs onto its outputs. Similarly, a network fails if it cannot realize, partially or completely, a mapping of its inlets onto its outlets.

Evidently, for the ordinary Clos network to be fully operational over the period of time τ, *all* of its switches must be operational over the same period of time τ. For

simplicity, assume that all the switches have the same reliability r. Therefore, the reliability of the ordinary network, assuming statistical independence (independent failure events), is

$$R_{ordinary} = r^{2k+m} \tag{6.1}$$

where $2k + m$ is the number of switches in the ordinary Clos network.

For the FTC, the network will remain fully operational if up to one switch in every stage fails. Let R_0, R_1 and R_2 be the reliabilities of stages 0, 1, and 2, respectively. Clearly, the three stages are statistically independent. Thus the reliability of the network is

$$R_{FTC} = R_0 R_1 R_2 \tag{6.2}$$

The reliability of the first stage, R_0, is the probability that at least k out of the $k + 1$ first stage switches, will be operational. Alternatively, if F_0 is the probability that the first stage fails, then

$$R_0 = 1 - F_0 \tag{6.3}$$

For stage 0 to fail, given that there is one extra switch, at least two switches will have to fail. This is a case of binomial distribution or Bernoulli trials [68], for which F_0 can be written as

$$\begin{aligned} F_0 &= \sum_{i=0}^{k-1} \binom{k+1}{i} r^i f^{k+1-i} \\ &= \sum_{i=0}^{k-1} \binom{k+1}{i} r^i (1-r)^{k+1-i} \end{aligned} \tag{6.4}$$

where $\binom{k+1}{i}$ is the combination of $k + 1$ taking i at a time.

Substituting in Equation 6.3 and realizing that $R_0 = R_2$ due to symmetry, it follows that

$$R_0 = R_2 = 1 - \sum_{i=0}^{k-1} \binom{k+1}{i} r^i (1-r)^{k+1-i} \tag{6.5}$$

A similar analysis shows that the reliability of the middle stage is

$$R_1 = 1 - \sum_{i=0}^{m-1} \binom{m+1}{i} r^i (1-r)^{m+1-i} \tag{6.6}$$

Substituting Equations 6.5 and 6.6 in Equation 6.2 yields,

$$R_{FTC} = \left\{1 - \sum_{i=0}^{k-1} \binom{k+1}{i} r^i (1-r)^{k+1-i}\right\}^2 \left\{1 - \sum_{i=0}^{m-1} \binom{m+1}{i} r^i (1-r)^{m+1-i}\right\} \tag{6.7}$$

Equations 6.1 and 6.7 thus represent the reliabilities of the two networks, the ordinary and the fault tolerant. They are plotted in Figure 6.5 for $m = 6$ and $r = 0.98$. The reliability of both networks drops as N increases. This is due to the fact that m is constant and therefore a larger N implies a larger number of switches in the network (at least in the two outer stages). Intuitively, the more components the network has, the less reliable it is. It is, therefore, understandable why the reliability of both networks falls as N increases. However, it can be seen that for the same N, $N > 0$, the reliability of the FTC is higher than that of the ordinary Clos network.

It can also be seen from Figure 6.5 that as N increases, the reliability of the SFT becomes considerably higher than that of the ordinary network. That is because, the higher the number of switches of the network is, the higher is its vulnerability to failure. The existence of one more switch in the FTC makes a single failure in the network insignificant. Therefore, the FTC is recommended for networks where there are a large number of switches.

To see the effect of the reliability of the individual switch on the reliability of the ordinary network, and on the need for an FTC, Equations 6.1 and 6.7 are replotted in Figure 6.6 for $r = 0.8$. Notice that the horizontal axis is different from that of Figure 6.5. It can be seen, first, that the reliabilities of both networks are much less

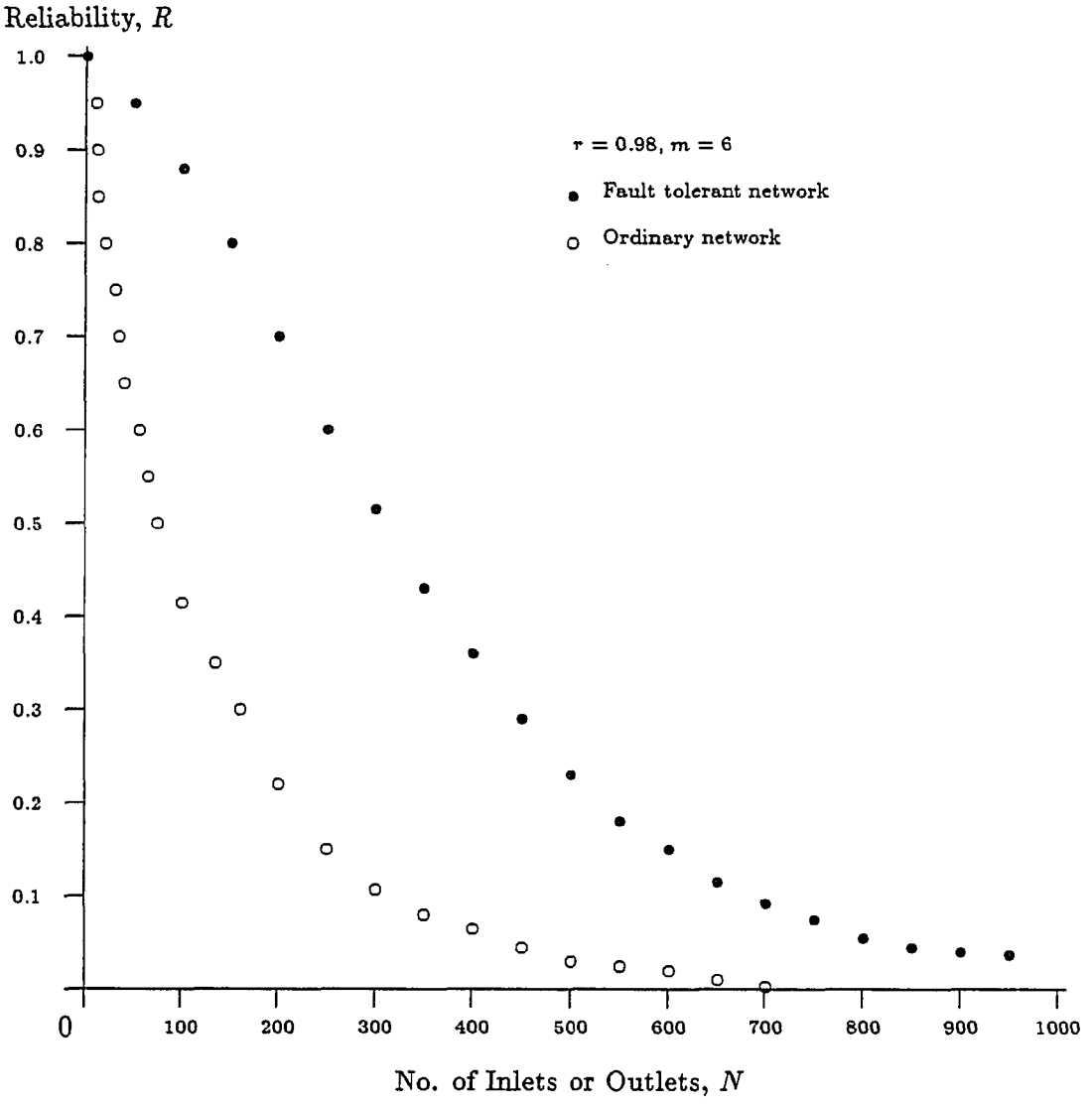


Figure 6.5: Reliability vs. N for both the ordinary network and the FTC, for $r = 0.98$

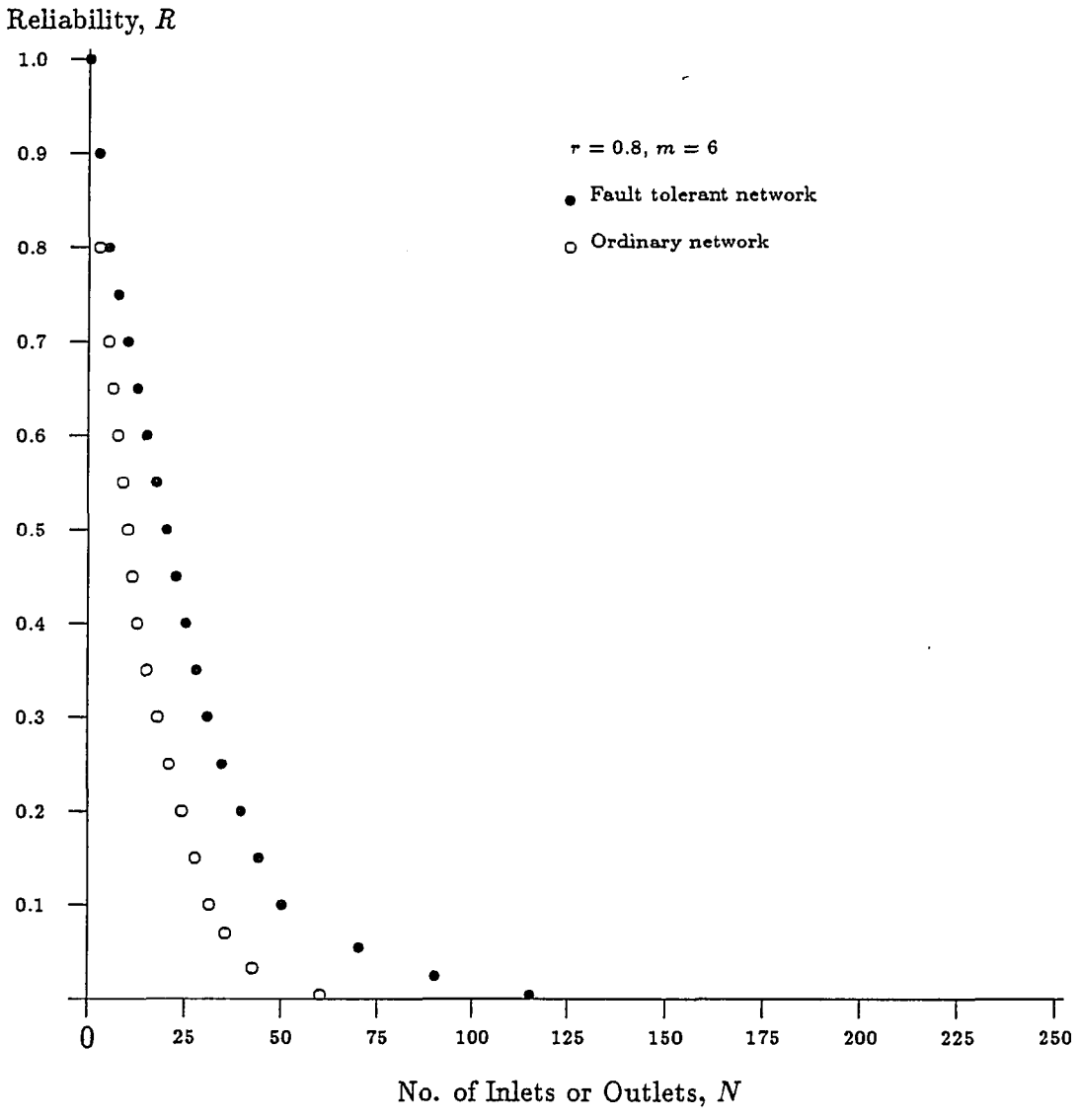


Figure 6.6: Reliability vs. N for both the ordinary network and the FTC, for $r = 0.8$

than those of Figure 6.5. That is understandable because the switches represent the building blocks for the network, and the reliability of the network is determined mainly by the reliability of its switches. Second, notice that the reliability of the FTC is greatly higher than that of the ordinary network over a wider range of N than the case in Figure 6.5. Indeed, the FTC is more beneficial for networks with poor switch reliabilities. In the limiting case, $r = 1$, there is clearly no need for any fault tolerance (recall that what is being said about switches, includes in fact both switches and links).

6.5 Generalization to More Than One Extra Switch per Stage

When more than one switch is added to every stage, in the same manner described for the FTC, greater reliability is expected. To verify that, Equation 6.7 will be generalized to the case where x switches are added to each of stages 0 and 2, and y switches are added to stage 1.

Using the same procedure used to derive Equation 6.7, it can be shown that the reliability of the new network, R_{more}, is

$$R_{more} = \left\{1 - \sum_{i=0}^{k-1} \binom{k+x}{i} r^i (1-r)^{k+x-i}\right\}^2 \left\{1 - \sum_{i=0}^{m-1} \binom{m+y}{i} r^i (1-r)^{m+y-i}\right\} \tag{6.8}$$

This equation is used in Figures 6.7 and 6.8 to show the ratio of the reliability of a fault tolerant Clos network and the reliability of its ordinary version. First, Figure 6.7 shows the reliability ratio for four cases, namely, when 1, 2, 3, and 4 extra switches per stage are used (that is,$x = y = 1$, 2, 3 and 4). In all cases the reliability of the individual switch is $r = 0.8$. Since m is fixed, the horizontal

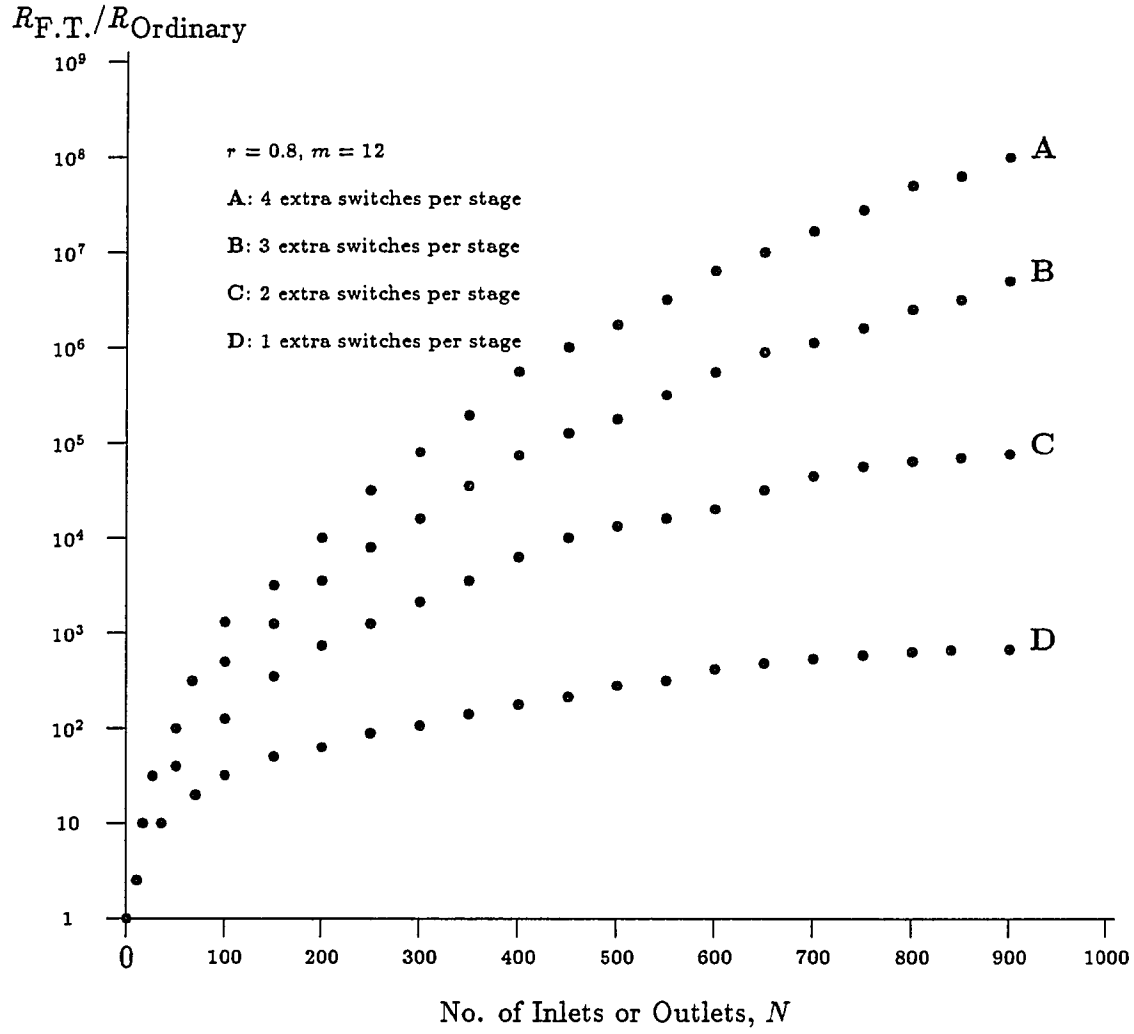


Figure 6.7: Gain in reliability vs. N for various fault tolerant networks with $r = 0.8$

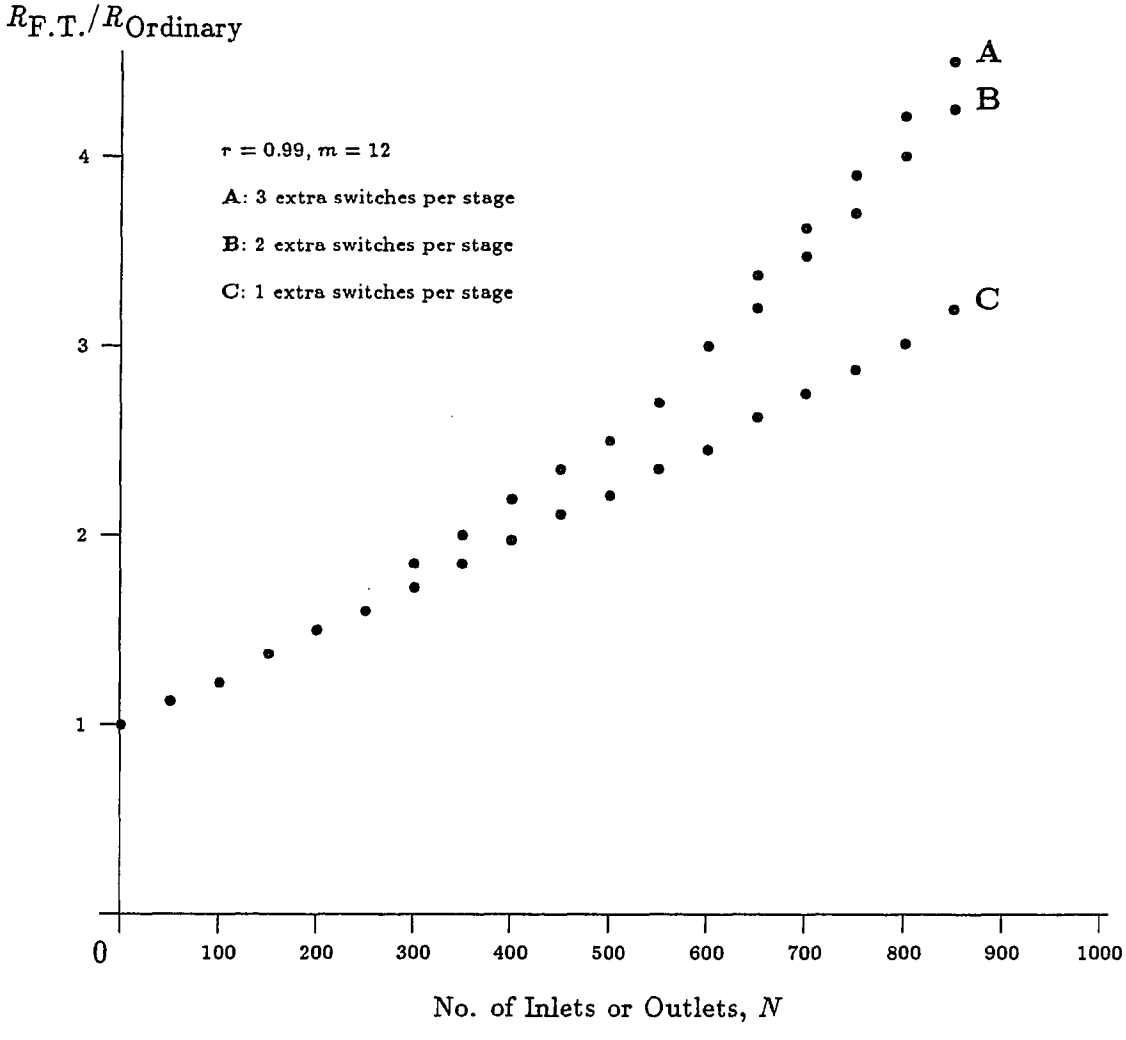


Figure 6.8: Gain in reliability vs. N for various fault tolerant networks with $r = 0.99$

axis really represents the number of switches in the network. It can be seen how the reliability of the Clos networks can be increased by many orders of magnitude just by adding a few switches. The gain in reliability monotonically increases as N increases as concluded before. However, this increase tends to saturate as N becomes higher and higher. It can be also seen that as the number of extra switches per stage increases, the gain in reliability always increases. At $N = 0$ there is no gain in reliability regardless of the number of extra switches, because $N = 0$ means there is no network.

It was mentioned earlier that if the reliability of the individual switch increases, the reliability of the network, ordinary or fault tolerant, increases. This fact is demonstrated in Figure 6.8, which is similar to Figure 6.7 except that $r = 0.99$. It can be seen that if r is so large, the addition of more than one switch per stage is unwarranted. Unlike the case in Figure 6.7, where the addition of one more switch increased the overall reliability of the network by orders of magnitude, the addition of one more switch in Figure 6.8 increases the gain only slightly. In fact, the curve for the network with $x = y = 4$ could not be drawn here because it coincided with curve for the network with $x = y = 3$ throughout the range of N in the figure. The figure also shows that for small N, adding any number of extra switches per stage yields the same gain in reliability. Therefore, it can be concluded that when the reliability of the individual switches is high, there is no need for adding excessive hardware, especially when N is small.

It is obvious from Figures 6.5 through 6.8 that adding more switches per stage is more advantageous when the number of switches in the network is large. For Clos networks with a small number of switches (implied by small N), the addition of one switch per stage would be sufficient. Adding more switches per stage can be seen to increase the overall reliability of the network. However, reconfiguration of the

network would be more difficult and time consuming. Moreover, the extra switches would increase the hardware of the network and complicate its design.

6.6 Discussion

This chapter shows the design and performance of a fault-tolerant Clos network, the FTC. Clos networks are used mainly to realize permutations. Without any fault tolerance, if a switch in the network fails, the network is rendered inoperative and the system has to be interrupted to put the network back to work. With the fault tolerance introduced here, the network can continue its work uninterrupted with the presence of a fault. That is possible simply because the FTC can reconfigure itself dynamically, by changing the settings of the multiplexers and demultiplexers and using the adaptive permutation translation scheme presented. The defective item can then be repaired during the time at which the system is unused. Besides the fault tolerance the FTC provides, the reliability of the network is greatly enhanced. High reliability means more system availability (the time of an uninterrupted operation). It is seen from the analysis that using this fault tolerance approach is most beneficial when

1. the reliability of the individual switches is poor

2. the number of switches in the network is large

As far as reliability is concerned, adding more than one switch per stage is recommended to a certain number of extra switches. This number depends on the number of switches in the network and the reliability of the individual switch, and can be determined for an optimum value. However, putting a large number of extra switches per stage adds significantly to the network hardware and routing complexity.

Chapter 7

Conclusions

7.1 Summary

This thesis has focused on fault tolerance for interconnection networks in general, and for three networks in particular. The three networks are: the Baseline network, the Beneš network and the Clos network. These three networks have found wide interest in the past three decades. Fault tolerance has become a consideration only recently, after large-scale multiprocessor started to become a reality.

The thesis started by defining a generalized MIN model which was later used systematically to put in perspective the MINs considered in the thesis. This rigorous foundation was a key step to understanding how a given MIN can be made fault-tolerant. In devising fault tolerance techniques for MINs, one should meet two common criteria. First, the fault tolerance mechanism should not add significantly to the hardware complexity of the system. Second, the mechanism should not significantly degrade performance under both normal conditions and faulty conditions.

The two fault tolerance techniques presented in this thesis meet the above mentioned criteria. The two add to the wealth already in the literature. However, they both have features which are unique to them. Taken together they offer a reasonable solution to the fault tolerance problem in a large number of MINs.

Fault tolerant MINs developed according to one of the two techniques suggested possess these important features:

- Using the same switches: The fault-tolerant networks are constructed from the same basic switches the ordinary networks use.
- Using the same routing algorithms: The fault-tolerant networks use the same routing algorithms as the ordinary networks.
- Having the same hardware and routing time complexities: The hardware and routing complexity of the fault-tolerant networks are the same as those of the ordinary ones.

7.2 The SFT Technique

The primary advantage of the SFT technique is that it is not MIN-specific. This means that it can be applied to any MIN with characteristics similar to those of the generalized MIN model. As has been shown, the SFT technique is useful not only under faulty conditions, but also under normal conditions. Among the functions that a bus in an SFT network can perform are broadcasting and blockage relief. These two functions are important in multiprocessor operation. It was shown that if two buses are added to the system in an SFT network, and if the two buses are used under normal conditions to relieve blockage, more permutations can be realized with the help of the two buses than on the original network. Also in this enhanced SFT network, the full recovery property is possible on networks with binary switches.

The cost of the SFT technique is minimal, as it does not require any switch design. Moreover the bus is totally invisible under normal conditions, which causes no negative impacts on routing while there are no faults.

7.3 The Fault-Tolerant Clos (FTC) network

The FTC is suggested as an alternative to using the SFT technique on Clos networks. The main reason is that switch sizes in Clos networks can be so large that using the SFT technique would result in a severely poor performance under faulty conditions. The FTC is characterized by ease of operation and by using little additional hardware. It is shown how the addition of only three switches to the network considerably increases the reliability of the network. Another advantage of the FTC is full recovery. This is particularly important in Clos networks, as the Clos network is primarily a permutation network. Having only the full access capability as a fault tolerance criterion would not be acceptable for a Clos network.

7.4 Open Problems

On the way to solving any problem, one often sees problems that were not noticeable before. In the case of the work done in this thesis, some problems have been observed, and as such they can make good research areas. First, the SFT technique was extended only to two standby buses. A possible SFT approach for networks that use large switches, would be in the form of using more than two standby buses. The optimum number of buses for a given network can be found. Controlling access to such large number of buses, as well studying the performance of the system as a whole would be of interest. Developing such a scheme for the Clos network and comparing it with an FTC of the same size would clarify which approach is more appropriate.

Another extension that can be made to the SFT technique is to make it tolerate more than one fault. This again can be done by increasing the number of buses to be larger than the number of processors affected by at least two worst case failures.

Aside from the fault tolerance problems, some other problems have been observed. In Chapter 5 for example, the number of permutations blocked in a Baseline network in exactly ξ paths, $\mathcal{P}_N^{(\xi)}$, was calculated for $\xi = 0$, 1,2. It is interesting to calculate $\mathcal{P}_N^{(\xi)}$ for all other values of ξ, namely, $3 \leq \xi \leq N - 2^{\lceil \nu/2 \rceil}$.

One last problem concerning the FTC, presented in Chapter 6, is to develop a new routing algorithm that takes advantage of its extra paths available under normal conditions. Such an algorithm could run in less time than those mentioned in Chapter 6, because of the flexibility resulting from the extra paths.

Bibliography

[1] M. Abidi and D. Agrawal. "On Conflict-free Permutations in Multi-stage Interconnection Networks", *Journal of Digital Systems*, vol. V, Summer 1980, pp. 115-134.

[2] G. Adams and H. Siegel. "The Extra Stage Cube: A Fault-Tolerant Interconnection Network for Supersystems", *IEEE Transactions on Computers*, vol. C-31, no. 5, May 1982, pp. 443-454.

[3] _____. "Modifications to Improve the Fault Tolerance of the Extra Stage Cube Interconnection Network", *Proceedings of the 1984 International Conference on Parallel Processing*, 1984, pp. 169-173.

[4] G. Adams, D. Agrawal and H. Siegel. "A Survey and Comparison of Fault-tolerant Multistage Interconnection Networks", *Computer*, June 1987, pp. 14-27.

[5] D. Agrawal. "Testing and Fault Tolerance of Multistage Interconnection Networks", *Computer*, April 1982, pp. 41-53.

[6] _____. "Graph Theoretical Analysis and Design of Multistage Interconnection Networks", *IEEE Transactions on Computers*, vol. C-32, no. 7, July 1983, pp. 637-648.

[7] S. Andresen. "The Looping Algorithm Extended to Base 2^t Rearrangeable Switching Networks", *IEEE Transactions on Communications*, vol. COM-25, no. 10, October 1977, pp. 1057-1063.

[8] J. Baer. "Multiprocessing Systems", *IEEE Transactions on Computers*, vol. C-25, no. 12, December 1976, pp. 613-641.

[9] S. Bandyopadhyay, et. al. "A Cellular Permuter Array", *IEEE Transactions on Computers*, vol. C-21, no. 10, October 1972, pp. 1116-1119.

[10] G. Barnes and S. Lundstrom. "Design and Validation of a Connection Network for Many-Processor Multiprocessor Systems", *Computer*, December 1981, pp. 31-41.

[11] K. Batcher. "The Flip Network in STARAN™", *Proceedings of the 1976 International Conference on Parallel Processing*, 1976, pp. 65-71.

[12] V. Beneš. "On Rearrangeable Three-Stage Connecting Networks", *The Bell System Technical Journal*, vol. XLI, no. 5, September 1962, pp. 1481-1492.

[13] _____. *Mathematical Theory of Connecting Networks and Telephone Traffic*, New York, Academic Press, 1965.

[14] L. Bhuyan. "A Combinatorial Analysis of Multibus Multiprocessors", *Proceedings of 1984 International Conference on Parallel Processing*, August 1984, pp. 225-227.

[15] _____. "Interconnection Networks for Parallel and Distributed Processing", *Computer*, June 1987, pp. 9-12.

[16] C. Cardot. "Comments on *A Simple Algorithm for the Control of Rearrangeable Switching Networks*", *IEEE Transactions on Communications*, vol. COM-34, no. 4, April 1986, p. 395.

[17] J. Carpinelli. *Interconnection Networks: Improved Routing Methods for Clos and Beneš Networks,* Ph.D. Thesis, Rensselaer Polytechnic Institute, Troy, NY, August, 1987.

[18] _____. "Applications of Edge-Coloring Algorithms to Routing on Parallel Computers", *Proceedings of the 3rd International Conference on Supercomputing*, May, 1988.

[19] J. Carpinelli and Y. Oruç. "On the Equivalence of Edge Coloring and Matrix Decomposition Algorithms for Routing in Clos Networks", *Submitted for publication*

[20] _____. "Some Group Theoretic Results Towards a Linear-Time Set Up Algorithm for Beneš Networks", *Proceedings of the 20th Annual Conference on Information Sciences and Systems*, March, 1986.

[21] _____. "Parallel Set-Up Algorithms for Clos Networks Using a Tree-Connected Computer", *Proceedings of the 2nd International Conference on Supercomputing*, May, 1987.

[22] _____. "Matrix Decomposition Algorithms for Dynamic Topology Reconfiguration in Parallel Computers", *Proceedings of the 4th International Conference on Supercomputing*, April, 1989.

[23] J. Carpinelli, C. Lin and M. Singh. "APAP: The Arithmetic Pipeline Analysis Package", *Proceedings of the 19th Annual Pittsburgh Conference on Modeling and Simulation*, May, 1988.

[24] _____. "APAP: A Computer-based Tool for Analyzing Data Flow in Arithmetic Pipelines", *Proceedings of the 1988 Frontiers in Education Conference*, October, 1988.

[25] T. Chen. "Parallelism, Pipelining, and Computer Efficiency", *Computer Design*, vol. 10, no. 1, January 1971, pp. 69-74.

[26] W. Chu. *Advances in Computer Communications and Networking*, Artech House, Dedham, Ma., 1979.

[27] L. Ciminiera and A. Serra. "A Connecting Network with Fault Tolerance Capabilities", *IEEE Transactions on Computers*, vol C-35, no. 6, June 1986, pp. 578-580.

[28] C. Clos. "A Study of Non-blocking Switching Networks", *Bell Systems Technical Journal*, vol. 32, no. 2, March 1953, pp. 406-424.

[29] C. Das and L. Bhuyan. "Bandwidth Availability of Multiple-Bus Multiprocessors", *IEEE Transactions on Computers*, vol. C-34, no. 10, October 1985, pp. 918-926.

[30] W. Davis. *Operating Systems: A Systematic View, 2nd Edition*, Addison-Wesley, Reading, Ma., 1983.

[31] D. Dias and J. Jump. "Analysis and Simulation of Buffered Delta Networks", *IEEE Transactions on Computers*, vol. C-30, no. 4, pp. 273-282.

[32] ———. "Packet Switching Interconnection Networks for Modular Systems", *Computer*, December 1981, pp. 43-53.

[33] ———. "Augmented and Pruned $N \log N$ Multistage Networks: Topology and Performance", *Proceedings of the 1982 International Conference on Parallel Processing*, 1982, pp.10-11.

[34] T. Feng. "A Survey of Interconnection Networks", *Computer*, December 1981, pp. 12-27.

[35] T. Feng and C. Wu. "Fault-Diagnosis for a Class of Multistage Interconnection Networks", *IEEE Transactions on Computers*, vol. C-30, no. 10, October 1981, pp. 743-758.

[36] M. Flynn. "Very High-Speed Computing Systems", *Proceedings of the IEEE*, vol. 54, December 1966, pp. 1901-1909.

[37] J. Fraleigh. *A First Course in Abstract Algebra*, Reading, MA, Addison-Wesley, 1967.

[38] H. Gabow. "Using Euler Partitions to Edge Color Bipartite Multigraphs", *International Journal of Computer and Information Science*, vol. 5, 1976, pp. 345-355.

[39] I. Gazit and M. Malek. "Fault Tolerance Capabilities in Multistage Network-Based Multicomputer Systems", *IEEE Transactions on Computers*, vol. 37, no. 7, July 1988, pp 788-798.

[40] G. Goke and G. Lipovski. "Banyan Networks for Partitioning Multimicroprocessor Systems", *Proceedings of the 1st Annual Symposium on Computer Architecture*, December 1973, pp. 21-28.

[41] S. Golomb. "Permutation by Cutting and Shuffling", *SIAM Review*, vol. 3, October 1961, pp. 293-297.

[42] T. Hallin and M. Flynn. "Pipelining of Arithmetic Functions", *IEEE Transactions on Computers*, vol. C-21, no. 8, August 1972, pp. 880-886.

[43] J. Hopcroft and R. Karp. "An $n^{\frac{5}{2}}$ Algorithm for Maximum Matchings in Bipartite Graphs", *SIAM Journal on Computing*, vol. 2, no. 4, December 1973, pp. 225-231.

[44] A. Jajszczyk. "A Simple Algorithm for the Control of Rearrangeable Switching Networks", *IEEE Transactions on Communications*, vol. Com-33, no. 2, February 1985, pp. 169-171.

[45] M. Jeng and H. Siegel. "A Fault-Tolerant Multistage Interconnection Network for the Multiprocessor Systems Using Dynamic Redundancy", *Proceedings of the 6th International Conference on Distributed Computing Systems*, 1986, pp.70-77.

[46] L. Kinny and R. Arnold. "Analysis of a Multiprocessor System with a Shared Bus", *Proceedings of the 5th Annual Symposium on Computer Architecture*, April 1978, pp. 89-95.

[47] C. Kruskal and M. Snir. "The Performance of Multistage Interconnection Networks for Multiprocessors", *IEEE Transactions on Computers*, vol. C-32, no. 12, December 1983, pp. 1091-1098.

[48] M. Kubale. "Comments on *Decomposition of Permutation Networks*", *IEEE Transactions on Computers*, vol. C-31, no. 3, March 1982, p. 265.

[49] V. Kumar and S. Reddy. "Augmented Shuffle-Exchange Multistage Interconnection Network", *Computer*, June 1987, pp. 30-40.

[50] T. Lang, M. Valero and I. Algre. "Bandwidth of Crossbar and Multiple-Bus Connections for Multiprocessors", *IEEE Transactions on Computers*, vol. C-31, no. 12, December 1982, pp. 1227-1233.

[51] D. Lawrie. "Access and Alignment of Data in an Array Processor", *IEEE Transactions on Computers*, vol. 24, no. 12, December 1975, pp. 1145-1155.

[52] J. Lilienkamp, D. Lawrie and P. Yew. "A Fault Tolerant Interconnection Network Using Error Correcting Codes", *Proceedings of the 1982 International Conference on Parallel Processing*, 1982, pp. 123-125.

[53] J. Lenfant. "Parallel Permutations of Data: A Beneš Network Control Algorithm for Frequently Used Permutations", *IEEE Transactions on Computers*, vol. 27, no. 7, July 1978, pp. 637-647.

[54] G. Lev, N. Pippenger and L. Valiant. "A Fast Parallel Algorithm for Routing in Permutation Networks", *IEEE Transactions on Computers*, vol. C-30, no. 2, February 1981, pp. 93-100.

[55] W. Lin and C. Wu. "Design of a 2×2 Fault-Tolerant Switching Element", *Proceedings of the 9th Annual Symposium on Computer Architecture*, 1982, pp. 181-189.

[56] H. Lorin. *Parallelism in Hardware and Software*, Prentice-Hall, Englewood Cliffs, N.J., 1972.

[57] M. Mano. *Computer System Architecture, 2nd edition*, Prentice-Hall, New Jersey, 1982.

[58] M. Marsan, G. Blbo, G. Conte and F. Gregoretti. "Modelling Bus Contention and Memory Interference in a Multiprocessor System", *IEEE Transactions on Computers*, vol. C-32, no. 1, January 1983, pp. 60-72.

[59] R. McMillen and H. Siegel. "Performance and Fault Tolerance Improvements in the Inverse Augmented Data Manipulator Network", *Proceedings of the 9th Annual Symposium on Computer Architecture*, April 1982, pp. 63-72.

[60] T. Mudge, J. Hayes and D. Winsor. "Multiple Bus Architectures", *Computer*, June 1987, pp. 42-48.

[61] T. Mudge et al. "Analysis of Multiple Bus Interconnection Networks", *Proceedings of the 1984 International Conference on Parallel Processing*, August 1984, pp. 228-232.

[62] T. Mudge and H. Al-Sadoun. "A Semi-Markov Model for the Performance of Multiple-Bus Systems", *IEEE Transactions on Computers*, vol. C-34, no. 10, October 1985, pp. 934-942.

[63] D. Nassimi and S. Sahni. "A Self-Routing Beneš Network and Parallel Permutation Algorithms", *IEEE Transactions on Computers*, vol. 30, no. 5, May 1981, pp. 332-340.

[64] V. Neiman. "Structure et Command Optimales de Reseaux de Connexion sans Blocage", *Annales des Telecommunications*, vol. 24, July-August 1969, pp. 232-238.

[65] D. Opferman and N Tsao-Wu. "On a Class of Rearrangeable Switching Networks, Part I: control Algorithm", *Bell Systems Technical Journal*, vol. 50, no. 5, May-June 1971, pp. 1579-1600.

[66] Y. Oruç. *Interconnection Networks: Group Theoretic Modeling*, Ph.D. Thesis, Syracuse University, Syracuse, NY, 1983.

[67] K. Padmanabhan and D. Lawrie. "A Class of Redundant Path Multistage Interconnection Networks", *IEEE Transactions on Computers*, vol C-32, no. 12, December 1983, pp. 1099-1108.

[68] A. Pages and M. Gondran. *System Reliability: Evaluation and Prediction in Engineering*, Springer-Verlag, New York, 1986.

[69] J. Patel. "Performance of Processor-Memory Interconnections for Multiprocessors", *IEEE Transactions on Computers*, vol. C-30, no. 10, October 1981, pp. 771-780.

[70] R. Pearce, J. Field and W. Little. "Asynchronous Arbiter Module", *IEEE Transactions on Computers*, vol. 24, no. 9, September 1975, pp. 931-932.

[71] M. Pease. "The Indirect Binary n-Cube Microprocessor Array", *IEEE Transactions on Computers*, vol. 26, no. 5, May 1977, pp. 458-473.

[72] C. Raghavendra and A. Varma. "INDRA: A Class of Interconnection Networks with Redundant Paths", *Proceedings of the 1984 Real Time Systems Symposium*, 1984, pp. 153-164.

[73] H. Ramanujam. "Decomposition of Permutation Networks", *IEEE Transactions on Computers*, vol. C-22, no. 7, July 1973, pp. 639-643.

[74] B. Raw. "Program Behavior and the Performance of Interleaved Memories", *IEEE Transactions on Computers*, vol. C-28, no. 3, March 1979, pp. 191-199.

[75] S. Reddy and V. Kumar. "On Fault Tolerant Multistage Interconnection Networks", *Proceedings of the 1984 International Conference on Parallel Processing*, 1984, pp. 155-164.

[76] J. Shen and J. Hayes. "Fault-Tolerance of Dynamic-Full-Access Interconnection Networks", *IEEE Transactions on Computers*, vol. C-33, no. 3, March 1984, pp. 241-248.

[77] H. Siegel and S. Smith. "Study of Multistage SIMD Interconnection Networks", *Proceedings of the 5th Annual Symposium on Computer Architecture*, April 1978, pp. 223-229.

[78] H. Siegel. "Interconnection Networks for SIMD Machines", *Computer*, vol. 12, June 1979, pp. 57-65.

[79] H. Siegel and R. McMillen. "The Multistage Cube: A Versatile Interconnection Network", *Computer*, December 1981, pp. 65-76.

[80] H. Stone. "Parallel Processing with the Perfect Shuffle", *IEEE Transactions on Computers*, vol. C-20, no. 2, February 1971, pp. 153-161.

[81] S. Thanawastien and V. Nelson. "Interference Analysis of Shuffle/Exchange Networks", *IEEE Transactions on Computers*, vol. C-30, no. 8, August 1981, pp. 545-556.

[82] P. Tobias and D. Trindade. *Applied Reliability*, Van Nostrand Reinhold, New York, 1986.

[83] M. Valero, et. al. "A Performance Evaluation of the Multiple-Bus Network for Multiprocessor Systems", *Proceedings of ACM Conference Performance Evaluation*, August 1984, pp. 200-206.

[84] V. Vizing. "On an Estimate of the Chromatic Class of a p-graph", Diskret. Analiz., no. 3, 1964, pp. 25-30.

[85] C. Wu and T. Feng. "On a Class of Multistage Interconnection Networks", *IEEE Transactions on Computers*, vol. C-29, no.8, August 1980, pp. 694-702.

[86] _____. "The Reverse-Exchange Interconnection Network", *IEEE Transactions on Computers*, vol. C-29, no.9, September 1980, pp. 694-702.

[87] _____. "The Universality of the Shuffle-Exchange Network", *IEEE Transactions on Computers*, vol. C-30, no. 5, May 1981, pp. 324-332.

[88] K. Yoon and W. Hegazy. "The Extra Stage Gamma Network", *Proceedings of the 13th Annual Symposium on Computer Architecture*, 1986, pp. 175-182.