5-31-1993

The postbus fault tolerant CLOS network

Udayabhanu Sarangapani
New Jersey Institute of Technology

ABSTRACT

The Postbus Fault Tolerant Clos Network

by

Udayabhanu Sarangapani

The trend in modern computing is to develop multiprocessor systems with hundreds, even thousands, of processors and memory modules. The task of providing communication paths among all these units is not a trivial one. For a small number of functional units, direct connections could be used but for large systems interconnection networks have to be used. Multistage Interconnection Networks (MINs), provide a dynamic means for interconnecting processors and memory in a multiprocessor system. These networks are built with switches in each stage.

The Clos network is a well defined family of MINs and consists of three stages. The ordinary Clos network has no fault tolerance capability. This thesis work presents the design for a modified Clos network by incorporating hardware redundancy. The excess hardware is in the form of an extra switch in the middle stage, demultiplexers and multiplexers in the outer stages and two sets of buses.

Algorithms are developed to set the states of the demultiplexers and multiplexers. It is shown that the proposed design is able to withstand one faulty switch in each stage and still retain the property of full recovery, i.e., the network is still able to realize any given input-output permutation.

THE POSTBUS FAULT TOLERANT CLOS NETWORK

by
Udayabhanu Sarangapani

A Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Electrical Engineering

Department of Electrical and Computer Engineering

May 1993

APPROVAL PAGE

The Postbus Fault Tolerant Clos Network

Udayabhanu Sarangapani

May 1993

5/13/93

Dr. John D. Carpinelli, Thesis Adviser — Date
Assistant Professor of Electrical and Computer Engineering, NJIT.

5/12/93

Dr. Sotirios Ziavras, Committee Member — Date
Assistant Professor of Electrical and Computer Engineering, NJIT.

5/13/93

Dr. Michael Palis, Committee Member — Date
Associate Professor of Electrical and Computer Engineering, NJIT.

BIOGRAPHICAL SKETCH

Author: Udayabhanu Sarangapani

Degree: Master of Science in Electrical Engineering

Date: May 1993

Date of Birth:

Place of Birth:

Undergraduate and Graduate Education:

- Master of Science in Electrical Engineering,
  New Jersey Institute of Technology, Newark, NJ, 1993

- Bachelor of Engineering in Electronics
  Bangalore University, Bangalore, India, 1990

Major: Electrical Enginering

This thesis is dedicated to

my parents

ACKNOWLEDGMENT

I wish to express my deep felt gratitude to my advisor, Dr. John Carpinelli. Time and again he put me back on the right track. Working with him has been an enlightening and wonderful experience.

Special thanks are due to the members of my committee: Dr. Sotirios Ziavras and Dr. Michael Palis. Their many suggestions have enhanced the quality of this thesis.

I would like to acknowledge the support and encouragement of my family. Their innumerable sacrifices have made this work possible.

Finally, I am thankful to God, without whose Grace nothing is possible.

TABLE OF CONTENTS

Chapter **Page**

1 INTRODUCTION....1
1.1 Parallel Processing....1
1.2 The Need for Interconnection Networks....2
1.3 Fault Tolerance....3
1.4 Motivation....3
1.5 Outline....4
2 NOTATION AND FAULT MODEL....5
2.1 Notation....5
2.2 Fault Model....6
3 MIN IMPLEMENTATIONS....7
3.1 The Baseline Network....7
3.1.1 Design of the Baseline Network....7
3.1.2 Routing the Baseline Network....8
3.2 The Clos Network....10
3.2.1 Design of the Clos Network....10
3.2.2 Routing the Clos Network....10
3.3 The Benes Network....13
3.3.1 Design of the Benes Network....13
3.3.2 Routing the Benes Network....14
3.4 Other MIN Implementations....14
4 FAULT TOLERANT MINS....16
4.1 The Extra Stage Cube (ESC) Network....17
4.2 The Simple Fault Tolerant Baseline (SFTB) Network....18
4.3 The Fault Tolerant Clos (FTC) Network....20
4.4 Fault Detection and Location....21

Chapter **Page**

5 THE POSTBUS FAULT TOLERANT CLOS (PFTC) NETWORK.....22
5.1 Design of the PFTC.....22
5.2 Fault Recovery on the PFTC.....23
5.3 Reliability Analysis.....28
5.3.1 Fundamentals of Reliability.....28
5.3.2 Reliability Analysis of the PFTC.....30
5.4 Discussion.....32
6 CONCLUSIONS.....33
6.1 Summary.....33
6.2 Future Work.....33
BIBLIOGRAPHY.....35

LIST OF FIGURES

Figure		Page
3.1	Shuffling 8 Objects	facing 7
3.2	8 x 8 Baseline Network	facing 8
3.3	8 x 8 Omega Network	facing 8
3.4	Legal states of a binary cell	9
3.5	8 x 8 Clos Network	facing 10
3.6	8 x 8 Benes Network	facing 13
4.1	8 x 8 Extra Stage Cube Network	facing 17
4.2	8 x 8 SFTB Network	facing 18
4.3	9 x 9 Fault Tolerant Clos Network	facing 20
5.1	9 x 9 Postbus Fault Tolerant Clos (PFTC) Network	facing 22
5.2	Reliability Curves with r = 0.98	facing 31
5.3	Reliability Curves with r = 0.99	facing 32

CHAPTER 1
INTRODUCTION

1.1 Parallel Processing

As the processing capacity of computers increases, so does the complexity of the problems that these machines are asked to solve. As a result there will always be a demand for more computational power. In the early years of electronic computing this advance was consistently achieved by improving the architecture of the processor. These improvements soon brought the designers to the physical limits of the devices and it was realized that the answer to further performance enhancement lay in a different direction. This approach involved carrying out the computational tasks in a parallel manner. Different avenues of achieving parallelism have been explored and many of them are used in combination.

Software parallelism was introduced as a way of maximizing the throughput of expensive systems. The idea was to keep the physical resources of a computer continually busy. To this end, multiple processes were loaded onto the system and these processes shared the resources in a manner determined by the operating system [14]. 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.

Hardware parallelism involves the concept of having multiple functional units. This sort of parallelism can occur at the computer level, sub-processor level or at the processor level. When parallelism takes place at the computer level, it is called distributed computing [12]. The computational load is distributed among many computers. Different computers may be doing different kinds of tasks or the load may be distributed in a symmetric manner. In either case, all these computers are connected by a communication network and work independently and asynchronously. They exchange data and results through the connecting network.

One way of achieving parallelism at the sub-processor level is by using pipelining. In this implementation a processor performs the four basic operations on an instruction - instruction fetch, decode, execute and write back of the result - in an overlapped manner. An instruction is fetched and sent to the decoding unit. While it is being decoded the first one is being executed and so on. In the steady state, ideally, one instruction is being executed per clock cycle.

On the processor level, parallelism is achieved by having multiprocessors [5]. There are basically two types of architectures - shared memory and message passing [19]. In the shared memory system all the processors share the common memory and communication between processors is achieved through shared variables in the memory. In the message passing type of multiprocessors, each processor has its exclusive memory and interprocessor communication is achieved through direct connection or through a communicating network.

1.2 The Need for Interconnection Networks

The modern trend in parallel computing is massive parallelism - the use of thousands of processors in a system. Clearly, connecting all these processors directly is a Herculean task. In the shared memory architecture, the common memory is divided into modules and conceptually it is possible for all processors to share a common system bus that lets them access all the memory modules [20]. However, if the number of processors is relatively high, the system performance is degraded, since only one processor can use the bus at a time. The alternative is to use crossbar switches. An N X M crossbar switch has N inputs and M outputs. Conceptually it can be thought of as two crossed sets of parallel conducting bars placed one above the other. To complete a connection between an input and an output, the corresponding crossed bars are connected by a switch. The main drawback of this scheme is its high cost for high values of N and M [9].

Between the two extremes lies the compromise candidate: the Multistage Interconnection Network (MIN). MINs are formed by stages of small switches.

Each stage is connected to the next through a set of links. By the proper setting of these switches any inlet can be connected to any outlet. These networks are similar to those used in telephone switches and much of the basic theories derive from the work done in early telecommunication research.

1.3 Fault Tolerance

Practical implementations of MINs involve large numbers of processors which are used in high performance computers. Such MINs have a relatively high hardware complexity and there is a chance that one or more of the components may fail. Any such failure could severely degrade system performance. The network may become unable to realize any arbitrary inlet-outlet connection. Thus there is a need for a degree of fault tolerance. In a fault tolerant MIN the occurrence of a fault lowers system performance but does not cause the system to crash. This is called graceful degradation and is the basic criterion for any fault tolerance design.

1.4 Motivation

Fault tolerance is of critical importance in large systems. These systems are generally very expensive and it is of crucial importance to reduce their down time. The Clos network is an attractive alternative to such interconnection schemes as point-to-point connections and crossbar switches, especially when one is considering large systems. Hence it is critical that these networks have some degree of fault tolerance.

Except for the work done by Nassar [29], no work has been done on fault tolerance schemes for the Clos network. The motivation for this thesis was to find an alternative method to make this family of networks fault tolerant, one that uses fewer switches.

The following chapters describe the design and operation of the Postbus Fault Tolerant Clos (PFTC) Network. This network can withstand one fault in each stage and exhibits the property of graceful degradation.

1.5 Outline

The remainder of this thesis is organized as follows. Chapter 2 lists the symbols used in this work, along with their meanings. The definition and various aspects of a fault tolerance model are discussed. Chapter 3 is a survey of some of the well known MIN implementations. The structure and routing techniques of the Baseline, Benes and Clos networks are discussed. In Chapter 4, some of the existing fault tolerance schemes are presented. The construction and operation of the Extra Stage Cube (ESC), the Simple Fault Tolerant Baseline (SFTB) and the Fault Tolerant Clos (FTC) networks are elaborated upon. This chapter also discusses the concepts of fault detection and location. In Chapter 5, the Postbus Fault Tolerant Clos (PFTC) network is described in detail. The design and routing algorithms for this network are explained. Fault recovery for the various types of faults that can occur is analyzed. This chapter includes a reliability analysis of the PFTC. Basic concepts of reliability are presented as background material. The PFTC is compared with the ordinary Clos network in terms of network reliability and it is shown that the PFTC performs more reliably than the Clos network under all circumstances. Chapter 6 lists the conclusions derived from the work done in this thesis. Suggestions for future work in this direction are given.

CHAPTER 2
NOTATION AND FAULT MODEL

2.1 Notation

The following notation is used in this thesis.

i,i^* : (general indices), inlet number in a permutation

j,j^* : (general indices), outlet number in a permutation

$X(i,j)$: 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, the number of inputs to a first stage switch or number of outputs from a third stage switch

n : in a Clos network, the number of outputs from 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

P : (the given) permutation

P^* : (the translated) permutation

$\binom{i}{j}$: tuple corresponding to inlet i and outlet j in the given permutation P

$\binom{i^*}{j^*}$: tuple corresponding to inlet i and outlet j in the translated permutation P^*

f_0: index number of faulty switch in stage 0 (integer value)

f_1: index number of faulty switch in stage 1 (integer value)

f_2: index number of faulty switch in stage 2 (integer value)

2.2 Fault Model

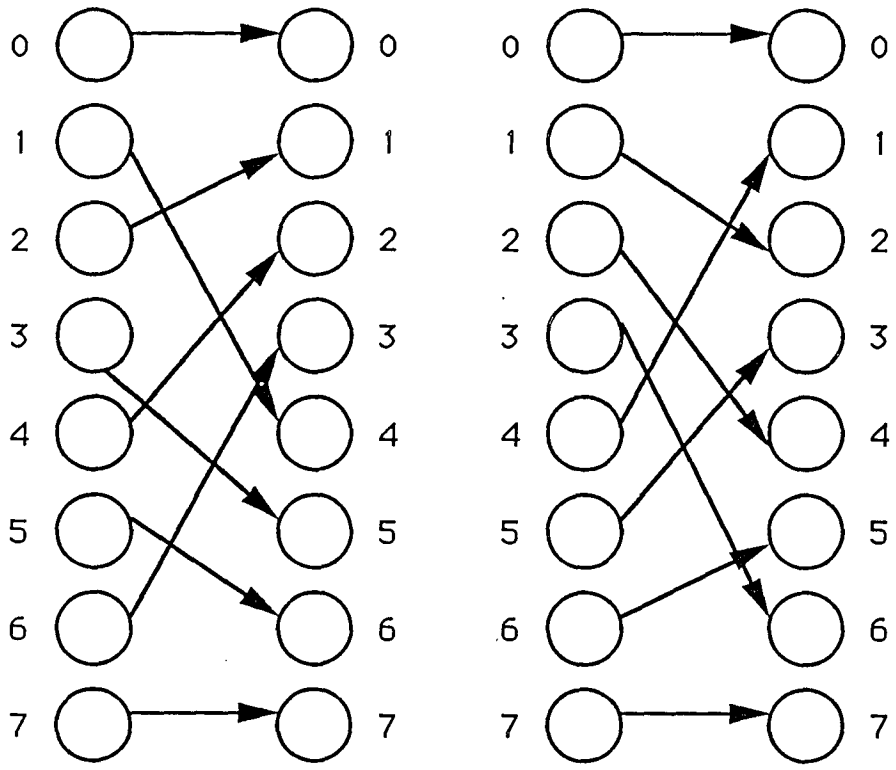
Fundamental to the design of a fault tolerant MIN is the definition of a fault tolerance model [2,22]. It contains three elements: the fault model, the fault tolerance criterion and the fault tolerance size.

The fault model identifies all the possible faults that can occur in the network. Thus, the fault model states the types of faults that the proposed design can recover from. In this work, the fault model is defined as follows.

1. Any basic network component can fail. This means that any of the switches and links can fail. A link fails if it is disconnected from either of the switches to which it should be connected. A switch can fail in several ways. For instance, it could be stuck in legal state. This could happen to be the desired state, but the switch is unresponsive to its control unit. A switch could be stuck in a partially legal state, in which case a subset of its inputs and outputs could be connected together. It could also happen that two inputs are connected together and two outputs are connected together. All these cases imply that the switch is not responsive to its control unit and are treated as switch failures.
2. The extra hardware can fail but its failure rate is incorporated into those of the switches to which it is connected.

The fault tolerance criterion is the condition that must be met for the network to be called fault tolerant. One criterion is full access retention. This means that after a fault occurs, any inlet must still be able to access any outlet. This does not guarantee that a given inlet-outlet permutation is realizable in one cycle of operations. In this work, a stronger criterion - full recovery is used. This means that even in the presence of a fault, any inlet-outlet permutation is realizable in one cycle.

The maximum number of faults a system can suffer and still meet the fault tolerance criterion defines its fault tolerance size. The proposed design can tolerate three faults, one in each stage. Thus it is 3-fault tolerant.



a) 4-shuffle of 8 objects b) 2-shuffle of 8 objects

Figure 3.1: Shuffling 8 objects

CHAPTER 3
MIN IMPLEMENTATIONS

Since the time that research in interconnection networks started, there have been a large number of MINs proposed. It is not possible to describe each MIN in detail, hence only a few of the well known ones will be described. The structure and routing of the Baseline, Benes and Clos networks are presented in this chapter.

3.1 The Baseline Network

3.1.1 Design of the Baseline Network

The shuffle family of MINs [36,41], of which the Baseline network is a member, is characterized by the use of the same switch structure and layout. It is built up of 2 x 2 switches. For a network of size N there are $g = \log_2 N$ stages, with $N/2$ switches in each stage. The important trait of this class of MINs is that the switches of any stage $i+1$ can be interchanged so that the links between stages i and $i+1$, where $0 <= i <= g-1$, form a 2-shuffle of the terminals of one stage into those of the other. This is the way in which networks with seemingly different topologies in this family can be obtained from one another [40]. The representative networks of this family include the Baseline [39], Omega [23], Shuffle Exchange [37], STARAN^{TM} Flip [6] and the SW Banyan [16] networks.

Consider N objects. Let $N = pq$. To obtain the p-shuffle of the N objects, proceed as follows [17]. Suppose that the pq objects are cards in a deck. Divide the cards into p piles, each with q cards. Arrange the piles in a row, in any arbitrary order. Starting with the first pile, remove the top card of each pile and create a new pile by placing the cards one on top of the other in the same order in which they were removed. Repeat this for all q cards in each pile. Now we are left with only one pile containing $pq = N$ cards. The ordered cards of this new pile is the

facing 8

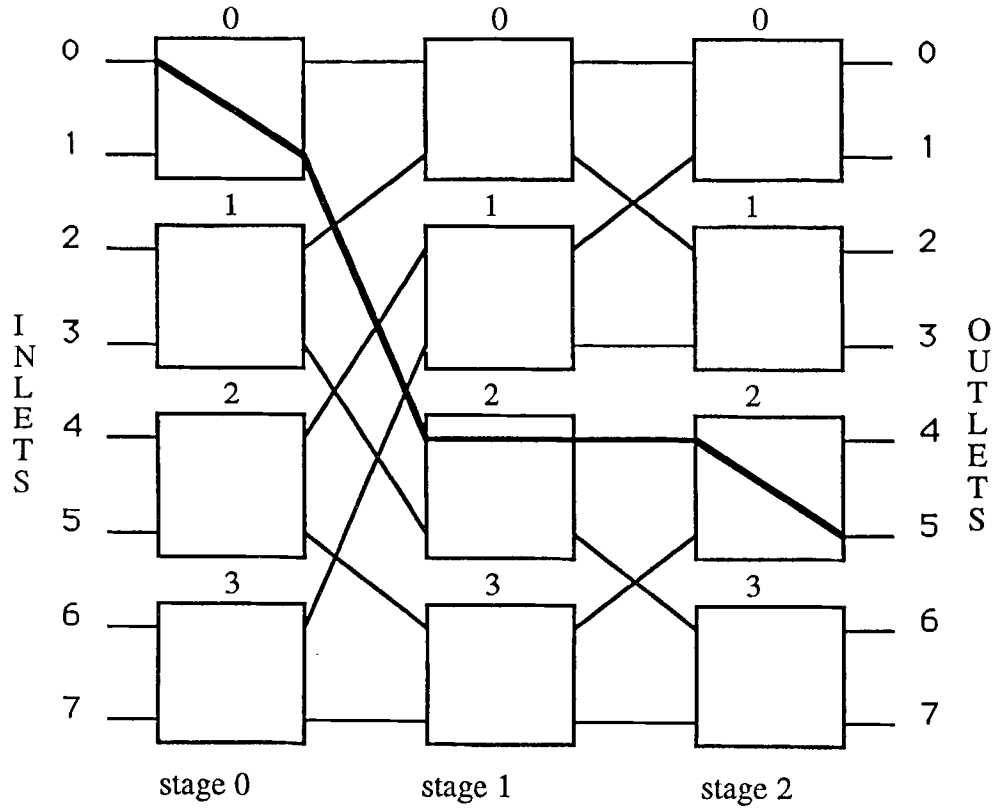


Figure 3.2: 8 x 8 Baseline Network

facing 8

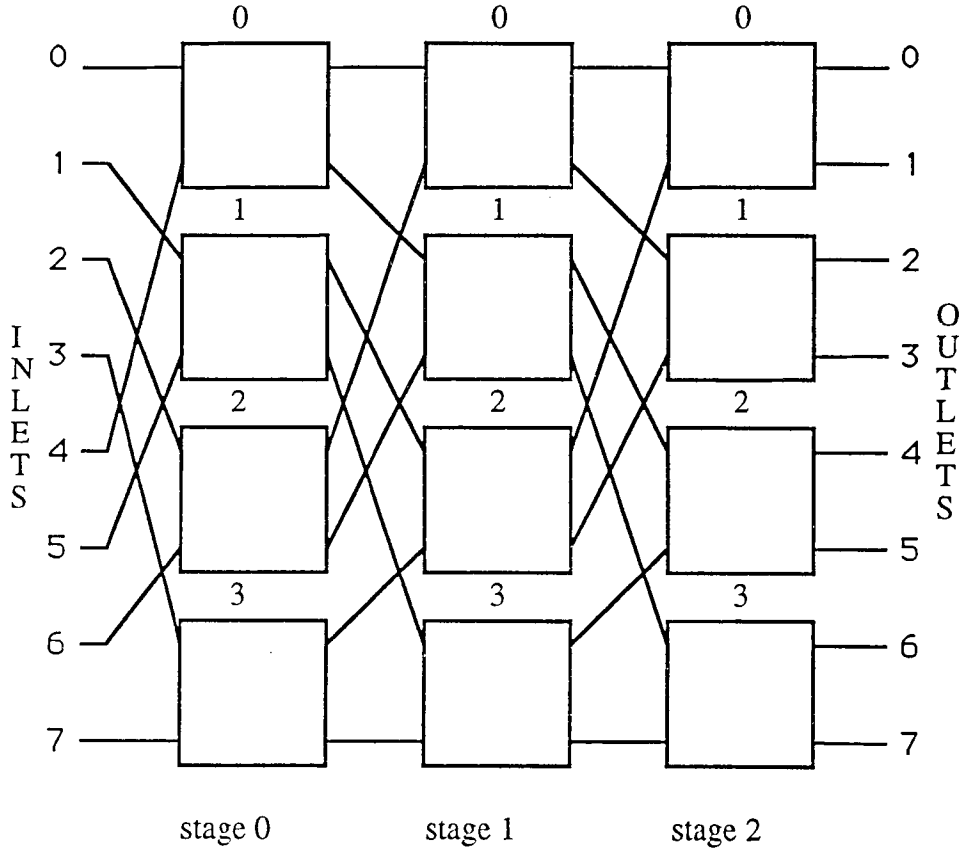


Figure 3.3: 8 x 8 Omega Network

p-shuffle of the original deck. If $p = 2$, the shuffle is called perfect. It should be noted that a p-shuffle followed by a q-shuffle gives back the original configuration. Figure 3.1 shows a 4-shuffle and a 2-shuffle of 8 objects.

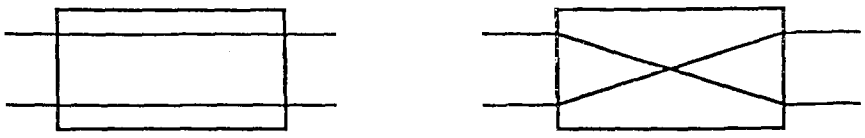
An 8 x 8 Baseline network is shown in Figure 3.2. It consists of $\log_2 8 = 3$ stages, each stage having 8/2 = 4 switches of size 2 x 2. The stages are labeled 0,1,2 from the left and in each stage the switches are labeled from the the top as 0,1,2,3. This is the labeling scheme that is adopted in all the networks in this thesis. The terms 'inlet' and 'outlet' refer to the network terminals and 'input' and 'output' refer to those of each individual switch. Also, for all MINs, inlets are on the left side of the network and outlets are on the right. It should be noted, however, that these terms are used only for ease of understanding since data flows in either direction.

Looking at the Baseline network, 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. If the switches 1 and 2 of stage 2 are interchanged, we get a network with 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 outputs of stage 2 and if the inlets are used as outlets and vice versa, then the resulting network is called the Omega network [23]. Figure 3.3 shows an 8-input Omega network.

3.1.2 Routing the Baseline Network

The building blocks of a Baseline network are the 2 x 2 crossbar switches. These are also called binary cells or binary switches. A binary cell can assume either of two legal states. Figure 3.4 shows a binary switch. 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 connections are reversed: the upper input is connected to the lower output and vice versa. The routing bit decides to which output an input must be connected to. Normally, if the routing bit is a 0, the input is connected to

the upper output and if the bit is 1, it is connected to the lower output. For this reason, the upper output is sometimes called the 0 output and the lower output is called the 1 output. If the inputs have identical routing bits, then there is contention



a) straight state b) cross state

Figure 3.4: Legal states of a binary cell

and one of the inputs must be blocked. This limitation makes the Baseline network a blocking network, in the sense that not all sets of path can be established between the inlets and outlets. The path between any given inlet and outlet is unique.

The Baseline network is self-routing. A routing tag is sent to the switches and they assume appropriate states to connect to switches in the next stage. To establish a path between inlet S and outlet D, we need to send the g-bit binary representation of D on inlet S. The integer D can be represented by the bit pattern

$$D = d_{g-1}\, d_{g-2} \ldots\ldots\ d_1\, d_0 \qquad (3.1)$$

The bit d_i will control the switch in stage $g - 1 - i$, $0 <= i <= g-1$. An example is shown in Figure 3.2. Suppose that inlet 0 should be routed to outlet 5. The 3-bit representation of 5, 101 is fed to the inlet 0, along with the data. This 3-bit string representing the destination is called the routing tag. The bits 1, 0 and 1 control switches in stages 0, 1 and 2 respectively. It has been shown that this type of bit representation for the routing tag can be realized and unscrambled in $(2\log_2 N - 1)$ passes or less [18].

The main disadvantage of the Baseline network is that not all permutations can be realized. However it has been observed that the most common permutations needed in parallel computing are realizable by this network family [26].

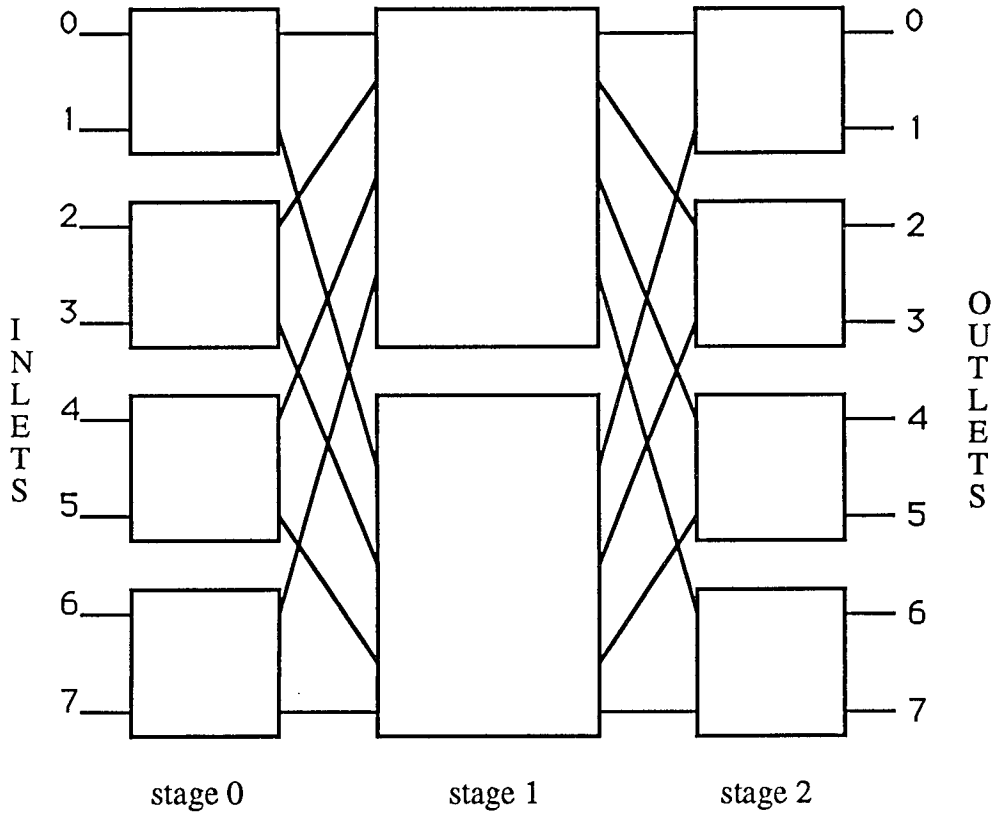


Figure 3.5: 8 x 8 Clos Network

3.2 Clos Network

3.2.1 Design of the Clos Network

Figure 3.5 shows an 8 x 8 Clos network. This is a family of non-blocking networks built up of three stages numbered 0, 1 and 2 [13]. A non-blocking network is one that can realize any inlet-outlet permutation. Stages 0 and 2 are also referred to as the outer stages and stage 1 is sometimes called the middle stage. A network is of size N if it has N inlets and N outlets. The 0th stage is made up of $k = N/m$ switches where each switch is of size m x n (i.e., with m inputs and n outputs). The middle stage has n switches each of size k x k. The final stage has k switches of size n x m. The three stages are connected by interstage links in such a manner that any switch in a given stage has access to all the switches in the next stage. In general, for a Clos network, $n >= m$. If $n = m$ the resulting network is called the Clos network or the ordinary Clos network. However, in the modified version presented in this thesis, $n > m$ and this gives the model some fault tolerance.

3.2.2 Routing the Clos Network

There is a central routing unit which receives a mapping, in the form of a permutation, and then determines the switch settings to realize it. This is not a trivial procedure and is often a time consuming process. The only possible way to avoid conflicts is to set the middle switches first and then the outer stage switches. There are three approaches to solving the problem: the group theoretic approach [33], the direct matrix decomposition approach [11] and the graph theoretic approach [27]. Only the matrix method is used in this work. An example will illustrate the procedure.

Let the permutation to be realized be given by

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

The first member of each tuple refers to the inlet and the second member to the outlet that it should be connected to. For example, the tuple $\binom{0}{5}$ implies that inlet 0 should be routed to outlet 5. In this example there are 9 inlets and 9 outlets, therefore $N = 9$. Let $m = n = 3$. The direct matrix decomposition approach starts by constructing an N X N matrix, I, from the permutation as follows:

$I[i,j] = 1$ if inlet i is to routed to outlet j

$= 0$ otherwise

where $I[i,j]$ is the element in row i and column j of the matrix I, $0 <= i,j < N$.

Thus in the example

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

Next, we partition I into k x k sectors, each of size m x m, and construct a k x k matrix H_m from I as follows:

$H_m[i,j]$ = sum of the m x m elements in sector i,j of I.

In the example,

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

The next objective is to decompose H_m into m matrices of size k x k such that there is only one 1 in each row or column. All other entries are 0s. Each

decomposed matrix will give the proper settings for one of the switches of stage 1. Once the switches in the middle stage are set, the switches in the outer stages can be set to realize the given permutation. Many algorithms have been proposed to decompose H_m in the general case, but in this work the one proposed by Neiman [30] will be used. Its basic idea is as follows.

1. Starting with the leftmost column of H_m, a non-zero element is marked in each column in such a way that there is no more than one marked element in each row. The k x k matrix formed by replacing the marked elements with 1s and filling in 0s elsewhere represents the settings for one of the switches in the middle stage. If it is not possible to mark an element in a particular column, that column is skipped and the procedure is continued on the next column. In this case another pass over the matrix is needed. This pass repeats a process of marking and unmarking elements of the matrix until m elements are marked. This process of unmarking and marking is continued until all columns are marked. In the example, one possible way of marking H_3 is as follows.

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

This yields one of the decomposed matrices which is given by

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

2. In the next step, each marked element in H_m is decremented by 1 to obtain the matrix $H_{m\text{-}1}$ and the algorithm is applied to $H_{m\text{-}1}$ to obtain the settings of a second switch in the middle stage. $H_{m\text{-}2}$ is then formed and the algorithm is applied recursively until H_1 is obtained. This matrix itself represents the settings of one of the middle stage switches.

facing 13

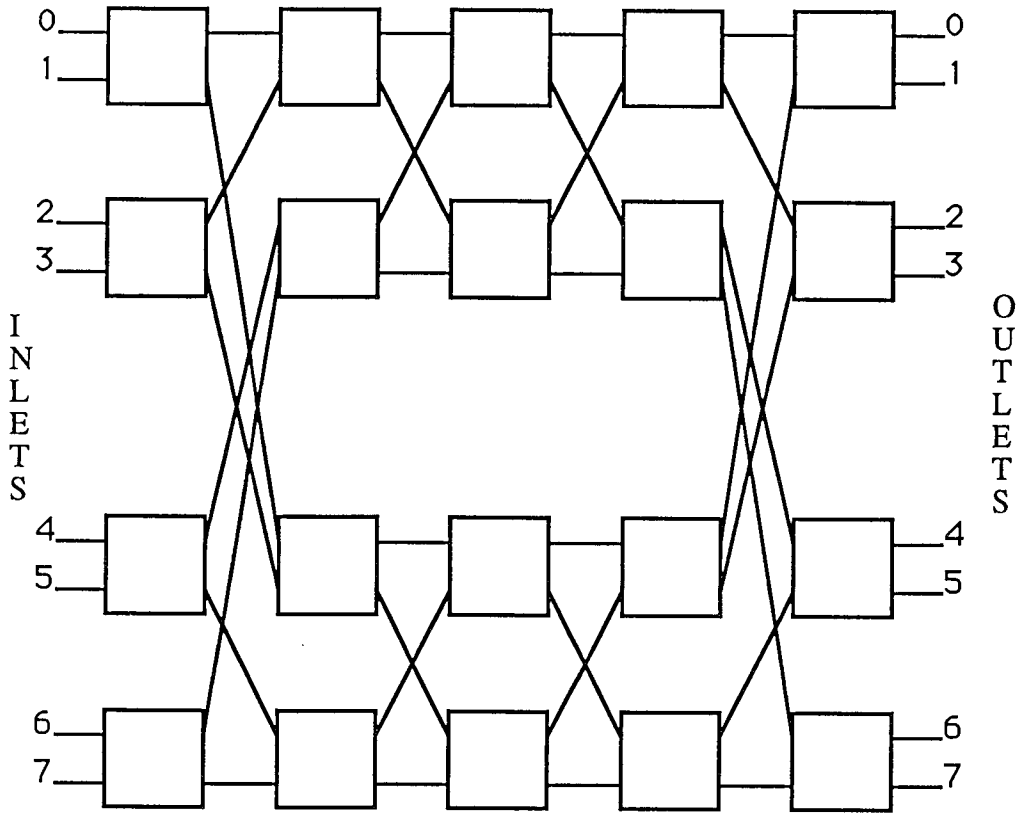


Figure 3.6: 8 x 8 Benes Network

In the example H_2 is given by

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

The complete decomposition of H_3 is obtained by decomposing H_2. Thus

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

The matrices on the right hand side give the settings of the three switches in the middle stage. If an element in row i and column j is 1, it indicates that the ith input is to be connected to the jth output in the switch. If the element is 0 it means the corresponding input-output pair are not to be connected together. After the middle stage is set, the two outer stage switches are set accordingly. For a given permutation, once the middle stage is set, the settings of the other two stages are straightforward to derive.

3.3 The Benes Network

3.3.1 Design of the Benes Network

The Benes network can be derived from a Clos network with $n = m = 2$ and $k = 2^p$, for some positive integer $p > 1$, by recursively decomposing each switch in the middle stage into a 3-stage Clos subnetwork whose outer stage switches are of size 2 x 2. This decomposition is continued until every switch in the network is of size 2 x 2 [7,8]. Figure 3.6 shows an 8 x 8 Benes network.

We derive the Benes network from the Clos network shown in Figure 3.5 by replacing both the middle stage switches with a 3-stage Clos subnetwork. There are 5 stages in the network, each containing 4 switches. In general, a Benes network will have $g = (2\log_2 N - 1)$ stages where N is the network size. Each

stage has $N/2$ switches, each 2 x 2. The Benes network can be viewed as two back to back Baseline subnetworks, with one of the two middle stages eliminated. This leads one to think that similar routing techniques can be used. This is indeed the case.

3.3.2 Routing the Benes Network

There are broadly two approaches taken to route Benes networks: distributed routing and central routing. As was mentioned earlier, the Benes network bears a close resemblance to the Baseline network. Hence the self-routing tag technique could be used in this case too, albeit in a modified way. This approach does not allow every permutation to be realized. The time complexity of the distributed routing algorithm is $O(\log_2 N)$.

The central routing algorithm takes advantage of the fact that at any stage j, $0 <= j <= (g\text{-}1)/2$, the stages between j and $2(\log_2 N) - j - 2$ form two Clos subnetworks of size $N/2^j$ each. The most famous algorithm to adopt this approach is known as the looping algorithm [4,31]. The name derives from the nature of the algorithm. The algorithm is initiated by arbitrarily setting one of the outer stage switches and recursively working towards the middle stage switches. The looping algorithm has a time complexity of $O(N\log_2 N)$. This is higher than that of the distributed routing algorithm; however, the advantage is that the network behaves like a non-blocking one under this algorithm. A new approach, double coset decomposition, is used to identify new properties of Benes networks and speed up routing for some permutations [10].

3.4 Other MIN Implementations

Apart from the three MIN implementations discussed above, there are many other types of MINs that can be found in the literature. The Gamma network is one such network. This is a MIN with redundant paths between some inlet-outlet pairs. It has

a cube network as a substructure. This network is controlled by a similar routing tag algorithm as the Baseline network.

A fairly new MIN implementation that has been proposed is the B-Network [25]. This is derived from the Gamma network and has backward links to provide backward paths for the requests blocked by contentions. The cube structure of the Gamma network is preserved but the direction of all other links are reversed. The routing technique and hardware complexity are identical to those of the Gamma network, while its performance is enhanced.

CHAPTER 4

FAULT TOLERANT MINS

In this chapter, an overview of some of the main fault tolerance designs that have been proposed will be discussed. The general concepts of fault tolerance in MINs will be presented. The inevitable tradeoff between hardware complexity and degree of fault tolerance can be understood from this chapter.

In general, any system can be made fault tolerant by adding extra hardware. The extreme solution is to duplicate the system - use one system and when that fails, switch to the other one. This approach is used when absolutely no performance degradation is acceptable. However, the obvious disadvantage of this solution is the high cost. It should be noted that only one system will be in use at a given time. Thus, it is clear that duplication is not a viable alternative.
In practice, MINs are made fault tolerant by adding extra hardware, such that the extra cost and increased size are acceptable. Adding more hardware normally translates to less severe performance degradation under faulty conditions. The objective in all fault tolerance schemes in MINs is to achieve graceful degradation. This means that when a network component fails, the network should not break down completely; it should still be functional, even though its performance may be reduced. The second goal of fault tolerance techniques is that the performance under normal conditions should not be adversely affected.

Most of the fault tolerance techniques proposed in the literature have been restricted to the shuffle family. The Kappa network has been suggested as a way to provide fault tolerance to the Gamma network by adding extra links [21]. The extra stage Gamma network employs an extra stage to provide redundancy and is one-fault tolerant [24]. A scheme applicable to a wide class of MINs has been suggested [31]. This method connects switches in the same stage together, thus providing alternate paths at every stage. Nassar has proposed a non-MIN specific

facing 17

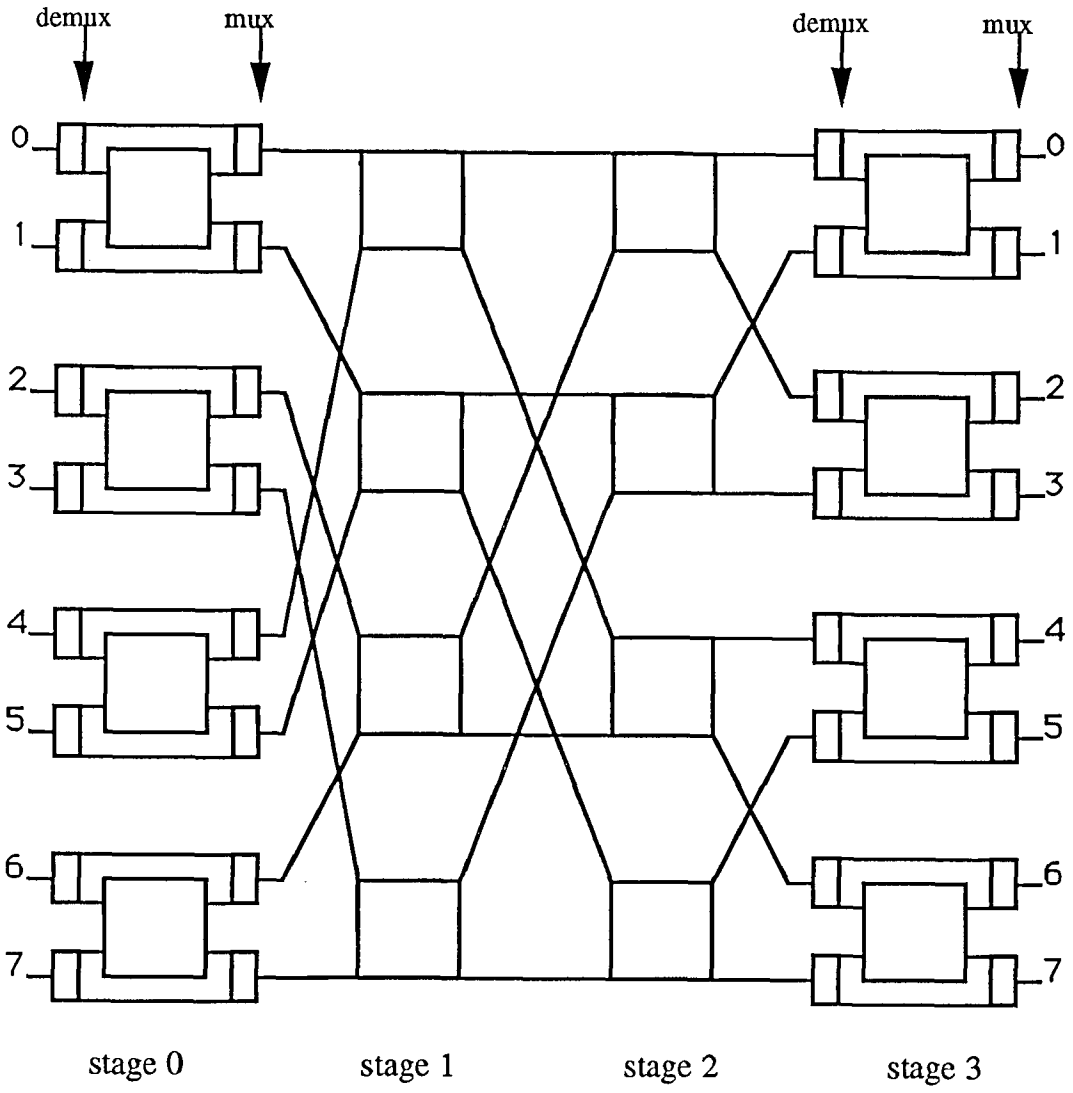


Figure 4.1: 8 x 8 Extra Stage Cube Network

technique that uses bypass buses under faulty conditions [29]. He has also proposed a fault tolerant Clos network that adds an extra switch in each stage [29]. Both these MINs are discussed later in this chapter. Also presented in this chapter is the construction and fault model of the Extra Stage Cube, a very well known fault tolerant MIN.

4.1 The Extra Stage Cube (ESC) Network

The Extra Stage Cube (ESC) is a fault tolerant implementation for the shuffle family [1]. Shown in Figure 4.1 is an ESC implementation on a member of this family, the Generalized Cube Network. Stage 0 is the extra stage. The inlets are connected to a set of demultiplexers of size 1 x 2. One of the outputs of each demultiplexer is connected to the corresponding switch in stage 0. The other output bypasses this switch. It is connected to a series of multiplexers on the output side of the switch. A similar arrangement is made for stage 3. Either of these stages can be switched on or off at will by properly setting the states of the multiplexers and demultiplexers. In the normal condition, stage 0 is bypassed. If a switch in stage 3 develops a fault, that stage is disabled and stage 0 is enabled. If a switch in a stage other than 0 or 3 is faulty, that particular stage is turned off and both stages 0 and 3 are used. It should be noted that in general, for a cube network, there are $(g + 1)$ stages, where $g = log_2 N$. In the example under discussion, $g = 3$ and there are 4 stages in the network. The extra stage provides a set of redundant paths between any inlet-outlet pair.

The Generalized Cube adopts a self routing technique. A routing tag is generated by the bitwise exclusive-OR of the two integers S and D representing the source and destination respectively. The tag has g bits. A switch in stage i examines the ith bit of the tag and assumes one of two states, 0 or 1. A '0' implies a straight connection, while a '1' puts the switch in the crossed state. If the two inputs of a switch have identical routing bits, contention occurs and one of the inputs has to be blocked. However, for the ESC, there is no way of knowing

facing 18

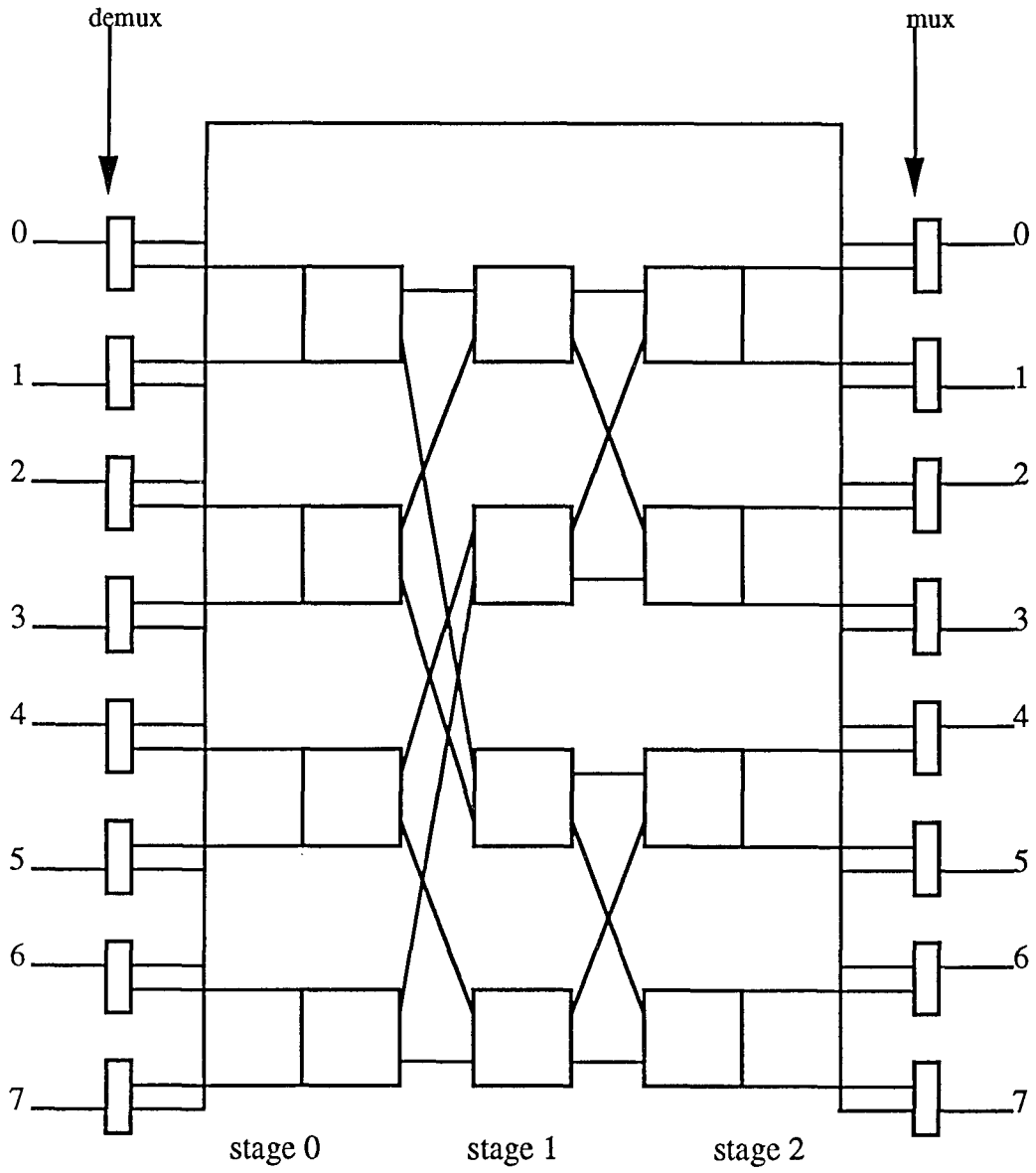


Figure 4.2: 8 x 8 SFTB Network

before hand which stage is going to be disabled. In the normal working condition stage 0 is disabled, but the occurrence of a fault in any stage forces stage 0 to be enabled. This dictates that dynamic routing techniques be employed. The ESC generates a dynamic $(g + 1)$ bit routing tag T', other than the normal routing tag T, for each inlet-outlet path. Obviously, all processors must be informed about the location of the fault. In this thesis, it is assumed that there is some circuitry to detect the location of a fault and to notify each processor.

The fault model for the ESC is given below.

1. Any network component can fail.
2. Any component of the extra hardware can fail. But their failure rates are incorporated into those of the corresponding network switches.
3. Faulty components are unusable.
4. Faults occur independently.

The fault criterion is full access retention. Any inlet can be routed to any outlet in the presence of a fault. The fault size of the ESC is 1.

The advantages of the ESC can be summarized as follows. It is made up of simple binary cells and is easy to operate. The multiplexers and demultiplexers have to be arranged only once after the occurrence of a fault.

The disadvantages are the following. The extra hardware complexity is relatively high: $N/2$ extra switches and $4N$ multiplexers and demultiplexers are needed. The ESC has a fault tolerance criterion of full access retention; any arbitrary permutation cannot be realized in the presence of a fault. The generation of a new routing tag after a fault occurs requires extra time. This decreases the performance.

4.2 The Simple Fault Tolerant Baseline (SFTB) Network

This is a non-MIN specific fault tolerance technique proposed by Nassar [29]. In the following discussion, this technique is applied to the Baseline network, but in fact, it can be applied to any MIN. The main idea behind this approach is to combine the two types of interconnection mechanisms that can exist in a

multiprocessor system: a single common bus or a MIN. The MIN is the primary interconnection mechanism and the bus is used by data that would normally pass through the switch that develops a fault. The result is to produce a network which has all the advantages of a MIN. Under normal working conditions, this network behaves exactly like a MIN, with the bus being invisible.

The SFTB is shown in Figure 4.2. An external bus bypasses the network and connects the inlet side and the outlet side. Each inlet is connected to both the network and the bus through a 1 x 2 demultiplexer. On the other side, each outlet is also connected to both the network and the bus through a 2 x 1 multiplexer. Under no-fault conditions, the states of the multiplexers and demultiplexers are such that the inlets and outlets are connected through the network. Generally, the '0' state corresponds to this configuration. When this state is '1', the inlets and outlets are connected through the external bus. When no faulty switch exists, the SFTB uses the routing algorithm of the ordinary Baseline network. Upon the occurrence of a fault, the SFTB has to be reconfigured. It is assumed that there is some mechanism to detect and locate faults. After a faulty switch is detected, its location is broadcast to all processors. At the start of each memory cycle, each processor must first find if the defective switch lies in its path. If it does not, then the processor starts the memory cycle as it would under normal conditions. If a faulty switch exists along its path, the processor will use the standby bus to access the memory. This is done by setting the state of the demultiplexer corresponding to the inlet to '1'. This connects the inlet to the external bus. On the outlet side, the multiplexer corresponding to the outlet that the particular inlet must be routed to is also set to state '1'. This completes the path between the inlet and its corresponding outlet through the bus.

It should be noted that the fault tolerance criterion for the SFTB is full access retention. This implies that when a fault occurs, all the inlets may not be routed to their corresponding outlets in one memory cycle. Referring to the Figure 4.2, it can be seen that if the switch corresponding to inlets 4 and 5 fails, at least

facing 20

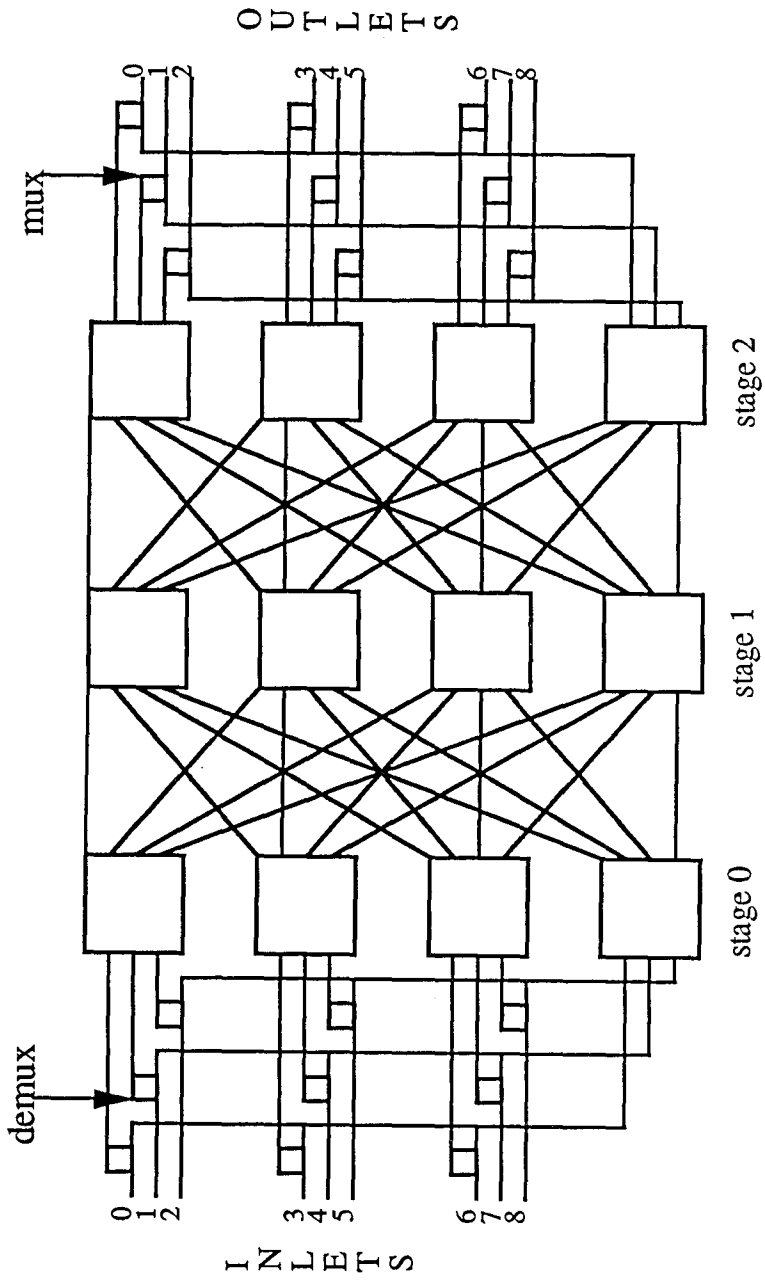


Figure 4.3: 9 x 9 Fault Tolerant Clos Network

two cycles are needed to complete all connections.

The advantage of the SFTB is its simplicity and ease of operation. This technique is applicable to any MIN. Under normal conditions, the routing algorithm of the Baseline network can be used. There are some disadvantages. The fault tolerance criterion of the SFTB is full access retention. This implies that under faulty conditions more than one memory cycle is needed. This problem can be eliminated by the use of the Enhanced SFTB [29], in which there are two external buses instead of one. Bus loading is also a concern for synchronous systems.

4.3 The Fault Tolerant Clos (FTC) Network

An ordinary Clos network of size N x N has $k = N/m$ switches in stages 0 and 2 where m is the number of inputs of a first stage switch. Each switch in stage 0 is of size m x n and each switch in stage 2 is n x m. In the middle stage there are n switches of size k x k . The Fault Tolerant Clos (FTC) network [29] is derived from the Clos network as follows. In the outer stages, switches with $n = m + 1$ are used. This implies that an extra switch is needed for the middle stage. In addition, an extra switch is added in each outer stage. Each switch is of the same size as the switches of the stage to which it is added. These extra switches provide fault tolerance to the corresponding stages by way of their redundancy. When a switch develops a fault, the extra switch in that stage is used to perform the routing. The network inlets are connected to the inputs of the first stage via 1 x 2 demultiplexers. On the outlet side, the outputs of the last stage are connected to the outlets through 2 x 1 multiplexers. Figure 4.3 shows an FTC derived from a 9 x 9 Clos network. Each switch in the original network is of size 3 x 3.

The FTC can be reconfigured in such a manner that its pre-fault connectivity is regained. The basic idea behind the implementation is that when a switch fails in any or all of the stages, the affected switch is disabled and the extra switch in that stage is used. As mentioned earlier, Clos networks can be routed using matrix decomposition methods. The FTC also uses the same approach. In the event of a

switch failure, the matrix generated from the original permutation is no longer valid. The original matrix assumes $m = n$, which is not the case in the FTC. A new permutation has to be generated from the original one, taking into account the faulty switches in the network. The translated permutation is used in the generation of a new matrix. This matrix is decomposed using standard matrix decomposition techniques and the switch settings of the middle stage are extracted.

The FTC offers increased network reliability at relatively low cost. Three extra switches and $2k$ multiplexers and demultiplexers are required. The fault tolerance criterion for the FTC is full recovery. This feature is particularly important in Clos networks, since these networks are primarily permutation networks.

4.4 Fault Detection and Location

Fault detection and location are two important issues related to fault tolerant MINs. There must be a method to know when and where a fault has occurred and a scheme to pass this information to all the processors. There are two ways in which faults can be detected and isolated: the online method and the offline method. The offline method basically consists of applying a test pattern at the input and comparing the output with the expected values [3,15]. This could be done at the start of each network cycle and can slow down the performance appreciably. The online techniques, on the other hand, are much faster but require sophisticated hardware. The techniques employed in this case could be either parity checking [34] or data bits checking [27]. In this thesis it is assumed that there is some mechanism to detect and locate faults and to inform the processors of the fault location.

facing 22

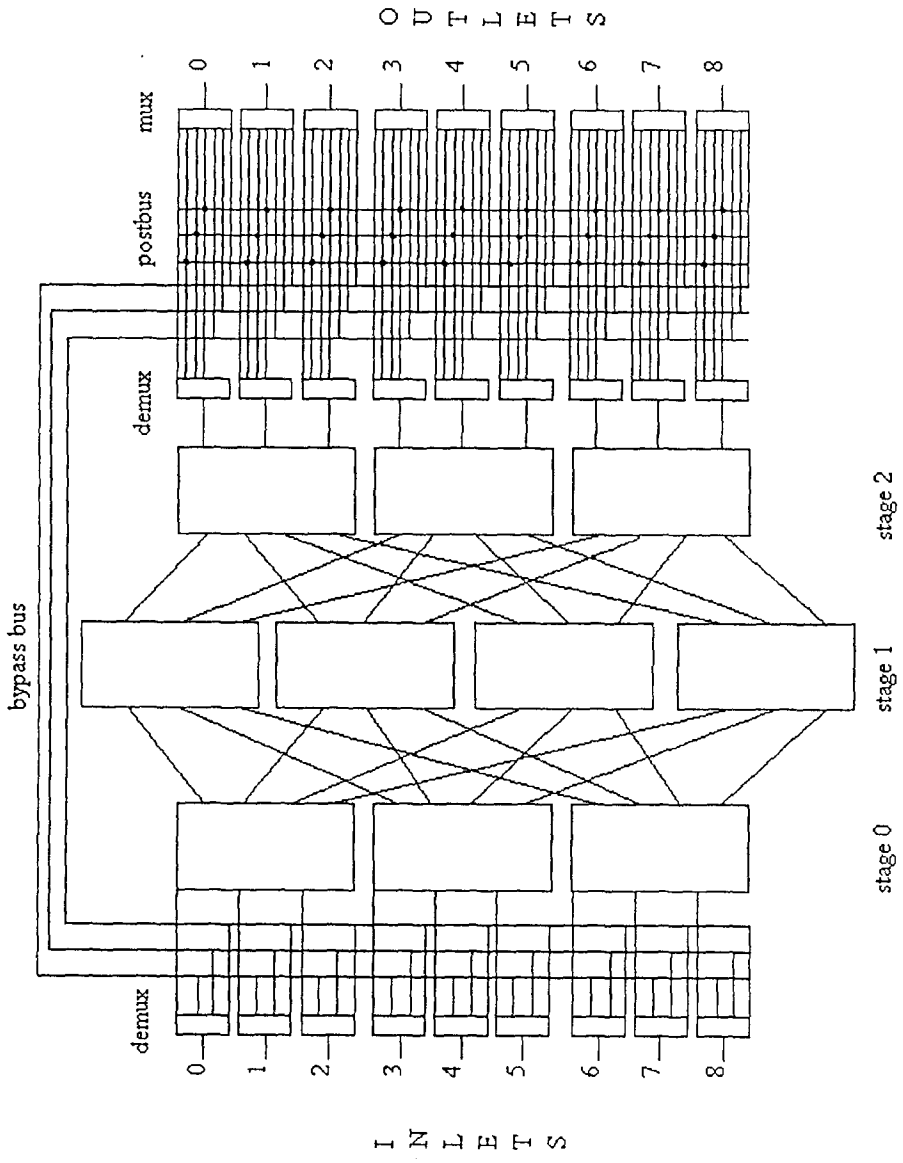


Figure 5.1: 9 x 9 Postbus Fault Tolerant Clos (PFTC) Network

CHAPTER 5

THE POSTBUS FAULT TOLERANT CLOS (PFTC) NETWORK

A Clos network inherently has some fault tolerance in the middle stage. This is due to the fact that a switch in this stage is connected to all the switches in the outer stages. However, this only offers full access retention. Also, a fault in either of the outer stages cannot be tolerated. The Postbus Fault Tolerant Clos (PFTC) network design proposed in this chapter satisfies the criterion of full recovery and can also withstand faults in the outer stages.

5.1 Design of the PFTC

A 9 x 9 PFTC is shown in Figure 5.1. The idea behind the design is to make each stage of the network one switch fault tolerant. Fault tolerance is built into each stage independently by treating each stage as a system by itself. This is a valid assumption since, in all reliability and failure analyses, system components are considered to fail independently.

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

1. Any switch or link can fail.
2. Extra hardware in the form of multiplexers and demultiplexers and external links can fail. Their failure rates are incorporated into those of the switches with which they are associated.
3. Faulty components are unusable.

Consider the network Figure 5.1. N is equal to 9 and m is 3. A set of m buses goes across from the inlet side to the outlet side, bypassing the three-stage network. N demultiplexers interface the network inlets and the first stage of the network. These demultiplexers are of size 1 x (m +1). In the normal condition, when all the network switches are functional, these demultiplexers are in state 0 and

the inlets are connected to the first stage switches. In the case of a faulty switch, these demultiplexers take on any of the (m +1) states. The bypass buses and the demultiplexers provide fault tolerance to the first stage of the Clos network. It should be noted that the terms 'multiplexers' and 'demultiplexers' are conceptual; data flows in either direction and the same piece of hardware behaves both as a multiplexer and a demultiplexer under different contexts.

To make the second stage of the network fault tolerant, the property of the Clos network that $n >= m$ is put to use and n is taken to be (m +1). There is now one extra output from each switch of stage 0 and all these outputs are fed as inputs to an extra switch in the second stage. The fact that a switch in this stage is connected to all the switches of stages 0 and 2 is exploited to make the middle stage fault tolerant. If a switch goes bad, the extra switch is activated. Thus at any given time, there are only k switches operational in stage 1.

The final stage is made fault tolerant in the following way. A set of N 1 x (m +1) demultiplexers are connected to the outputs of the third stage. The other end of the demultiplexers are connected to the set of m postbuses and a set of N multiplexers of size ($2m$ +1) x 1. The remaining inputs of the multiplexers form the final network outlets. In the normal working condition the states of all the multiplexers and demultiplexers are 0. The extra hardware is transparent to the network and it behaves exactly like an ordinary Clos network.

5.2 Fault Recovery on the PFTC

As noted earlier, the proposed design has a fault tolerance criterion of full recovery. In other words, even after the occurrence of a fault in each of the stages of the network, any given inlet-outlet permutation can be realized. The occurrence of a fault in a middle stage switch, either by itself or in conjunction with those in the outer stages, is easily handled. All that has to be done to get around the problem is to press the extra switch in the stage into service. Hence this case is not discussed further. Essentially there are three non-trivial fault conditions that can exist.

1. First stage switch faulty, i.e f_0 exists.

2. Third stage switch faulty, i.e f_2 exists.

3. A switch in each of the outer stages is faulty, i.e f_0 and f_2 both exist.

The fault recovery mechanism is given by defining the states of the switches and the multiplexers and demultiplexers. If the first case occurs, i.e f_0 exists, then the inputs to f_0 are diverted to the set of bypass buses. On the outlet side, the corresponding multiplexers select the appropriate lines.. In the second case, the inlets that correspond to the outputs of f_2 are routed through the bypass bus and on the outlet side the multiplexers that are associated with f_2 assume appropriate states to select the inlets. If the third case occurs, i.e a switch in each of the outer stages fails, then the given permutation P has to be translated to obtain P^*. In this case both P and P^* have to be used to solve the routing problem. Given below is an algorithm to translate P to obtain P^*.

Let the given permutation P be represented by the tuples (i,j), $0<=i,j<N$. This means that inlet i should be routed to outlet j. The translated permutation P^* is given by the tuples (i^*,j^*), $0 <= i^*,j^* < N$.

int $i,j,N,i^*,j^*,m,x,y,f_0,f_2$;
For each tuple (i^*,j^*) in P
{
IF (!($m * f_0 <= i < m * f_0 + m$) OR ($m * f_0 <= i < m * f_0 + m$ AND $m * f_2 <= j < m * f_2 + m$))
RETAIN THE TUPLE (i,j) AS IT IS;
ELSE IF ($m * f_0 <= i < m * f_0 + m$ AND !($m * f_2 <= j < m * f_2 + m$))
{ For each permutation (x, y) in P
IF ($m * f_2 < y < m * f_2 + m$ AND !($m * f_0 < x < m * f_0 + m$))
INTERCHANGE j and y
}}

It happens that Neiman's algorithm defines the states of the switches even under faulty conditions with minor modifications. The translated permutation is used in the algorithm. If a middle stage switch fails, the algorithm is applied to the

remaining working switches plus the extra one. If an outer stage switch fails it does not matter what its internal state is, since that switch will be bypassed. Thus all that needs to be done to define the fault recovery mechanism is to show how the states of the demultiplexers and multiplexers change under faulty conditions. Algorithms to set the states of the multiplexers and demultiplexers are given below.

Case 1: First Stage Switch f_0 Faulty

```
int i, j, N, m, f0, STATE_DEMUX[N], STATE_MUX[N];
int STATE, STATE_A, STATE_B;

/* Define the states of the first column of N demultiplexers */

for ( i = 0; i < N; i++) /* scan each (i,j) tuple of the given permutation P */
{
 if (i >= m * f0 && i < m * f0 + m) /* if inlet feeds a faulty switch */
 {
  STATE_DEMUX[i] = STATE; /* use the bypass bus */
  STATE++;
 }
 else
  STATE_DEMUX[i] = 0; /* use the network */
}

/* Define the states of the second column of N demultiplexers */

for (i = 0; i < N; i++)
{
 if (i >= m * f0 && i < m * f0 + m)
  STATE_DEMUX[j] = TRISTATE; /* tristate the demux on the outlet side */
 else
  STATE_DEMUX[j] = 0;
}

/* Define the states of the column of N multiplexers */

STATE = m;
for (i = 0; i < N; i++)
{
 if (i >= m * f0 && i < m * f0 + m)
 {
  STATE_MUX[j] = STATE; /* take input from the bypass bus */
  STATE--;
 }
 else
```

```
   STATE_MUX[j] = 0; /* use regular network outputs */
}
```

Case 2: Third Stage Switch f_2 Faulty

```
int i, j, N, m, f2, STATE_DEMUX[N], STATE_MUX[N];
int STATE, STATE_A, STATE_B;

/* Define the states of the first column of N demultiplexers */

STATE = 1;
for (i = 0; i < N; i++)
{
 if (j >= m * f2 && j < m * f2 + m) /* if the j val. of the (i,j) tuple corr. to the
                                       faulty switch */
 {
  STATE_DEMUX[i] = STATE; /* prepare to use bypass bus */
  STATE++;
 }
 else
  STATE_DEMUX[i] = 0;
}

/* Define the states of the second column of N demultiplexers */

for (i = 0; i < N; i++)
{
 if (j >= m * f2 && j < m * f2 + m) /* if given outlet corr. to faulty switch */
  STATE_DEMUX[j] = TRISTATE; /* tristate the output */
 else
  STATE_DEMUX[j] = 0;
}

/* Define the states of the column of N multiplexers */

STATE = m;
for (i = 0; i < N; i++)
{
 if (j >= m * f2 && j < m * f2 + m)
 {
  STATE_MUX[j] = STATE;
  STATE--;
 }
 else
  STATE_MUX[j] = 0;
}
```

Case 3: Faulty Switches f_0 and f_2 in the Outer Stages

```
int i, j, N, m, f0, f2, STATE_DEMUX[N], STATE_MUX[N];
int STATE, STATE_A, STATE_B;

/*Define the states of the first column of demultiplexers */

STATE = 1;
for (i = 0; i < N; i++)
{
 if (i >= m * f0 && i < m * f0 + m)
 {
  STATE_DEMUX[i] = STATE;
  STATE++;
 }
 else
  STATE_DEMUX[i] = 0;
}

/* Define the states of the second column of N demultiplexers */

STATE = 1;
for (i = 0, i* = 0; i < N; i++, i*++) /* scan the given and the translated
                                         permutations simultaneously */
{
 if (j >= m * f2 && j < m * f2 + m) /* if the outlet in the given permutation corr. to
                                       faulty switch f2 */
  STATE_DEMUX[j] = TRISTATE;
 else
 {
  if (j != j* && i >= m * f0 && i < m * f0 + m)
  {
   STATE_DEMUX[j] = STATE;
   STATE++;
  }
  else
   STATE_DEMUX[j] = 0;
 }
}

/* Define the states of the column of N multiplexers */

STATE_A = m;
STATE_B = m + 1;
for (i =0, i* = 0; i < N; i++, i*++)
{
 if (i >= m * f0 && i < m * f0 + m)
 {
```

```
  STATE_MUX[j] = STATE_A;
  STATE_A--;
 }
 else
 {
  if (j != j* && j >= m * f2 && j < m * f2 + m)
  {
   STATE_MUX[j] = STATE_B;
   STATE_B++;
  }
  else
   STATE_MUX[j] = 0;
 }
}
```

5.3 Reliability Analysis

5.3.1 Fundamentals of Reliability

The reliability of a system [34,38], $r(t)$, is defined as the probability that the system does not fail within time t. In this work, the network reliability, R, will be defined mathematically as follows:

$$R = 1 - F \tag{5.1}$$

where F is the probability that a system fails within time t. Note that the time factor has been omitted in Equation (5.1). This is because we are interested in comparing the reliabilities of two systems and since the time parameter must be the same for both, we can ignore it.

Real systems can be considered to be made up of functional blocks. For example, each stage in a MIN is made up of switches and the MIN itself is made up of stages. It is assumed that these blocks are statistically independent, i.e the failure of one does not affect those of the others. This was also the philosophy adopted in designing the PFTC, where each stage was considered a system by itself and was made fault tolerant independently of the other stages. Consider a system with n blocks. If the system fails when even one block fails, then the system is considered to be a serial one. The reliability of this system is given by:

$$R = \prod_{i=1}^{n} R_i \tag{5.2}$$

where R_i is the reliability of each block i, $1 <= i <= n$. If, one the other hand, a system with n blocks fails only when all the blocks fail, it is considered a parallel system and the reliability of the system is the probability that at least one block is operational. This can be derived easily by considering the probability of failure of each of the blocks. For a system with n blocks, the probability of all the blocks failing is given by:

$$F = \prod_{i=1}^{n} F_i \tag{5.3}$$

where F_i is the probability of block i failing, $1 <= i <= n$. F_i is given by:

$$F_i = 1 - R_i \tag{5.4}$$

Hence we have,

$$F = \prod_{i=1}^{n} (1 - R_i) \tag{5.5}$$

Using the above relation in Equation (5.1), we have for the reliability of a parallel system with n blocks

$$R = 1 - \prod_{i=1}^{n} (1 - R_i) \tag{5.6}$$

In most cases pertaining to fault tolerant systems, a system has n parallel blocks, but needs at least m of them to be operational. This is a case of binomial distribution. The reliability of the system can be computed by considering the probability that the system fails. This can happen when the system has between 0 and m - 1 blocks defective.

This probability is given by the relation

$$F = \sum_{i=0}^{m-1} {}^{n}C_i R^i (1-R)^{n-i} \tag{5.7}$$

where ${}^{n}C_i$ is the number of combinations of n items taken i at a time. The reliability of the system is the complementary event. Thus,

$$R = 1 - \sum_{i=0}^{m-1} {}^{n}C_i R^i (1-R)^{n-i} \tag{5.8}$$

5.3.2 Reliability Analysis of the PFTC

In this section, the reliability of the PFTC is computed and compared with that of an equivalent Clos network. It is shown that the PFTC offers greater reliability for all values of N. An ordinary Clos network has $2k + m$ switches. The network fails even if one switch fails. If the reliability of each switch is denoted by r, the reliability of the Clos network is given by

$$R_{Clos} = r^{2k+m} \tag{5.9}$$

To compute the reliability of the PFTC, the reliability of each stage is first computed and then all the three individual reliabilities are multiplied together to get the system reliability. This can be done because the PFTC can be considered to be a series system with three blocks, each block being one stage of the network. The system fails even if one stage fails. Thus,

$$R_{PFTC} = R_0 R_1 R_2 \tag{5.10}$$

where R_0, R_1 and R_2 represent the reliabilities of stages 0,1 and 2 respectively. These quantities are calculated by considering F_0, F_1 and F_2 which represent the probabilities of failure of each of the three stages. Stages 0 and 2 have k switches each and need at least k - 1 to remain operational. In other words, they can function with one switch being defective. Replacing n with k, m with k - 1 and R with r in Equation (5.7), we obtain,

facing 31

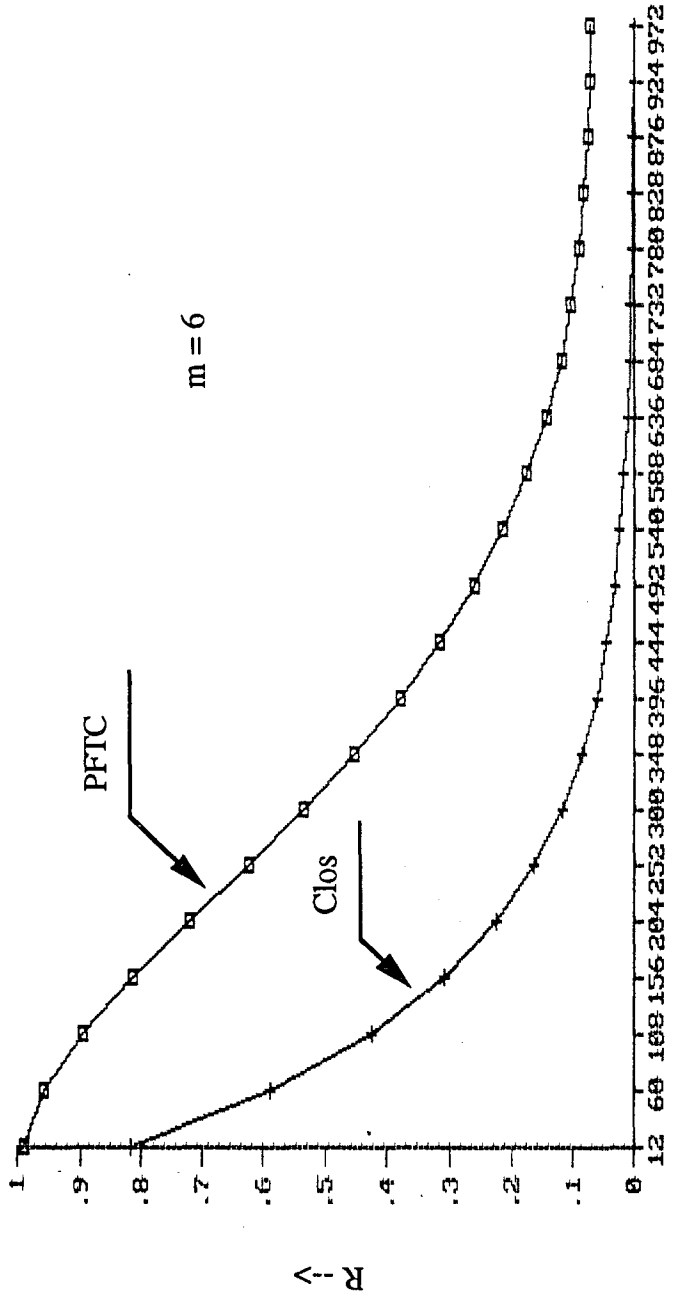


Figure 5.2: Reliability Curves with $r = 0.98$

$$F_0 = F_2 = \sum_{i=0}^{k-2} {}^{k}C_i \, r^i \, (1 - r)^{k - i} \tag{5.11}$$

where r is the reliability of each switch in stages 0, 1 and 2. The reliabilities of stages 0 and 2 are given by the complementary events, i.e

$$R_0 = 1 - F_0 = 1 - F_2 = R_2 \tag{5.12}$$

Thus,

$$R_0 = R_2 = 1 - \sum_{i=0}^{k-2} {}^{k}C_i \, r^i \, (1 - r)^{k - i} \tag{5.13}$$

The middle stage has $m + 1$ switches and needs at least m of them to remain operational. Replacing n by $m + 1$ and R by r in Equation (5.7), the failure probability of stage 1 is given by,

$$F_1 = \sum_{i=0}^{m-1} {}^{(m+1)}C_i \, r^i \, (1 - r)^{m+1-i} \tag{5.14}$$

Hence the reliability of the middle stage is given by

$$R_1 = 1 - \sum_{i=0}^{m-1} {}^{(m+1)}C_i \, r^i \, (1 - r)^{m+1-i} \tag{5.15}$$

The reliability of the entire network is given by,

$$R_{PFTC} = \left\{ 1 - \sum_{i=0}^{k-2} {}^{k}C_i \, r^i \, (1 - r)^{k - i} \right\}^2 \left\{ 1 - \sum_{i=0}^{m-1} {}^{(m+1)}C_i \, r^i \, (1 - r)^{m+1-i} \right\} \tag{5.16}$$

The reliability curves for $r = 0.98$ and $r = 0.99$ have been plotted in Figures 5.2 and 5.3 respectively for a 9 x 9 PFTC and an equivalent Clos network. The value of m has been taken to be equal to 6. It is seen that for the entire range of values for N, the PFTC has a better reliability. It is also observed that as the value of N increases,

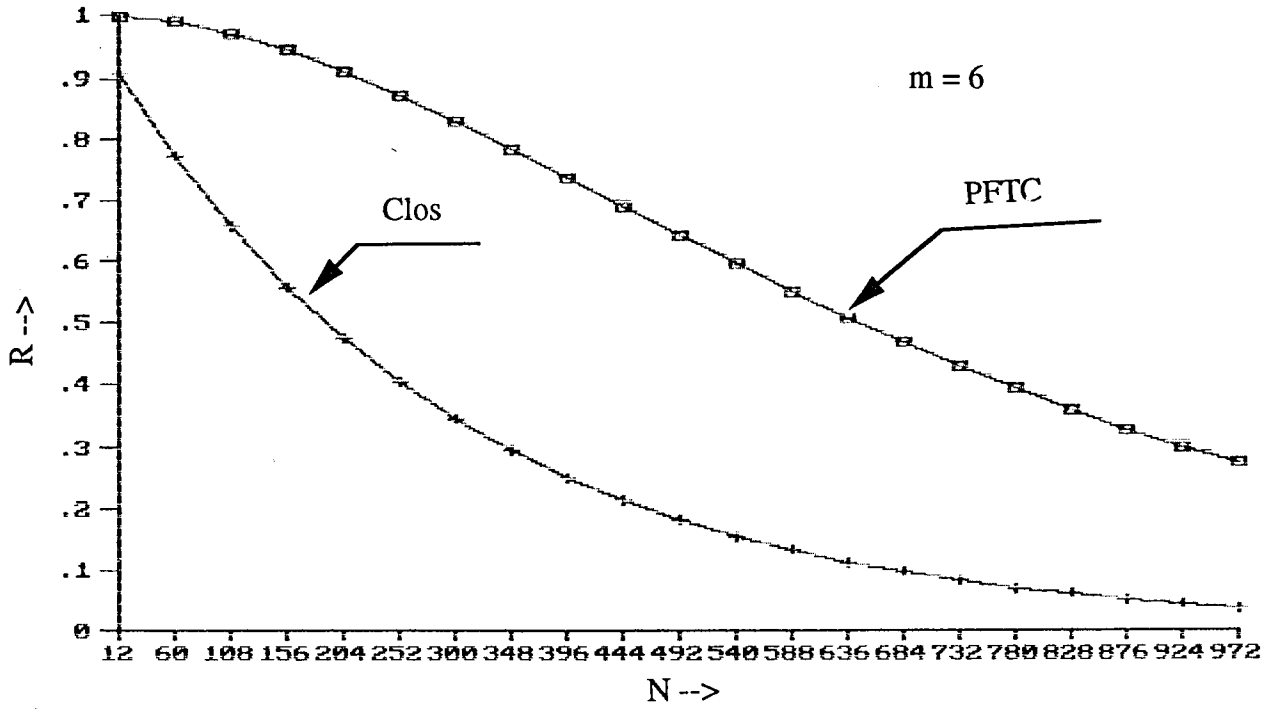


Figure 5.3: Reliability Curves with $r = 0.99$

the curve drops towards zero. This is to be expected, since the larger the network, the more components it has and the lower its reliability.

5.4 Discussion

This chapter described the design and performance of a fault tolerant Clos network. It is seen that the PFTC continues to function in the presence of faults.

Interconnection networks are made up of a large number of switches and greater the number of switches, lesser the system reliability. The PFTC offers increased network reliability, since it does not breakdown when a fault occurs. It satisfies the criterion of graceful degradation in the following way. When no fault exists, the extra hardware is transparent to the network and normal routing methods are used. When one of the outer stages develops a fault, only the states of the multiplexers and demultiplexers have to be set. This takes some extra time and performance decreases. When both the outer stages develop a fault, in addition to setting the states of the multiplexers and demultiplexers, the given permutation has to be translated. This incurs additional time and the performance decreases further, but the network is still functional.

CHAPTER 6

CONCLUSIONS

6.1 Summary

This paper has proposed a scheme to make the widely used Clos network fault tolerant. Fault tolerance in MINs assumes greater significance today as the trend towards massive parallelism continues. Performance under faulty conditions could be severely degraded and the system may be rendered useless for real time applications.

Any fault tolerance scheme should stick to two guiding principles. First, the performance under normal conditions should not be affected significantly and second, the cost of extra hardware should be minimal. The PFTC meets the above two criteria. With all the demultiplexers and multiplexers in state 0 under normal conditions, the extra hardware is not visible to the basic network and normal performance is not significantly diminished. Also, the cost of the extra hardware, which is in the form of demultiplexers and multiplexers, does not add much to the total cost.

Another important feature of the PFTC is that the same routing algorithms that are used for the ordinary Clos network are applicable here. All the advances in this field can be taken advantage of. For the same reason similar routing time complexities are at work.

The bypass bus makes it possible to carry out broadcast operations. Though it is not included in this work, the algorithms for setting the states of the demultiplexers and multiplexers should be relatively simple.

6.2 Future Work

One research area that could be explored regards the development of new routing algorithms that take advantage of the extra hardware under normal conditions.

These could run in faster time.

The fundamental concept behind the PFTC is not MIN-specific. it could just as easily be applied to any other MIN. For example, the Baseline and the Benes networks could be made fault tolerant this way.

The algorithms to set the states of the demultiplexers and multiplexers are implemented through software by a control unit. If they are implemented in hardware the speed up factor would be considerable, but the cost would increase. Reliability analysis needs to be performed to determine the probability of faults and to see whether the increased cost would be offset by the speed up factor.

BIBLIOGRAPHY

1. Adams, G. 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.

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

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

4. Andresen, S. "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.

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

6. Batcher, K. "The Flip Network in STARAN^{TM}," *Proceedings of the 1976 International Conference on Parallel Processing*, 1976, pp. 65-71.

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

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

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

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

11. Carpinelli, J. and Y. Oruc. "Matrix Decomposition Algorithms for Dynamic Topology Reconfiguration in Parallel Computers," *Proceedings of the 4th International Conference on Supercomputing*, April, 1989.

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

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

14. Davis, W. *Operating Systems: A Systematic View, 2nd Edition*, Addison-

Wesley, Reading, Ma., 1983.

15. Feng, T. 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.

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

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

18. Huang and Tripathi. "Self-routing Techniques in Perfect Shuffle Exchange Networks Using Control Tags," *IEEE Transactions on Computers*, February 1988, pp. 251-256.

19. Hwang K. *Advanced Computer Architecture*. McGraw-Hill, Inc., 1993.

20. Kinny, L. 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.

21. Kothari and Prabhu. "The Kappa Network with Fault Tolerant Destination Tag Algorithm," *IEEE Transactions on Computers*, May 1988, pp. 612-617.

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

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

24. Lee and Hegazy. " The Extra Stage Gamma Network," *IEEE Transactions on Computers*, November 1988, pp. 1445-1450.

25. Lee and Yoon. "The B-Network: A MIN with Backward Links," *IEEE Transactions on Computers*, July 1990, pp. 966-969.

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

27. Lev, G., 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.

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

29. Nassar, H. *Fault-Tolerant Interconnection Networks for Multiprocessor Systems*, Ph.D. Thesis, New Jersey Institute of Technology, Newark, NJ, May 1989.

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

31. Nian, T., et al. "Fault-Tolerant Interconnection Networks via Chaining," *IEEE Transactions on Computers*, April 1988, pp. 458-462.

32. Opferman, D. 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.

33. Oruc, Y. *Interconnection Networks: Group Theoretic Modeling*, Ph.D. Thesis, Syracuse University, Syracuse, NY, 1983.

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

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

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

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

38. Tobias, P. and D. Trindade. *Applied Reliability*. Von Nostrand Reinhold, New York, 1986.

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

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

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