University Of London,

Imperial College Of Science And Technology,

Department Of Computing.

THE ANALYSIS AND DESIGN OF SWITCHING NETWORKS FOR HIGHLY PARALLEL COMPUTER SYSTEMS

by

A.J. Field.

This thesis submitted in part fulfillment for the requirements of the degree of Doctor Of Philosophy, October 1984.

子曰：

學而不思則怠、

思而不學則惘

ABSTRACT

This thesis examines the application of multistage interconnection networks to highly parallel computer systems which may contain many thousands or tens of thousands of component processors. The cost and complexity of the network becomes prohibitively high when existing network design techniques are applied to very large-scale systems. A number of performance models for interconnection networks are derived. This analysis covers some interesting variations on conventional network designs as well as that of a new class of networks which are capable of exploiting locality of reference whilst still providing low-cost global communication. The general properties of this class of networks is described. The analysis of asynchronously controlled full crossbar switches with request blocking is also described. A new approach to network design is then presented. This approach enables the efficient bit-serial control of the network which naturally reduces the size and complexity of the network by allowing larger single-chip component switches to be constructed. The design of the 'XS1' prototype network switching chip, which incorporates these design techniques, is described. Network interfacing and performance are also examined. Finally, network fault tolerance is discussed. We show how by extending any interconnection network so that each attached component has multiple links into the network, a fault tolerant topology can be realised which does not require fault location in order to achieve fault avoidance. We describe a novel technique for concurrent error detection in bit-serially controlled networks which does not require error detection codes to be transmitted through the network. A technique for dynamically replacing individual faulty switches in a fault tolerant network is also described.

Acknowledgements

I would like to take this opportunity to thank all the members of the ALICE research group who over the past three years have become not only respected colleagues but also great friends. My thanks go to Mike Reeve, Pete Harrison, Ian Moor and Victor Wu who have fed me with numerous ideas, Penguin biscuits and cups of dark roast coffee for the past three years, and to Roger Bailey, whose threats of physical violence made the task of writing up so much easier to come to terms with. A special "thank you" goes to Alison Surry who took all the words and somehow managed to get most of them typed out in the right order, to Duncan Haysom who in drawing the diagrams saved me many long hours of frustration, and to Graham Fletcher who turned the XS1 from paper to silicon in one go.

Special thanks also goes to the Science And Engineering Research Council Of Great Britain who have supported both myself and the ALICE project for the past three years.

Above all, however, I would like to thank my supervisor, John Darlington, and my close colleague, Martin Cripps, without whom this thesis would most certainly not have been completed. Their endless encouragement and undying enthusiasm is an inspiration to us all.

TABLE OF CONTENTS

Chapter 1: Introduction. 1

1.1. Exploiting Concurrency 3
1.2. Dynamic Communication Networks 4

Chapter 2: Network Operation And Analysis. 7

2.1. General Properties Of Interconnection Networks 7
2.1.1. Topology 7
2.1.1.1. Discussion Of Topology 10
2.1.2. Irregular Topologies 11
2.1.3. Routing 12
2.1.4. Blocking 14
2.1.5. Switching Mode 15
2.2. Performance Analysis 16
2.2.1. NetworksWithNon-UniformLoad 16
2.2.1.1. Assumptions Of The Model 17
2.2.1.2. A Recurrence Equation For Generating T_N 19
2.2.1.3. Sparse Networks 24
2.2.2. k-Channel Network Analysis 26
2.2.3. Lambda Network Analysis 34
2.2.3.1. Lambda Network Characteristics 36
2.2.3.2. Analysis 38
2.2.3.2.1. Effects Of Locality 44
2.2.4. The Analysis Of Asynchronous Systems 47
2.2.4.1. The Analysis Of An Asynchronous Crossbar Matrix 48
2.2.4.2. Towards Asynchronous Multistage Network Analysis 55

Chapter 3: Self-Clocking Networks. 57

3.1. Design Techniques 57
3.1.1. Serial Switching 60
3.1.2. Self Clocking Networks 61
3.2. The Design Of The XS1 Network Switching Device 64
3.2.1. Description Of The XS1 65
3.2.1.1. The D (Data) And C (Clock) Lines 66
3.2.1.2. The R (Reset) Line 67
3.2.1.3. The A (Address Valid) Line 67
3.2.1.4. The B (Burst) Line 68
3.2.2. The XS1 Routing Cycle 69
3.2.2.1. The End Of Path Protocol 72
3.2.3. XS1 Logic 72
3.2.3.1. The Address Generation Unit, G 75
3.2.3.2. The Arbitration Unit, A 76
3.2.3.3. The Pass (P) And Multiplexer (M) Units 77
3.2.4. Serial Switching, Self Clocking And Structure Independence 78
3.2.5. Interface Design 80
3.2.5.1. XS1 Level 0 Interface Description 80
3.2.5.2. Interface Operation 82
3.2.6. Performance 84
3.2.6.1. Timeouts 85
3.2.7. Future Developments 86
3.2.7.1. Asynchronous Burst Clock Generation 86
3.2.7.2. Tri-Level Logic Implementations 86

Chapter 4: Fault Tolerance In Self Clocking Networks. 89

4.1. Fault Models 89
4.1.1. Permanent Faults 90
4.1.1.1. "Stuck-At" Faults 90
4.1.1.2. Permanent Link Faults 91
4.1.2. Transient Faults 91
4.1.2.1. Transient Switch Faults 91
4.1.2.2. Transient Link Faults 91
4.1.2.3. Synchronisation Failure 92
4.2. Existing Fault Tolerant Schemes 93
4.2.1. Self Testing Switches 93
4.2.2. Extra-Stage Networks 93
4.2.3. Multiple Plane Networks 94
4.3. A Fault Tolerant Scheme For Self Clocking Networks 95
4.3.1. Providing Multiple Paths 97
4.3.1.1. Address Transformation 97
4.3.1.2. Extending The Cube 97
4.3.1.3. Irregular Topologies 102
4.3.2. Error Detection And Fault Avoidance 103
4.3.2.1. Error Detection Mechanisms 105
4.3.2.1.1. Path Building Failure 105
4.3.2.1.2. Addressing Failure 106
4.3.2.1.3. Data Transfer Error 106
4.3.2.2. NIC Operation 107
4.3.2.3. Finding Fault-Free Paths 111
4.3.2.4. Multiple Faults 113
4.3.3. Coping With Multiple Faults 114
4.3.3.1. Fault Location 114
4.3.3.2. Isolating Faulty Switches 116
4.3.3.3. Fault Repair 120
4.4. Summary 121

Chapter 5: Summary And Conclusions 123

REFERENCES 128

Appendix 1: A Routing Algorithm For Lambda Networks 132

Appendix 2: Asynchronous Network Simulator Commands 139

LIST OF FIGURES

Fig.	Description	Page
2-1	An Interconnection Network, Size 16, Degree 2.	8
2-2	Sparse Network Performance	25
2-3	A K-Channel Network Switch With Degree x	26
2-4	K-Channel Network Performance	33
2-5	A Lambda Network, Size=32, Degree=2	35
2-6	Lambda Switch Loading	39
2-7	Lambda Network Performance	44
2-8	Lambda Network Performance With Locality	46
2-9	Queueing Network Model For Single Asynchronous Switch	49
2-10	Asynchronous Full-Crossbar Performance	55
3-1	2x2 Switching Element Schematic	58
3-2	The ALICE Machine Schematic	65
3-3	XS1 Schematic	66
3-4	Path Building	70
3-5	The Arbitration Clock Generator	74
3-6	XS1 Slice Schematic	74
3-7	The Address Generation Unit	75
3-8	The Arbitration Unit	76
3-9	Level 0 Interface For Transputer/Transputer Interconnection	81
3-10	Self Clocking Network Performance Curves	86
4-1	A Cube-Based Fault Tolerant Network, N=32, x=2, k=2.	103
4-2	Example Fault Tolerant Set-Up	104
4-3	NIC Status Register Format	115
A1-1	A Lambda Network Switch	136

LIST OF NUMBERED EQUATIONS

Eqn.	Defined
[2-1]	19
[2-2]	20
[2-3]	21
[2-4]	21
[2-5]	23
[2-6]	30
[2-7]	31
[2-8]	45
[2-9]	51
[2-10]	52
[4-1]	105

LIST OF TABLES

A1-1	Lambda Network Control Strings	135

CHAPTER 1

Introduction

It is now widely acknowledged that concurrency in program evaluation provides the only means of significantly increasing processing speed.

Cooperation between the component processors of a parallel machine relies on inter-processor communication, and the design of communication systems, or communication networks, for multiprocessor architectures has been a subject of growing interest in recent years. This thesis examines the a class of communication networks in the context of highly parallel systems where communication among many hundreds, thousands, or even tens of thousands of processing components is required.

Feng [Fen82] categorises interprocessor communication into static and dynamic types, the former corresponding to distributed, and generally loosely coupled systems where communication is only between adjacent processors which are arranged into a processor interconnection graph, and the latter corresponding to systems where a separate communication subsystem is employed to achieve processor coupling. The latter types of system are often, although not always, closely coupled, with the communication system providing each processing device with a 'multiport' view of a shared memory system.

Static communication networks are often employed where the structure of the problems being solved closely reflect the structure of the machine (i.e. the topology of the processor interconnection graph). Virtual tree machines [BuS81], which are very effecient at evaluating divide-and-conquer problems are good examples of such systems. Note that there is generally no global communication: subcomputations (together with their data) are

spawned from one processor to an adjacent processor in the graph and results are fed back via the same connection when evaluation is complete.

Dynamic communication networks have been proposed for various SIMD [Bar81, Lan76,LaS76,LaV82,Law75,Len78,NaS80,NaS82,Sie79,YeL81], MIMD [Dij81,JuD81, MAS81,McS80,MSi80,Pat79], dataflow [ADI83,GWG80] and reduction [DaR81, KeL79] machines. In these machines, each system component is connected to a separate communication network and communication between two arbitrary components is achieved by establishing either a physical or a virtual channel through the network between the two along which data can be transferred.

In these machines subcomputations gain access to a data structure not by being passed a copy of the structure, as is often the case in static networks, but by making a direct (or indirect) reference to a shared copy of the structure when the subcomputation is invoked. As a consequence of sharing data structures in this way, computation soon becomes 'detached' from the data on which they operate. The communication network provides the mechanisms for efficiently accessing shared (global) data items which may be located anywhere in the system, and for the general coordination of concurrent tasks by the passing of control information among the systems' components. The distribution of the processes or computations may also be facilitated by the network.

This thesis is concerned with the design and analysis of a class of dynamic interconnection networks, which are communication networks analogous to high speed telephone exchanges.

The particular applications considered are those systems which obtain high performance by exploiting very large degrees of parallelism and which have the property of extensibility whereby a guaranteed increase in performance can be achieved by simply incorporating more processors into the system.

These systems may contain tens or even tens of thousands of component processors depending on the computing power required.

A number of network designs have been suggested for use in SIMD and MIMD environments [Bar81,Bar82,LiW82,MAS81,Pat79,WLL82], but we argue that these designs are impractical and inappropriate for systems of any significant size. Networks based upon existing principles then become very large, complex, expensive, and inherently unreliable. This thesis examines a very different approach to network design and proposes techniques for both reducing network complexity and enhancing network reliability.

1.1 Exploiting Concurrency

Considerable research has been done into the problem of exploiting concurrency but this has often led to frustration because of the inherently sequential nature of the programming languages used to formulate the problems being solved. Some language extensions have been introduced (for example, parallel DO, and Fork-and-Join in FORTRAN) to enable the programmer to make better use of the parallelism available in the machine, but this greatly complicates the task of programming - the user requiring a detailed knowledge of the machine in order to best exploit its concurrent capabilities.

Although a number of concurrent programming languages have been developed (for example Modula [Wir83] and Occam [IMS84]), one of the most promising solutions to this problem has evolved through the study of declarative (side-effect free) programming languages [Bur75,Hen80,DHT81]. Such languages possess a clean mathematical semantics which enhances program clarity and which significantly aids the task of program development and automatic transformation [BuD77,Dar81]. Additionally, these languages exhibit the property of referential transparency, one consequence of which is the potential for concurrent evaluation. These issues are eloquently

discussed by Backus in [Back78], and a number of machines designed to exploit this concurrency have been proposed, for example [BuS81,DaR81, KeL79, Mag79].

It is this class of so called 'declarative language support architectures' to which this thesis is particularly addressed, although the issues raised are applicable to all network-based systems employing very large numbers of component processors.

1.2. Dynamic Interconnection Networks.

A dynamic interconnection network is a communication system in which any network input link can be connected to any network output link by appropriately setting the switch, or switches, from which the network is constructed. This differs from static network where the links between two components are passive and cannot be reconfigured for direct connection to other components [Fen82].

The simple shared bus is one extreme form of dynamic network. This is effective provided the number of components attached to the bus is small. As the number increases the bus reaches a saturation point and bus bandwidth, and hence system performance, levels off [KiA78].

In terms of performance, the most effective dynamic communication system is the full crossbar matrix [LVA82,MeC80,MuM82] which may be viewed as a multiple bus system in which each of the connected components has its own unique communication bus. The full crossbar forms the opposite extreme of dynamic communication networks and has been used in several systems (for example the CDC Cyber 170 [Tho70] and Cmmp [FuH78] systems). Again, however, its use is limited, this time because of the very large ($O(N^2)$) costs associated with its construction.

Single and multistage interconnection networks provide a cost-effective compromise between the extremes of the shared bus and the full crossbar matrix. These networks are composed of an interconnected array of switching devices each of which is effectively a small crossbar switch. By appropriately setting these switches, any network input link can be dynamically connected to any network output link.

A number of subclasses of network have evolved. These include the rearrangeable (non-blocking) Clos [Clo53] and Benes [Ben65] networks, the PM2I subclass which includes the Data Manipulator [Fen73], the ADM [McS80, McS81] and Gamma [PaR82] networks, the single-stage (perfect shuffle) networks [Sto71], and the so called 'N log N' multistage interconnection networks [DiJ82] which include, for example, the Omega [Law65], Baseline [WuF80], SW-Banyan [GoL73], indirect binary n-cube [Pea77] and Generalised n-cube [SiM81] networks.

It is this latter class of networks which are considered in this thesis and throughout the remainder of the thesis we shall use the term 'interconnection network' to refer solely to the 'N log N' class of networks although strictly the term is generic, covering the Benes, Clos, PM2I, full crossbar networks, and so on.

The remainder of the thesis is divided into three parts:

Chapter 2 describes the general characteristics of the 'N log N' class of dynamic interconnection networks which are of interest to the thesis. Network performance analysis is also studied and four interconnection network variations are examined and compared.

Chapter 3 describes two design techniques, 'self clocking' and 'serial switching', which are techniques aimed at reducing the cost and complexity of the interconnection network whilst maximising the network flexibility.

This chapter includes a description of a network implementation which has been designed and built as part of the ALICE reduction machine project [DaR81].

Chapter 4 examines the (much neglected) issues of fault tolerance and describes how this can be obtained in interconnection networks by introducing error detection mechanisms and by exploiting some of the topological properties of these networks to facilitate fault avoidance and fault repair.

Finally, the summary and conclusions of the thesis are laid out in Chapter 5.

CHAPTER 2

Network Operation And Analysis

Interconnection networks are a cost-effective means of providing arbitrary point to point communication in a multiprocessor or multicomputer environment, offering greater architectural simplicity and flexibility than the full crossbar matrix which they aim to simulate.

Section 2.1 of this chapter describes the general properties of interconnection networks with regards to their physical and operational characteristics. Section 2.2 is concerned with network analysis and examines the performance of four variations on conventional network implementations.

2.1. General Properties Of Interconnection Networks.

2.1.1 Topology

A regular interconnection network is rectangular array of interconnected full-crossbar switches of arbitrary, but fixed size. In the remainder of this thesis, the switch size - which we generally refer to as the switch degree - will be denoted by x.

A regular interconnection network of size N and degree x consists of $n=\log_x N$ stages of crossbar switches with N/x switches in each stage. The total number of switches in the network is consequently $(N/x)\log_x N$. Each switch in the network has x input ports and x output ports so that the total number of input and output ports in each stage of the network is equal to N. An example of a simple interconnection network of size 16 and degree 2 is shown in Figure 2-1.

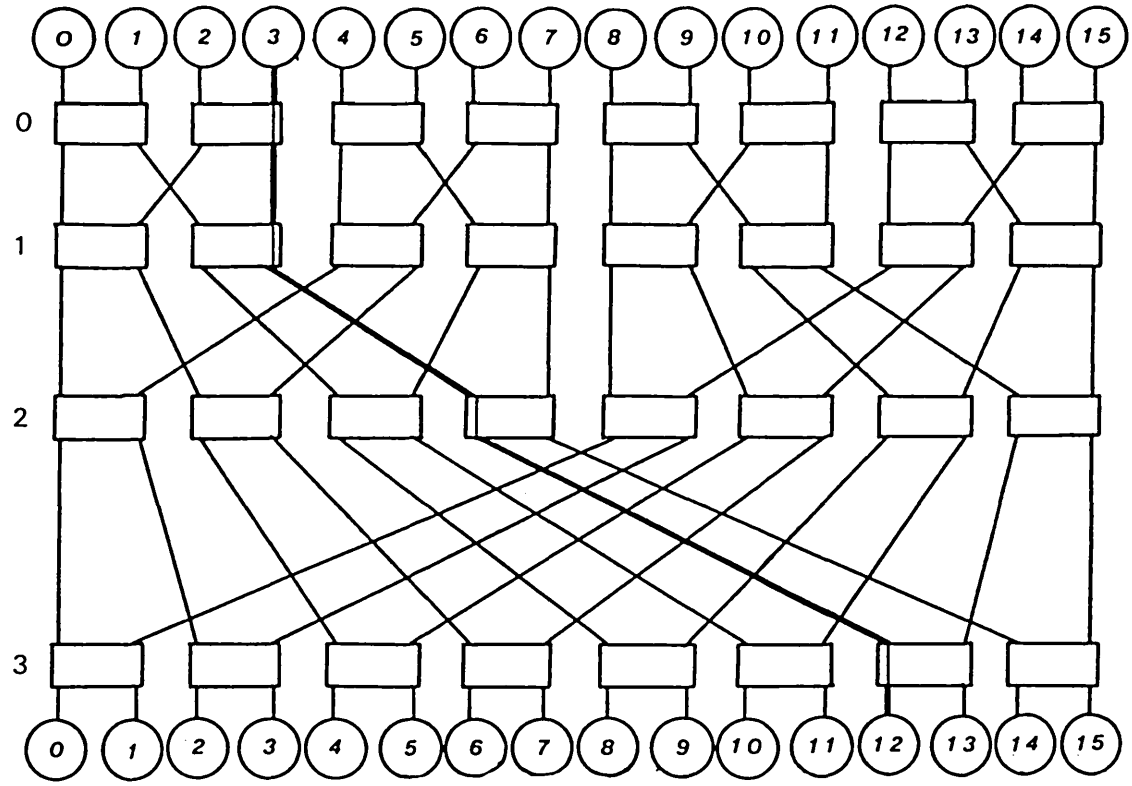


Figure 2-1: An Interconnection Network, Size 16, Degree 2.

The interconnection topology describes the way in which the input and output ports of adjacent stages are connected. Note that there are no connections between the switches of a single stage. Three types of interconnection topology are frequently referred to in the literature; these we now define.

Let the n stages of the network be indexed from 0 to n-1 with stage 0 corresponding to the input side of the network. Now let the jth input link of stage s be labelled $I_s(j)$, j=0..N-1, s=0..n-1, and let the jth output link of stage s be correspondingly labelled $O_s(j)$, j=0..N-1, s=0..n-1.

The interconnection (or permutation) between adjacent stages, say s and

s+1, may be viewed as a function mapping each of the $O_s(j)$ to exactly one of the $I_{s+1}(j')$, $s \in [0..n-2]$, $j,j' \in [0..N-1]$, i.e.

$$P(s) : O_s(j) \rightarrow I_{s+1}(j')$$

where P is the permutation function for stage s of the network.

We describe the permutation by specifying the relation between j and j' above for a given value of s for each of the three permutation functions. To do this we shall write j as a base x number, which, for a network with n stages will contain n digits. So let j be written:

$$j = \langle j_{n-1}, j_{n-2}, .., j_1, j_0 \rangle, \quad j_k \in [0..x-1], \; k=0..n-1$$

We can now express the three permutation functions as functions which transform the component digits of j:-

1. The CUBE Permutation.

$$P(s) : O_s(\langle j_{n-1}, j_{n-2}, .., j_{s+2}, j_{s+1}, j_s, .., j_1, j_0 \rangle) \rightarrow$$
$$I_{s+1}(\langle j_{n-1}, j_{n-2}, .., j_{s+2}, j_0, j_s, .., j_1, j_{s+1} \rangle), \quad s=0..n-2$$

i.e. exchange digits 0 and (s+1) of j.

(This permutation is so called since the couplings between stages s and s+1 of the network are the same as the couplings in the (s+1)th dimension of an x-wide n dimensional hypercube [MSi80, Pea77].

2. The Shuffle Permutation

$$P(s) : O_s(\langle j_{n-1}, j_{n-2}, .., j_1, j_0 \rangle) \rightarrow$$
$$I_{s+1}(\langle j_{n-2}, .., j_1, j_0, j_{n-1} \rangle), \quad s=0..n-2$$

i.e. left rotate the digits of j.

(This permutation is so called since the case x=2 produces a mapping analogous to that produced by shuffling, or interleaving, two halves of a card deck whose face values are the integers 0..N-1. This topology with x=2 forms the basis of Lawrie's omega network [Law75].)

3. The Partial Shuffle Permutation

$P(s) : O_s(<j_{n-1},..,j_{s+2},j_{s+1},j_s,..j_1,j_0>) ->$

$I_{s+1}(<j_{n-1},..,j_{s+2},j_s,..,j_1,j_0,j_s+1>), \quad s=0..n-2$

i.e. perform the shuffle permutation on digits 0..s+1 of j.

(This permutation forms the basis of the baseline topology [WuF80] which has been used to demonstrate the topological equivalence of various N log N networks.)

For example, starting at input port 3 (0011) in the network of Figure 2-1 (which has a Partial Shuffle topology) we have that:

$O_0<0011> -> I_1<0011>$

$O_1<0011> -> I_2<0110>$

$O_2<0110> -> I_3<1100>$.

2.1.1.1. Discussion of Topology

The three permutations defined above form the basis of all regular interconnection networks . A number of these networks use inverted forms of the above topologies, for example the Generalised Cube topology [SiM81] which is simply an inverted form of the indirect x-ary n-cube (which itself is a generalisation of the binary n-Cube of Pease [Pea77]). Note that the Cube permutation function as defined above requires an additional permutation on the output side of the network to 'realign' the port addresses so that they are contiguous at the network outputs. The same is true of an inverted shuffle topology, although the inverted baseline topology requires no output permutation, regardless of the orientation of the network.

An interesting property of the shuffle permutation is that the interconnection is independent of the stage number. Consequently, an extension to a shuffle interconnected network of size N requires the

existing interconnections to be rewired so as to retain the shuffle topology in the extended network. This limits its applications in reconfigurable systems. Also, since the interconnection is the same for all stages, the network may be compressed into a single stage topology in which the network outputs are fed back to the inputs. This configuration is termed a 'single stage' shuffle-exchange network [LaN76,WuF81], and functions by repeatedly circulating requests through the single stage until the required destination node has been reached. Various algorithms have been developed to run directly on a single stage shuffle processor array [Kuh80]. The use of these networks for more general interprocesor communication has been examined in [KDJ83].

2.1.2. Irregular Topologies

Of particular significance in the next chapter is the class of irregular network topologies which can be obtained by incorporating switches of varying size into a single network. Irregular networks are important when large switches are being used to construct a network. If only switches of degree x are being used then only networks of size x^n, n=1,2,... can be constructed. If intermediate sizes are required, then it must be possible to integrate stages of switches of degree y<x. Such a network then has no fixed degree; the term hybrid network will often be used to refer to such irregular configurations.

As a simple example, consider the problem of constructing a Cube-based network of size $2x^n$ using n stages of switches of degree x and one stage of switches of degree 2.

The rules for interconnecting the n stages of switches of degree x are simply obtained from the regular permutation function for the Cube as was given above. Further extending this network by a factor of two requires adding an additional stage of switches of degree 2. In the regular Cube

network new stages are (ordinarily) added to the bottom (output) stage of the network. The permutation function required between the old output stage and the new degree-2 stage is obtained by viewing the original regular network as being of size 2^r for some r. (An integer value of r will always exist since x is a power of two [Pat79].) The extension is then viewed as a normal extension to a network of degree 2, i.e. the required permutation function is simply that of the Cube, with x=2. Note that if the existing cube is inverted (yielding the Generalised Cube topology), then the extra stage will be added at the top of the network, but the permutation function will be the same.

A wide variety of irregular networks can be conceived. Networks composed of many different sized switches are possible and these can, in theory, occur anywhere in the network so long as the network can be consistently addressed.

An exhaustive categorisation of irregular topologies is impossible, suffice it to say that the ability to construct hybrid networks adds considerable flexibility to system design. A network of size 2^n for any n, for example, may be constructed using a combination of switches of size 2, 4, 8, 16... and so on if required by the system. In the next chapter we propose a design methodology which enables arbitrarily complex hybrid systems to be built without destroying the functionality of the network as a whole.

2.1.3. Routing

The process of establishing a path through the network involves individually selecting or addressing exactly one switch in each stage of the network.

It is easy to see from Figure 2-1 that an interconnection network is a system of N interwoven 1:x demultiplexor trees (of fixed, or mixed degree),

and the path building process is similar to that used in conventional tree traversal. Only the required network output port address need be specified in order to establish a path from any source (network input port) to any destination (network output port).

Consider a regular network of size N and degree x. Let the source node address be:

$$S = \langle s_{n-1}, s_{n-2}, .., s_1, s_0 \rangle \quad s_i \in [0..x-1],\ i=0..n-1$$

and the destination node address be:

$$D = \langle d_{n-1}, d_{n-2}, .., d_1, d_0 \rangle \quad d_i \in [0..x-1],\ i=0..n-1$$

A path from S to D is established by using d_i (or d_{n-i-1} depending on the network) to select stage i of the network. Since each switch in a regular network has x input and x output ports, d_i is sufficient to specify exactly one of the switches' output ports. It is easy to show that this rule correctly steers the request to its addresed destination node:-

Consider, for example, the Generalised Cube topology which is an inverted form of the Cube topology defined in 2.1.1 above in which d_{n-i-1} is ordinarily used to select stage i of the network, i=0..n-1. In the Generalised Cube, the input (top) stage of the network is traditionally labelled stage n-1 and the output (bottom) stage of the network - stage 0. Using our notation, the permutation function, G, for stage s is given by:-

$$G(s) : O_s(\langle j_{n-1}, .., j_{s+1}, j_s, j_{s-1}, .., j_1, j_0 \rangle) \rightarrow I_{s-1}(\langle j_{n-1}, .., j_{s+1}, j_0, j_{s-1}, .., j_1, j_s \rangle),\ s=1..n-1$$

After stage n-1 of the network has been selected (using d_{n-1}), the request is steered to the stage n-1 output port labelled:

$$\langle s_{n-1}, s_{n-2}, .., s_1, d_{n-1} \rangle.$$

From the G(n-1) permutation rule, this port is linked to the stage n-2 input port labelled:

$$\langle d_{n-1}, s_{n-2}, .., s_1, s_{n-1} \rangle .$$

After stage (n-k) has been selected similarly, the request is steered to the stage n-(k+1) input port labelled:

$$\langle d_{n-1}, d_{n-2}, .., d_{n-k}, s_{n-(k+1)}, .., s_1, s_{n-k} \rangle .$$

Consequently, after stage 1 (=n-(n-1)) has been selected, the request arrives at input port

$$\langle d_{n-1}, d_{n-2}, .., d_1, s_1 \rangle$$

of stage 0 of the network. Thus supplying the last routing address, d_0, to the corresponding switch then steers the request to the network output port labelled

$$\langle d_{n-1}, d_{n-2}, .., d_1, d_0 \rangle \quad \text{Q.E.D.}$$

In an irregular network, the bit string representing the destination address is simply partitioned irregularly so that the length of each substring, d_i, used for addressing purposes may be different. In the case where each stage of the network contains switches of fixed degree but the stages themselves are of mixed degree, then the routing scheme may be viewed as being based on mixed radix addressing [BhA82]. A folded network of this sort reduces to an alpha structure as defined in [BhA82].

2.1.4. Blocking

The paths from a single source node to the set of all destination nodes of a network form a 1:N demultiplexor tree. Because the demultiplexor trees of all the available input nodes share common nodes in the interconnection graph it is possible that two requests originating from separate source

nodes and addressed to separate destination nodes both require to use the same output port of one particular switch in the network. Because in a demultiplexor tree there is only one path between any input and output node, one of these requests must be denied and a blockage is said to have occurred. Until the successful request 'releases' this output port, the blocked request must either 'hang' or cancel. Path building may be held up in any stage in the network mas a result of blocking.

The degree of blocking experienced in a network is a key factor affecting network performance. This isssue is raised in 2.2. below where we use probabilistic techniques to analyse the contention in a variety of interconnection network configurations.

2.1.5. Switching Mode

An interconnection network may operate in either a circuit switched or a packet switched mode of communication, and the arguments concerning both approaches are similar to those debated in the context of other communication systems. Basically, packet switching offers the advantages of reduced link contention since the packets themselves are buffered in successive stages of the network so that the links used to transmit the packet may be immediately released for further traffic. An excellent examination of packet switching techniques has been presented by Jump and Dias for both single [KDJ83] and multistage interconnection networks [JuD81, DiJ81].

In this thesis we concentrate on circuit switched interconnection networks in which a physical (rather than virtual) channel is established between the source and destination nodes. We do not wish to argue that this approach is generally superior since the most suitable mode is dependent very much on the application. Circuit switching yields faster access times [Bar81], allows for arbitrarily large volumes of data to be transferred in

a single transaction and also facilitates bidirectional (full duplex) communication between source and destination.

2.2. Performance Analysis

This section examines interconnection network performance for networks with varying physical and operational characteristics.

We examine four network models. The first considers an extension to the model given by Patel [Pat79] in which we derive equations for the throughput of a conventional switching network operating in a cyclical mode with non-uniform input load. The second model examines the effect of providing multiple channels on each port of the switching nodes of a conventional network. The third model covers the analysis of new class of interconnection structures called Lambda networks, which we define, and the fourth investigates the application of queueing theory to the analysis of asynchronous networks. The design of asynchronous networks is taken up in Chapter 3.

2.2.1. Networks with Non-Uniform Load

This analysis relates to networks which operate in a **cyclical** mode. A network cycle describes a time frame in which the entire network is synchronously routed through and data is transferred along those paths which have been established without incurring blockage [Pat79]. Consequently this analysis is particular to circuit-switched implementations.

The analysis serves two important functions:

1. For networks which naturally operate in a cyclical mode, for example [Bar81], it yields an accurate measure of the networks throughput.

2. It provides a valuable reference model by which different network configurations can be compared. The cyclical model is adopted in both the next two sections; a direct comparison of the performance of the various networks is then possible.

2.2.1.1. Assumptions of the Model

The following model is a generalisation of that given in [Pat79] which covers the analysis of delta networks.

The environment considers networks which operate on a cyclic basis. For the time being we shall consider a regular network of size N and degree x in which N_i of the input ports and N_o of the output ports are actually connected to system components. n will be used to denote the number of stages in the network, i.e. $n=\log_x N$.

At the start of a network cycle, requests are synchronously presented to each of the networks' input ports. These requests are then passed through each of the $\log_x N$ switching stages of the network simultaneously. Certain of the requests will be blocked at intermediate switches in the network. Those which are not blocked form physical channels betwen the input and output ports of the network. The network then enters a data transfer phase during which data is passed (possibly in both directions) across the network. At the end of data transfer, all paths are simultaneously released and the cycle repeats.

We lift the restriction imposed by Patel's model and consider a system in which the inputs may be unequally loaded by the components attached to the network. This enables the performance of certain types of heterogeneous systems to be predicted. In particular, the analysis subsumes both the original analysis given by Patel and that of **sparse** networks, in which only a subset of the input and output ports of the network are connected to system components.

Let the cycles be enumerated by $\mathcal{C}$ = {0,1,...} and let E(c,k) be the event: "there exists an arrival on channel k at the start of cycle c ($c \in \mathcal{C}$, $0 \leq k \leq N-1$)". The extended definition of Patel's model is then as follows:-

1. E(c,k) and E(c',k') are assumed to be independent, $c \neq c'$, $k \neq k'$. Note that this implies that blocked requests are 'lost', i.e. requests are assumed to be submitted to a port independently of any previous transactions on that port.
2. The probability that a request is submitted on input port k at the start of a cycle is P_k, k=0..N-1.
3. All output ports connected are equally likely to be addressed, that is no output ports are 'favoured' in preference to others.

Note that if only a subset of the output ports are connected to system components, then these are assumed to be evenly distributed across the outputs. That is, each switch in the last stage of the network is assumed to have the same number of connected ports. Note also that assumption 1 is not realistic since in practice blocked requests are almost certain to be resubmitted in the next cycle. However, simulation suggests that this assumption introduces only a small inaccuracy in the predicted performance of the network.

The performance measure we are concerned with here is normalised throughput, T_N, as defined in [JuD81]. This is the proportion of the output ports which are active during each cycle and is also accurate as a measure of the degree of blocking in the network. From the normalised throughput, the acceptance probability, P_A, [Pat79] can be easily obtained from:

$$P_A = \frac{N_0 \, T_N}{\sum_{k=0}^{N-1} P_k}$$

and the actual bandwidth BW (in requests passed per second) from:

$$BW = \frac{N_0\ T_N}{t_c}$$ where t_c is the network cycle time.

2.2.1.2 A Recurrence Equation for Generating T_N

We begin by considering the top stage of switches (stage 0) and viewing each switch in isolation. Label the switches of the top stage of the network 0..(N/x)-1, and consider some switch w, $w \in [0..N/x-1]$. Now label the input ports associated with that switch k, k+1 .. k+x-1. (Clearly, k=wx.) We wish now to derive the normalised throughput of this switch given that the input loads on the inputs to the switch are P_k, P_{k+1},..., P_{k+x-1}, as specified by the model. Define $T_N(s,w)$ as the normalised throughput of switch w of stage s. $T_N(n-1,w)$ then represents the normalised throughput of switch w in the output stage of the network.

Let R be a random variable denoting the number of requests present at switch w at the start of a given cycle. Then, we have:

$$Pr\{R=0\} = \prod_{l=k}^{k+x-1} (1-P_l)$$

$$Pr\{R=1\} = \sum_{l=k}^{k+x-1} P_l \prod_{\substack{m=k \\ m\neq l}}^{k+x-1} (1-P_m)$$

$$= \prod_{l=k}^{k+x-1} (1-P_m) \sum_{l=k}^{k+x-1} \frac{P_l}{(1-P_l)}$$

and so on. Generally:

$$Pr\{R=r\} = \prod_{l=k}^{k+x-1} (1-P_l) \sum_{i_1=k}^{k+x-r} \sum_{i_2=i_1+1}^{k+x-(r-1)} \cdots \sum_{i_r=i_{r-1}+1}^{k+x-1} \prod_{m=1}^{r} \frac{P_{i_m}}{(1-P_{i_m})} \quad [2-1]$$

$T_N(0,w)$ can be interpreted as the average load on each output port of the switch. Let U(r) be the probability that a given output link of the switch contains a request given that r requests are present at the inputs to the switch. Then U(r) is given by:

$$U(r) = 1 - \left(\frac{x-1}{x}\right)^r$$

Thus, we have for $T_N(0,w)$:

$$T_N(0,w) = \sum_{r=0}^{x} Pr\{R=r\}.U(r)$$

$$= \sum_{r=0}^{x} \prod_{l=k}^{k+x-1}(1-P_l) \sum_{i_1=k}^{k+x-r} \sum_{i_2=i_1+1}^{k+x-(r-1)} \cdots \sum_{i_r=i_{r-1}+1}^{k+x-1} \prod_{m=1}^{r} \frac{P_{i_m}}{(1-P_{i_m})} \quad x$$

$$1 - \left(\frac{x-1}{x}\right)^r$$

$$= 1 - \sum_{r=0}^{x} \prod_{l=k}^{k+x-1}(1-P_l) \sum_{i_1=k}^{k+x-r} \sum_{i_2=i_1+1}^{k+x-(r-1)} \cdots \sum_{i_r=i_{r-1}+1}^{k+x-1} \prod_{m=1}^{r} \frac{(x-1)\,P_{i_m}}{x(1-P_{i_m})}$$

[2-2]

In order to simplify this expression, we consider the following product term:-

$$D = (a_0+b_0) \times (a_1+b_1) \times .. \times (a_{n-1}+b_{n-1})$$

This expands in a similar manner to the Pr{R=r} terms given above. In fact, the component of D which contains exactly r 'a' terms is given by:

$$C_r = \prod_{l=0}^{n-1} b_l \sum_{i_1=0}^{n-r} \sum_{i_2=i_1+1}^{n-(r-1)} \cdots \sum_{i_r=i_{r-1}+1}^{n-1} \prod_{m=1}^{r} \frac{a_{i_m}}{b_{i_m}} \qquad [2\text{-}3]$$

Thus D may be rewritten as:

$$D = \sum_{i=0}^{n} C_i$$

Therefore, by substituting $a_j=(x-1)P_j/x$ and $b_j=(1-p_j)$, j=0..x-1, we arrive at an equivalent equation for $T_N(0,w)$ i.e.

$$T_N(0,w) = 1 - \prod_{j=k}^{k+x-1} \left(\frac{(x-1)p_j}{x} + (1-p_j) \right)$$

$$= 1 - \prod_{j=k}^{k+x-1} \left(1 - \frac{p_j}{x} \right) \qquad [2\text{-}4]$$

Now, if $p_j=p$ for all j=k..x+k-1, then equation [2-4] becomes:

$$T_N(0,w) = 1 - \left(1 - \frac{p}{x} \right)^x$$

which is consistent with the result given in [JuD81].

The set of $T_N(0,w)$, w=0..(N/x)-1 characterises the normalised throughputs for each switch of the first stage of the network and these formthe input load to the switches of the next stage (stage 1) of the network.

By using the values for the $T_N(0,w)$ and by applying the process recursively, the equation for $T_N(1,w)$ can be found. From the values of $T_N(1,w)$ w=0..(N/x)-1, the values of the $T_N(2,w)$ w=0..(N/x)-1 can be found, and so on. An interesting property of these networks is that:

$T_N(n-1,w) = T_N(n-1,w')$ for <u>all</u> $w,w'=0..(N/x)-1$.

Observe that although the inputs to the network were unevenly loaded, the normalised throughput of <u>all</u> the networks outputs is the same. This result, (which holds for all networks based on the topologies of 2.1.1.1 but <u>not</u> for all delta networks), stems from the partitioning properties of the three important permutation rules defined in 2.1.1.1:-

Consider stage 0 of the network. This stage partitions the rest of the network into x independent subnetworks [Sie80]. That is, stages 1..n-1 form x independent networks which have no switches in common. The outputs of each switch in stage 0 of the network are fed into the top stage of independent subnetworks. No two links from such a switch connect to the same subnetwork.

Consider now switch w in stage 1. This is linked to by exactly x disjoint switches in stage 0. Let the stage 0 switch connected to input port k of switch w be labelled w_k', k=0..x-1. The w'_k can be found from the permutation function of the network. Consider, for example, the Cube topology defined in 2.1.1.1. If w is written as a base x number, $\langle w_{n-1},w_{n-2},..,w_2,w_1\rangle$, then w'_k is given by:

$$w'_k = \langle w_{n-1},w_{n-2},..,w_2,k\rangle, \quad k=0..x-1.$$

Now consider some w'_k, $k \in [0..x-1]$. The complete set of stage 0 output ports associated with switch w'_k are those labelled $\langle w_{n-1},w_{n-2},..,w_2,k,d\rangle$ d=0..x-1. These outputs are attached to the stage 1 switches labelled $\langle w_{n-1},w_{n-2},..,w_2,d\rangle$, d=0..x-1.

Consequently, the dth output port of <u>each</u> of the w'_k, k=0..x-1, all 'meet' at the same switch in stage 1, that is, at switch $\langle w_{n-1},w_{n-2},..,w_2,d\rangle$. Note that:

1. Each of the switches of stage 1 labelled $\langle w_{n-1},w_{n-2},..,w_2,d\rangle$, d=0..x-1 lie in independent subnetworks formed by stages 1..n-1.
2. Each of the switches of stage 1 labelled $\langle w_{n-1},w_{n-2},..,w_2,d\rangle$, d=0..x-1 have identical input loads, since their inputs originate from the same set of switches in stage 0.

Since the same argument applies to all switches in stage 1, it follows that each of the x subnetworks formed by stages 1..n-1 have identical loading patterns i.e. for each stage 1 switch (forming a top-stage switch of exactly one subnetwork), with input loads, $L_0,..,L_{x-1}$, there exists exactly one corresponding switch in each of the other x-1 subnetworks with the same input loads, $L_0,..,L_{x-1}$. The above argument can now be applied (recursively) to each of the x subnetworks. Since each stage partitions the network into x independent subnetworks and since the network has a total of $n=\log_x N$ stages, it follows that after the last stage has been traversed, the throughput of all switches in the last stage of the network is the same. Consequently, $T_N(n-1,w)=T_N(n-1,w')$ for all w,w'=0..(N/x)-1.

Note that the equation for T_N is correct, providing each output port of the network is being used. Consider now the case where $N_o<N$, and where each switch in stage n-1 of the network has c of its outputs connected, so that $N_o=(N/x)c$. Following the same sequence of steps as was used to generate equation [2-4], the value of T_N is found to be:-

$$T_N = 1 - \prod_{j=0}^{x-1} \left(1 - \frac{L_j}{c} \right) \qquad [2\text{-}5]$$

where L_m is the input load to link m of each switch in the last stage.

As a result of the uniform load sharing properties of these networks, the value of T_N is valid for all connected output ports of the network. The value of T_N may also be interpreted as the percentage load on each connected output link of the network. Thus the total network bandwidth and

acceptance probability may be expressed as was given in 2.2.1.1.

2.2.1.3. Sparse Networks

As was stated earlier, the above analysis subsumes both Patels' original delta network analysis and that of sparse network analysis in which only a subset of the network input/output ports are connected. The throughput of such networks can be obtained by setting the input load values of all unconnected input ports to zero, and by using equation [2-5] to determine the throughput of the last stage of switches, where only a subset of the output ports of each stage n-1 switch are connected. Note that for a single switching element of degree x in which only y of the inputs are used, the load presented to the switch when each of the y inputs are saturated (that is, have input loads of 1) is equivalent to a load of:-

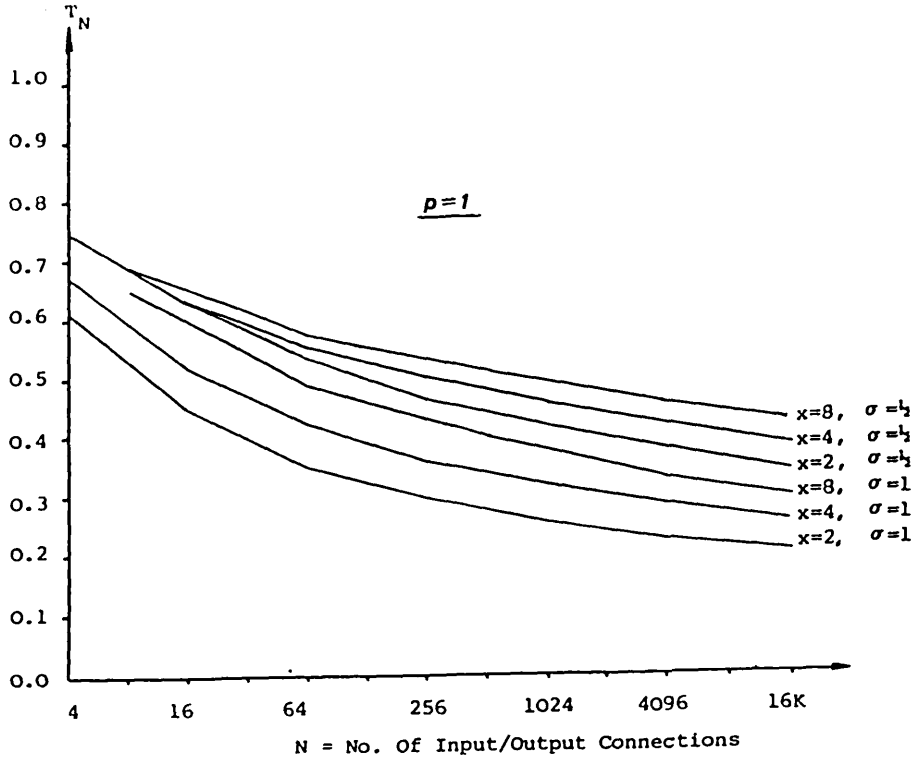
$$x\left(1 - \left(1 - \frac{1}{x}\right)^{y/x}\right)$$

on <u>each</u> of the x ports of an equivalent fully populated switch; <u>not</u> (y/x) as might be predicted. This (incorrect) assumption has been assumed made on at least one occasion in the literature.

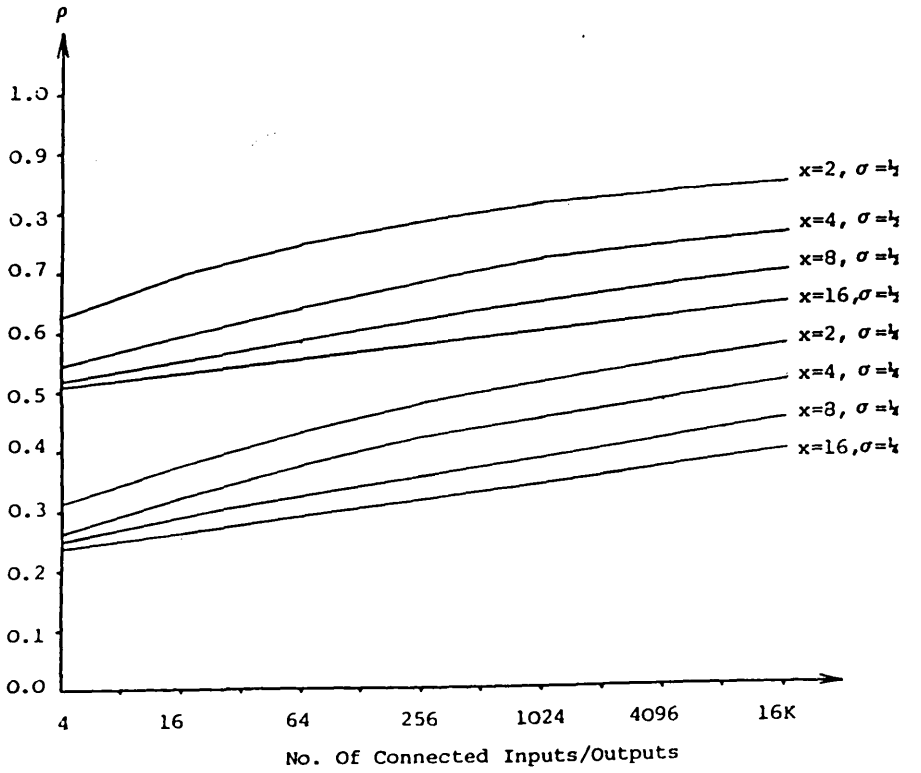
Sparse networks have been proposed as a means of reducing network contention [Bar81]. Figure 2-2(a) shows a comparison of conventional and sparsenetwork performances for varying network sizes[1] and switch degrees. In these curves each connected input is assumed to be saturated (i.e. p=1 for all connected inputs). The parameter, σ, denotes the ratio of connected inputs (and outputs) to network size. Thus σ=1 corresponds to a conventional (non-sparse) configuration. Figure 2-2(b) shows the <u>relative</u> performance/cost curves for the same sparse networks. The parameter, ρ, denotes the ratio of sparse network performance/cost to conventional network performance/cost. The cost is defined to be the

---

[1] In the performance curves, N denotes the number of connected inputs and outputs in the network <u>not</u> the total number of network inputs/outputs.



(a) Performance Comparison With Conventional Networks.



(b) Relative Performance/Cost Curves.

Figure 2-2: Sparse Network Performance.

number of switches required to connect a given number of system components. Thus, ρ is given by:

$$= \sigma \frac{T_N(\text{sparse})}{T_N(\text{non-sparse})}$$

Although sparseness reduces contention, these curves show that the performance gain does not outweigh the required cost increase of $1/\sigma$. Sparse networks are, however, easily implemented and require no additional hardware in either the switches or the network interfaces.

2.2.2 k-Channel Network Analysis

We now examine an interesting variation on traditional network topology and consider the analysis of networks in which each switch port has k channels, or links, associated with it instead of the usual one.

Figure 2-3 illustrates a single switching element of degree x in which each of the x input and output ports contains k links.

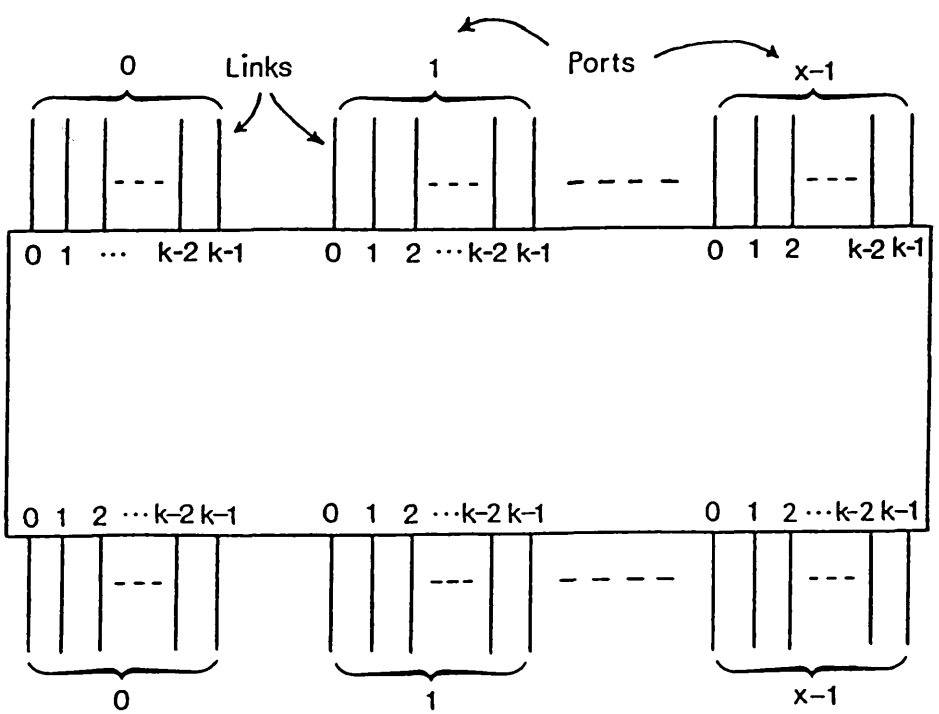


Figure 2-3: A k-Channel Network Switch With Degree x.

A k channel network operates in the same manner as a traditional network except that up to k requests can now be passed through a switch port simultaneously instead of only one. We wish to examine to what extent the additional channels reduce network contention and how the resulting performance compares with that of equivalent conventional networks.

There are two ways (not necessarily mutually exclusive) of attaching the system components to such a network:

1. If the system components themselves have multiple communication ports, then the network can be made fully populated, with each component supplying k communication ports for network connection, thereby utilising all $k.x^n$ network input/output ports.

2. If the components have only one communication port, then a sparse connection scheme may be used on the input side of the network so that only one of the k available channels of that port is used, and a k:1 multiplexor used on the output side of the network to concentrate the k network port channels into the (single) destination component port. It is possible for each of the input ports to be connected to independent system components, but multiplexors must always be provided at the output ports.

In order to make a meaningful comparison of k-channel networks with conventional and sparse networks, we consider configurations in which only one of the k network input links associated with an input port is connected to a system component.

Again, we assume the cyclical model as described above where probability of a request being presented to a channel of an input port of the network at the start of a cycle is p for all channels. Since these channels are independent, we may immediately write the probability of there being r

requests at the input channels of a given top-stage switch at the start of a cycle as:

$$\binom{x}{r} p^r (1-p)^{x-r}$$

Given that there are r requests at the inputs to a switch, we now wish to know the probability that an arbitrary output link is selected. To do this, we consider the problem in two parts. Firstly, we derive the probability of m of the r requests being directed to an arbitrary output port (a port here is a collection of k links), then from this we generate the individual output link utilisations.

Given that r requests are present at the inputs to the switch, the probability that exactly m are directed to an arbitrary output port is given by:

$$\binom{r}{m}\left(\frac{1}{x}\right)^m \left(1 - \frac{1}{x}\right)^{r-m}$$

Given that there are m requests at an output port, the mean utilisation of each of the output links associated with that port is defined to be:-

$$\frac{m}{k}$$

since two requests cannot both be directed to the same output link.

For the sake of the model, we assume that a request proceeding to an output port, randomly selects one of the currently unused port channels (if one exists). If $m>k$ then the utilisation of each output channel is 1. Consequently the equation for the mean output channel utilisation must consider two cases: $m \leq k$ and $m>k$. Thus, we get for the mean utilisation, U:

$$U = \sum_{i=0}^{k-1} \Gamma(i) \binom{i}{k} + \sum_{j=k}^{x} \Gamma(j)$$

where:

$$\Gamma(n) = \sum_{r=0}^{x} \binom{x}{r} p^r (1-p)^{x-r} \binom{r}{n} \left(\frac{1}{x}\right)^n \left(1 - \frac{1}{x}\right)^{r-n}$$

Now observe that:

$$\binom{x}{r}\binom{r}{n} = \frac{x!}{r!(x-r)!} \cdot \frac{r!}{n!(r-n)!} = \binom{x}{n}\binom{x-n}{x-r}$$

So that we have for $\Gamma(n)$:

$$\Gamma(n) = \binom{x}{n} \left(\frac{p}{x}\right)^n \left(1 - \frac{p}{x}\right)^{x-n}$$

and consequently for U:

$$U = 1 - \sum_{i=0}^{k-1} \binom{x}{i} \left(\frac{p}{x}\right)^i \left(1 - \frac{p}{x}\right)^{x-i} \left(1 - \frac{i}{k}\right)$$

This analysis is correct for a single switch network. However, the equation for U above cannot be used recurrently as in the analysis of 2.2.1. The problem is that although the output ports of a given switch generate statistically independent input processes to the next stage, the channels within the port are not independent. The value of U formulated above is actually an <u>average</u> load taken over the whole port. We cannot, from this average alone, infer that each output link of a given port places a load of U on the next stage of the network independently. So, in forming a recurrence equation to solve for total networkthroughput, we

must consider ports, not links within ports:

Thus, for a given switch, s, we define the following:

1. R_r: The probability that a total of r requests are present on the (kx) input links of s.
2. T_r: The probability that a total of r requests are present on the (k) links of an output port of s.
3. $\emptyset_r$: The probability that a total of r requests are present on the (k) links of an arbitary input port of s.
4. $\emptyset_r'$: The probability that r requests are present on the (k) links of an arbitrary output port of s.

(Since the ports are independent, each input port will have the same value of $\emptyset_r$ and each output port will have the same value of $\emptyset_r'$.

Considering, now, an arbitrary switch in which up to kx requests may be present at the switch inputs, we have:

$$T_r = \sum_{n=r}^{kx} R_n \binom{n}{r} \left(\frac{1}{x}\right)^r \left(1 - \frac{1}{x}\right)^{n-r}$$

and,

$$\emptyset_r' = \begin{cases} T_r & r<k \\ 1 - \sum_{i=0}^{k-1} T_i & r=k \end{cases} \qquad [2\text{-}6]$$

We are now aiming to derive a recurrence equation for the $\emptyset_r'$ in terms of the $\emptyset_r$ r=0..k-1.

R_i is dependent on the way in which requests are distributed over the input links of the switch. Thus, define a vector:

$\underline{n} = \langle n_0, n_1, .., n_{x-1} \rangle$ and let $|\underline{n}| = \sum_{i=0}^{x-1} n_i$

where n_t denotes the number of requests present on port t of a switch, t=0..x-1. R_i, i=0..kx can then be expressed as:

$$R_i = \sum_{\substack{\underline{n} \\ s.t.\ |\underline{n}|=i}} \prod_{j=0}^{x-1} \emptyset_{n_j} \qquad i=0..kx \qquad [2\text{-}7]$$

and the normalised throughput of the output multiplexer by :-

$$T_N = 1-\emptyset_0$$

The required recurrence equation is then obtained from [2-6] by substituting equation [2-7] for R_i. Initially, since only one of the (k) links of a given input port is attached to a system component the $\emptyset_i$ are given by: $\emptyset_0=1-p$, $\emptyset_1=p$, $\emptyset_j=0$, j=0..k.

The equation defining R_i (equation [2-7]), however, is very hard to simplify. The values of the R_i can be computed explicitly, although equation [2-7] should really be considered as a generating function.

For small values of x, it is relatively easy to generate an equational form for the recurrence equation. As an example, consider the simple case with x=k=2. For given values of $\emptyset_0$, $\emptyset_1$ and $\emptyset_2$, equation [2-7] gives us the following values for $R_0..R_4$:

$$R_0 = \emptyset_0^2$$

$$R_1 = 2\ \emptyset_0\ \emptyset_1$$

$$R_2 = 2\ \emptyset_0\ \emptyset_2 + \emptyset_1^2$$

$$R_3 = 2\ \emptyset_2\ \emptyset_1$$

$$R_4 = \emptyset_2^2$$

We then have for the T_r, r=0..4:

$$T_r = \sum_{n=0}^{4} \frac{R_n}{2^n} \binom{n}{r}$$

which (as may be verified by the reader) gives the following recurrence equation for the $\emptyset_i$':-

$$\emptyset_0' = \emptyset_0 \left(1 - \frac{\emptyset_2}{2}\right) + \frac{\emptyset_1}{4} (1 - \emptyset_0) + \frac{\emptyset_2^2}{16}$$

$$\emptyset_1' = \emptyset_0 (\emptyset_1 + \emptyset_2) + \frac{\emptyset_1}{2} \left(\emptyset_1 + \frac{3\emptyset_2}{2}\right) + \frac{\emptyset_2}{4}$$

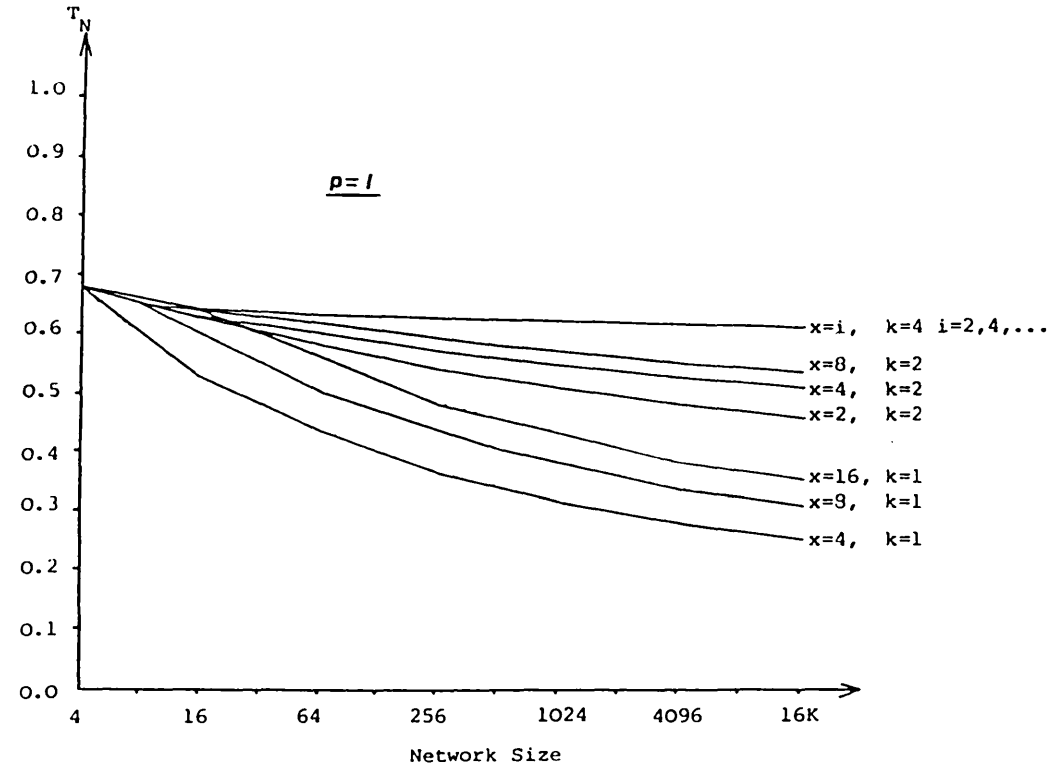
$$\emptyset_2' = 1 - (\emptyset_0' + \emptyset_1')$$

The normalised throughput of the network is obtained by repeatedly applying the above equations, setting $\emptyset_0=\emptyset_0'$, $\emptyset_1=\emptyset_1'$ and $\emptyset_2=\emptyset_2'$ after each iteration and repeating the process as many times as there are stages in the network.

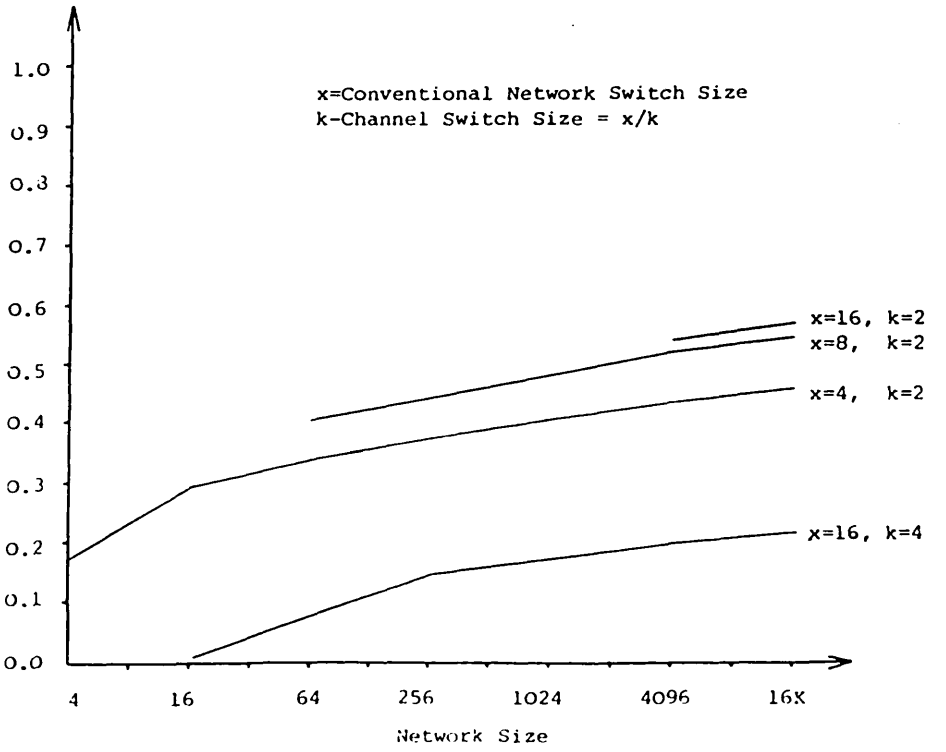
Figure 2-4(a) compares the throughputs of conventional and k-channel interconnection networks for varying network sizes (measured by the number of connected inputs) and switch degrees. These graphs assume that all connected inputs present a load of 1 (i.e. a request is submitted on every cycle).

In terms of raw performance, the k-channel configurations offer higher throughput than conventional networks, but they require larger switch packages because of the additional communication links associated with each port. A conventional switch of degree x thus requires approximately the same number of pins as a k-channel switch of degree x/k.

The effect of this is revealed in the relative performance/cost curves shown in Figure 2-4(b). In these curves the relative performance/cost metric, ρ, is given by:-



(a) Performance Comparison With Conventional Networks.



(b) Relative Performance/Cost Curves

Figure 2-4: K-Channel Network Performance.

$$\rho = \frac{\log_x N}{k.\log_{x/k} N} \cdot \frac{T_N(\text{k-channel})}{T_N(\text{single channel})}$$

In Figure 2-4(b) the performance/cost figures for conventional networks of degree x are compared with those of k-channel networks with degree x/k for varying values of k>1.

2.2.3. Lambda Network Analysis.

In a conventional interconnection network the input and output ports of the network occur at opposite extremes of the network. The 'distance', in terms of the number of switches traversed, between a source component attached to the input side of the network and a destination component attached to the output side of the network is the same for all source/destination pairs - i.e. $\log_x N$.

We now introduce Lambda networks which offer the benefits of global communication yet which also allow locality between communicating components to be exploited. Lambda networks are homomorphic (one-sided) networks which are similar in topology to conventional interconnection networks except that both the input and output links of the network occur on the same side of the network. Figure 2-5 shows an example of a Cube-based Lambda network of size 32 and degree 2.

In Figure 2-5, the system components are shown at the bottom of the diagram and each has both a link into and a link out of the network via the bottom stage of the network, although the input and output links could be attached to independent components. Requests submitted by the components are fed upwards until the addressed destination node can be reached from the

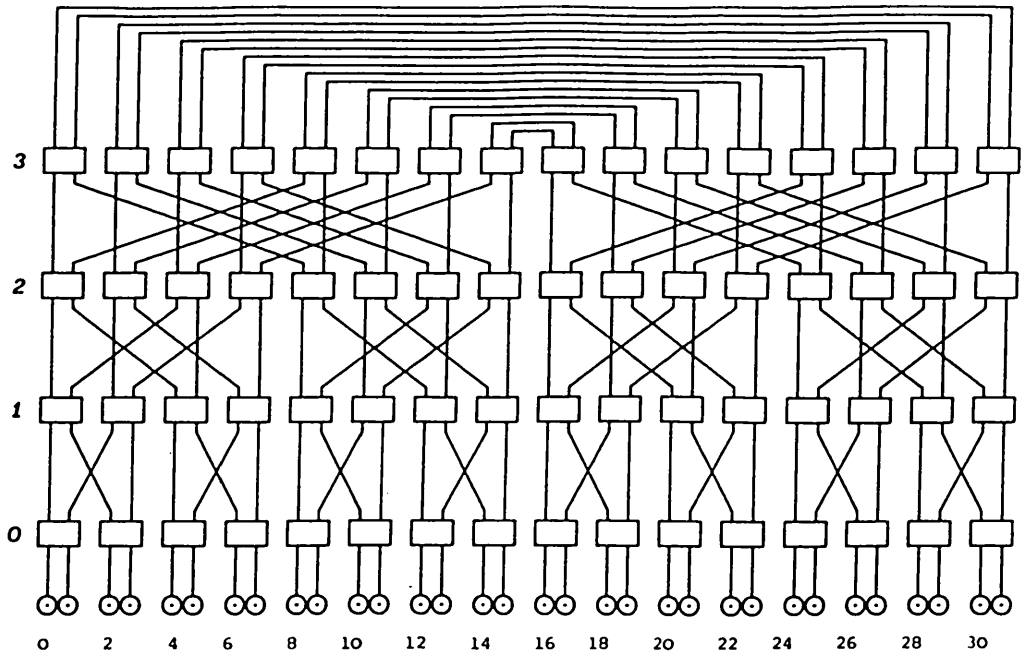


Figure 2-5: A Lambda Network, Size=32, Degree=2.

currently selected switch by turning round ('pivoting') and traversing the network in the opposite direction. The switches on the downward path are selected in the usual way i.e. the switch select addresses are derived from the destination node address. An algorithm for path building in a Lambda network is described in Appendix I.

Unlike conventional networks, it is possible for a request to reach its required destination by traversing less than the usual $\log_x N$ switches. (In the extreme, if the source and destination addresses are the same then the network need not be accessed at all.) We now wish to know whether the ability to take these 'short cuts' through the network has any effect on the network performance for a network of given size and degree. Firstly, though, we describe some of the general characteristics of these networks which are important to the analysis given in 2.2.3.2 below.

2.2.3.1. Lambda Network Characteristics.

A regular Lambda network of size 2N and degree x consists of two identical and interconnected subnetworks located side by side each of which has one of the topologies specified in 2.1.1.1. There are thus $n=\log_x N$ stages in the network. In a Lambda network, a channel between two switches, say S_1 and S_2, consists of two independent links - one from S_1 to S_2 and the other from S_2 to S_1.

The bottom stage of switches (stage 0) is designated the input/output side of the network which provides 2N network input links and 2N network output links. Each output link from the top stage (stage n-1) of one subnetwork is connected to exactly one input link of the other subnetwork. Note that the permutation here is arbitrary, a sufficient permutation rule being, for example:-

$$O\hat{}_{n-1}(j) \rightarrow I_{n-1}(2N-j-1), \qquad j=0..2N-1,$$

where O and I have the same meaning as in 2.1.1. tagged with a '^' if the corresponding link is an upgoing link.

The visibility of switch w is defined to be the set of network outputs which can be reached by traversing a downward path from w. During path building from some source node, S, to some destination node, D≠S, successive stages of the network are traversed, from S, in the upward direction until a switch, w, is reached from which D is visible. The request then pivots about w and traverses the downward path toward D. If stage n-1 is reached before D has become visible, then D must be connected to an output link in the other subnetwork comprising the Lambda network. The request is therefore passed over to the other subnetwork via the links at the top of the network. Because each subnetwork is equivalent to a conventional network when being traversed in the downward direction, the request can enter the subnetwork anywhere in the top stage (stage n-1). Thus the choice of permutation rule between the two subnetworks at the top

is arbitrary.

Observe that:-

1. A request only pivots <u>once</u>.
2. A request traversing $I\hat{}_s(j)$ never traverses $O_s(j)$ since pivoting occurs as soon as the destination becomes visible.

But most significantly,

3. If a request is to be steered upwards from some switch, w, then the choice of which upgoing output link to take is <u>arbitrary</u>. i.e. all upgoing traversals are contention free.

This third point we now prove in the following theorem which relates to a Cube-based Lambda network. Similar results can also be derived for non cubic topologies.

<u>Theorem 2-1:</u> Let the stages of the network be labelled 0..n-1 with stage 0 associated with the input/output side of the network. Let the interconnection topology be that of the Cube i.e.

$$O\hat{}_s(\langle j_{n-1},..,j_{s+2},j_{s+1},j_s,..,j_1,j_0\rangle) \rightarrow$$
$$I_{s+1}(\langle j_{n-1},..,j_{s+2},j_0,j_s,..,j_1,j_{s+1}\rangle),\quad s=0..n-2$$

Let $W_{s,j}$ be switch j in stage s, $j \in [0..N-1]$, $s \in [0..n-1]$. Then: each switch in stage i, i=s+1..n-1 accessible from $W_{s,j}$ has the same visibility.

<u>Proof:</u> Assume that we select $O\hat{}_s(a)$, $xj \leq a < x(j+1)$, through which to steer the request. Let $\langle a_{n-1},..,a_{s+2},a_{s+1},a_s,..,a_1,a_0\rangle$ be the base-x expansion of a. From the definition of the Cube topology, the request is steered to $I_{s+1}(a')$, where $a'=\langle a_{n-1},..,a_{s+2},a_0,a_s,..,a_1,a_{s+1}\rangle$. The stage s+1 switch associated with this input link is $W_{s+1,a''}$ where $a''=\langle a_{n-1},..,a_{s+2},a_0,a_s,..,a_1\rangle$. The downgoing output links of $W_{s+1,a''}$ are thus the outputs $O_{s+1,k}$ where k takes all values $\langle a_{n-1},..,a_{s+2},a0,a_s,..,a_1,Q\rangle$, Q=0..x-1. Theselink to the $I_s(k')$ where k' takes all values $\langle a_{n-1},..,a_{s+2},Q,a_s,..,a_1,a_0\rangle$, Q=0..x-1.

Applying the rule repeatedly for each stage down to stage 0, it is easy to see that the visibility of $W_{s+1,a''}$ is the set of network outputs labelled $\langle a_{n-1},a_{n-2},..,a_{s+2},Q_{s+1},Q_s,..Q_0\rangle$, $Q_i=0..x-1$, $i=0..s+1$. Generally, the visibility of each switch of stage t, $t=s+1..n-1$, accessible from $O\hat{}_s(a)$ (of $W_{s,j}$), $xj\leq a<x(j+1)$ is given by $\langle a_{n-1},a_{n-2},..,a_{t+1},Q_t,Q_{t-1},..,Q_1,Q_0\rangle$. Since $a_{t+1}..a_{n-1}$ is the same for <u>each</u> of the upgoing outputs of $W_{s,j}$ it follows that these visibilities are the same, i.e. the choice of which upgoing output link to take is arbitrary.

[]

Thus, a sufficient routing procedure is for all requests arriving on $I\hat{}_s(j)$ <u>not</u> pivoting in stage s to be sent out on $O\hat{}_s(j)$. This is assumed in the analysis which follows.

2.2.3.2. Analysis.

In the analysis of Lambda networks we assume the same cyclical mode of operation as in 2.2.1 and 2.2.2 above. To simplify the analysis, we consider a uniform input load: at the start of a cycle, the probability that a request is presented to a network input is p for all inputs. For the time being we shall assume that the probability that a request on input i addresses output j, $i,j=0..N-1$ is the same for all j. The effects of locality of reference are considered in 2.2.3.2.1. below. Also, in order to correctly model the behaviour of Lambda networks, we assume that requests never traverse the same switch port twice. This implies that a request steered upwards from stage s to stage s+1 via port P is <u>not</u> directly routed from stage s+1 to stage s back through P. Thus, if the requests' source and destination addresses are the same, it is assumed that the network is not accessed at all.

In this analysis the principle performance measure is the normalised throughput, T_N, of the network, as above.

Figure 2-6 shows the layout of one component switch of a Lambda network of degree x. Here, the switch is shown with the input and output connections separated so that the direction of the flow of requests is from the top of the switch to the bottom.

In generating T_N we are concerned with the mean utilisation of the downward links at the bottom (i.e. input/output) stage of the network.

The mean utilisation of the downgoing output links of a switch in stage s (say $\emptyset_s$) is dependant on:-

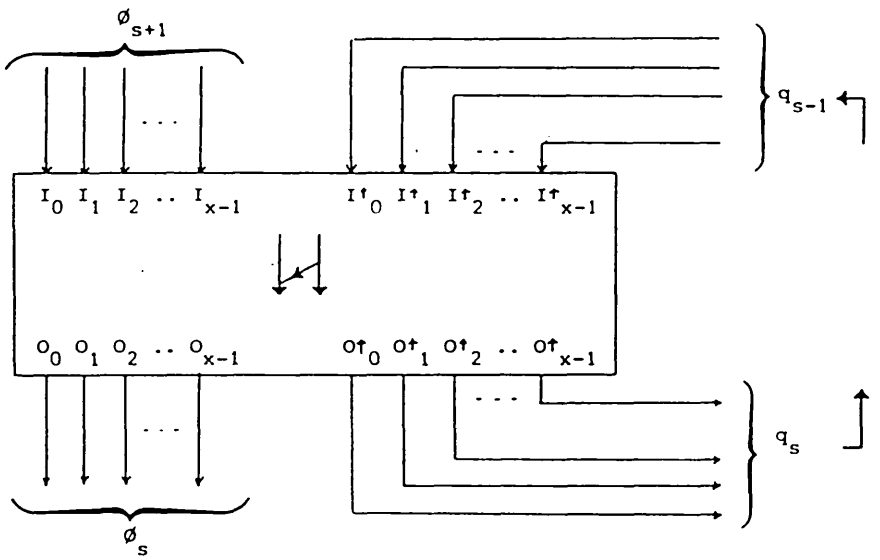


Figure 2-6: Lambda Switch Loading.

1. The utilisation of those switch inputs originating from a higher stage (stage s+1), i.e. $\emptyset_{s+1}$.
2. The utilisation of those switch inputs originating from a lower stage (stage s-1), say q_{s-1}.

In addition to the downgoing outputs of the switch (with mean utilisation $\emptyset_s$), the switch also has outputs which are fed into switches in stage s+1.

The utilisation of these links is, by our definition, q_s.

Note that requests arriving at a switch from above will never be routed back up again. Consequently, q_s is independent of $\emptyset_{s+1}$.

It is convenient to separate the q_{s-1} terms into two components, q_s and r_s so as to simplify the expression for $\emptyset_s$. r_s represents the load imposed on the downgoing links of the switch by those requests which pivot at the switch and begin traversing the network downwards. Thus,

$$q_{s-1} = q_s + r_s, \qquad s=1..n-1.$$

Consider now a switch in the bottom stage (stage 0) of the network. At the start of a cycle, the probability that the component attached to that switch generates a request is p, as defined by the model.

Now, with probability 1/N (assuming a random addressing distribution), the destination address will be equal to the source address and the request can bypass the network altogether.

Consequently, the r_0 component is:-

$$r_0 = \frac{p(x-1)}{N}$$

and the q_0 component:-

$$q_0 = p\left(1 - \frac{x}{N}\right)$$

Note that $r_0+q_0=p(1-(1/N))$ not p since we have excluded all requests whose source and destination addresses are the same.

Now, for each stage encountered going up into the network, the r loads will be multiplied x-fold since the visibility of stage s is x times that of stage s-1, s=1..n-1. Thus generally,

$$r_s = \frac{px^s(x-1)}{N} \qquad s=0..n-1$$

Similarly, we get for q_s:-

$$q_s = p(1 - \frac{x^{s+1}}{N}) \qquad s=0..n-1$$

Note that $r_s+q_s=q_{s-1}$ as can be easily verified.

One of the boundary conditions occurs at the top stage (stage n-1) of the network. Here, in a Lambda network, we expect the value of q_{n-1} to be p/2 since under the random addressing assumption, exactly half of all requests generated in one subnetwork will be ultimately routed to the other subnetwork. This is easily verified:-

$$q_{n-1} = p(\frac{1-x^n}{N})$$

$$= p(1 - \frac{x^{\log(N/2)}}{N})$$

$$= p(1 - \frac{N}{2N})$$

$$= p/2 \qquad \text{Q.E.D.}$$

Note that the q_{n-1} of one subnetwork form the $\emptyset_n$ of the other subnetwork so that:-

$$\emptyset_n = q_{n-1} = \frac{p}{2}$$

This is used as the starting point for the iteration.

From the values of r_s and $\emptyset_{s+1}$ the throughput of each stage s switch can be found. However, we cannot simply apply equation [2-4] to the problem. This is because a request which pivots in stage s (thereby contributing to the r_s component) cannot be steered back through the link from which it came. Consequently, requests arriving at the switch from below are not evenly distributed among the outputs of the switch. This is not true of requests which have pivoted somewhere higher in the network, (i.e. the $\emptyset_{s+1}$)

so we shall consider, for the time being, just the contribution to $\emptyset_s$ made by r_s.

Consider a single switch in stage s and consider only those requests which pivot about the switch and let $\emptyset_s'$ be the contribution made to $\emptyset_s$ by such requests.

If there are m pivoting requests present at the switch inputs then:

1. Pr{a pivoting request is present on the kth upgoing input} = m/x

2. $$\Pr\{\text{output port k contains request}\} = 1 - \left(\frac{x-2}{x-1}\right)^m$$

Hence:-

3. $$\Pr\{\text{output k accessed}\} = \binom{m}{x}\left(1 - \left(\frac{x-2}{x-1}\right)^{m-1}\right) + \left(1 - \frac{m}{x}\right)\left(1 - \left(\frac{x-2}{x-1}\right)^m\right)$$

$$= 1 - \left(\frac{x-2}{x-1}\right)^m \left(1 + \frac{m}{x(x-2)}\right)$$

Therefore, we get:-

$$\emptyset_s' = \Pr\{\text{a request exists at an arbitrary output}\}$$

$$= \sum_{m=0}^{x} \binom{x}{m} (r_s)^m (1-r_s)^{x-m} \left(1 - \left(\frac{x-2}{x-1}\right)^m \left(1 + \frac{m}{x(x-2)}\right)\right)$$

$$= 1 - \sum_{m=0}^{x} \binom{x}{m} (r_s)^m (1-r_s)^{x-m} \left(\frac{x-2}{x-1}\right)^m$$

$$- \sum_{m=0}^{x} \binom{x}{m} (r_s)^m (1-r_s)^{x-m} \left(\frac{x-2}{x-1}\right)^m \cdot \frac{m}{x(x-2)}$$

$$= 1 - \left(1 - \frac{r_s}{x-1}\right)^x - \frac{1}{x(x-2)} \cdot \sum_{m=0}^{x} \binom{x}{m} \left(\frac{r_s(x-2)}{x-1}\right)^m \cdot m \cdot (1-r_s)^{x-m}$$

Now, $$\binom{x}{m} m = \frac{m\,x!}{m!(x-m)!} = \frac{x(x-1)!}{(x-m)!(m-1)!} = x\binom{x-1}{m-1}$$

Substituting, we get for $\emptyset_s'$,

$$\emptyset_s' = 1 - \left(1 - \frac{r_s}{x-1}\right)^x - \frac{1}{x-2} \cdot \sum_{m=1}^{x} \binom{x-1}{m-1} \left(\frac{r_s(x-2)}{x-1}\right)^m \cdot (1-r_s)^{x-m}$$

$$= 1 - \left(1 - \frac{r_s}{x-1}\right)^{x-1}$$

This is, perhaps, intuitive since for pivoting requests, only x-1 of the switch output ports are 'candidates' for that request.

Since the requests contributing to $\emptyset_{s+1}$ are independent from those contributing to r_s, equation [2-4] can now be applied, which yields:-

$$\emptyset_s = 1 - \left(1 - \frac{\emptyset_{s+1}}{x}\right)^x \left(1 - \frac{r_s}{x-1}\right)^{x-1}$$

with the boundary cases:-

$$\emptyset_{n-1} = 1/2 \quad \text{and} \quad r_{n-1} = \frac{px^{n-1}(x-1)}{N}$$

Figure 2-7 shows the values of T_N for conventional interconnection networks and Lambda networks for varying x and N. This shows Lambda networks to have higher throughput than conventional networks, but, as with 2-channel networks, a Lambda network switch requires approximately twice as many pins as a conventional switch. Thus, the relative performance/cost curves shown in Figure 2-8(b) below favour conventional networks. The benefits of Lambda networks, in terms of both performance and performance/cost become more apparent when the inherent locality in these networks can be exploited.

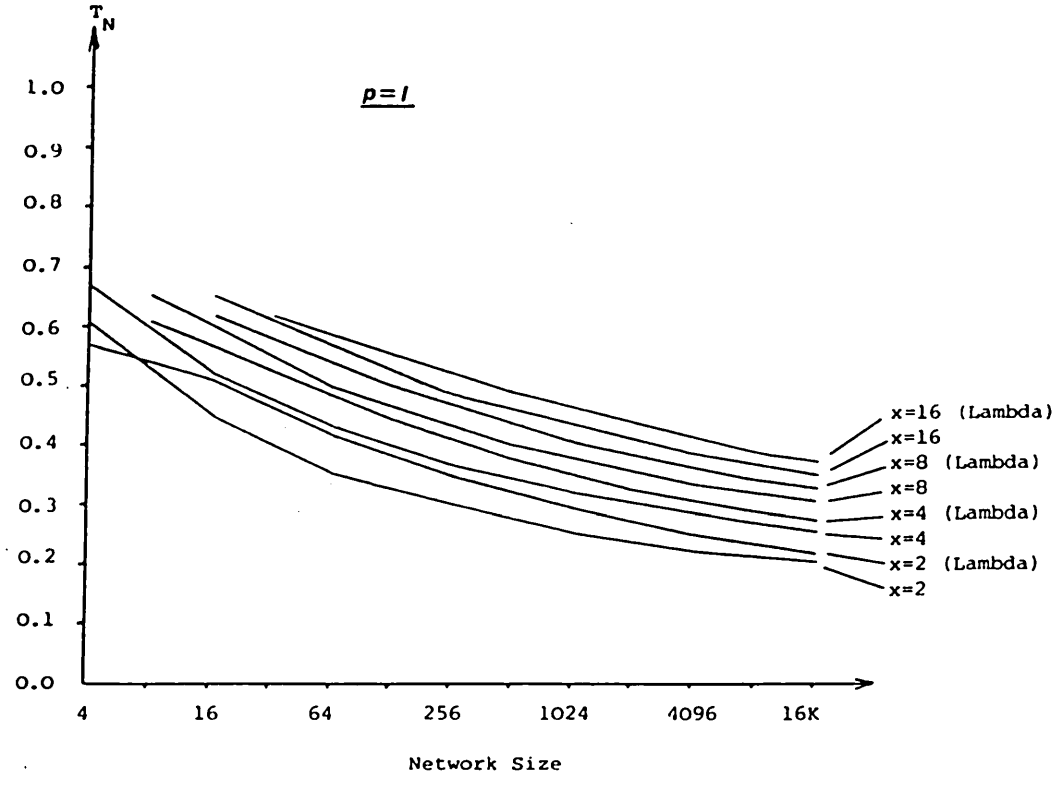


Figure 2-7: Lambda Network Performance.

2.2.3.2.1. Effects Of Locality.

Locality in a Lambda network has the effect of reducing the average number of stages traversed before pivoting occurs over the number traversed under a random addressing distribution. Thus it affects the value of r_s defined above (and hence the value of q_s) for each stage.

In order to provide a simple means of quantifying the locality in a Lambda network we assume that V_s - the probability that a request pivots in stage s - has a 'straight line' distribution of the form:-

$$V_s = P_{DL} + G.(s+1) \quad s=0..n,$$

where n is the number of stages in the network, P_{DL} is the probability of direct locality i.e. the probabilty that destination=source, V_n is the probability that a request pivots implicity (i.e. by traversing the links at the top of the network), and G is the line gradient, subject to:-

$$P_{DL} + \sum_{s=0}^{n} V_{s+1} = 1 \qquad [2\text{-}8]$$

r_s is then given by

$$r_s = pV_s$$

By rearranging equation [2-8] we have:-

$$P_{DL} = \frac{1}{n+2} - \left(\frac{G(n+1)}{2}\right)$$

G is maximal when P_{DL}->0 and minimal when P_{DL} approaches the value of V_n when G is maximal. i.e.

$$G_{max} = \frac{2}{(n+1)(n+2)} \quad \text{and} \quad G_{min} = -G_{max}.$$

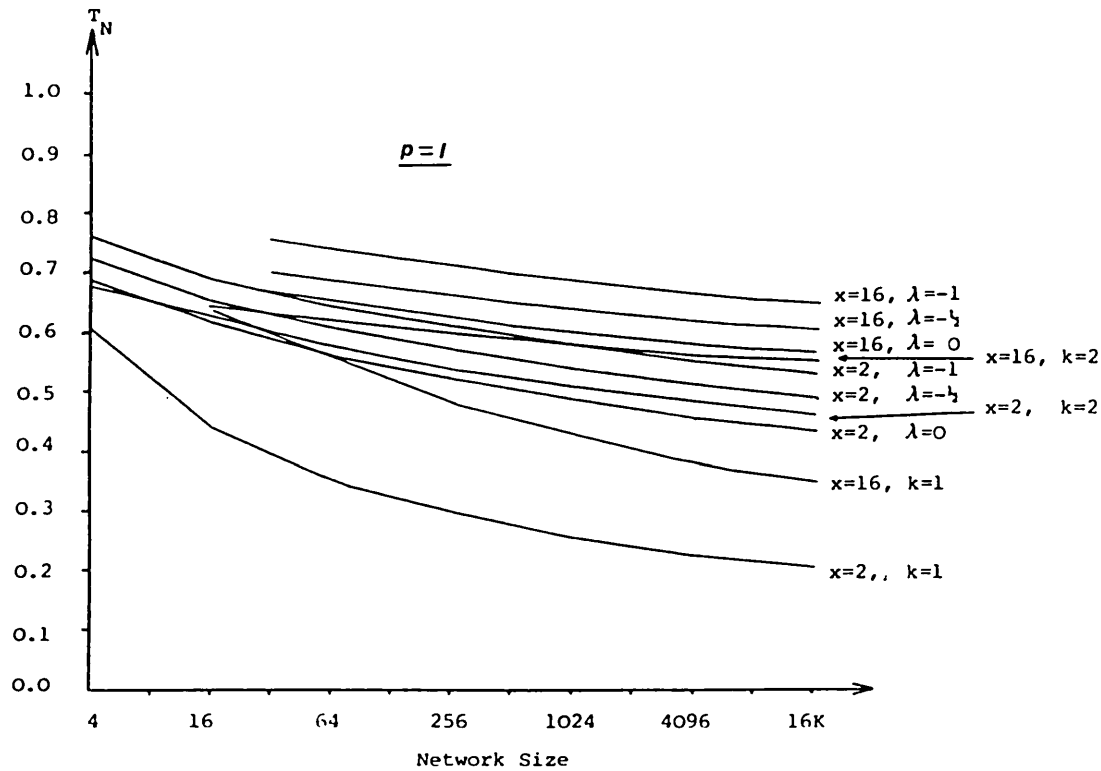
Therefore, the locality in the network can be specified by a single parameter, λ, $-1 \leq \lambda \leq 1$ from which we get:-

$$G = \lambda G_{max} = \frac{2\lambda}{(n+1)(n+2)}$$

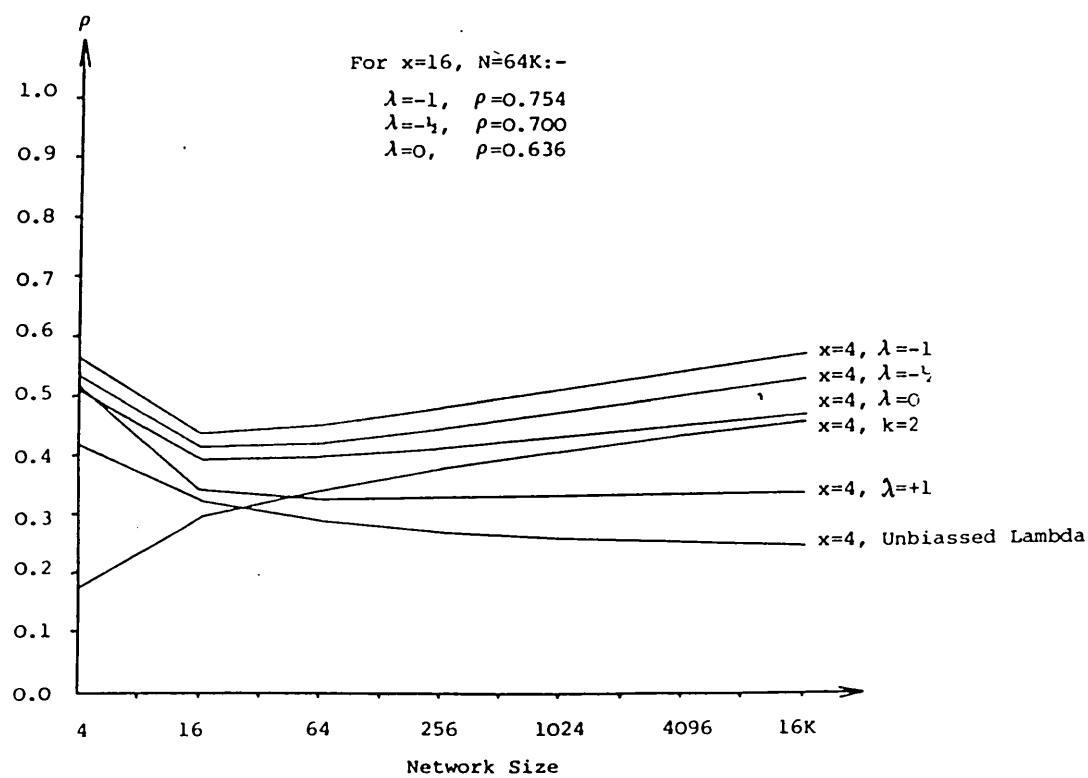
$$P_{DL} = \frac{1}{n+2} - \frac{G(n+1)}{2}$$

Thus, the more negative λ is, the greater the locality described. Note that with $\lambda = 0$, $P_{DL} = V_s = (1/n+2)$, s=0..n.

Figure 2-8(a) compares the performance of conventional networks, 2-channel networks and 'biassed' Lambda networks with varying values of λ, and Figure 2-8(b) shows the relative performance cost curves for the same configurations in addition to those of Lambda networks under a random addressing distribution. As with k-channel networks, the larger switches required to implement Lambda networks affects the cost more than the benefits of the network affect the performance. Consequently, the performance/cost curves favour conventional networks for networks of any practical size.



(a) Performance Comparison With Conventional Networks.



(b) Relative Performance Cost Curves.

Figure 2-8: Lambda Network Performance With Locality.

2.2.4 The Analysis of Asynchronous Systems

The previous sections of this chapter have been concerned with the analysis of networks which operate cyclically (and consequently synchronously). We now address the analysis of networks in which all activity (path building, data transfer and path release) in the network is asynchronous. Such networks are beneficial where the volume of information required to be sent through an established channel varies from one transaction to the next. In such systems there is no notion of a network cycle, and as a result the corresponding analysis is more complex.

We begin by describing the model:

1. We concern ourselves solely with circuit-switched networks in which a physical channel is established across the network before information is passed. Packet switching has been examined in [JuD81].
2. All network input/output ports are assumed to be connected to system components. Components attached to the input ports of the network (source components) may present requests to the network at any time. The time between the completion of one request and arrival of the next request is assumed negative-exponentially distributed with mean t_i. All source components are assumed identical in this respect.
3. Once a channel has been established between a source component and a destination component (i.e. one attached to a network output port), information is exchanged for a mean time of t_x and the channel is then released. The channel-hold time (i.e. exchange time) is again assumed to be negative-exponentially distributed.
4. Requests remain in the network until they have successfully reached their addressed destination node. A blocked request is not removed from the network.

5. The switching times are assumed to be insignificant in comparison to the channel-hold time and may therefore be ignored.

The principle performance measures of interest are the network bandwidth (in requests completed per second) and, as in the above sections, the acceptance probability, P_A.

A similar analysis has been attempted in [WLL82] although this assumed a synchronous control mechanism in which blocked requests were assumed lost.

2.2.4.1 The Analysis of an Asynchronous Crossbar Matrix.

In this section we derive exact results for both the bandwidth and acceptance probability for an asynchronous full-crossbar network. For completeness, we shall consider an asymmetric crossbar comprising x input ports and y output ports as shown schematically in Figure 2-8(a). To model the circuit-switched nature of the switch operation, we distinguish active input ports, which currently have a submitted request pending completion, from inactive ports, which are awaiting the next request arrival on that port. An inactive port is modelled as a server with service rate:

$$\lambda = 1/t_i$$

Channel communication is modelled by viewing each output port of the switch as being a server with service rate:

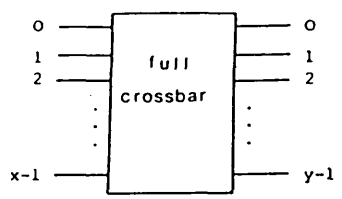
$$\mu = 1/t_x$$

Blockage in the switch is modelled by considering each server to have an associated queue of capacity x-1. Requests which attempt to obtain service from a busy server join the queue associated with the server.

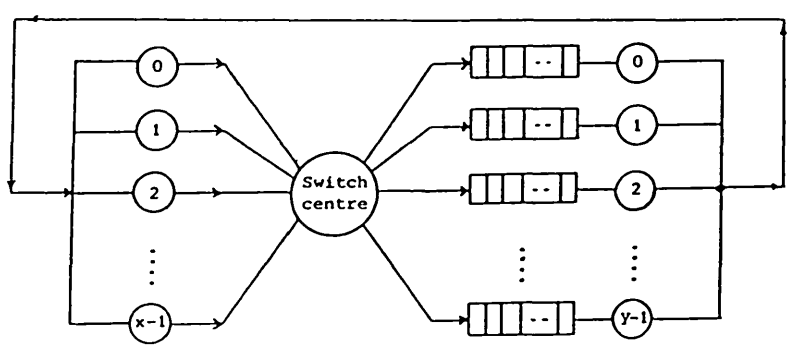
When the current request completes service, a queued request is arbitrarily

chosen and serviced. The request currently in service may not be preempted by any arriving requests. A request which completes service effectively reactivates the input port on which the request arrived. This port then becomes a candidate for further arrivals. Figure 2-9(b) shows a queuing network model of the switch.

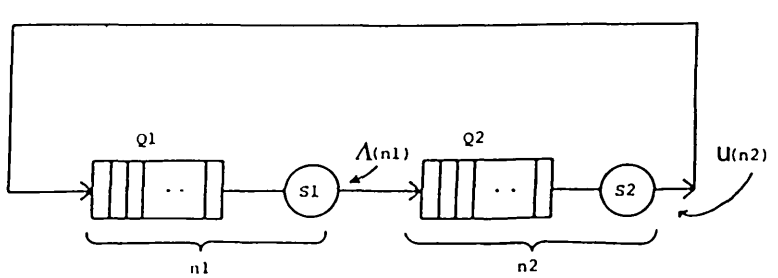
To simplify the analysis, we reduce this network to a two-stage feed-forward closed queueing network as shown in Figure 2-9(c). Arrivals to the switch are imagined as being generated by a single server (S1 in the diagram) whose inputs are those requests fed back from the output ports of the switch on completion.



(a) Switch Schematic



(b) Internal Queueing Model.



(c) Closed Feed-Forward Equivalent Of (b).

Figure 2-9: Queueing Network Model For Single Asynchronous Switch.

The state of the system at time t is a tuple $\langle n_1, n_2 \rangle$ where n_1 is the number of requests associated with S1 and its queue, Q1 (i.e. the number of inactive input ports in the switch), and n_2 is the number of requests either in service or currently queued for service (i.e. the number of active input ports in the switch). At all times n_1 and n_2 satisfy:

$$n_1 + n_2 = x$$

The arrival rate of the reduced model (i.e. the service rate of S1) is state dependent and is given by:

$$\Lambda(n) = \lambda n \qquad (0 \leq n \leq x)$$

The composite service rate, U, for S2 is rather more difficult to obtain since it must include the way in which the (n_2) requests are distributed among the servers in Figure 2-9(b). We state without proof a result given by Harrison [Har83]:

$$U(n) = \frac{ny\mu}{n+y-1} \qquad (0 \leq n \leq x)$$

Let $P(n_1,n_2)$ be the probability that the system is in the state $\langle n_1, n_2 \rangle$. The balance equation for this system is given by:

$$P(n_1,n_2)[\Lambda(n_1) + U(n_2)] = P(n_1+1,n_2-1)\,\Lambda(n_1+1) + P(n_1-1,n_2+1)U(n_2+1)$$

We use the well-known result given in [Kob78] namely that at equilibrium the steady state probability, $\pi(n_1,n_2)$ of being in state $\langle n_1, n_2 \rangle$ is:

$$\pi(n_1, n_2) = \frac{1}{G}\left(\prod_{m=1}^{n_1} \frac{1}{\Lambda(m)}\right)\left(\prod_{n=1}^{n_2} \frac{1}{U(n)}\right)$$

G is a normalising constant which ensures that the probabilities sum to unity, i.e.:

$$G = \sum_{\substack{\text{all } \langle n_1, n_2 \rangle \\ n_1+n_2=x}} \left(\prod_{m=1}^{n_1} \frac{1}{\Lambda(m)} \right) \left(\prod_{n=1}^{n_2} \frac{1}{U(n)} \right)$$

Now:

$$\prod_{m=1}^{n_1} \frac{1}{\Lambda(m)} = \prod_{m=1}^{n_1} \frac{1}{\lambda m} = \left(\frac{1}{\lambda} \right)^{n_1} \cdot \frac{1}{n_1!}$$

and:

$$\prod_{n=1}^{n_2} \frac{1}{U(n)} = \prod_{n=1}^{n_2} \frac{n+y-1}{ny\mu} = \left(\frac{1}{y\mu} \right)^{n_2} \binom{n_2+y-1}{y-1}$$

Thus:

$$\pi(n_1, n_2) = \frac{\left(\frac{1}{\lambda} \right)^{n_1} \frac{1}{n_1!} \left(\frac{1}{y\mu} \right)^{n_2} \binom{n_2+y-1}{y-1}}{\sum_{i=0}^{x} \left(\frac{1}{\lambda} \right)^{i} \left(\frac{1}{y\mu} \right)^{x-i} \frac{1}{i!} \binom{x-i+y-1}{y-1}}$$

$$= \frac{(n_2+y-1)!}{n_1!\, n_2! \sum_{i=0}^{x} \frac{(x-i+y-1)!}{i!\,(x-i)!} \left(\frac{1}{\lambda} \right)^{i-n_1} \left(\frac{1}{y\mu} \right)^{x-n_2-i}} \qquad [2\text{-}9]$$

(Note that the balance equation is valid for any work preserving queueing discipline.)

Now that we have the steady-state probabilities, we can derive equations for the switch bandwidth and acceptance probability.

The switch bandwidth is simply the average number of requests serviced at the second server of Figure 2-9(c) per unit time, and is given by:

$$BW = \sum_{n=0}^{x} \pi(x-n,n) \cdot U(n)$$

Expanding the π and U terms, this becomes:

$$BW = y\mu \sum_{n=1}^{x} \frac{n_2+y-1!}{(x-n)!\ (n-1)!\ \sum_{i=0}^{x} \frac{(x+y-i-1)!}{i!\ (x-i)!} \left(\frac{y\mu}{\lambda}\right)^{n+i-x}}$$

To find the equation for acceptance probability, we must use both the original and the reduced models of Figure 2-9. Equation [2-9] gives us the equilibrium probability of being in state $\langle n_1,n_2\rangle$. The arrival rate in this state is given by $\Lambda(n_1)$. The acceptance probability is the probability that an arriving request immediately goes into service at the switch outputs. Given that we are in the state (n_1,n_2), let $\gamma(n_2)$ be the probability that an arriving request immediately selects an idle server.

Let q_i be a random variable denoting the number of requests at switch output i, i=0..y-1.

Without loss of generality, assume that an arriving request, R, selects switch output 0 and let $\underline{N}=\langle N_1,N_2\rangle$ be the random variable denoting the equilibrium state vector with R removed.

The acceptance probability can then be expressed as:-

$$P_A = Pr\{q_0=0\}$$

$$= \sum_{n=0}^{x-1} Pr\{q_0=0 \mid \mathbf{M}=\langle x-n-1,n\rangle\}.Pr\{\mathbf{M}=\langle x-n-1,n\rangle\}$$

$$= \sum_{n=0}^{x-1} \mathcal{V}(n).\pi(x-1-n,n)$$

from [LaR80].

Now consider just the output side of the switch and the case where there are n<x requests currently either in service or queueing for service. The state of this subsystem may be viewed as a vector:

$$\mathbf{r}(n) = \langle r_0, r_1, .., r_{y-1}\rangle$$

where r_i represents the total number of requests at the server and in the queue of service station i, i=0..y-1. Clearly:

$$\sum_{i=0}^{y-1} r_i = n$$

Now, since each request passed through the switching centre of Figure 2-9(b) is routed to service station i with probability 1/y for all i=0..y-1, the visitation rate [Kob78] is the same for all servers. Since each server has the same service rate, μ, the ratio of visitation rate to service rate is also the same for all servers. Thus the extended states:

$$\langle x-n, r_0, r_1, .., r_{y-1}\rangle, \qquad n=0..x$$

(which includes the number of requests present at the inputs of the switch) each occur with equal probability.

The equation for $\mathcal{V}(n)$ in equation [2-10] consequently reduces to a 'ball-in-bag' problem, namely:

$\nu(n)$ = Pr {a server is idle given there are n requests already at the outputs of the switch}

$$= \frac{\text{Number of ways of arranging n balls in y-1 bags}}{\text{Number of ways of arranging n balls in y bags}}$$

$$= \frac{\binom{y+n-2}{y-2}}{\binom{y+n-1}{y-1}} = \frac{y-1}{y+n-1}$$

Observe, now, an interesting result when the switch is saturated, that is when $\lambda \to \infty$. Here, requests which complete service are immediately resubmitted to the switch, so that $\pi(0,x)$ approaches unity. In the limiting case, we have:

$$P_A \to \nu(x-1) = \frac{y-1}{x+y-2}$$

Consequently, for square switches, in which x=y:

$$P_A \to 1/2 \text{ as } \lambda \to \infty$$

Note that this is independent of the switch size!

It is interesting to compare this result with the same figure for cyclical systems. In the latter, P_A is not the same for all sizes of switch but does have an asymptotic limit given by:

$$\lim_{x \to \infty} 1 - \left(1 - \frac{1}{x}\right)^x$$

$$\to 1 - \frac{1}{e} = 0.632 \qquad \text{[JuD81]}$$

In Figure 2-10 the network bandwidth is expressed as the completion rate per switch port and is normalised with respect to the service rate, μ. The absolute switch bandwidth is obtained by multiplying the normalised bandwidth (BW_N for a given value of N and λ/μ) by $x\mu$. Figure 2-10 also shows the behaviour of P_A with varying N and $\lambda:\mu$ ratios.

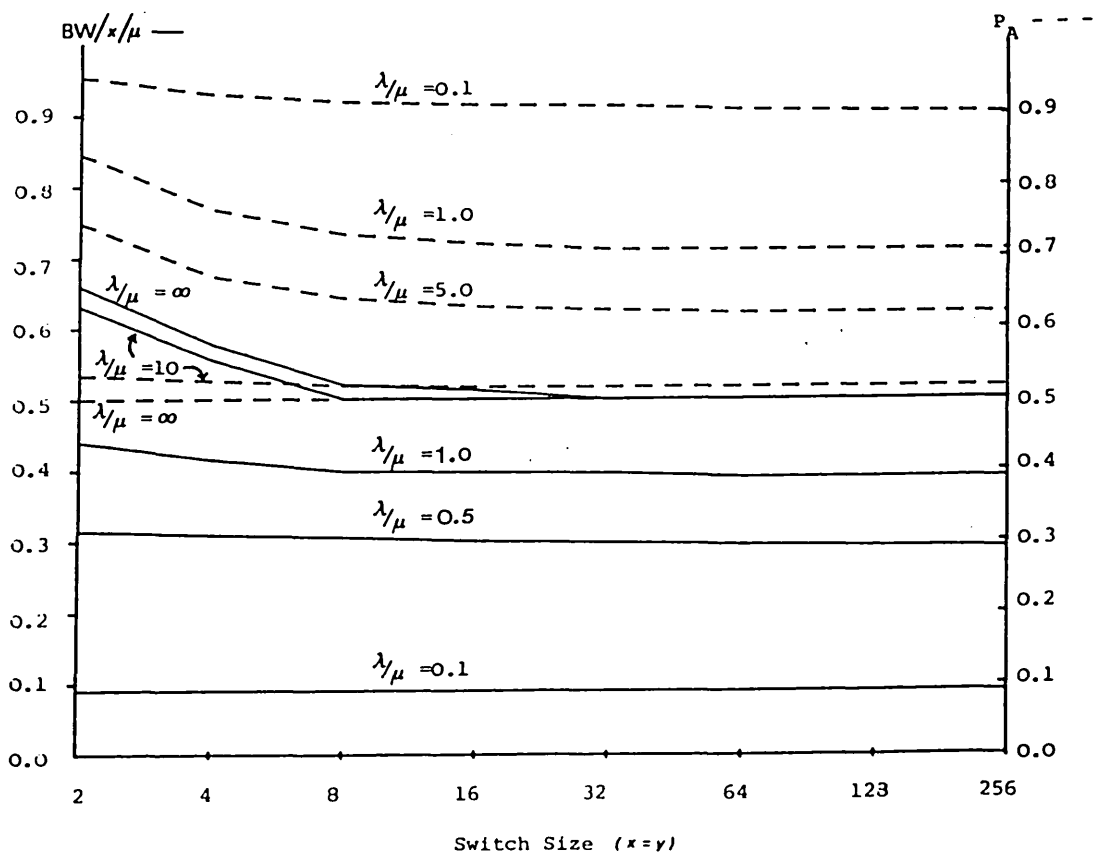


Figure 2-10: Asynchronous Full-Crossbar Performance.

2.2.4.2 Towards Asynchronous Multistage Network Analysis

The above analysis provides exact results for the bandwidth and acceptance probability of a single asynchronous switch. However, extending the model to cover multistage networks presents a problem which, to date, we have been unable to solve. Consider, for example, a two-stage network. To

obtain the network bandwidth and acceptance probability we can reduce the network to a simple two-stage queueing arrangement with each stage corresponding to one stage of the network. Each of these queuing stages can then be modelled by a two-stage feed-forward queuing network as was done above. This reduces the size of the state space, and the balance equations can be easily written down.

Provided we know the arrival rate of requests to the second server then we can apply the results of 2.2.4.1. to obtain the overall network throughput and acceptance probability. This arrival rate, however, depends on the departure rate from the first server, which, in turn, is dependent on the effective service rate of the second server (as opposed to the devices connected to the outputs of the second server), which we don't know.

So, we end up with an unknown value in the system of equations, namely the rate of traffic flow between the two stages. This problem occurs in all multistage networks and this has prevented the analysis of such networks. Although there appears to be no obvious solution to this problem, further research may yield, at least, some reasonable approximations. We leave this as an open question.

CHAPTER 3

Self-Clocking Networks

3.1. Network Design Techniques.

Figure 3-1 shows a schematic diagram of a switching element, taken from [Pat79]. This design has been widely adopted and forms the basis of all currently proposed network designs although a number of extensions to the control plane have been proposed in [LiW82, MAS81, WLL82]. A network of such devices operates as follows:-

At the start of a network cycle, all source components wishing to submit a request to the network place their respective destination addresses on the data busses of the switching elements to which they are connected at the top stage of the network. (Ordinarily, each network input port is associated with only one source component.) The request lines (REQ in the diagram) are then raised on the switch control inputs. All requests currently present at the inputs to the network then 'ripple' through the network until either they reach their addressed outputs successfully or they become blocked due to competition for a common switch output link somewhere in the network. Each switch uses one bit from its data bus (which holds the network routing address) to specify the switch output port required by the request. If the request can be successfully steered to this output port, then both the control and data lines are switched through accordingly. The REQ line then propogates through to the next switch (attached to that output port), indicating to that switch the presence of a request on its corresponding input. Since the data bus is also switched through to this switch the network routing address is also made available to it. This switch then undergoes the same operations, this time using a different bit in the data path to select the device. If, on the other hand, the request is blocked i.e. if the switch output port it wishes to

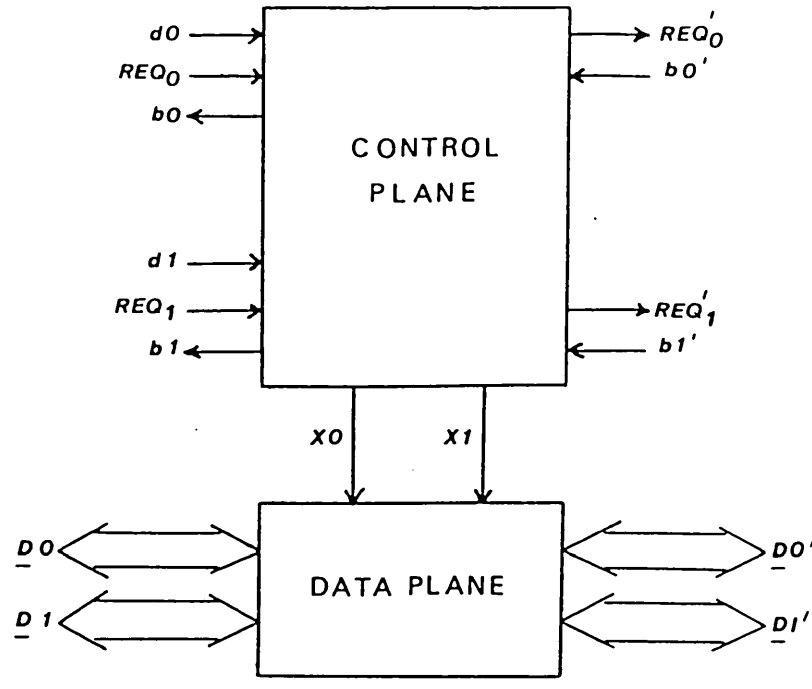


Figure 3-1: A 2x2 Switching Element Schematic.

use is currently busy then the 'request blocked' (b) line (which is ordinarily LOW) is raised. This signal propogates back to the source node. After a time, dependent on the size of the network, the request will reach its designated output port (if, of course, it has not been blocked during the addressing process).

Ordinarily, the network operates synchronously, that is, all requests are presented to the network at the same time. The model has been extended slightly, so that network addressing and data transfer is controlled by alternate cycles of a two-phase clock [WLL82]. The 4x4 switch design described in [MAS81] is architecturally similar except that at each stage two bits from the data plane are used to select a device.

The advantages of this approach are:-

1. Because the entire network routing address is available on the data bus during path set up, the switching time is very fast and the logic required to select a switch is very simple. Nearly all the complexity

lies in the bus gating logic.

2. Because the data path is wide (limited by pin constraints), the time taken to transfer data through an established channel is small.

Consequently, this design can lead to very high performance networks. However, there are serious drawbacks to this approach which become more significant as the size of the network is increased:

1. As a consequence of employing wide data paths to carry the routing address and through-channel data, the size of switch which can be constructed from a given size of package is limited. This means that very large numbers of switching elements are required to implement a network of any significant size. Multiple data planes which share the same control signals may be employed, but this only makes the problem worse; If k data planes are used then the network cost increases by approximately O(k).
2. Because the number of switches and, more significantly, the number of interconnections is so high, the resulting network is inherently less reliable. (Reliability is discussed in Chapter 4.)
3. As the network gets larger and the number of stages increases, the problem of skewing in the data path becomes a significant problem. This is particularly true of multiple-plane networks where the data path is distributed over many independent ICs. This inherently limits the rate at which data can be transferred through an established channel.
4. Because the network is synchronous, a global control clock must be provided for control. Some implementations require individual switches to be strobed, for example [WLL82]. As the network is made larger, clock distribution becomes a significant problem.
5. Because the network cycle time is dependent on the size of the network, an extension to the network will require an adjustment to the

source timing logic.

So, whilst this approach may be suitable for small scale, static, SIMD and MIMD machines, we argue that their use in the very large-scale, extensible, multiprocessors which are of interest here is limited.

3.1.1. Serial Switching

We observe from the description of the switch of Figure 3-1 that:-

1. During path set up, the data bus serves only to 'hold' the network routing address for the currently selected device.
2. During data transfer, the switch select address and 'blocked status' lines in the control path of each switch are idle.

We now introduce the concept of serial switching which is a technique aimed at making more efficient use of the available pins on a switch package. In a serially-switched network, there is no explicit data path. Instead, the data path and control path are unified so that they both share the same pins on a single package; the width of the data path is inherently reduced as a result and there is only one network plane. During path set up, the control lines control network addressing and path building; during data transfer, some of the control lines change roles and become data transfer lines. The result is that for a given size of package, we are able to imbed a significantly larger switch than is possible with conventional switch design techniques, and each pin in the control/data path is fully utilised both during path set up and during data transfer.

The immediate consequence of serial switching is a reduction in network cost: fewer stages and fewer switches per stage are required to implement a network of a given size. Network contention is reduced, and, perhaps most significantly, the network is made inherently more reliable, requiring fewer components and far fewer interconnections than existing implementations.

Although reducing the width of the data path inherently reduces the switching speed and the through-channel data rate, this is offset by the fact that there is now less contention in the network. For a given size of package, reducing the width of the control/data path by a factor of k means that the size of switch which can be accomodated in that package is increased by a factor of approximately k. For a given size, N, of network, this reduces the network cost by a factor of approximately:-

$$\frac{k\log_x N}{\log_{kx} N}$$

which is >k. In addition to the reduced contention in the network the problem of relative signal skew is also reduced (in the limiting case of bit serial control and communication the skew problems are eliminated altogether). Thus, serially switched networks offer the potential for improved cost/performance when compared to conventional implementations.

3.1.2. Self-Clocking Networks

Let us now take a closer look at the implications and problems associated with the serial switched approach:

The ability to construct large switches has obvious benefits in terms of network cost, contention and reliability, but it offers less flexibility in the choice of network size. A regular network of degree x may only be of size x^n, n=0,1,2.. If intermediate sizes are required then irregular or 'hybrid' networks must be made possible.

In a serially-switched network, because there is no data bus to hold the network routing address during path set up, the individual switch select addresses($\log_2 x$ bits for a switch of degree x) must be loaded from the source on demand. (The network routing address could be passed from switch

to switch but this is unnecessary and time consuming and requires buffering logic to be provided in the switches.) If the network has an irregular topology, then clearly each address fetch may require a different number of select address bits to transmitted by the source. Somehow the source must 'know' how many bits are required by each fetch operation. This presents an interesting problem: the source node cannot determine how many bits are required on each address demand unless it has a-priori knowledge of the topology of the network. If the stages of the network are not of a fixed degree then the source must further take into consideration where the request is being sent to since it may encounter different sized switches in transmitting to different destinations. Consequently, the source node (or its interface to the network) would have to be very complex, performing synchronisation on each demand, predetermining the address pattern to be sent to the network based upon the network topology and destination node, and further transmitting the right number of bits to each switch for it to be selected. The partitioning of the routing address would most likely have to be done in software which introduces a time overhead in addition to the hardware overheads required for the control of the network.

The proposed solution to these problems lies in a modest extension, not to the source nodes or their interfaces, but to the component switches of the network. A Self-Clocking network exploits the fact that successive select addresses for successive stages of a network are held contiguously in the network routing address regarldess of exactly how many bits are required by each switch on a given path. In a self-clocking switching element, a 'demand' for a select address consists of a pulse train or burst clock equal in length to the number of address bits required to select that device. By maintaining the routing address in a shift register (or similar device) at the source, this pulse train can be coupled to the shift register clock input so that the required address component is automatically extracted from the register. This will also align the

remainder of the routing address at the 'front' of the register for subsequent switches to clock out. Observe that:-

1. No synchronisation or control clock is required at the source. The network itself provides the clocks required to control the transmission of address bits from the source register.
2. The scheme functions independently of the size (which is bounded by the length of the source address register) and topological configuration of the network. Each switch 'knows' its own size and can therefore provide the correct number of burst clock pulses.
3. In theory, only two wires are required to perform the address fetch operation: one to transmit the pulse train back to the source, and one to carry the select address back to the switch. After path set up, the burst clock and select address transmission lines may be coupled to the source and destination components to form a bidirectional (full duplex) serial communication channel.
4. The resulting network is naturally asynchronous. No global control clock is required, so the problems of exact phase and frequency matching of the control clocks does not arise. Note that the asynchrony is achieved with very modest overhead. The burst clock is simple to implement and overcomes many of the problems associated with asynchronous control.

In the next section, we clarify these ideas by describing an implementation of a network based upon these design techniques. Note that serial switching and self clocking design techniques may be readily applied to k-channel and Lambda networks as described in Chapter 2. The description which follows, however, assumes a conventional, single channel network implementation since this configuration yields the minimal pin count per package and results in the best performance/cost figures when compared to the other two variations.

3.2. The design of the XS1 network switching device

The XS1 was originally designed as a building block for a network used for device interconnection in the prototype ALICE reduction machine [DaR81]. A schematic diagram of the machine architecture is shown in Figure 3-2. The principle system components are reduction (processing) agents and intelligent storage segments. Load sharing is done via a distribution network which is ring based; agents and storage segments communicate via a switching network which is composed of interconnected XS1 devices.

The characteristics of the network are summarised below:

1. The network employs serial switching and self-clocking protocols as were described above.
2. The network is circuit switched, i.e. physical channels are established through the network between the source of a request and its designated destination. This enables arbitrary volumes of data to be transferred through the network and also facilitates bidirectional (full duplex) communication. The return channel can also be used for concurrent error detection. The mechanisms for achieving this are described in Chapter 4.
3. The network is asynchronous. There is no global clock to control network operation. Source components may submit requests to the network at any time convenient to them. Individual switches may consequently receive requests on their input ports at any time. Each switch in the network may, if necessary, operate at a different speed.
4. The network is arbitrarily expandable. Since all switches in the network function asynchronously, there are no timing or synchronisation problems associated with the expansion process. The prototype ALICE implementation is based around a 64 port network although systems of size 4, 16, 64, 256 etc. are easily obtained. Note that XS1 is a 4x4 switching element.

5. The resulting network design philosophy yields inherent structure independence. This basically enables the system and network components to function without knowledge of the topological structure or size of the network. This is explained in more detail in section 3.2.4.

3.2.1 Description of the XS1

The XS1 is a custom-designed self-clocking network switching device of degree 4 fabricated in Emitter-Coupled-Logic (ECL) technology. The device has no explicit data path, i.e. it is a serially switched device. The XS1 forms a building block to enable arbitrary sized interconnection networks to be constructed.

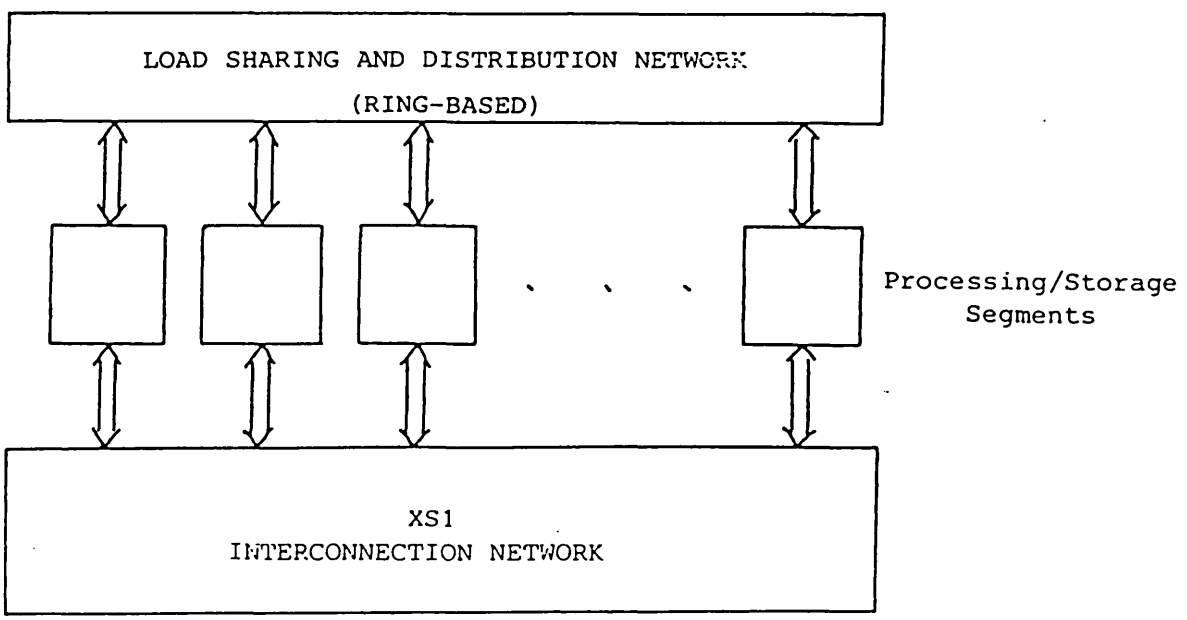


Figure 3-2: The ALICE Machine Schematic.

The XS1 has four upper (U) ports and four lower (L) ports, and functions to enable a 1:1 coupling to be achieved between any arbitrary pair of U and L ports. At most one U port may be connected to a given L port at any time. The device is logically divided into four slices labelled 0..3. Each slice has associated with it one U port and one L port. Internally, each slice is connected to all other slices in the device, i.e. U_i is not necessarily always coupled to L_i. Figure 3-3(a) shows a schematic diagram of the XS1 and Figure 3-3(b) shows the control lines associated with each U and L port. These consist of five wires labelled A, B, C, D and R. C, D and R are forward-directed (i.e. from U to L); A and B run in the opposite direction. In Figure 3-3, these control lines are tagged with I and O, for 'input' and 'output', according to whether the line forms an input line or an output line to the slice. These five wires are now described.

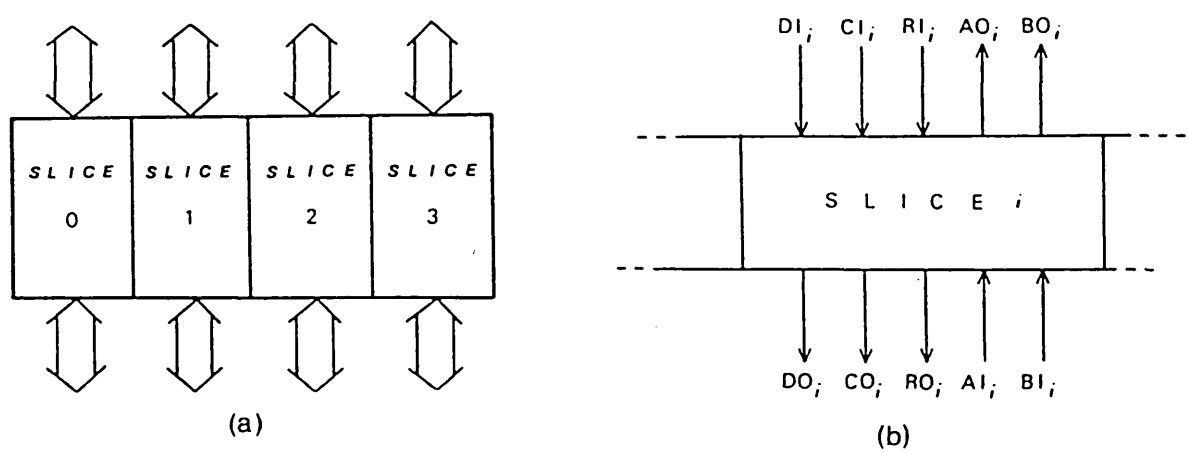


Figure 3-3: XS1 Schematic

3.2.1.1. The D (Data) and C (Clock) lines

The D and C lines are path-matched data and clock lines which serve two purposes. During path building these wires form part of the control path of the XS1. The D wire carries the address bits required to select the device. Since the XS1 is of degree 4, this select address always consists

of two bits. However, to simplify the on-chip logic, a start bit is also transmitted with each select address. Thus each time an XS1 is selected, a three-bit packet is issued to the appropriate port of the device.

The C wire carries a clock signal which is one half cycle out of phase with respect to the signals on the D wire. This enables the select address bits to be clocked into the XS1 without the need for a data-synchroniser on chip. This again simplifies the internal logic of the device.

Once a path has been established through the network the D and C lines form a two-line matched-path communication channel between the source node and the addressed destination node. Because the wires are delay-matched, they can be used to transmit 'clocked' data through the network. Exactly how these wires are used is dependent on the implementation.

3.2.1.2 The R (Reset) Line

The reset line is used in much the same way as the request line in conventional switch design, the only difference being that it is active LOW. The RI input of an inactive U port is always HIGH, holding the port in its reset state. The arrival of a request on that port is signalled by a transition in the RI line to its set state (i.e. LOW); that port is then enabled to receive select address bits on D and C. RI is held in this state for as long as the source node requires the channel through the network. The U port is released by setting the RI line back to its reset state. The associated U port then remains inactive until RI is set LOW by some subsequent request. Note that at any time during the path building or data transfer phases, the RI line may be set HIGH. That is, the XS1 allows the asynchronous cancellation of requests.

3.2.1.3. The A (Address Valid) line

As with the D and C lines, the A line has a dual role. During path

building, the A line acts as an addressed value (or acknowledge) line which is set HIGH when a select address has been clocked into the XS1 through the D and C wires, and LOW when a path through the device has been successfully established. The two select address bits transmitted to the XS1 on the D wire denote the address of the L port through which the associated request is to be steered. The A line is set LOW as soon as this L port becomes free. In theory, this protocol alone (which we refer to as the A-protocol) is sufficient for network control.

Once a path has been established through the network, the A wire provides a return channel from the addressed destination node back to the source node.

3.2.1.4. The B (Burst) line

The B line provides the XS1 with the self-clocking properties which were described earlier. When a U port is inactive, the associated BO line is held in its passive state, which is LOW. When a request arrives on the U port, the corresponding RI line transits from HIGH to LOW as already described. This transition causes the XS1 to administer a three pulse burst signal back to the source on the B wire. If the network routing address is held in a shift register at source, then by coupling the BO wire (at the topmost stage of the network) to the clock input of this shift register, the required select address bits are automatically clocked out of the register as and when the currently selected XS1 device is ready to accept them. This protocol is referred to as the B-protocol. Note that a three pulse signal is issued to enable the required start bit to be clocked out as well. This assumes of course that the start bit(s) have already been introduced into the network routing address. This is clearly awkward to maintain, particularly if subsequent implementations allow hybrid topologies to be used and arbitrarily changed according to the needs of the system. So, the A and B protocols have been designed to allow 'automatic' start bit insertion which can be achieved by using both the A and B

protocols in conjunction. A simple extension to the source/network interface enables the first pulse of each pulse train appearing on the B wire to be 'picked off' and used to generate the required start bit. This combined protocol (which we refer to as the C-protocol) is described more fully section 3.2.5 on interface design.

To provide complete symmetry in the device, the A and B lines are also delay-matched through the XS1. Consequently, once a path has been established through the network, the A and B lines form an exact mirror of the D and C lines, except that they run in the opposite direction, i.e. from the destination to the source. Their use is, again, dependent on the implementation.

By overlaying the data and control paths in this way we obtain optimal utilisation of the available pins on the package. The XS1 is housed in a single 48 pin DIL ceramic package.

3.2.2. The XS1 routing cycle

A regular interconnection network can be produced by interconnecting the U and L ports of a number of XS1 devices according to one of the topological rules described in Chapter 2. To obtain a single channel through such a network from a source node (SCE), to some destination node (DST), exactly one switch in each stage of the network must be selected. The XS1 routing cycle describes the sequence of events which take place in order for a coupling to be made between a U port and an L port of each switch on the path between SCE and DST.

Consider now set-up depicted in Figure 3-4. Here, the network contains three stages, S1, S2 and S3, labelled sequentially from top to bottom. The three switches which occur on the path between SCE and DST are labelled X1, X2 and X3 as shown.

Consider now the routing cycle of X2 in the diagram. Suppose that device X1 has just been selected and that the request from SCE has just arrived at the inputs to X2. Assume that the U port on which the request arrives is in slice i of X2, and that U_i is to be coupled to L_j for some j. Initially, the lines of port i are in their 'passive' states i.e. RI_i=HIGH (U_i in reset state); DI_i=HIGH (no select addresses have yet reached X2); CI_i=HIGH; BO_i=LOW; AO_i=LOW (U_i is ready to receive a request).[1]

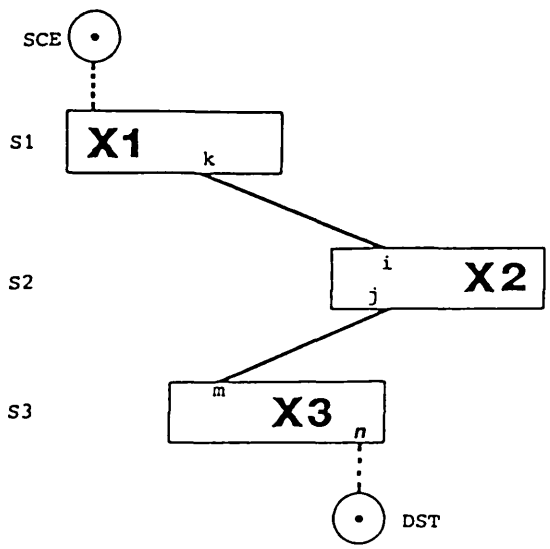


Figure 3-4: Path Building

The routing cycle of X2 then proceeds as follows (Note that the discussion assumes the use of the full self-clocking (C) protocol for network control.):-

1. RI_i->LOW. At this point X1 has become transparent and the output signals on L_k in the diagram are those present at SCE.
2. The transition in RI_i enables U_i. The burst signal, BO_i is then generated by the device. At SCE, the first pulse in BO_i is used to generate a start bit (which is a 0); the remaining two pulses clock

[1] (The i subscript signifies that the control line is in slice i.)

select address bits from a shift register at SCE.

3. The burst clock is reflected back into the network on the C line at the source together with the start bit, and select address bits from the shift register on the D line. The leading edge of the burst pulses clock out select address bits from SCE, while the falling edge of the reflected burst pulses clock the address bits into X2.

4. The three bits arriving on DI_i of X2 are clocked into a three bit internal register. The address valid (A) line is raised when the start bit hits the end of the register.

5. U_i now attempts to 'claim' L_j (the address, j, is held in the internal register). A global engaged status line E_j indicates the current status of L_j. If L_j is currently engaged, i.e. if there exists a request on some other U port of X2 which is currently coupled to L_j, then the request on U_i is blocked. When L_j is released, U_i may contend for it with other requests waiting on the release of L_j. The XS1 employs a dynamic-priority, starvation-free arbitration mechanism which guarantees a maximum upper bound on the blockage time for a given request if the request is held in the 'wait' state by SCE. This is described below.

6. As soon as U_i successfully claims L_j, E_j is toggled marking L_j as 'busy'. All lines on U_i are then coupled to the corresponding lines on L_j, i.e. RI_i->RO_j, DI_i->DO_j, CI_i->CO_j, AI_j->AO_i and BI_j->BO_i. Coupling AO_j to AO_i causes a transition to be observed in the AO line at source. This prepares SCE to generate a start bit on the first pulse of the next burst clock (in this case from X3). Coupling RI_i to RO_j causes a transition in the RI_m input to X3. X3 then undergoes a similar routing cycle to that of X2. Note that X2 is now transparent.

3.2.2.1. The End-Of-Path Protocol

Once X3 has been selected, the RO_n output line in the diagram (which is linked directly to DST) makes a negative transition from HIGH to LOW. This signals to DST the presence of a request at its associated network output port. DST then responds by issuing a single pulse on the corresponding B wire, whilst holding the A line LOW. This is observed back at SCE and causes an interrupt to be issued (possibly to the source processor). SCE and DST are now directly connected via the C and D lines in the forward direction, and the A and B lines in the reverse direction. The network is now transparent to both SCE and DST, and they may communicate freely in both directions across the network.

The path through the network is held for as long as is required to complete the conversation between SCE and DST. When this conversation has ceased, SCE releases the path by setting the RI line HIGH back at X1. This immediately disengages all the U/L couplings made by the request. L_k, L_j and L_n of X1, X2 and X3 respectively may then be 'claimed' by other requests awaiting their release. SCE is then free to submit a further request to the network and DST is free to receive further requests from the network.

3.2.3. XS1 Logic

One of the major problems associated with the operation of the XS1 is arbitration. Because the four U ports in the device operate asynchronously there is the problem of two requests, on different U ports, both trying to claim the same L port simultaneously. This is resolved by means of a system of arbitration clocks which control the activities of each slice. A single externally generated high speed clock, $\emptyset_x$, is driven on chip and separated into four internal clocks, $\emptyset_0..\emptyset_3$. These clocks are arranged so that $\emptyset_i$, i=0..3, is HIGH on only one in every four cycles of $\emptyset_x$. Furthermore, the clocks are phase shifted so that no two clock signals are

ever HIGH at the same time. This set up can be viewed abstractly as a single 'token' which is passed cyclically around the four slices in turn, i.e. from slice 0 to slice 1 to slide 2 to slice 3 and then back to slice 0 again. The arbitration protocol is simple: All incoming requests are synchronised with respect to $\emptyset_x$ and the U port of slice i, U_i, i=0..3, may only claim and release L ports when it has the token, that is, when $\emptyset_i$ is HIGH. Both claiming and releasing of L ports is done on the rising edge of $\emptyset_i$ which has two important consequences:

1. Since the $\emptyset_i$ are distinct it is not possible for two U ports to claim the same L port simultaneously.
2. The L port release timing guarantees that arbitration is starvation-free. If U_i releases L_j it can only do so if it has the token. When L_j has been released, the token is passed on to the neighbouring slice of slice i in the chain. U_i cannot now perform another claim operation until it has the token again. Clearly, if other U ports wish to claim L_j, they are all guaranteed to be given the chance since the token must flow through them before returning to slice i.

The arbitration is said to have 'dynamic priority' since an arriving request can 'jump the queue' for a busy, and already demanded, L port if the U port on which it arrives happens to be 'closer' in the token chain to the current owner of L_j than any existing blocked requests for L_j. This arbiter involves minimal on-chip logic complexity whilst still ensuring stravation-free characteristics. Simulations have shown that the resulting network throughput is indistinguishable from that which would be experienced as a result of employing a more complex first-come-first-served arbitration scheme. This is predictable since the arbiter is work preserving in the queue-theoretic sense.

All incoming requests on a U port must be synchronised with respect to these arbitration clocks. The arbitration clock generator circuit is shown in Figure 3-5.

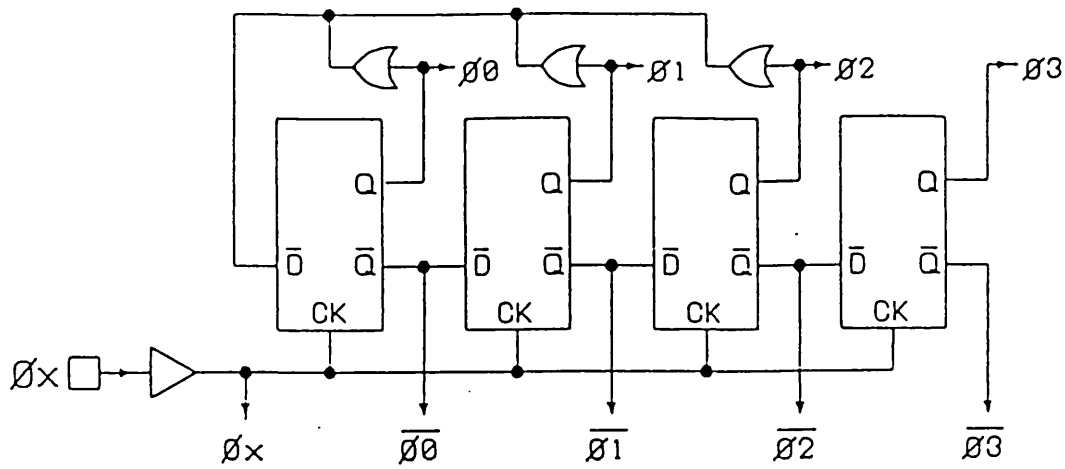


Figure 3-5: The Arbitration Clock Generator

A schematic diagram of one of the four slices of the XS1 is shown in Figure3-6. A slice contains four types of component each of whos' operation is now described:-

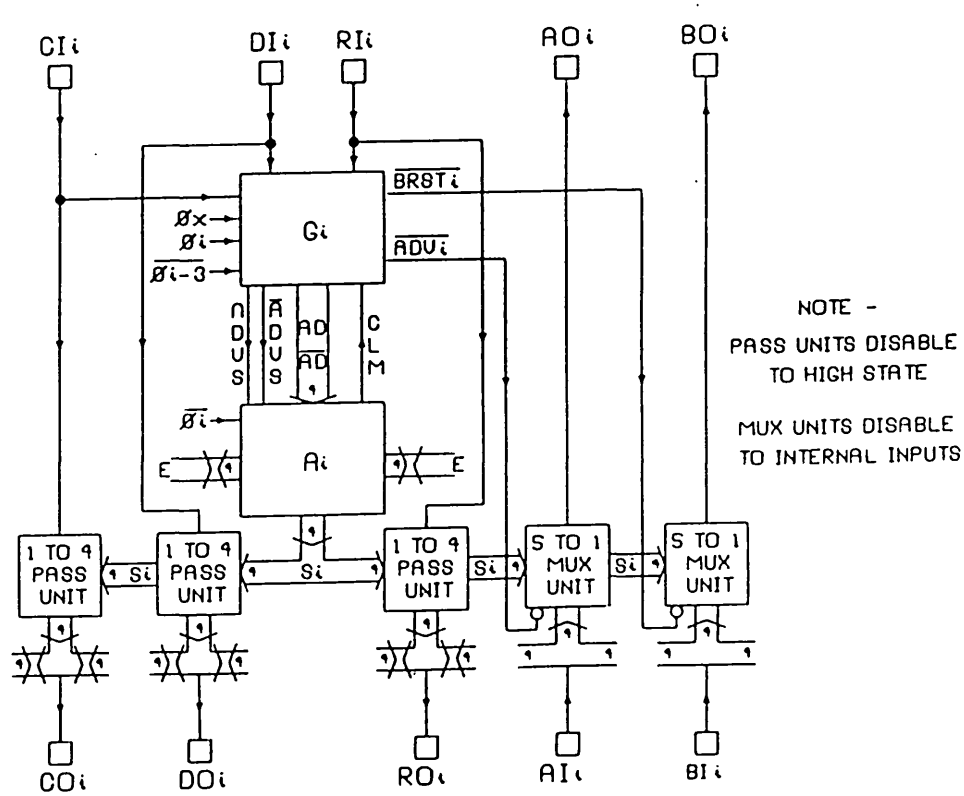


Figure 3-6: XS1 Slice Schematic.

3.2.3.1 The Address Generation Unit, G

The RI, DI and CI inputs to U_i in the device are all inputs to the corresponding G unit. The G unit is responsible for acquiring select address bits from the source node. The burst clock generator is present in G, as is the select address shift register. Figure 3-7 shows the logic diagram for this unit. The burst signal is generated from the two bistables and the four input OR gate at the top left of the diagram. The OR gate shown, passes $\emptyset_x$ for exactly three cycles after which it is disabled by the $\overline{Q}$ output of the second bistable. The three bistables in the centre of the diagram form the select address (and start bit) shift register. The two address bits, AD0 and AD1 are further latched by the two buffers shown at the top centre of the diagram. When loading the select address, the arrival of the start bit (a zero) at the far end of the shift register generates the address valid signal, ADVS which is ultimately

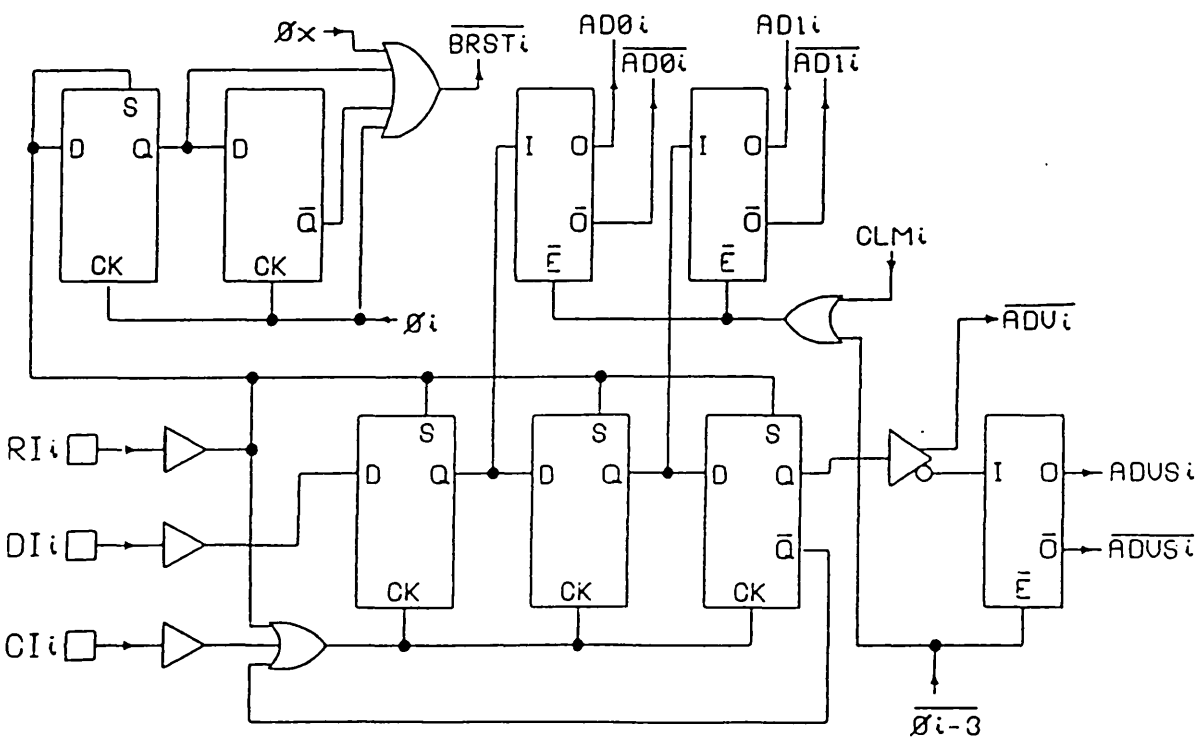


Figure 3-7: The Address Generation Unit.

routed back to the source on the AO wire. ADVS is also buffered. ADVS, AD0 and AD1 must be buffered in this way so as to enable a new set of select address bits to be clocked into the device whilst the L port claimed by a previous request is being released. A full description of the XS1 timing may be found in [CrF83].

3.2.3.2. The Arbitration Unit, A

A schematic diagram of the arbitration unit is shown in Figure3-8. The inputs to the arbitration unit are the external RI line, the ADVS, AD0 and AD1 lines generated by the G unit, and four L port engaged-status lines $E_0..E_3$ which are global to all four slices. If E_j is HIGH, then L_j is currently 'busy' i.e. there exists some U_i currently coupled to L_j.

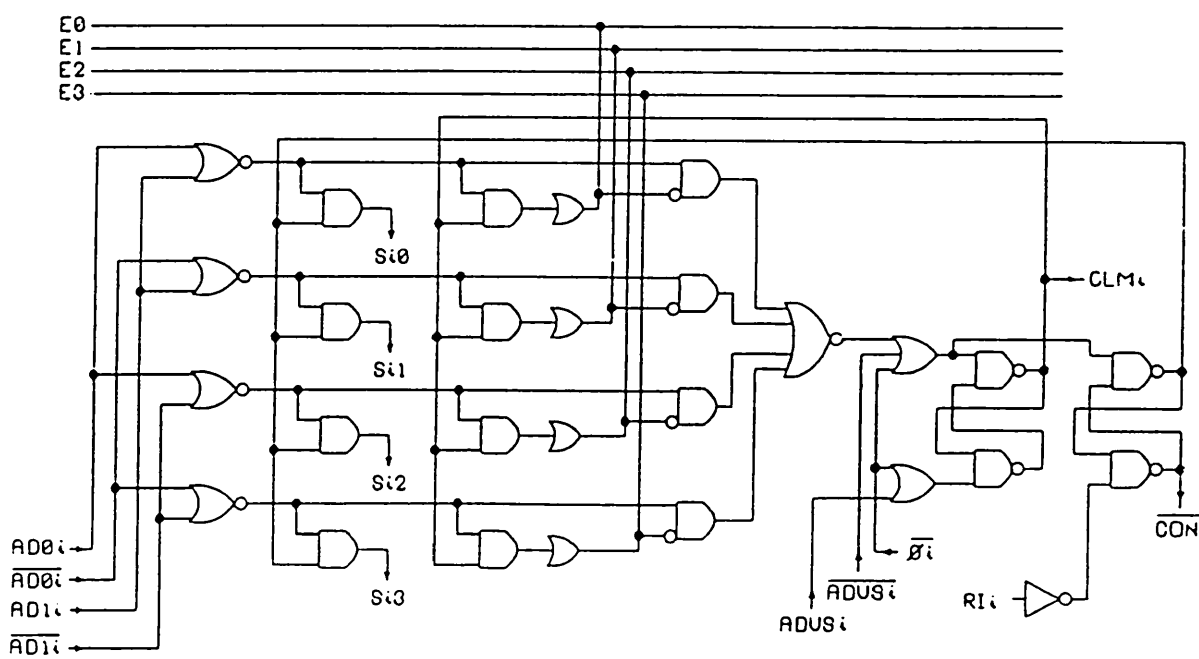


Figure 3-8: The Arbitration Unit.

The arbitration unit simply examines the status of the addressed L port on each arbitration clock edge, i.e. each time the 'token' passes. If the corresponding E line indicates that this port is free, then the port is

claimed, and the corresponding E line is set HIGH indicating to other requests in the device that that port is now engaged. When this claim has been successfully achieved, one of the select lines, $S_0..S_3$ in the diagram, is raised. This transition results in the lines on the U port being coupled to the corresponding lines on the addressed L port. When this port is eventually released, the coupling is broken and the arbitration unit waits for the next appropriate arbitration clock edge (i.e. it waits for the token) before lowering the relevant E line. Whilst this is being done, another set of select address bits may be arriving in the G unit. The AD0, AD1 and ADVS lines must be held stable during this time, hence the double buffering in the G unit.

3.2.3.3. The Pass (P) and Multiplexer (M) Units

The pass and multiplexer units contain the logic necessary to couple together an arbitrary pair of U and L ports. The required couplings are specified by the select lines generated by the arbitration unit. A HIGH signal on S_{ij} causes the five wires associated with U_i to be linked to the corresponding wires of L_j. The multiplexer units here are 5 to 1 multiplexers. Before the device has been selected by a request, the address valid and burst signals (internally - the ADVS and BRST signals) must be propogated back to the source on the A and B wires respectively. The extra input to these units enables this to be achieved whilst retaining the delay-matched property of the A and B lines through the chip.

A more detailed description of the internal logic of the XS1 together with timing diagrams explaining the interaction between the slice components and other XS1 devices may be found in [CRF83].

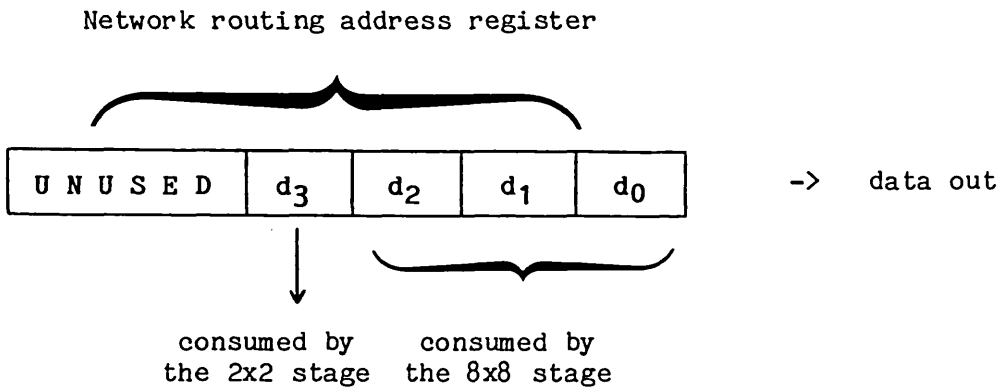
Although we have adopted a very narrow data path for the device (two wires in either direction), the routing cycle time (i.e. the time taken to request a device, fetch the select address bits, perform arbitration and

achieve coupling between and U and L port) is extremely fast (85ns without blocking). The A,B,C and D lines through an established channel each have a bandwidth of approximately 150Mbits/sec. Because of the delay-matched paths of C and D, and of A and B, this corresponds to a 150MHz self-clocked data communication channel in each direction. Performance is covered in more detail in 3.2.6. where the effects of blocking on the throughput of the XS1 and similar devices is considered.

3.2.4. Serial Switching, Self Clocking and Structure Independence

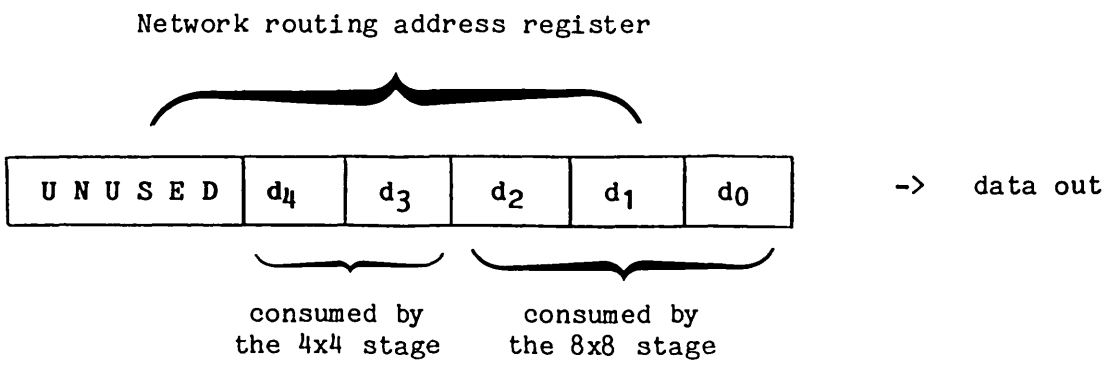
An important feature of this approach to switch design is that the surrounding system need not be aware of the size or topological structure of the interconnection network. Consider now a self-clocking network whose building block switches are of mixed degrees. We envisage providing a suite of different sized switches, each designed with the same self-clocking capabilities described above. All that is required to be known about the network is its maximum potential size, i.e. we must know how large to make the shift registers at the source nodes. Once this has been established the network can be configured in any way which suits the designer. As described in Chapter 2, large switches will generally be used wherever possible, firstly to reduce contention and increase performance, and secondly to reduce the total network chip count. If the requirements of the machine are such that one or more stages of smaller switches are desirable, however, then such switches can easily be introduced. Each switch extracts from the source shift register only those bits required to select it. Additionally, because he destination node provides an end of path signal the network can have arbitrary depth; the network 'tells' the source node when it has completed building the requested path. Consequently, the system operates independently of the network size.

For example, in a hybrid network composed of one stage of 8x8 switches and one stage of 2x2 switches the network routing address will be 'consumed' in the following way:-



Note that address bits are clocked out from the least significant end of the network address. This is to preserve structure independence (the index of the most significant used bit in the network address is dependent on the network size).

Now suppose that a network of size 32 is required, possibly by making an expansion to the existing network. This could either be achieved by using an extra stage of 2x2 switches, or by replacing the existing 2x2 switches by 4x4 switches and simply doubling the number of 8x8 switches in the network. Consider the latter example. The fact that the 2x2 no longer exist and the fact that the network is now twice its original size is of no importance to the system components or their interfaces. All that happens now is that one extra bit is consumed from the network routing address register, thus:-



If a particular stage contains switches of varying degrees then the network still operates correctly. This time, two requests may generate different demand patterns if they are required to traverse different sized switches in the same stage.

Structure independence significantly eases the task of network construction and reconfiguration. The network now assumes the properties of a 'black box' which functions correctly and at its maximum speed irrespective of the environment in which it is being used.

3.2.5. Interface Design

The network controlling mechanisms may be viewed on two distinct 'levels', which we term level 0 and level 1, corresponding to the path building and data transfer phases of a transaction. Level 1 provides mechanisms for enabling data transfers between the source and destination components. This level is dependent on the system components. Level 0 facilitates control of the interconnection network. The design of the level 0 logic concerns the interaction with the network via the self-clocking protocols described above and is independent of the attached system components. This section concentrates on level 0 since it is common to all interface designs. Implementation-dependent level 1 designs have been explored elsewhere for other networks, for example [WLL82].

3.2.5.1. XS1 Level 0 Interface Description.

In order to simplify the discussion of the XS1 interface logic, we consider a particular implementation which demonstrates the minimal logic required to control an XS1 network.

We consider an interfacing device whose role is to provide, as the result of the path building process, a single bit-serial channel in each direction

through the network. In the example, the D and A wires from the XS1 have been selected for this purpose although the choice is arbitrary.

In systems where the components have built-in serial communication channels a level-0 design provides all the necessary logic to control network operation. Such is the case with the ALICE machine where the systems components are formed by interconnected INMOS Transputers [IMS84]. Level 1 corresponds to the Transputer serial link controller. The role of the interconnection network is to provide a dynamic coupling between the serial links of two independent Transputers.

The logic diagram for this interface is shown in Figure 3-9 . The network routing address is supplied to the interface via a parallel interface of arbitrary width. The serial input and output wires forming a link are also supplied to the interface. When a path has been established to the addressed destination Transputer, the interrupt (ATTN) line is

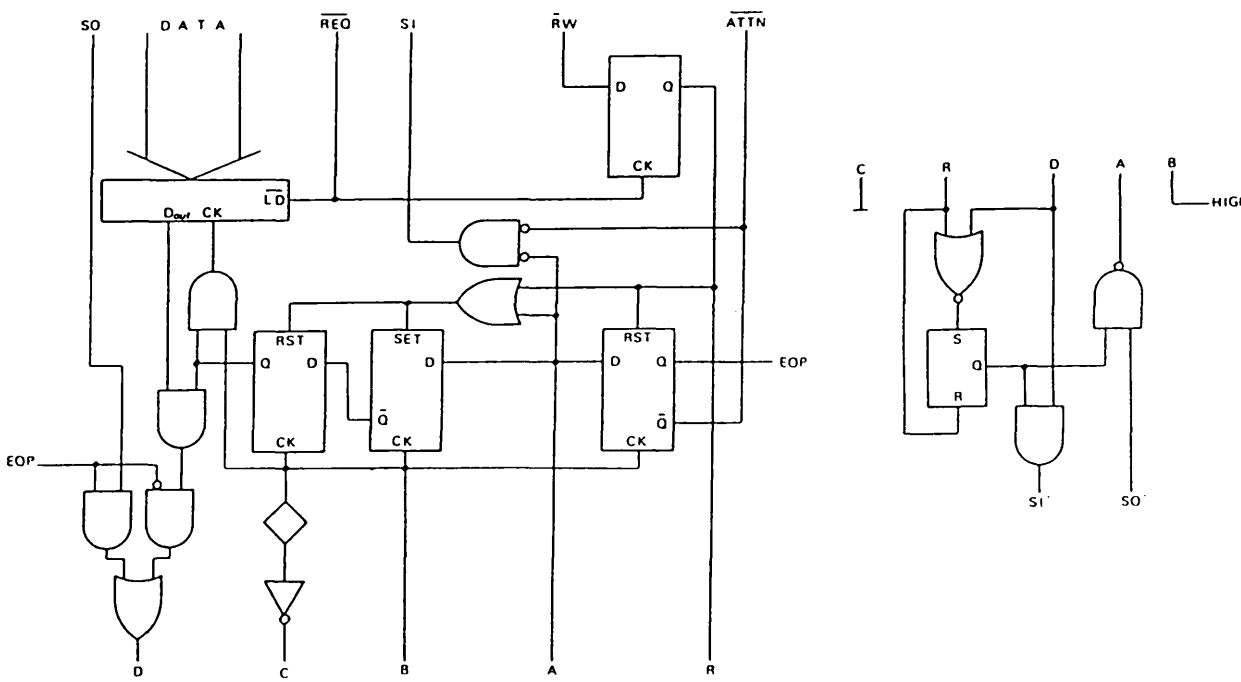


Figure 3-9: Level 0 Interface For Transputer/Transputer Interconnection

raised at which point the serial links of both source and destination Transputers are connected. The interface shown is a combined network input/output interface. Thus two serial links are provided by each Transputer to its corresponding interface. Each Transputer may then submit requests to and receive requests from the network concurrently.

3.2.5.2. Interface Operation.

When a (source) Transputer wishes to communicate with some other (destination) Transputer in the system, it writes the address of the destination Transputer to its associated interface. This write operation automatically initiates the self-clocking protocols in the network. Path building then proceeds automatically. The source Transputer waits for the interrupt to occur on its ATTN input which is generated directly from the end-of-path signal generated by the destination interface. At this point the network (and the interface) is transparent and the two Transputers can communicate as if they were hard-wired together. The established channel through the network is released by performing a 'read' operation on the interface register. This has the effect of resetting the XS1 request (R) line thereby freeing the channel.

Viewing the Transputer/interface bus as an occam [IMS84] channel, the following occam code is sufficient to control a network transaction:-

At the transmitting end:

```
PAR
  -- Any other activities..
  SEQ
    Parallel.Port     ! Target.Node          -- Write network address
    ATTN              ? ANY                  -- Wait for interrupt
    Serial.Link       ! <some data>          -- Send data
    Serial.Link       ? <possible reponse>   -- Optional response
    Parallel.Port     ? ANY                  -- Release network channel
  -- Any other activities..
```

At the receiver:

```
PAR
  -- Any other activities..
  SEQ
    Serial-Link       ? <some data>           -- Input message
    Serial-Link       ! <possible response>   -- Optional response
  -- Any other activities..
```

Note that the NIC logic may be incorporated into a custom interfacing device, although, it is easy to construct this using discrete logic. (The interface shown in Figure 3-9 can be implemented using the equivalent of five 74-series TTL ICs.)

Observe that everything in the NIC is asynchronous; no control clocks or synchronisations are required since all timing information is provided by the network itself.

This defines the minimum interface required to be able to integrate an XS1 network into a parallel computer system. The speed (i.e. the eventual throughput) of the network is dictated by the speed of the serial links. The maximum speed attainable by this interface is the XS1 D/A wire specification speed of 150 MHz. If the processor serial links are slower than this, and greater bandwidth is required, then a separate high-speed NIC with a parallel interface to the processor must be provided. This NIC could simply provide the same (or similar) functions as are provided by the Transputer serial links, only at higher speed (for example by exploiting the fast off-chip serial communication capabilities of ECL, GaAS, etc. technologies and by using the path-matched properties of D and C, and A and B to self-clock the data through the network).

If only unidirectional block transfers are required (as is the case with many declarative systems architectures), then the NIC could be buffered so that the whole block is present in the NIC <u>before</u> the transaction is initiated into the network. Then, once the channel is established between the source and destination NICs, the block transfer can proceed at NIC

speeds. This minimises the 'channel-hold' time through the network and therefore minimises overall network latency. In this mode, the network is operating at its maximum speed, during both the path set up and data transfer phases of the transaction. Multiple buffers can be included in the NICs to enable the next transaction(s) to be assembled whilst the previous one(s) are being completed. This minimises the delay between successive transactions being initiated. The buffers in the NICs can be filled (at the source end) and emptied (at the destination) by memory mapping from the respective processors or, more appropriately, by DMA. The problems of network interfacing are raised again in Chapter 4 where the requirements of a fault-tolerant network interface are examined.

Note that because the XS1 allows the asynchronous cancellation of requests it is possible for the interface to implement a timeout facility whereby blocked requests are cancelled and later retried. This yields a marginal performance improvement, as described below, but does not guarantee starvation-free routing behaviour.

3.2.6 Performance

Analytical models for obtaining asynchronous network perfomance are still being developed. At the present time, the predictions of self-clocking network performance have been achieved through simulation, although single-switch configurations have been successfully analysed as described in Chapter 2.) The following performance figures were obtained from an interactive event-driven simulation program written in PASCAL and developed primarily for the rapid acquisition of performance estimates for arbitrary self-clocking networks through user interaction.

The operating environment is specified by the user and can be modified from run to run. The list of system commands and alterable operating parameters are given in Appendix 2.

The significant network operating parameters are:-

N - The size of the network

x - The degree of the network (only regular topologies are exmamined)

I- The mean time taken to present a new request to the network following the completion of the previous request.

T - The mean time taken to complete a transaction after path building.

The significant performance statistics referenced here are:-

BW_N - The normalised bandwidth, which is the proportion of the data transmission clock frequency actually used for data transfer.

TT_N - The normalised mean transaction time, which is the mean transaction time expressed in data transmission clock periods.

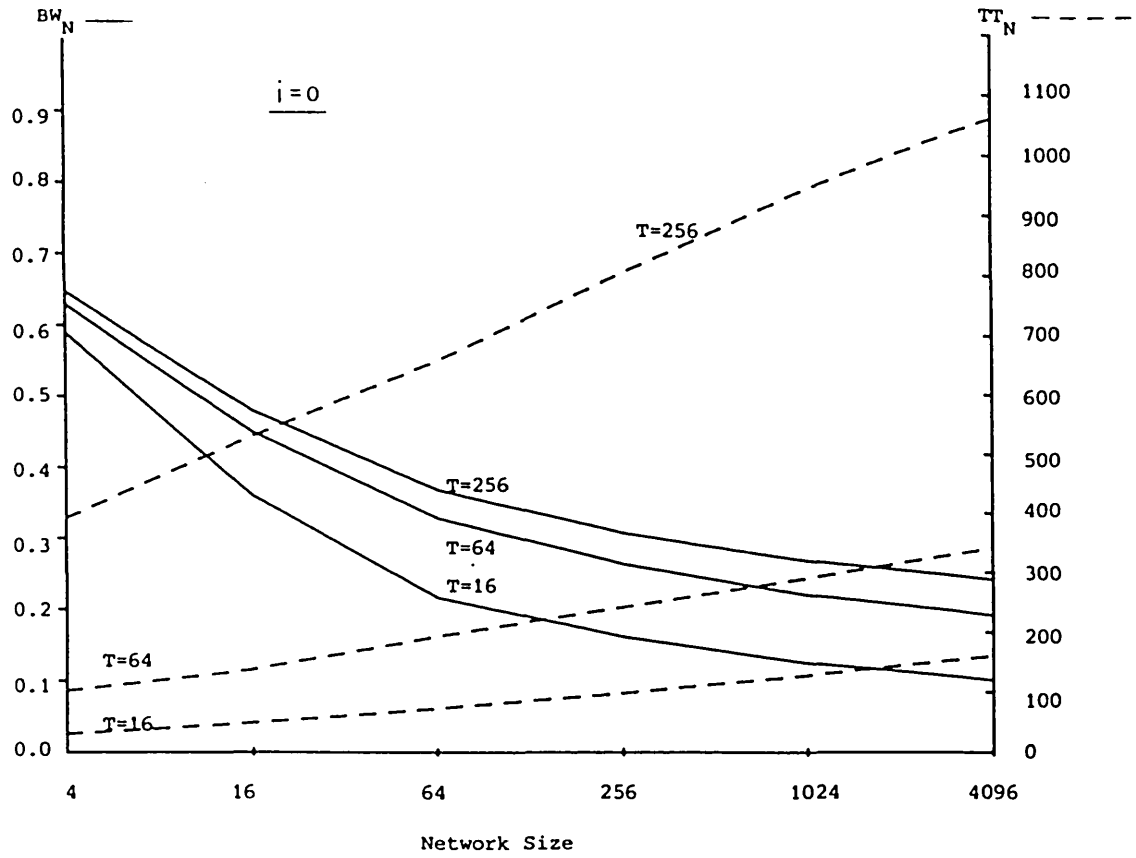
In each simulation run, the switch control clock, ($\emptyset_x$ in the XS1) is assumed to operate at frequency f. This enables the performance metrics to be expressed independently of the network clock frequency. Except for very short transfers, the transaction time is dominated by the path set up and data transfer times, so the switching speed (determined by $\emptyset_x$) does not contribute significantly to the overall transaction time.

Figures 3-10(a) through 3-10(c) show the behaviour of BW_N and TT_N for:-

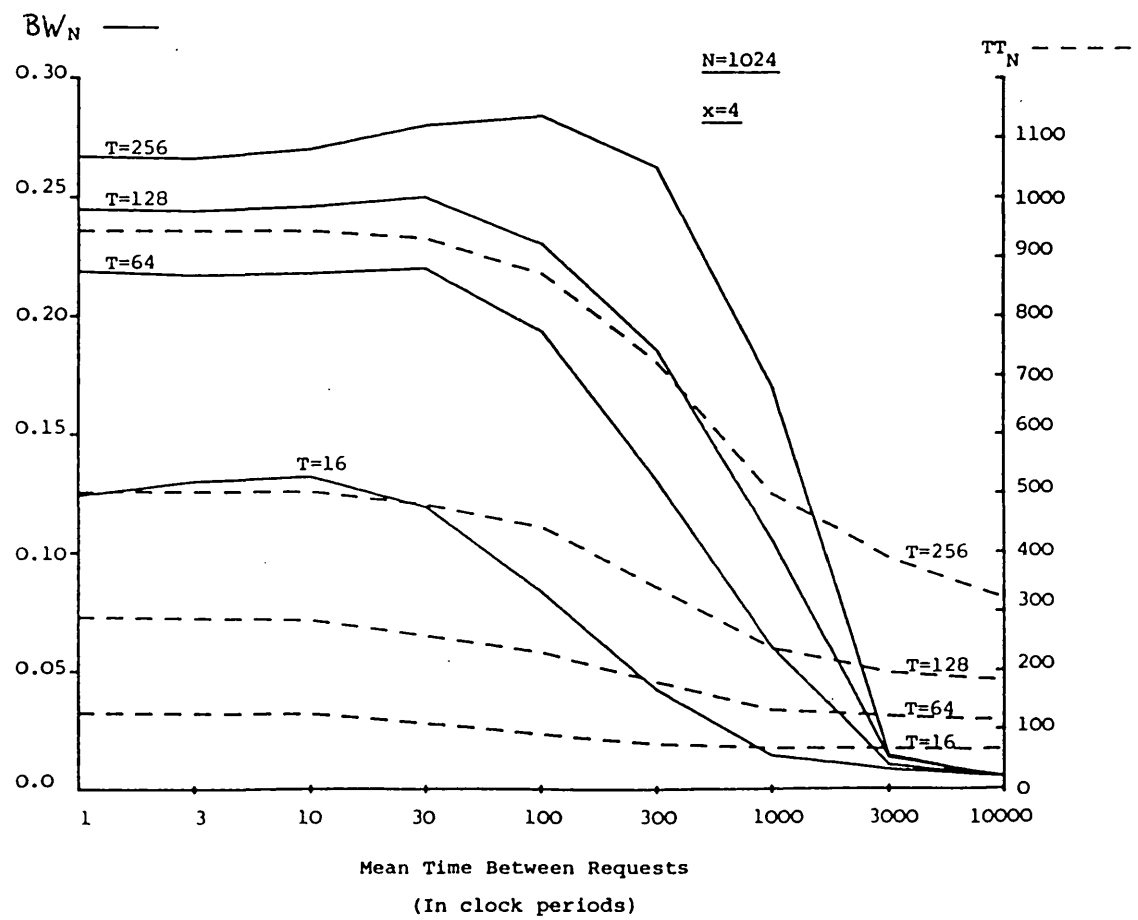
1. x=4, I=0 (e.g. the XS1) with varying N and T.
2. x=4, N=1024 with varying I and T.
3. N=4096, I=0 with varying T and x.

3.2.6.1. Timeouts

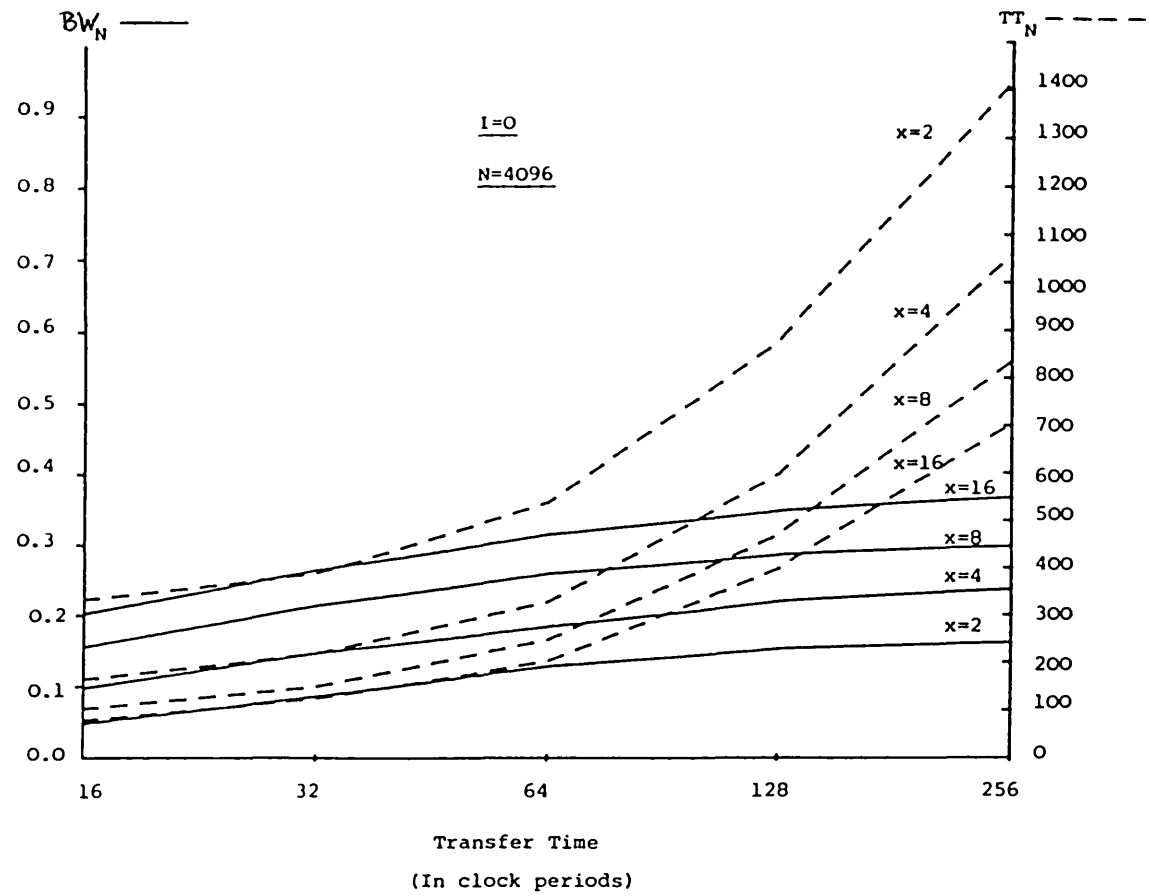
As described above, the XS1 allows the source node interface to asynchronously cancel a request and then resubmit it. This mechanism can be analysed by use of the SET V, SET Y and SET H options in the simulator (see



(a) XS1 Performance With Increasing Network Size And Transfer Length.



(b) XS1 Performance With N=1024. cont....



(c) Network Performance With Varying Switch & Transfer Sizes.

Figure 3-10: Self Clocking Network Performance

Appendix 2). There appears to be no 'optimum' rule for establishing the value of the V, Y and H parameters, particularly when the network transfers are of varying size. However, by experimentation with the simulator it has proven possible to enhance the network throughput by up to approximately 12% for some configurations. It is questionable whether the additional complexity in the interface required to implement the timeout scheme can be justified with only comparatively little gain in performance at stake. Also, in order to implement the scheme the minimum network path building time must be known to the interface and the timeout parameters must be to be re-programmed after a reconfiguration of the network so as to retain the optimal performance.

3.2.7. Future Developments

We conclude this chapter by describing two enhancements to the XS1 design which may be incorporated into future self-clocking network implementations.

3.2.7.1. Asynchronous Burst Clock Generation

In the XS1 two synchronisations are required in order to select a switch in the network. However, by extending the width of the data/control path, it is possible to reduce this to just one synchronisation by having the burst generator clock supplied externally and synchronised with respect to the network Reset line. The extended six line interface would operate exactly as before except that the negative transition in the network Reset line corresponding to an arriving request need not now be synchronised with respect to $\emptyset_x$ to generate the burst clock. Instead the required number of burst clock pulses supplied on the additional clock input (generated by the previous stage) can be 'picked off' and passed back to the source without requiring synchronisation. This enhances the switching speed of the network and reduces the probability of errors occurring due to (transient) synchronisation failure prior to the generation of the burst clock.

3.2.7.2. Tri-Level Logic Implementations.

An obvious way to improve the current XS1 design is to further reduce the width of the control/data path from five wires (six wires if the above enhancement is made) so as to further improve the ratio of switch size to package size. As described earlier, self clocking techniques can be implemented with just three wires: D and C can be contracted into a single phase-encoded, or isochronous transmission line, and the A and B lines reduced to just the burst signal, dispensing with the acknowledge line altogether. However, by doing so, many of the attractive properties of the self clocking scheme are lost. Phase encoding relies on some form of phase-locked loop decoding on chip which is awkward to implement.

Transmission of data without including an explicit clock (isochronous transfer) is less reliable and relies on an accurate matching of the source and destination clocks to make it work. The lack of an acknowledge line in the control path means that an end of path signal (easily generated in the five wire configuration) requires more complex coding techniques which must be adhered to by both the burst clock and acknowledge signal generators on chip, and the end of path signal generator at the destination interface. Finally, as a result of there being no explicit clock line in either the forward or reverse data paths after path set up, the through-channel data transmission speed is necessarily reduced (phase encoding, for example requiring a channel bandwidth of 2f Hz in order to transmit data at a rate of f Hz).

Despite this, however, a three line data/control path could be used by employing tri-level logic to overcome all of the above problems, and provide additional advantages over the XS1 design.

A tri-level logic implementation of the XS1 would employ only three data/control wires at each switch port: a tri-level combined Reset/Burst clock input line, R/Ø with logic states LOW, MID and HIGH, say, a tri-level unified data/clock (D/C) line, and similarly, a tri-level unified burst/acknowledge (B/A) line. In the passive state, the R/Ø line is in its HIGH logic state. The conventional Reset signal is signalled by a transition from the HIGH to MID logic state and this is then followed by a continuous clock between the MID and LOW logic states. This clock is used to generate the burst clock. For all data transmission on the A/B or C/D lines, a MID->LOW transition is be interpreted as a 'clock data LOW' signal, and a MID->HIGH transition as a 'clock data HIGH' signal. Consequently the full bandwidth of the transmission line is made available for data transfer.

Since the burst and acknowledge signals are mutually exclusive, the burst

clock could be generated using MID<->HIGH transitions on the A/B wire, and the acknowledge signals, by using MID<->LOW transitions, for example. The end of path signal is then easily generated by, for example, issuing a double pulse from MID->LOW on the A/B wire.

Using this technique, the full suite of self-clocking network protocols can be implemented using only three wires per slice. This results in a pin count of only 6x+P+1 where x is the switch degree, P is the number of power pins and the extra pin is for the arbitration clock input.

Reagrdless of whether the implementation uses a three, five or even six wire control/data path, the use of self-clocking network principles results in networks which are inherently smaller, cheaper, more cost-effective and more reliable than equivalent conventional implementations.

CHAPTER 4

Fault Tolerance In Self-Clocking Networks

Despite the growing interest in interconnection networks in recent years, comparatively little attention has been paid towards making these networks fault tolerant. In this chapter, we consider this problem, and describe a highly efficient means of obtaining fault tolerance which is particularly appropriate to the asynchronous self-clocking networks which were introduced in Chapter 3.

The fault problem in interconnection networks may be appreciated by examining a typical interconnection network, for example that shown in Figure 2-1. For any pair of source and destination nodes there is only one path through the network which links the two. If a (permanent) fault is introduced anywhere in the switches or interconnections on this unique path, then the network ceases to provide complete interconnectivity. In small systems, where the number of switches and the number of device interconnections required in the network is comparatively small, the fault problem is less significant. In highly parallel systems, however, a large number of switches and an even larger number of interconnections are required. At this point the fault problem must be taken seriously if the system is to operate without requiring frequent down time for repair. Note that we consider here only the fault tolerance of the interconnection network; fault tolerance within the components attached to the network is not addressed.

4.1. Fault Models

An interconnection network may go wrong for a number of reasons; the types

of fault which may occur may be broadly categorised into transient and permanent fault types.

4.1.1. Permanent Faults

4.1.1.1. "Stuck-at" Faults

The classical fault model for digital circuits is a single line stuck at either a logical 0 or a logical 1 [Agr82]. The stuck-at fault model is applicable to all digital systems and accounts for a majority of SSI and MSI failures and is applicable to all VLSI systems even though transient faults predominate here (see 4.1.2.1).

A stuck-at fault occurring in the control/data path terminals of a device in an interconnection network will either cause:

1. A request to be stuck as the result of the inhibition of the self-clocking protocol signals between the network an the interface control logic.
2. The request to be steered to the wrong network output port.
3. Data to be corruped as it is transferred through an established network channel.

Errors of type 1 may be caused by any stuck-at fault in the B or C terminals of a switch, a stuck-at-one fault in the R terminal of a switch, or a stuck-at fault in the A terminal of a switch (or interface) causing the end of path protocol to be corrupted. Errors of type 2 may be caused by a permanent stuck-at fault at one of the D-wire terminals of a switch or by a stuck-at fault at an A terminal causing address bits to be clocked out as start bits (if start bits are used). Errors of type 3 may be caused by stuck-at faults in any of the A,B,C or D lines depending on the nature of the communication through an established channel. Note that a stuck-at-zero fault at an R terminal will effectively set up a permanent channel

from an internal switch of the network through to some destination interface. Any information sent to the associated faulty switch port will be immediately transmitted to this destination node. The request will then fail because no end of path signal will be generated.

4.1.1.2. Permanent Link Faults

Permanent faults in the links of the interconnection network will produce effects similar to those produced by stuck-at faults. The fault scenario is as described above.

4.1.2. Transient Faults.

4.1.2.1. Transient Switch Faults.

Although the stuck-at fault model is recognised as being applicable to SSI and MSI technologies, field studies indicate that between 90 and 98% of all detected faults in VLSI are transient and are induced as the result of interactions between adjacent elements in densely packed layouts over a limited area [MST79]. Although the stuck-at fault model is still significant, and must be considered, it would appear that transient errors predominate. Transient ('soft') errors in an interconnection network may cause stuck requests, incorrect routing as a result of soft errors during path set up, or data transfer errors as a results of soft errors occurring during data transfer.

4.1.2.2. Transient Link Faults.

A majority of faults occuring in data transfer paths have been shown to be unidirectional (either 1s becoming 0s or 0s becoming 1s) [CSS73]. Broken connections, the shorting of signal lines to power or ground and the loss of power all cause unidirectional errors during data transmission, be it during the path set up or data transfer phases of a transaction.

4.1.2.3. Synchronisation Failure.

In any asynchronous system, the failure to synchronise two phase-independent signals will cause undeterminable voltage signals to be induced in the system ('metastability', 'glitching', 'boggling' etc.). Although the synchronisation failure probability can be reduced by the use of high loop gain bistables (as in the XS1 device) and long settling times, the possibility of failure still remains. In the XS1, for example, synchronisation failure will cause one or more burst clock signals to be lost resulting in the associated request becoming 'stuck' at the currently selected switch.

In the proposed fault tolerant scheme we assume the following fault models based upon the scenarios listed above:

1. Stuck-at faults at the terminals of any switch in the interconnection network or at the terminals of the network interfaces which link into the network.
2. 'Soft' faults within any switch in the network causing:
   (a) The wrong switch output port to be selected during path building.
   (b) The (transient) corruption of a control signal during path building causing a request to become 'stuck'.
   (c) Unidirectional errors in the data as it is transferred through an established network channel.
3. Either permanent faults in the links of the network (or links between the network interfaces and the peripheral stages of the network), or transient link faults causing unidirectional errors in data transmission.
4. Synchronisation failure prior to the generation of the burst clock.[1]

---

[1] This can be eliminated as described in 3.2.7.

Note that bidirectional data transfer errors can be handled provided the number of bits corrupted from 0 to 1 is different from the number of bits corrupted from 1 to 0. Note also that faults present at the junctions or links between the interfaces and the system components are not considered.

4.2. Existing Fault Tolerant Schemes

Before describing the proposed approach to fault tolerance we first comment on related work in the area of fault tolerance network design.

4.2.1. Self Testing Switches

In SIMD systems, the interconnection network is controlled synchronously. In [LiW82], the synchronous property of such networks is used to periodically perform 'test cycles' on the network. In this test cycle each switching element performs a self test operation in the network and if a fault is recorded during this test, then backup logic within the device is switched in. Multiplexers and demultiplexers are provided within the switch to achieve this. After the test cycle, the network resumes normal operation. All switches are, at that point, guaranteed to be fault free.

Whilst the test cycle correctly diagnoses any faults within the switches of the network, there is no provision for fault handling during normal operation. Additionally, the implementation of the scheme requires considerable increase in the complexity of the switch. Most of the switch logic is concerned with implementing the self test, i.e. generating test patterns, detecting faults and performing recovery. It is questionable whether this makes the switch any more reliable than before. The scheme appears impractical for switches of degree larger than 2.

4.2.2. Extra Stage Networks

An elegant scheme for enhancing network reliability has been proposed in

[AdS82] and elsewhere in [WLL82]. In these designs an extra stage of switching elements are provided at the input side of the network. Initially, the extra stage is enabled onto the network inputs by demultiplexer units attached to each extra-stage switch, and the output stage of the network is disabled via multiplexer units attached to each switch. Faults detected in the internal stages of the network can be bypassed by switching in the extra stage which yields a second (fault-free) path between the source and destination. Faults in the topmost stage of the network are bypassed by switching in the extra stage and switching out the topmost stage. Complete connectivity is preserved at all times.

The 'extra-stage' approach is very economical, but relies on the ability to detect errors to the level of individual switches so that the correct choice of redundant paths can be made after reconfiguration. It is never clear, though, whether a fault lies in output to a switch in some stage, S, the input to a switch in stage S+1, or in the link between the two. Furthermore, the scheme does not cater for multiple faults affecting the same source/destination path, or faults in the terminals, links or (de)multiplexers at the peripheral stages of the network.

4.2.3. Multiple Plane Networks

An obvious way of obtaining fault free paths in an interconnection network is to have two or more independent networks and simply switch a new network plane into operation whenever a fault is detected in the currently operating plane [Agr79]. Multiplane networks of this sort are by default fault tolerant. This is an effective, although not very economical means of achieving fault tolerance since each redundant paths introduces only one extra potential path between an arbitrary source and destination.

However, a method of exploiting multiplane networks in bit slice configurations has been proposed in [LLY82]. Here, the network is assumed to already exist in three dimensions: each word to be transferred across

the network is bit sliced into packets and these packets are passed through the network synchronously with each packet being submitted to an independent data plane. Error correcting codes are used within each packet to enable the destination node interfaces to perform automatic error correction of the entire word transferred. An entire network plane can fail without causing the network to fail, and can consequently be replaced while the other planes continue to operate.

The scheme exploits the packet redundancy inherent when large words are being transferred in bit slice fashion. The scheme is not applicable to single plane systems, and is only suitable for synchronous network implementations.

4.3. A Fault Tolerant Scheme For Self-Clocking Networks.

None of the existing approaches to achieving fault tolerance is suitable for the class of networks which are of interest here, namely the asynchronous serially-switched networks which were introduced in Chapter 3. The asynchronous nature of each switch in the network makes it impractical to perform on line self testing (fault avoidance), whether by using self testing switches, or 'test stimuli' techniques as described in [Agr82,WuF79]. 'Extra-Stage' networks could be adapted for asynchronous implementations, but these rely on exact fault location followed by a complete reconfiguration of the network. What is desirable is a distributed error detection and isolation mechanism which can be exercised by each system component independently of, and asynchronously with respect to the other components in the system. In the following sections we describe how error detection, fault avoidance and fault recovery can be provided in a self-clocking network, although many of the techniques may applied to existing designs.

In a fully populated interconnection network, only one path exists between

a source node and an arbitrary destination node. (This is obvious since the set of paths from a source node to the set of all destination nodes forms an x-ary demultiplexor tree.) A failure in any switch or link between switches in the network will prevent certain source/destination couplings from being made. Consequently, a necessary condition for fault tolerance is the presence of multiple paths between all source/destination pairs. We now propose an extended network topology in which each system component is allocated one or more additional links into the network. This applies to both source and destination components. The number, $k \leq x$, of network links at each component is termed the degree of fault tolerance.

The existance of k links at the source components results in there being potentially k distinct paths between each source node and a given network output port. However, since each destination component also links to k independent network output ports, there are potentially a total of k^2 distinct paths between each pair of source and destination nodes.

In this definition, two paths are said to be distinct if they share no internal network links in common. Each link between the peripheral stages of the network and the system components will be shared by k distinct paths. In addition, two paths are said to be independent if they share neither a common link nor a common switch in the network. A network is defined to have the path independence property if for every possible path from a source node to a given destination node there is at least one additional and independent path between the same two nodes. The path independence property thus guarantees the network to be fault tolerant in the case of all single faults.

The problem is now to find a general interconnection rule which guarantees this property regardless of the size or topology of the underlying network. We choose as an example a network based upon the Generalised Cube topology

and consider the problem of providing dual-port fault tolerance i.e. k=2. We choose to discuss the Generalised Cube since the proofs for this topology are slightly simpler than for other topologies. However, similar properties can also be derived for non Cube-based topologies and for cases with k>2. Initially, we shall consider only regular networks. Fault tolerance in irregular networks is discussed in 4.3.3.

4.3.1. Providing Multiple Paths.

4.3.1.1. Address Transformation

In a normal operating environment, a source node, s, obtains a channel between itself and some destination node, d, by issuing the address of d to its associated network interface. In the fault tolerant set up, the same destination node can also now be reached by routing the request to the other (generally, any of the other k) network output port(s) associated with the destination node. With k=2, the alternative address, denoted by d', is obtained by a simple, and consistant transformation, T, on d, i.e.

$$d' = T(d) \quad \text{for all } d.$$

Issuing either d or d' to the source interface will steer the request successfully to the destination node. Thus there are four ways to address the network in order to provide a coupling between a source and a destination node since either input port and either the transformed or untransformed version of the destination address may be used. We now demonstrate that there exists a topological rule which ensures that each of the paths taken by these four methods is distinct and that the network as a whole has the path indpendence property.

4.3.1.2. Extending The Cube.

The Generalised CUBE topology is similar to that of the CUBE topology described in Chapter 2. We use the traditional definition of the topology which assumes the stages of the network to be labelled such that stage 0

occurs at the output side of the network and stage n-1 at the input side of the network. To simplify the proofs, we use the more convenient and general notation for specifying the topology, as has been used in the previous chapters. The permutation function for stage s we shall denote by G_s and is defined as follows:

$$G_s : O_s(<j_{n-1},j_{n-2},..,j_{s+1},j_s,j_{s-1},..,j_1,j_0>) \rightarrow$$
$$I_{s-1}(<j_{n-1},j_{n-2},..,j_{s+1},j_0,j_{s-1},..,j_1,j_s>)$$

In a normal operating environment no permutation is required at the output side of the network, i.e. a routing address:

$$d = <d_{n-1},d_{n-2},..,d_1,d_0>$$

steers a request directly to the network output port labelled d. In the fault tolerant system with k=2, there are two network ports associated with each system component, thus there are additional permutations on both the inputs and outputs of the network.

The fault tolerant topology is obtained by applying the same permutation that occurs between the last two stages of the network between the source components and the top level of switches in the network. Thus in this example, we extend the Generalised Cube topology so that the G_1 permutation function is applied between the system source components and the top level of switches. In the fault tolerant scheme, the bits of the destination address are clocked out of the source NIC from the least significant end to preserve structure independence. (This is discussed in 4.4.) Thus at the output side of the network, a digit reversal permutation is applied to 'realign' the addresses.

The system components at the input and output sides of the nodes of the system are now connected to the network via adjacent pairs of network links

respectively before and after the input (G_1) and output (address bit reversal) permutations have been applied. Thus, for example, input links:

$$\langle l_{n-1},l_{n-2},..,l_1,l_0\rangle \text{ and } \langle l_{n-1},l_{n-2},..,l_1,l_0'\rangle \quad l_i=0..x-1, \quad i=1..n-1$$

are both associated with the same input node, where l_0 and l_0' differ only in their least significant bit. From here onwards the prime (') may be assumed to be the postfix operator meaning 'invert least significant bit'.

We now demonstrate that using the above interconnections, the 4 (i.e. k^2) possible paths between 2 arbitrary nodes in a network of arbitrary size and degree each access independent subsets of the network switches, with the exception of stages 0 and n-1 which must clearly be shared by 2 (i.e. k) paths each.

Firstly, we must demonstrate that each system component is connected (via its interfaces) to exactly 2 <u>independent</u> switches in stage 0 and stage n-1.

<u>Lemma 4-1</u>: Each input node links to 2 independent top level (stage n-1) switches.

<u>Proof</u>: Let $\langle i_{n-1},..,i_1,i_0\rangle$ and $\langle i_{n-1},..,i_1,i_0'\rangle$ be the two network links associated with source node i. After applying the input permutation, G_1, source node i links to inputs $\langle i_{n-1},..,i_0,i_1\rangle$ and $\langle i_{n-1},..,i_0',i_1\rangle$ of the top level (stage n-1) of switches. These are associated with the switches labelled $\langle i_{n-1},..,i_2,i_0\rangle$ and $\langle i_{n-1},..,i_2,i_0'\rangle$ respectively. Since $i_0 \neq i_0'$, these switches are independent.

[]

<u>Lemma 4-2</u>: A destination node $d=\langle d_{n-1},..,d_1,d_0\rangle$ can be accessed from all source nodes by supplying either the routing address d, or the address $d'=\langle d_{n-1},..,d_1,d_0'\rangle$.

<u>Proof</u>: The generalised cube which clocks from the least significant end of the routing address, steers a request with routing address d to

network output $<d_0,d_1,..,d_{n-1}>$. After realignment, it is steered to $<d_{n-1},..,d_1,d_0>$. Similarly, a request with routing address d' will be steered to $<d_{n-1},..,d_1,d_0'>$ after realignment. Since adjacent pairs of outputs are both connected to the same destination node, and since d_0 and d_0' differ only in their least significant bit, both d and d' will steer a request to the same destination node, d.

[]

Thus if path building to some target node fails with address d, an alternative address can be derived by inverting the least significant bit of the destination address. This defines the transformation function, T, described in 4.3.1.1.

We must now demonstrate that for all possible transactions from a given source node, s, the set of switches of stages 1 to n-1 which are accessed by issuing a request on one input channel, is disjoint from those accessed by issuing the same request on the other channel. This amounts to showing that the set of switches in stages 1 to n-1 which are accessible from either one of the network inputs of a given source node is disjoint from that accessible from the other input.

Lemma 4-3: Let $i=<i_{n-1},..,i_2,i_1>$ and $j=<j_{n-1},..,j_2,j_1>$ be the two stage n-1 switches to which with some arbitrary source node, i, is connected. From Lemma 4-1, $i_k=j_k$, k=2..n-1; $j_1=i_1'$. Let D_k be the set of all devices in stages n-1 through 1 which are accessible from switch k of stage n-1, k=0..N/x-1.

Then, $D_i \cap D_j=\emptyset$.

Proof: Denote by [s,w], switch w of stage s, s=0..n-1, w=0..N/x-1, and let $D_i(s)$ be the set of switches of stage s satisfying:

$$[s,w] \in D_i \iff [s,w] \in D_i(s).$$

(Note that $D_i(n-1) = \{[n-1,i]\}$ and $D_i(0) = \Sigma$, the set of all switches in stage 0 of the network.) D_i may then be expressed as:

$$D_i = D_i(n-1) \cup D_i(n-2) \cup .. \cup D_i(1).$$ *

Now start with [n-1,i]. The output links associated with [n-1,i] are those outputs of stage n-1 labelled $\langle i_{n-1},i_{n-2},..,i_1,\theta_1\rangle$, θ_1=0..x-1. Now, applying G(n-1) to these we have that:

$$D_i(n-2) = \{[n-2,\langle\theta_1,i_{n-2},..,i_1\rangle]\},\ \theta_1=0..x-1$$

From the definition of G(n-2) we have that:

$$D_i(n-3) = \{[n-3,\langle\theta_1,\theta_2,i_{n-3},..,i_1\rangle]\},\ \theta_1,\theta_2=0..x-1$$

Generally, $D_i(s)$ is given by:

$$D_i(s) = \{[s,\langle\theta_1,\theta_2,..,\theta_{n-(s+1)},i_s,..,i_1\rangle]\},\ \theta_y=0..x-1,\ y=1..n-(s+1)$$

Therefore, since $i_1\neq j_1$, and since $D_k(s)\cap D_m(s+1)=\emptyset$ for all k,m it follows that for all s=1..n-1, $[s,d]\in D_i(s) \Rightarrow [s,d]\notin D_j(s)$ s=1..n-1, hence, from $\ast$, $D_i\cap D_j=\emptyset$.

[]

Finally, we must demonstrate a similar result for the address transformation scheme, i.e. that:

Lemma 4-4: The set of switches of stages n-2 through 0 accessed by issuing a routing address, $d=\langle d_{n-1},..,d_1,d_0\rangle$ is disjoint from that accessed by issuing the address $d'=\langle d_{n-1},..,d_1,d_0'\rangle$ from the same source node.

Proof: Let the top level switch through which a request is issued be labelled $\langle i_{n-1},..,i_2,i_1\rangle$. The stage n-2 switch which is accessed by supplying the routing address d is then given by $\langle d_0,i_{n-2},..,i_2,i_1\rangle$. Similarly, $\langle d_0',_{n-2},..,i_2,i_1\rangle$ for the routing address d'. The proof then follows similar lines as for Lemma 4-3. This time, the sets corresponding to D_i and D_j of Lemma 4-3 are disjoint by virtue of the fact that $d_0\neq d_0'$. []

Thus:

Theorem4-1: Thefour possible paths between any two nodes have no stage 2..n-2 switch in common.

Proof: FromLemma 4-3, no stage 2..n-1 switch accessible from one ofthe

input ports of a given source node is accessible from the other. From Lemma4-4, no stage 2..n-1 switch traversed by issuing the routing address, d, from a given network input is traversed by issuing the address d' from the same input. Consequently, in these stages none of the four paths have any switches in common.

[]

Hence,

Theorem 4-2: Each of the four possible paths from a given source node to a given destination node is distinct and for each of these paths there exists exactly one additional and independent path between the same two nodes.

Proof: Follows directly from Theorem 4-1, and from Lemmas 4-2 and 4-4.

An example of a regular fault-tolerant network is shown in Figure 4-1. This example shows an extended Generalised CUBE network of size 16 and degree 2 with the additional permuatations for fault tolerance shown at the peripheral stages of the network. The four paths from source node 4/5 to destination node 8/9 are shown highlighted. This shows the four distinct paths which consist of two pairs of independent paths.

4.3.1.3. Irregular Topologies

In the previous chapters we have stressed the benefits of hybrid or irregular topologies which may contain stages of varying degrees. In the proposed fault tolerant scheme the only problem presented by irregular topologies is retaining the distinctive and independent properties of the k^2 paths through the network for each source/destination pair. However, this is not hard to achieve and can be seen from, for example, Figure 4-1. An irregular topology can be easily formed by replacing the switch arrays within each broken box shown by switches of degree 4. If required, the lower three stages could even be replaced by two switches of degree 8 in

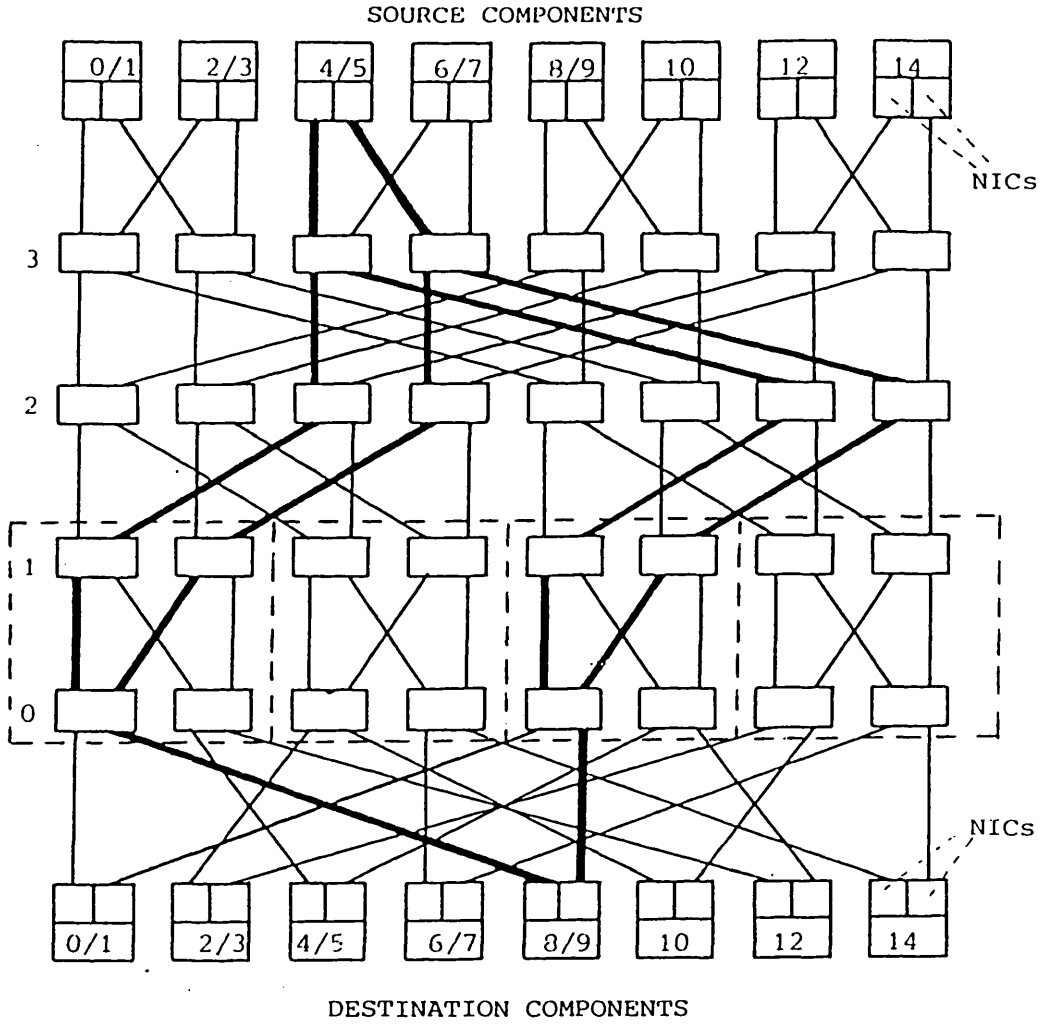


Figure 4-1: A CUBE-Based Fault Tolerant Network. N=32, x=2, k=2.

the obvious way. Four distinct paths are still retained, and each path has an independent counterpart so that the path independence property still holds. Provided the whole array is not replaced by just a single switch, the systematic replacement of subparts of the network by larger single switches will not affect the networks' ability to provide independent paths between each pair of source and destination components.

4.3.2. Error Detection And Fault Avoidance

Because the enhanced interconection topology described above providesfour (generally k^2) mutually independent paths between any pair of source and

destination ndoes, a permanent fault anywhere in the interconnection network can be avoided. The mechanisms for error detection and fault avoidance with the proposed approach are now decribed by means of an example of a fault tolerant network. The techniques for providing permanent fault avoidance are indepedent of the mechanisms for error detection and are described later.

The example set up is shown in Figure 4-2. and considers a self clocking network which is being used to transfer messages or blocks of data unidirectionally from the source components (attached to the input side of the network) to the destination components (attached to the output side of the network). Each system component is shown to have a separate network interfacing component (NIC) for each of its k (in this case 2) network input or output ports. These may, however be combined, and even integrated into the system components themselves. The blocks of data are assumed to be buffered in source NICs before the transaction is initiated into the network. Transferred blocks are further assumed to be buffered in the destination NICs. This yields a guaranteed data transfer rate after path set up which increases the network throughput. Multiple buffers could also be provided so that buffer filling and buffer emptying within an NIC can proceed concurrently. For simplicity, though, we consider only single

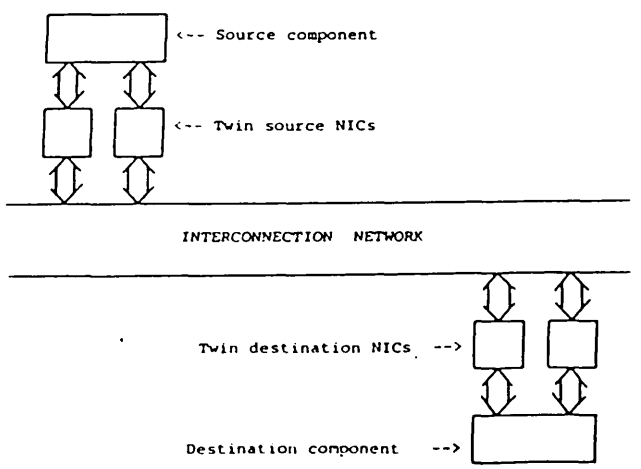


Figure 4-2: Example Fault Tolerant Set up.

buffering. The protocols at the interface between the NIC and system component are undefined, as is the logic required to perform the block transfer.

4.3.2.1. Error Detection Mechanisms.

Error detection in an interconnection network must be performed at three different levels corresponding to the path building, path verification and data transfer phases of a transaction. Path building error detection and path verification are invariant among different implementations. The mechanisms described for error detection during data transfer are dependent on the implementation of Figure 4-2 although they can be readily applied to other set-ups.

4.3.2.1.1. Path building Failure.

Stuck-at faults at a switch or interface terminals, permanent link faults, transient faults in the switches or links of the network and synchronisation failure will often cause a request to become stuck during path building as a result of the corruption of the self-clocking network control signals. Such errors are detected by a timeout mechanism implemented by the source component. The timeout period is a function of both the network size and the maximum network channel 'hold' time and must be greater than the maximum possible network transaction time. The maximum time between initiating a transaction and completing the transaction is equal to:

$$T = \frac{(N-1)s_{max}}{x-1} + Nt_{max} \qquad [4-1]$$

which is the value of T_{n-1} in the recurrence:

$$T_0 = s_{max}+xt_{max}, \quad T_i = s_{max}+xT_{i-1}, \quad i=1..n-1.$$

Here s_{max} is the maximum possible switching time through one switch of the

network (without blocking), N is the size of the network and t_{max} is the maximum time taken between obtaining a path and completing the transfer of the transmitted data. This is a pathological case since it relies on every component trying to reach the same network output port at the same time. However it defines the necessary minimum timeout period which must be set when a transaction is initiated. In the event of a timeout occurring, the 'partly' claimed channel is cancelled and the request is later retried (see below).

4.3.2.1.2. Addressing Failure.

When an end of path signal is generated by a destination node interface, there is no guarantee that the correct destination has been reached. Certain faults in the network may cause a corruption of the routing address information causing the request to be steered to the wrong output. Consequently, immediately after path building is complete, the requested and the obtained destination addresses are compared. The claimed channel is released if any discrepency is observed, and status information is reported back to the source node indicating the nature of the failure.

4.3.2.1.3. Data Transfer Error.

If the address check succeeds then control is passed to the data transfer unit (level 1) in the interface where the block transfer proceeds. Regardless of any errors which may occur during data transfer, the entire block is always transferred from the source NIC to the destination NIC. Because the fault model assumes unidirectional faults in the data transmission path, a Berger check [Ber61] is performed on the data as it is transmitted. The Berger checksum is simply the number of logical zeros within the data block and is optimal in the sense that there is no unidirectional error detecting code with fewer check bits for the same number of data bits transferred. The Berger check has been proposed in

[FAH83] for interconnection networks with conventional control mechanisms but it is particularly suitable for serially-switched networks since the checksum can be accumulated 'on the fly' by means of a counter triggered by the signals transmitted on the forward data/clock path (the D and C control lines) through the network. The checksum logic required is thus very simple. At the end of the block transfer, the received checksum is compared with the transmitted checksum as described below. If the comparison fails then the network channel is released and a status word is made available to the source node. If the comparison succeeds then the transaction is complete and the claimed channel can be released normally.

Note that because data transfer errors are (hopefully) infrequent, the use of continuous block transfer protocols, as opposed to handshaking protocols, will, in the long run, yield higher overall throughput since no redundancy is required. Additionally, block transfers require fewer (if any) synchronisations between the sender and the receiver.
These mechanisms are explained more fully in the next section which describes the operation of the fault tolerant network interfacing component (or NIC).

4.3.2.2. NIC Operation.

The following discussion describes the operation of the NICs from the point at which the buffer of the source NIC has been filled. There may be additional buffers in the NIC which may be filled whilst the current buffer is being emptied; indeed, the block transfer may be done directly from the source main memory to the destination main memory via conventional DMA. This is not important to the discussion. We merely consider the mechanisms for error detection as the block is transferred from the source NIC to the destination NIC.

The error detecting NIC may be viewed as containing three levels of network

control. Level 0 is exactly as described in Chapter 3 except that status information is now maintained to assist the process of fault diagnosis. (Fault location and fault repair are described in 4.3.3.) Level 0 contains the routing address register used by the self-clocking protocols. Writing the destination address (possibly transformed) to this register initiates the transaction by pulling the Reset line LOW into the network. At this same time, the source component begins the timeout. The routing address is also copied to an address comparison register located in the intermediate level of the NIC shown in Figure 4-3.

From now on the self-clocking protocols take effect. Any fault in the network causing the request to become stuck during path building is detected by the timeout in the source node (although the timeout logic may be included in the NIC). During path building the signals present on the A (acknowledge) and B (Burst) lines from the network are monitored and counts of the number of transitions occurring on these lines is maintained in the form of a status register held within the NIC. Additional status information is also held as described below. If the request times out then the source component reads the contents of this register and passes it on to a system monitor where it can be used to assist in fault location. The A count indicates the number of stages traversed to date. The B count should be consistant with the A count and indicates the total number of burst clocks received by, i.e. the total number of address bits transmitted from, the routing address register. Any discrepency in these two counts may assist fault diagnosis.

If path building completes successfully, then an end of path signal is sent by the destination NIC and the signal will be made available in the status register. This signal disables level 0 of the source NIC and enables the network A/B and C/D lines through to the intermediate level where path verification is performed. Following the end of path signal, the

destination NIC immediately transmits its own address (which is written to a register internal to the NIC during system initialisation) back to the source node. The received address is loaded into the internal level of the NIC where it is checked against the intended destination address which was loaded when the transaction was initiated. If the two addresses differ then an error has occurred either during path building or during the transmission of the destination address. Both conditions are errors so in the event, the established channel is released and an interrupt is sent to the source component. The end of path signal and the result of the address check are added to the status register so that appropriate actions for recovery in the event of failure may be undertaken.

If the address check succeeds then control is transferred to the data transmission logic (level 1 of the interface as defined in Chapter 3) and the data is transmitted across the network. Regardless of any errors which may occur during transmission, the entire block is <u>always</u> transmitted. The block is transferred in 'chunks' which may be bytes, words etc., each chunk being preceded by a start bit of one. These are serialised and self-clocked through the network on the C/D data path. Error detection is performed at the source and is achieved by <u>two</u> 'dynamic' Berger checksum counters maintained by level 1 of the source interface as follows:

When the destination address has been sent by the destination NIC for path verification, the destination NIC is set to receive incoming data on C/D and the data received (i.e. the data on D clocked by C) is then sent back to the source node on the A wire. The network is thus configured so that the source NIC receives all data it transmits. The two Berger checksum counters at the source NIC are triggered from the data transmitted on the C/D data path and received back on the A wire. Thus each data bit transferred makes a round trip from the source NIC through the network to

the destination NIC and then back to the source NIC. The last word of the block is preceded by a start bit of zero. After this last word has been transmitted, an additional 'end of block' signal is transmitted whereby a positive edge is provided on D whilst C is held HIGH. When the last word has been received at the destination, the destination NIC is configured to pass this end of block signal back to the source on the (as yet unused) B wire. The receipt of this signal on B at the source end signals that the Berger checksum counters are valid i.e. this mechanism preserves structure independence since the network latency is unimportant. The output from the comparitor is enabled onto the network D wire after the end of block signal has been sent and the B wire is enabled onto the C wire. Thus the received signal on B is immediately sent back to the destination. Consequently at the destination the signal on the C wire will clock a one on the D wire if the checksum succeeded and a zero if it failed. This signal either enables or disables the buffered data according to the result of the checksum.

Note that this scheme is very robust since it is impervious to both permanent and transient faults before the final 'buffer valid' signal is generated. However, this final validation signal must be fault-free. There is no way of guaranteeing a successful acknowledgement at the end of a transaction.

A signal received on the B wire at the source end causes an interrupt to be sent to the source component and, subsequently, the release of the channel. If the B signal was received prematurely then a flag is set in the status register indicating the fault. This aids the process of fault diagnosis.

In addition to the status register, the address validation register, the two Berger checksum counters and the original routing address register can also be accessed by the source component. A description of how this information is used in the event of failure is given in 4.3.3. below.

Observe that:-

1. Once the block transfer is intiated, the communication is 'self-timed' in the sense that no synchronisation is required in either the source or destination NICs. Only two synchronisations are required in total: one to transmit the destination address and one to begin the block transfer, although if the extended protocol described in 3.2.7. is used the first of these can be eliminated by using the burst generator clock at the output of the last stage to clock out the destination address.
2. The use of Berger checksums means that error detection is very simple and fast. Because the checksums are generated 'on the fly', the comparison is made just two network delays after the last data bit has been sent regardless of the size of the network. The Berger checksums are <u>not</u> required to be transmitted through the network.
3. There is no data redundancy required since the source NIC re-recieves all data that it transmits.

4.3.2.3. Finding Fault Free Paths

Under normal operation, both ports of the source and destination nodes can be used. To even the load at the network outputs, alternative requests submitted by the source nodes may use the transformed version of the destination addresses. This, in theory, makes it possible for two transactions to take place between the same source/destination pair concurrrently.

If a fault is detected by the source node during path set-up, the immediate effect will be for the source node to cancel the request (by reading the NIC status register) and then to retry it an arbitrary number of times. If the request now succeeds, then the error was transient. If the retry (or

retries) still fail then there is a permanent fault somewhere in the network which must be avoided. It is an important property of the proposed scheme that neither the location nor the cause of the fault need be known by the source node. The fault may be local to a single slice of one switch, or it may affect a whole (or several) switches, for example as a result of power failure. Furthermore, the fault may lie in a link between two switches or it could even lie in one of the network interfaces, although this is not covered by the fault model.

To avoid a permanent fault, the source component first attempts to use the address transformation to steer around the faulty part of the network. If the failed request used an untransformed address then the request will be retried using the transformed version of the address, and vice versa. Using the transformed (or, correspondingly, untransformed) address causes the request to take a distinct path to that taken by the original attempt, although the request still passes through the same switch in the top stage of the network as before. If this successfully steers the request to its required destination node, i.e. if it successfully avoids the fault, then all subsequent requests submitted to that port will use only transformed (untransformed) versions of the required destination addresses. This is balanced by now using only untransformed (transformed) addresses on the other network port. Thus, even after a fault has arisen, it is still possible for the network to provide two communication channels between all pairs of source and destination nodes. (Compare this with multiple plane networks where only one path would now exist). Note that:

1. Rather than restricting all subsequent transactions to using transformed addresses, a table could be provided in the source node indicating whether or not a transformation must be enforced in communicating with a particular destination node, although this may be costly to maintain.

2. If the fault lies in a switch in the last stage of the network or in the link between the last stage of switches and the destination node interface, then issuing the transformed (untransformed) version of the destination address will cause an error to be detected regardless of which source input port is used. In this case both source ports must issue only untransformed (transformed) addresses in order to avoid the fault.

If on the other hand the retry with (or without) transformation does not successfully avoid the fault, then the transaction can only be completed by using the other network input port associated with the source node. This implies that the fault lies either at the source NIC or at the top level switch to which the NIC is connected, or in the link between the two. By using now the other port, the alternative path obtained is guaranteed to avoid the fault.

Note that if an error status table is not used at the source node, then the network input port on which the fault was detected cannot now be used at all. Consequently, this port must be shut down until the failed device(s) have been replaced (see 4.3.3.).

4.3.2.4. Multiple Faults

Using the above scheme, all single faults in the network switches and links can be tolerated. However, a substantial number of multiple faults can also be tolerated. The only dual faults which cannot be catered for are those which affect both switches (or interfaces) to which a source or destination component are attached. That component then has no access to the network at all. With k=2, the network will still function with up to three permanent faults in the internal stages of the network in the worst case. Many more failures can be tolerated provided they do not result in

all four distinct paths between any pair of source and destination nodes being affected.

4.3.3. Coping With Permanent Faults.

In the previous discussions we have been concerned only with keeping the network operational in the event of a network failure. Over a long period of time, however, multiple errors in the network will make necessary some form of repair.

One approach is to simply wait for multiple faults to cause the network to fail and then instigate the repair by powering off the machine and replacing the faulty switches or connections. However, a far more desirable approach is to make these repairs 'on the fly' i.e. whilst the rest of the system remains operational. The network can then be maintained to be far less susceptible to failure from multiple switch and interconnection faults. In this section we describe a method for doing this which is compatible with the fault tolerant scheme described above. This process of maintainance is referred to as dynamic switch replacement.

4.3.3.1. Fault Location.

One of the benefits of the fault tolerant scheme from the point of view of the systems components is that neither the nature of the failure nor the exact location of the failed switch or interconnection need be known. Thus no fault location facilities need be provided by the components. In order to carry out a repair this information is established by either automatic or manual analysis of error reports transmitted by the components to a central monitoring processor.

All network errors detected by the system components are logged in the form of an error log table, which is transmitted to the central monitor for statistics-gathering and for permanent fault location. The error log table

has the following format:

<Source Node Address>

<Intended Destination Node Address>

<Error Code>

<NIC Status At Time Of Error>

{ <Additional Information> }

The NIC status register has the format shown in Figure 4-3 below:-

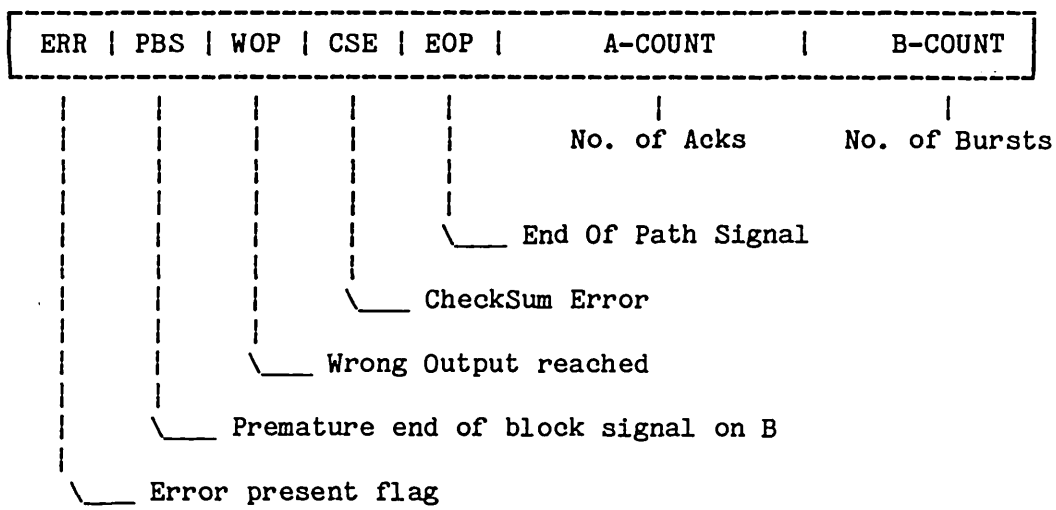


Figure 4-3: NIC Status Register Format.

The error code denotes the type of error recorded i.e. transient, permanent, incorrect routing, timeout etc. If the fault was permanent then the additional information is a dump of all the other NIC registers at the time of the error, together with the isolation mechanism which was used to avoid the fault (address transformation or port closure). If the fault was transient and resulted in the wrong output being reached then the additional information is the address validation register. If the error was a transient data transfer error then the additional information is the contents of the checksum register(s).

If a stuck-at fault or permanent link fault in either of the C or D lines is present between the last stage of switches and the destination node

interface then the block transfer checksum will fail on all requests directed to that destination interface. Note that a received and incorrect destination address of all 0s or all 1s, or a failure to receive an end of path signal when the A and B counts of the status register indicate that path building was complete when a timeout occurred, suggests a permanent fault in the A/B data path between the destination NIC and the last stage of switches. Stuck-at faults or permanent link faults in a C line higher in the network will cause path building failure and hence a timeout. A permanent fault in the A line higher in the network may cause incorrect routing if start bits are used by the self-clocking control protocols and will also inhibit the transmission of the end of path signal which is, again, detected by a timeout. The status register A-count field then indicates the location of the fault. Faults in a D line occurring anywhere other than at the outputs to the last stage will cause requests to be steered to the wrong network output. This is detected by the path verification logic. Permanent faults in the R line of the network will cause a data transfer error if the fault occurred after path building and a timeout error in all other cases.

Once the fault has been approximately located (observe that the exact location of the fault is often not possible although at most two devices (or their interconnections) will be candidates), the fault can be isolated and subsequently rectified by manual replacement.

4.3.3.2. Isolating Faulty Switches.

The problem of affecting a dynamic repair using the fault tolerant scheme described above is that although one particular switch port (or subset of switch ports) or the interconnections between two switch ports may have become unusable, the rest of that switch may still be operational. A partly faulty switch, therefore, might not be completely isolated. Before

any form of replacement can be undertaken, the switch must be made inaccessible from all of the source components in the system.

Consider again the extended Generalised Cube network of degree x with k=2, as described above. In this topology each source and destination component has two links to the input and output ports of the network respectively, as shown in Figure 4-1. Now label these two ports respectively the 'even' and 'odd' ports according to whether the corresponding network input (output) port to which they are attached is even or odd numbered after the input (before the output) permutation has been applied. Then:-

Lemma 4-5: The source node ports to which an arbitrary switch in stage n-1 of the network is attached are all labelled either 'even' ports or 'odd' ports.

Proof: Consider an even numbered switch $\langle i_{n-1},..,i_2,i_1\rangle$, $i_1=\langle b_{x-1},..,b_1,0\rangle$, $b_i \in [0,1]$, i=1..x-1, in stage n-1 of the network. The network input ports associated with this are thus the $\langle i_{n-1},..,i_2,i_1,\theta\rangle$, θ=0..x-1. Now, applying the inverse of the G_1 permutation function to the port addresses maps them respectively onto the source inputs numbered $\langle i_{n-1},..,i_2,\theta,i_1\rangle$, θ=0..x-1. Since b_0 of i_1 is 0, these are all even. Similarly, if we consider only odd numbered switches, then b_0 of i_1 will be 1 and hence the source output ports these switches connect to will all be odd.

[]

Lemma4-6: The destination node ports to which an arbitrary switch of stage 0 of the network is attached are all labelled either 'even' ports or 'odd' ports.

Proof: Is analagous to that of Lemma 4-5.

[]

We now wish to shaow that the set of switches of stages 1 to n-1 (inclusive) which can be accessed from the even ports of all the source nodes is disjoint to that which can be accessed from all the odd ports.

Thus:- Let L(j) be the set of switches of stage n-1 linked to from input j (before the G_1 permutation has been applied)

Theorem 4-4:↑ Let E and E' be respectively the sets of all even and odd ports. Let ϵ be the set of even numbered switches of stage n-1 and let ϵ' be the set of odd numbered switches of stage n-1. Let β_k be the set of switches in stage 1 to n-1 which are accessible from input port k (before the G_1 permutation is applied), and let D_k be defined as in Lemma 4-3. Finally, let Σ and Σ' be given by:

$$\Sigma = \bigcup_{e \in E} \beta_e \qquad \text{and} \qquad \Sigma' = \bigcup_{e' \in E'} \beta_{e'}$$

Then: $\Sigma \cap \Sigma' = \emptyset$.

Proof: For all $j \in E$: j & $j' \in E'$ are both associated with the same source node. From Lemma 4-5, $L(j) \in \epsilon$, and $L(j') \in \epsilon'$. From Lemma 4-3, $\beta_j \cap \beta_{j'} = \emptyset$. This implies that $D_{L(j)} \cap D_{L(j')} = \emptyset$. Thus, since $\epsilon \cap \epsilon' = \emptyset$ it follows that $\Sigma \cap \Sigma' = \emptyset$.

[]

And, similarly for address transformation:

Theorem 4-4: The set of switches of stages 0 to n-2 which are accessible by issuing only even destination addresses from all the source nodes is disjoint from that accessible by issuing only odd destination addresses from all the source nodes.

Proof: The proof of this is analogous to that of Theorem 4-3, using Lemma 4-3 and Lemma 4-6.

[]

Thus, by instructing all source nodes to either stop transmitting on their even or odd ports, or to submit only even or odd destination addresses to the network, any single fault in the network can be fully isolated and

therefore replaced.

The isolation mechanism is as follows:-

If the fault potentially lies in stage n-1 of the network (including the links between stage n-1 and the source node interface, and between stage n-1 and stage n-2) then either the even port or the odd port of each source node must be temporarily shut down depending on whether the fault was detected by an even or an odd port in the first place. This information is held in the error log record transmitted at the time of the error was detected. This caters for faults local to one slice (plus its interconnections) of a switch in stage n-1 of the network which were originally avoided by using address transformation. In all other cases, address tranformation will have caused all subsequent requests submitted from the failing source port to access a different subset of the switches of stages 0 to n-2 to that which contained the fault. Thus, from Theorem 4-4, the fault can be made inaccessible from all the source nodes by having each source node temporarily restrict the parity of all destination addresses transmitted. The error log record indicates whether only even or odd destination addresses should be used.

Note that if the fault is known to lie between stage 1 and stage n-1 (inclusive) then, from Theorem 4-3, either isolation mechanism can be used to isolate the fault from all the source nodes. If port closure is to be used, however, then the information on which port to shut down is not available in the error log record since the fault will have been avoided using address transformation. This information must be determined from the location of the fault and from the network topology.

Note that when rectifying multiple faults in the network, it may be necessary to perform the replacements in separate operations. There are some cases where the isolation commands transmitted to each source node

will prevent a source/destination transfer from being performed during the transplant. This occurs when, during the transplant, the only available path from a source node to a destination node is via another faulty device or interconnection which has already been made inaccessible from the source node. In such cases the source node must idle until the repair is complete. Of course, if the system is maintained so that only single faults are ever present then this situation will never arise!

4.3.3.3. Fault Repair.

The problems of physically accessing and replacing a faulty device in the network is closely allied to the problems of packaging. The network cards must have all their external connections provided by flexible cable to allow the cards to be removed without affecting unisolated devices present on the same card. Furthermore, power transistors must be provided for each switch to enable individual switches to be powered off and replaced independently of other switches on the same card. ZIF mounting of ICs is also desirable. Because the devices are powered off when they are replaced, these interconnections can also be tested during the repair. The issues of packaging are not covered further.

Once a repair has been made, a message can be sent from the system monitor to all the system components instructing them to resume normal operation. At this point the network will function correctly and any restrictions imposed on port use or destination address parity can be lifted, and the default operation mode resumed.

4.4. Summary.

This chapter has been concerned with the problem of fault tolerance in interconnection networks, with particular attention paid to self-clocking network implementations. In the proposed scheme, fault tolerance is broken down into three steps:-

1. Error detection
2. Fault Avoidance, and
3. Fault Repair.

Error detection in the network is performed by the system components themselves (catering for 'stuck' requests), and by the network interfaces, which are extended to include logic for path verification and dynamic error detection during the data transfer phase of a transaction.

Fault avoidance is achieved by incorporating an extension to the original network topology which guarantees the existance of multiple and independent paths through the network between arbitrary pairs of source and destination nodes. A novel feature of this scheme is that fault avoidance can be achieved without requiring any knowledge as to the location or the nature of the fault. The affected source component simply retries the request using address transformation or using its additional network input links (or both) until the request succeeds. In much the same way that self-clocking protocols yield structure independence, so the fault tolerant scheme suggests some notion of fault independence.

Indeed, structure independence can be preserved using the fault tolerant scheme. The address transformation scheme described in 4.3.1.1. functions by manipulating the lower end of the network routing addresses, the network address bits are clocked out from the least significant end of the register and the network implements the full suite of self-clocking protocols during path building. The only network-dependent parameter required to be known

is the timeout period which is a function of both the network size and the maximum data transfer time (t_{max} in equation 4-1).

A further important consequence of the fault tolerant scheme is that is simple to implement in terms of hardware. The use of self-clocked data in data transmission, and the use of twin dynamic Berger checksums makes the error detection logic very simple, very fast, and very reliable! Finally, the ability to dynamically replace faulty network components minimises the networks susceptibility to unavoidable multiple faults and hence maximises network availability.

CHAPTER 5

Summary And Conclusions

This thesis has examined how a class of dynamic multistage interconnection networks may be employed to provide arbitrary communication in very large parallel computer systems which may contain many hundreds, thousands or even tens of thousands of component processors.

We have variously exmamined the issues of performance, network design and integration, and network fault tolerance.

This work was primarily motivated by recent developments in so called 'declarative language support architectures' which promise to offer not only an improved programming environment, but also the ability to exploit very high degrees of parallelism in execution. Despite this relatively new approach to computation, many of the problems associated with conventional parallel processing systems remain. This thesis has been concerned with what is arguably the most critical aspect of these systems, namely that of copmponent interconnection and communication.

Chapter 2 described the general characteristics of the class of dynamic interconnection networks with which the thesis is concerned. This included sections on network performance analysis and the performance of a number of network configurations was investigated. A new class of interconnection networks - Lambda networks - were introduced which provide the same low cost global communication, as is provided by conventional interconnection networks, yet which also offer the potential to exploit locality of reference. From performance models of these and conventional networks, we demonstrated that under a random addressing distribution the

performance of Lambda networks is superior to conventional interconnection networks and that the performance increases in accordance with the degree of locality which can be preserved in the addresing scheme. It remains to be seen whether or not this locality can be effectively exploited by the surrounding system. Considerable work has yet to be done in this area which will involve a detailed study of 'typical' program behaviours, and run-time load balancing and data distribution mechanisms.

Chapter 3 examined the issues of network design and integration. A number of interconnection network designs have been offered in the literature, but we have argued that these designs are only practical for small and medium-scale parallel processing systems. Highly parallel systems containing very large numbers of processing devices have been proposed based upon these designs, but in reality these lead to unmanagable problems of cost, size, construction and wiring complexity and, perhaps most significantly, reliability. We believe that if interconnection networks are to be practical in very large-scale systems, then the effort should be put into reducing complexity rather than increasing it.

We proposed two design techniques aimed at overcoming the complexity of existing designs:-

1. Serial switching, which ensures maximal utilisation of available pins (which is the limiting factor in any switch design).
2. Self clocking of the network switches, which offers almost unlimited flexibility in the choice of network topology, size and configuration, and which overcomes the problems of bit-serial network control and many of the problems associated with asynchrony.

Networks incorporating these principles are inherently smaller, cheaper, more cost-effective and more reliable, and are naturally asynchronous so that the problems of global clock distribution do not arise.

Despite the adoption of very narrow data paths through the proposed network, very high performance can still be maintained. Chapter 3 described how future implementations based upon tri-level logic encoding techniques can reduce the control/data path width to just three wires, thereby allowing large switches to be built, and can maximise the serial transmission rates by eliminating relative signal skewing. As device speeds increase, skewing will become a limiting factor in performance. By adopting 'self-clocked' single line data paths, the network performance is limited only by technology rather than by the physics and practicalities of providing matched transmission lines.

The self-clocking and serial-switching design techniques were substantiated by describing an implementation of a network based around a custom-designed network switching chip called the XS1. A network of XS1s is being used for component interconnection in the ALICE graph reduction engine prototype which has been developed at Imperial College.

In addition to network design, we also raised the issue of fault tolerance, which to date has received comparatively little attention. We have stressed that fault tolerance is vital in an interconnection network where very large numbers of switching elements, and even larger numbers of network interconnections are required. Despite a growing awareness of the need for fault tolerance in interconnection networks, the existing proposals have all been focused on synchronous network implementations, and most cover only a subset of the fault models which are relevent to VLSI network implementations.

In the proposed scheme, fault tolerance is viewed as consisting of three stages: error detection, fault avoidance and fault repair. The error

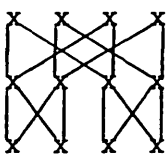
detection mechanisms we have proposed are fast, simple, and effective and are particularly well suited to serially-switched network implementations.

Fault avoidance is achieved by providing multiple paths between each pair of source and destination nodes by means of an extension to the network topology which exploits the general partitioning properties of interconnection networks. The resulting fault tolerant topology guarantees fault-free paths in the event of all single network failures and a rich variety of multiple network failures. Furthermore, the fault avoidance mechanisms require no knowledge of the nature or the location of the fault. We argue that this is critical since in many cases the exact location of a fault causing a permanent error condition cannot be determined. Thus, in keeping with the ideas of structure independence resulting from self-clocking techniques, the fault avoidance scheme may be considered as being fault independent. The scheme has been tailored for self clocking networks where the asynchrony of the network means that fault avoidance must be achieved on a 'per-component' basis, i.e. independently of other components in the system.

The fault tolerant interconnection topology also allows the dynamic on-line replacement of permanently faulty switches and the repair of faulty connections. We have described how by using status information from the network interface a single permanently faulty switch or interconnection can be located and then both electrically and logically isolated from the rest of the network and replaced, without the need to power off the machine, or destroy the full point to point interconnectivity of the remainder of the network.

Research into parallel systems architecture indicates that interconnection networks will have an important part to play in many future concurrent processing systems. However, there is a danger that with the rapidly

diminishing fabrication costs of integrated ciruit components, the interconnection network cost will eventually dominate the machine cost and will contribute to a majority of the machines' bulk, manufacturing complexity and operational unreliability. Unlike the system components where progressively more of the interconnection can be placed in silicon, the interconnection network benefits less from VLSI, primarily because the number of I/O pins required to build a switch is directly proportional to the degree of the switch. In this thesis, we have taken a more pragmatic look at the feasibility of interconnection networks than has been done to date. We conclude that although existing design techniques present problems of unmanagable cost and complexity, there is still much that can be done to bring this complexity under control. By adopting a very different and more economical approach to switch design and by incorporating modest and cost-effective extensions to the network to cater for faults, it is possible to develop very large networks which are cheap and compact, fast, highly reliable and which are well suited for integration into many present and future concurrent processing systems.



REFERENCES

[ADI83] Arvind, M.L. Dertouzos, R.A. Iannucci, "A Multiprocessor Emulation Family", Tech. Rep £ 302, MIT Lab. for Comp. Science, Sept. 1983.

[AdS82] G.B. Adams III & H.J. Siegal. "The Extra Stage Cube: A Fault Tolerant Interconnection Network For Supersystems", IEEE Trans. Comp. Vol. C-31 No.5 May 1982. pp 443-454.

[Agr82] D.P. Agrawal, "Testing and Fault Tolerance of Multistage Interconnection Networks", IEEE Computer, April 1982, pp 41-53.

[Agr79] D.P. Agrawal, "A Duplex System With Improved Performance", Proc. 1979 Conf. Information Science & Systems, March 28-30, pp 333-336.

[Bac77] J.F. Backus, "Can Programming Be Liberated From The von Neumann Style?", ACM Turing Award Lecture, Comm. ACM 21 (8), 1977, pp 613-641.

[Bar81] G.H. Barnes, "Design And Validation Of A Connection Network For Many-Processor Multiprocessor Systems". Computer, Dec. 1981, pp 31-41.

[Ben65] V.E. Benes, "The Mathematical Theory Of Connecting Networks", Academic Press, N.Y., 1965.

[Ber61] J.M. Berger, "A Note On Error Detection Codes For Asymmetric Channels", Information And Control, Vol. 4, March 1961, pp 68-73.

[BhA82] L.N. Bhuyan & D.P. Agrawal, "A General Class Of Processor Interconnection Strategies", 9th Ann. Symp. Comp. Arch., 1982, pp 90-98.

[BuD77] R.M. Burstall & John Darlington, "A Transformation System For Developing Recursive Programs", JACM, 24, 1, 1977, pp 44-67.

[Bur75] W.H. Burge, "Recursive Programming Techniques", Addison-Wesley, 1975.

[BuS81] F.W. Burton & M.R. Sleep, "Towards A Zero-Assignment Parallel Processor", 2nd Int'l Conf. on Dist'd Computing Systems, April 1981, pp 80-85.

[Clo53] C. Clos, "A Study Of Non-Blocking Switching Networks", Bell System Technical Journal, Vol.32, 1953, pp 406-424.

[CrF83] M.D. Cripps & A.J. Field, "The MARCH HARE Network Switching Device", Internal Report, Dept. Computing, Imperial College, May 1983.

[CSS73] R.W. Cook, W.H. Sisson, T.G. Stoney & W.N. Toy, "Design Of Self-Checking Microprogram Control", IEEE Trans. Comp., Vol.C-22, No.3, March 1973, pp 255-262.

[DaR81] John Darlington & M.J. Reeve, "ALICE: A Multiprocessor Reduction Machine For The Parallel Evaluation Of Applicative Languages", Proc. 1981 ACM/MIT Conf. on Functional Programming & Computer Architecture.

[Dar81] John Darlington, "The Structured Description Of Algorithm Derivations", Invited Paper, Int'l. Symp. on Algorithms, Amsterdam, 1981.

[DiJ81] D.M. Dias & J.R. Jump, "Packet Switching Interconnection Networks For Modular Systems", Computer, Dec.1981.

[DiJ82] D.M. Dias & J.R. Jump, "Augmented And Pruned N LOG N Multistage Interconnection Networks: Topology And Performance", Proc. Int'l Conf. on Parallel Processing, 1982, pp 10-12.

[FAH83] W.K. Fuchs, J.A. Abraham, K.H. Huang, "Concurrent Error Detection In VLSI Interconnection Networks", Proc. 10th Int'l Symp. Comp. Arch., 1983, pp 309-315.

[Fen73] T.Feng, "Parallel Processing Characteristics and Implementation of Data Manipulating Functions", Rome Air Development Centre report, RADC-TR-73-189, July 1973.

[Fen82] T. Feng, "A Survey Of Interconnection Networks", Computer, Dec. 1982, pp 12-27.

[FuH78] S.H. Fuller & S.P. Harbison, "The C.mmp Multiprocessor", Technical Report No. CMU-CS-78-148, Dept. Computer Science, Carnegie-Mellon University, 1978.

[Gol73] L.R. Goke & G.J. Lipovski, "Banyan Networks For Partitioning Multiprocessor Systems", Proc. First Ann. Computer Architecture Conf., Dec. 1973, pp 21-28.

[GWG80] J.R. Gurd, I.Watson & J.R.W Glauert, "A Multilayer Dataflow Computer Architecture", Internal Report, Dept. Comp. Sci., Univ. Manchester, 1978.

[Har83] P.G. Harrison, "Lecture Notes: Performance Modelling", Dept. Computing, Imperial College, 1984.

[Hen80] P. Henderson, "Functional Programming: Application And Implementation", Prentice-Hall International, 1980.

[IMS84] INMOS Ltd., "The occam Programming Language" and "The T424 Transputer", INMOS Publications, 1984.

[JuD81] J.R. Jump & D.M. Dias, "Analysis And Simulation Of Buffered Delta Networks", IEEE Trans. Comp., Vol. C-29, No. 9, Sept. 1980, pp 791-801.

[KDJ83] M. Kumar, D.M. Dias & J.R. Jump, "Switching Strategies In A Class Of Packet Switching Networks", Proc. 10th Int'l Symp. Comp. Arch., 1983, pp 284-285.

[KeL78] R.M. Keller, G. Lindstrom & S. Patil, "An Architecture For A Loosely-Coupled Parallel Processor", Technical Report, Dept. Comp. Sci., Univ Utah, Tech. Rep. £ UUCS-78-105-1978.

[KiA78] L.L. Kinney & R.G. Arnold, "Analysis Of A Multiprocessor System With A Shared Bus", CACM V. 21, No. 8, Aug. 1978.

[Kob78] H. Kobayashi, "Modeling and Analysis", Addison-Wesley Publishing Co., 1978.

[Kuh80] R.H. Kuhn, "Efficient Mapping Of Algorithms To Single Stage Interconnections", Proc. 7th Ann. Symp. Comp. Arch., 1980 pp 182-9.

[Lan76] T. Lang, "Interconnections Between Processors And Memory Modules Using The Shuffle-Exchange Network", IEEE Trans. Comp., Vol. C-25, No. 5, May 1976, pp. 496-503.

[LaR80] S. Lavenberg & M. Reiser, "Stationary State Probabilities at Arrival Instants For Closed Queueing Networks With Multiple Types Of Customers", Journal Applied Probability, 17, pp 1048-1061, 1980.

[LaS76] T. Lang & H.S. Stone, "A Shuffle-Exchange Network With Simplified Control", IEEE Trans. Comp., Vol. C-25, No. 1, Jan. 1976. pp.55-65.

[LaV82] D.H. Lawrie & C.R. Vora. "The Prime Memory System For Array Access", IEEE Trans. Comp.Vol., C-31 No.5 May 1982. pp 435-443.

[Law75] D.H. Lawrie, "Access And Alignment Of Data In An Array Processor", IEEE Trans. Comp., Vol. C-24, No. 12, Dec. 1975. pp. 1145-1155

[Len78] J. Lenfant, "Parallel Permutations Of Data: A Benes network Control Algorithm For Frequently Used Permutations", IEEE Trans. Comp. Vol C-27, No. 7, July 1978, pp 637-647.

[LiW82] W. Lin & C.Wu, "Design of a 2x2 Fault-Tolerant Switching Element", Proc. 9th Ann. Symp. Comp. Arch., 1982, pp 181-189.

[LLY82] J.E. Lilienkamp, D.H. Lawrie & P-C. Yew, "A Fault Tolerant Interconnection Network Using Error Correcting Codes", Proc. 1982 Int'l Conf. on Parallel Processing, Aug. 1982, pp 123-125.

[LVA82] T. Lang, M. Valero & I. Alegre, "Bandwidth Of Crossbar And Multiple Bus Conenctions For Mulitprocessors", IEEE Trans. Comp. Vol. C-31, No. 12, Dec. 1982.

[Mag79] G.A. Mago, "A Network Of Microprocessors To Execute Reduction Languages", Intl' Journal Of Comp. & Inf. Sciences, Vol.8, No.5, 1979.

[MAS81] R.J. McMillen, G.B. Adams & H.J. Siegal, "Performance And Implementation Of 4x4 Switching Nodes In An Interconnection Network For PASM", Proc. Int'l Conf. on Parallel Processing, 1981, pp 229-234.

[MeC80] C. Mead & L. Conway, "Introduction To VLSI Systems", Addison-Wesley, 1980.

[McS80] R.J. McMillen and H.J. Siegal, "MIMD Machine Communication Using The Augmented Data Manipulator Network", Proc. 7th Ann. Symp. Computer Architecture, June 1908, pp 51-58.

[MSi80] R.J. McMillen & H.J. Siegal, "The Hybrid Cube Network", Proc. Symp. on Dist'd Data Acquisition & Control, 1980, pp 11-22.

[McS81] R.J. McMillen & H.J. Siegal, "Dynamic Rerouting Tag Schemes For The Augmented Data Manipulator Network", Proc. 8th Int'l Conf. Comp. Arch., 1981, pp 505-516.

[MST79] S.R. McConnel, D.P. Siewiorek & M.M. Tsao, "The Measurement And Analysis Of Transient Errors In Digital Computer Systems", Proc. 1979 Intl' Symp. Fault Tolerant Computing. June 1979, pp 67-70.

[MuM82] T.N. Mudge & B.A. Makrucki, "Probabilistic Analysis Of A Crossbar Switch", Proc. 9th Ann. Symp. Comp. Arch., 1982, pp 311-320.

[NaS80] D. Nassimi & S. Sahni, "A Self Routing Benes Network", Proc. 7th Int'l Conf. Comp. Arch., 1980, pp 190-195.

[NaS82] D. Nassimi & S. Sahni, "Parallel Algorithms To Set Up The Benes Permutation Network", Proc. Workshop on Interconnection Networks for Parallel & Dist'd Processing, Wst. Laf. IN., 1980, pp 70-71.

[PaR82] D.S. Parker & C.S. Raghavendra, "The Gamma Network: A Multiprocessor Interconnection Network With Redundant Paths", Proc. 9th Int'l Conf. Comp. Arch., 1982, pp 73-80.

[Pat79] J.H. Patel, "Processor-Memory Interconnections For Multiprocessors" Proc. Sixth Annual Syp. Computer Architecture, April 1979, pp 168-177.

[Pea77] M.C. Pease III, "The Indirect Binary n-Cube Microprocessor Array", IEEE Trans. Comp., Vol. C-26, No. 5, May 1977. pp. 458-473.

[Sie79] H.J. Siegal, "Interconnection Networks For SIMD Machines", Computer, Vol.12, No. 6, June 1979, pp 57-66.

[Sie80] H.J. Siegal, "The Theory Underlying The Partitioning Of Permutation Networks", IEEE Trans. Comp., Vol C-29, No. 9, Sept. 1908.

[SiM81] H.J. Siegel, R.J. McMillen, "The Multistage Cube: A Versatile Interconnection Network", Computer Dec. 1981 pp 65-75.

[Sto71] H.S. Stone, "Parallel Processing With The Perfect Shuffle", IEEE Trans. Comp., Vol. c-20, No.2, Feb. 1971, pp 153-161.

[Tho70] J.E. Thornton, "The Design Of A Computer: The Control Data 6600", Scott, Foresman & Co., 1970.

[Wir83] N. Wirth, "Programming In Modula-2", Springer-Verlag 1983.

[WLL82] C-L Wu, W. Lin & M-C Lin, "Distributed Circuit Switching STARNET", Proc. Int'l Conf. on Parallel Processing, 1982, pp 26-33.

[WuF79] C. Wu, & T. Feng, "Fault Diagnosis For A Class Of Multistage Interconnection Networks", Proc. 1979 Int'l Conf. Parallel Processing, pp 269-278.

[WuF80] C. Wu & T. Feng, "On A Class On Multistage Interconnection Networks", IEEE Trans. Comp., Vol. C-29, No. 8, Aug. 1980, pp 694-702.

[WuF81] C. Wu, & T. Feng, "The Universality Of The Shuffle-Exchange Network", IEEE Trans. Comp., Vol. C-30, No. 5, May 1981.

[YeL81] P-C. Yew & D.H. Lawrie, "An Easily Controlled Network For Frequently Used Permutations", IEEE Trans. Comp., Vol. C-30, No. 4, April 1981.

APPENDIX 1

A Routing In Lambda Networks.

Routing in a Lambda network is based upon relative addressing. The routing addresses and control information supplied to the network is derived from both the source and destination port addresses. This differs from traditional networks where the destination address only is sufficient to control network switching. Consequently, each source node may generate different addressing information in order to steer to the same destination. The following section describes a routing algorithm for Cube-based Lambda networks based upon the self-clocking design techniques described in Chapter 3.

It is the purpose of the routing algorithm to produce routing addresses for both the upward and downward network traversals. The network control information required to traverse a Lambda network is assumed to consist of a (possibly empty) sequence of 1s (one for each upgoing stage traversed), followed by a 0 (to denote 'turn round here'), and then a bit pattern corresponding to the address required to steer the request down through the network to its destination. As each stage is traversed going up, successive bits of the routing address are picked off and supplied as control information to the switches in the network. A 1 indicates that the request should be steered upwards, and a 0, that it should pivot and begin traversing the downward path. Once the pivot has occurred, the remaining bits of the routing address represent the switch settings for the downward path. For example, referring back to Figure 2-3, the path from source 001 to destination 110 is given by

```
      1  1  1    0    1  1  0
                  ^
go up 3 stages    |  select downward path with 110
                  |
          turn around
```

A Software Algorithm

The algorithm to generate this address string may be easily expressed as a procedure which takes the source and destination addresses as inputs and returns the necessary routing address, R.

To generate the routing address for a transaction between source $S=\langle s_{n-1},s_{n-2},..,s_1,s_0\rangle$ and a destination $D=\langle d_{n-1},d_{n-2},..,d_1,d_0\rangle$, the principle problem is to locate the stage at which the request pivots and begins its downward path towards D. This is done by comparison of the componentdigitsofS and D:-

If S=D, the network may be bypassed altogether. Here, S and D are said to be in direct locality. If $S\neq D$, then a number of upgoing traversals of the network must be made in order to make D visible to S.

If $s_i=d_i$, i=1..n-1, $s_0\neq d_0$, then D must lie at one of the other x-1 downgoing outputs of the stage 0 switch to which S is attached. So, the request can immediately 'turn round' and be steered out through the downgoing ouptut link, d_0, of this switch.

If $s_i=d_i$ i=2..n-1, $s_1\neq d_1$, then traversing only one switch in the upward direction is not sufficient to make D visible from S. Consequently, the request must be steered further into the network by routing it up into one of the (x) stage 1 switches attached to the stage 0 switch associated with S. From this switch D is then visible. A downward path is then traversed by selecting this stage 1 switch with d_1 and the next (stage 0) switch reached with d_0.

Generally, if $s_i=d_i$ i=k..n-1, $s_{k-1}\neq d_{k-1}$, then the request must be passed upwards k stages before being turned around, and the address digits $d_{k-1},d_{k-2},..,d_1,d_0$ supplied, in that order, to each switch on the downgoing path as it is traversed. This is like routing through a subset of a

conventional interconnection network.

Note that if a request is ever required to traverse <u>all</u> n-1 stages of the network in the upward direction, then D must lie in the opposite half of the system to S. Consequently, upgoing paths out of the topmost stage of the network have only to connect to <u>any</u> of the topmost switches into the opposite half of the network to complete the connectivity of the system. The permutation given in 2.2.3.1. is one of many possible permutations.

A software algorithm for generating the routing control string in regular Lambda networks when S≠D is given below:-

```
Stack := 0
P := 0
y := log2x
Source >> y
(Destination >> y) -> Stack
WHILE Source≠Destination
  P PLUS 1
  Source >> y
  (Destination >> y) -> Stack
ENDWHILE
IF P=n THEN
  Stack << 1
  P MINUS 1
ELSE
  0 ->> Stack
TO P
  1 ->> Stack
ENDTO
```

Comment >> and << represent respectively right and left logical shift operations. (x >> y) -> z results in x and z both being shifted right by y places such that each bit shifted out of the least significant end of x is subsequently shifted into the most significant end of z. x ->> y causes bit x to be shifted into the most significant end of y from the left.

Algorithm A1-1: Routing Address Generation in the Lambda Network.

Note that the resulting address is read left to right: the first control bit is held in the most significant end of the stack. Table A1-1 below shows the routing addresses generated for all destinations by source node 6

on a Lambda network of size 16 and degree 2.

Source=0110_2 2N=16

Destination	Address Generated	
0000	110000	
0001	110001	
0010	110010	
0011	110011	
0100	1000	
0101	1001	
0110	--	{destination = source!}
0111	01	
1000	111000	
1001	111001	
1010	111010	
1011	111011	
1100	111100	
1101	111101	
1110	111110	
1111	111111	

Table A1-1: Lambda Network Routing Addresses for All Paths From Source 6.

Note that it is possible to explicitly include the extra (virtual) stage of switches absent in a Lambda network to form a Lambda* network which provides the full x^n input/output ports. This, too, is controllable using the algorithm above. A simple modification to the algorithm makes it suitable for controlling such a configuration. The only difference here is that the 0 'pivot' bit is required in all control strings.

Implementation In Hardware

Although the algorithm given above can be used to generate the network routing control string, its evaluation may incur a computation time overhead which is significant relative to the total network transaction time. The algorithm can, however, be implemented in hardware using the self-clocking techniques described in Chapter 3:-

Figure A1-1 shows the structure of a self-clocking Lambda network switching element. The switch is logically divided into upgoing (distribution) and

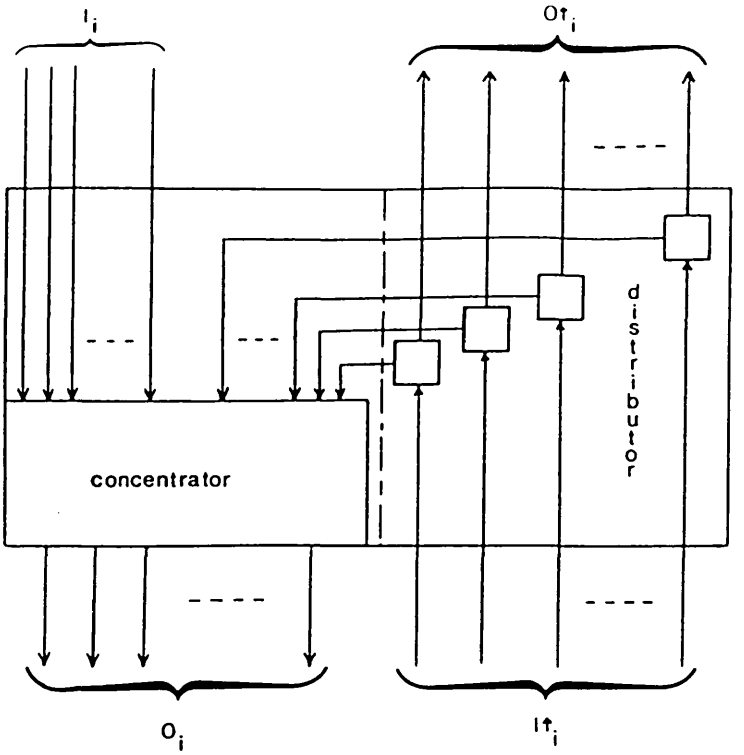


Figure A1-1: A Lambda Network Switch.

downgoing (concentration) logic. The distribution logic simply consists of 1:2 demultiplexers: if a request arriving on $I\hat{}_j$ pivots at the switch then it is demultiplexed into the switch concentrator; if the request does not pivot in the switch then it passed out on $O\hat{}_j$ where it ascends to a switch in then next highest stage of the network. The concentrator is similar to the crossbar matrix in a conventional network except that here the matrix concentrates 2x input ports onto x output ports.

The algorithm given above may be easily implemented in hardware in the network interface (or NIC). The NIC is required to contain three registers, named S, D and K for Source, Destination and stacK. The S and D registers are right-to-left shift registers and the K register is a two-way shift register. Each switch port consists of five wires, A,B,C,D and R which have the same meanings as for the XS1 described in Chapter 3.

At the start of a transaction the S register is loaded with the source address automatically from a permanent holding register, and the D register

is loaded with the destination address via an interface to the source component intiating the transaction. The K register is zeroised. When the registers have been loaded the interface lowers the R line into the network. The (stage 0) switch to which the interfacing component is attached responds by transmitting b burst clocks on the B line where b is the logarithm (base 2) of the degree of the switch. This burst clock is coupled to the clock inputs of the three registers. On each clock edge, S, D and K are shifted rightwards one place such that each bit overflowing from the least significant end of D is clocked into the most significant end of K. Throughout the shifting process, the contents of the S and D registers are compared. The output from the comparator is enabled onto the network D wire and the incoming burst clock from the network is enabled onto the C wire. When the final (bth) clock bit on C has been received at the switch an acknowledge signal is issued back to the interface on the A wire and the signal on the D wire is examined. This signal is:-

$$(s \gg b) = (d \gg b)$$

where s and d are respectively the source and destination addresses and 'M >> n' means shift M right n places.

If the S=D signal is HIGH then s and d differ <u>only</u> in their least significant b bits. If this is the case then d is visible from the stage 0 switch attached to s and the request must pivot at stage 0. If S=D is LOW then d is not yet visible from s and so must be fed higher into the network.

The signal on the D wire after the receipt of the b burst clocks is thus the 'pivot' signal; D=HIGH meaning 'pivot'. If the request is to pivot then, since the switch itself provided the burst clock, the information now held in K is sufficient to select the switch in the downward direction. The distributor in the switch couples the five wires on its input to the corresponding input of the concentrator. Thus the concentrator 'sees' a

transition on its R line and begins the normal self-clocking routing cycle as described in Chapter 3. At the interface, the K register is now set up to shift leftwards with the overflow from the most significant end of the register being enabled onto the network D wire and the S and D registers are disabled. The K register now behaves as the routing address register and the interface functions exactly as would a normal self-clocking controller.

If the request must be steered further into the network i.e. if the pivot signal is LOW at the switch and interface, then the distributor simply couples the fives wires (say on $I\hat{}_j$) to $O\hat{}_j$ and the cycle repeats, this time with the interface in communication with a switch in stage 1.

In a Lambda* implementation the S and D registers are bound to be equal at some time since there can be a maximum of m burst clocks and since each address is only m bits long. In a simple Lambda network, the interface must detect if the topmost stage has been passed (i.e. this is an <u>implicit</u> pivot). This could be done by simply inverting the acknowledge signal in the normal self-clocking protocol. Then, when the top stage is selected and the request is fed into a downgoing input link in the opposite half of the network, a double acknowledge will be observed at source. This could be used to indicate an implicit pivot. Alternatively, an extra contol line could be provided which is set HIGH only when an implicit pivot occurs. Observe that the routing mechanism preserves <u>structure independence</u> as described in Chapter 3 since at each stage going up sufficient bits are stacked in K to enable the network to be correctly addressed following the pivot. However, the switches of a given degree must all be of fixed degree.

APPENDIX 2

Network Simulator Decription

A number of simulators have been written for verifying theoretical models of network performance. The simulator described here is the largest of this suite of simulators and has been used extensively for predicting the performance of self-clocking networks with varying physical and operational parameters.

The simulator is event driven and there is one event for each of the major processes associated with network control and data transfer, as well as additional ('snap-shot') events solely responsible for accumulating statistics. The simulator is designed to be run interactively although a batch version exists for analysing very large network configurations.

When the simulator is run the initial network configuration is prompted for. Once the operational parameters have been defined, a number of optional commands can be issued to vary the degree of tracing and statistics-gathering performed during a simulation run. Help information is available on all commands. The simulator is begun by issuing the RUN command followed by the number of requests to be passed before termination. When this number of requests have been completed a results summary is displayed and control returns to the command interpreter. Any of the simulation parameters may be altered before re-running the simulator. The list of available commands on the system is shown in Table A2-1 and the list of alterable simulation parameters is shown in Table A2-2. In Table A2-1, the minimum abbreviation for unique identification of each command is shown in upper case. The simulation results displayed at the end of a run are listed below:-

* The observed mean block size transferred in bits
* The observed mean time between arriving requests to each input port.
* The mean observed path set up time
* The mean observed total network transaction time
* The total time taken to pass the specified number of requests
* The observed request completion rate
* The total number of requests experiencing blockage
* The proportion of requests succeeding without blockage
* The maximum observed path set up time
* The maximum observed total trasnaction time
* The maximum number of requests blocked at a switch at any time

Plus, the following optional or simulation-dependent summaries:-

* The observed utilisation of links specified by the OBserve command
* The observed total transaction time for each transfer class specified in the SET T V command
* The observed mean number of retries before holding the channel if the timeout mechanisms are in use.
* A plot of the distribution of transaction times for all requests passed.
* A plot of the distribution of network output port addresses (primarily for analysing the effects of biassing).
* Tracing information recorded during the simulation run. This includes the current simulated real time, the total number of requests passed, the total number of requests blocked, the number of requests currently blocked in the network, the number of requests in the system and, for each stage in the network, the total number of links in use and the mean stage utilisation.

Command Name	Parameters	Description
BIas	<output port> <% bias>	Bias the given network output port by given percentage. That port is then p% more likely to be addressed than unbiassed output.
BLip	<request count>	Causes simulated real time to be displayed at the terminal every given number of requests.
Debias		Resets all biasses
Exit		Exit simulation system
Help		Displays help information
OBserve	<stage number> <link number>	Monitors the utilisation of the given output link of the given stage.
OUtput	<file name>	Redirects results summary to specified results file.
PRintbias		Displays current biasses
PLot	<plot type code> {<destination>}	Plot either output port address (code A) or transaction time distribution (code T). Destination='/' forces plot to terminal.
RESults	{<destination>}	(re)Display results of last simualtion run. Destination='/' forces output to terminal rather than to results file.
RUn	<no. requests>	Run the simulator & pass given number of requests.
REDefine		Redefine all simulation parameters
SEt	<parameter code> <new value descr.>	Sets simulation parameter given by parameter code to value given in descriptor. (See Table A2-2).
STop		Turns off link observation (see OB)
SYstem		Display current simulator parameter settings.
Trace	{<interval> {<file name>}}	Every n time units, given by interval, causes trace information to be sent to given file. No parameter => turn off trace. New file name => redirect with same interval.

TABLE A2-1: Simulation System Commands.

Parameter Code	Parameter Value Descriptor	
N	<inputs>	No. of active network input ports.
M	<outputs>	No. of active network output ports.
W	<width>	Width of network, i.e. total number of network input/output ports.
C	<crossbar>	Size of each component crossbar switch
I	<I.A.T.> {<dist. flag>}	Inter-arrival time. Distribution flag='F' => fixed intervals; flag='E' => exponentially distributed inter-arrival times.
L	<low limit>	Transaction time histogram lower range limit.
U	<up limit>	Transaction time histogram upper range limit.
P	<period>	Data transmission clock period (in ATUs)
A	<fetch time>	Switch select address retrieval time.
T	<trans.length> {<dist flag>}	Total number of bits transferred across established network channel. Distribution flag='F' => fixed size blocks; flag='E' => exponentially distibuted block size; flag='V' => variable block size: user is prompted for block sizes & relative frequencies.
S	<interval>	Master statistics clock interrupt period. Link observations & traces done after interrupt (if set on).
V	<timeout>	Request timeout period. After timeout, request is cancelled, delayed & retried.
Y	<interval>	Maximum timeout to retry interval. After random backoff time (≤interval) request is retried.
H	<timeout count>	No. of retries before holding channel. After given number of timeout/backoff/ retries, timeout is disabled. (Avoids starvation>.

TABLE A2-2: Possible Arguments To SET Command