ADELAIDE UNIVERSITY

Department of Electrical and Electronic Engineering

Multiport Memory as a Medium for Interprocessor Communication in Multiprocessors

By

Nasser Asgari

B.Sc., M.Sc.

Thesis submitted for the degree of

Doctor of Philosophy

February 2003

Abstract

The performance of a multiprocessor greatly depends on the effectiveness of its interprocessor communication. Shared memory and message passing are two major communication architectures for multiprocessors. In shared memory systems, processors communicate by writing to and reading from a common memory. In message passing architectures, nodes communicate by passing messages through an interconnection network using send and receive commands. Both systems have their advantages and disadvantages. This study aims to explore the feasibility of using multiport memories for interprocessor communication based on message passing.

The individual ports of a multiport memory provide independent access to memory cells and can be used as communication links. In the communication structure proposed in this study, several nodes connected to a multiport memory can communicate in parallel without the overhead and delay of the bus architecture, or the interconnection network of a typical shared memory system. The small number of ports on multiport memory is a limiting factor that restricts the number of nodes connected to this structure.

The proposed structure can be scaled by using a hierarchy in which the nodes that are not connected directly can communicate through network controllers and other available multiport memories. In this structure, shared memory is used as a link for

message passing. In contrast to other shared memory systems, the small shared memory in this structure is exclusively used for communication purpose.

The first stage in evaluating the proposed structure was the design and implementation of a small multiprocessor called MultiCom. In this prototype system, four nodes were interconnected by a 4-port memory. The memory management of MultiCom could prevent the nodes from interfering with each other using static and dynamic allocations. In particular, dynamic allocation used fewer but larger buffers that could be assigned to any communication on demand, and full memory utilization was possible. It could also handle multicasting and broadcasting very efficiently. As dynamic allocation required a lock mechanism for allocating the buffers, in the absence of hardware locks on multiport memories, two new software locks for controlling the ownership of the multiport memory were designed and successfully tested. Using a basic communication protocol for MultiCom, the measured communication rate was 4.2 times faster than a system using serial links, 11 times faster than a system using dual-port memories, and 14 times better than a bus-based system. In addition, compared to a system using dual-port memories, the system enjoyed a four-fold reduction in cost.

A simulation model was designed to evaluate the performance of the scaled structure. The model showed that the structure was scalable for small systems in which all of the nodes were connected as a group using a single multiport memory. It also confirmed that the structure only required small amount of shared memory for message passing. However, the performance of the cluster structure in the original proposal in which several groups were connected using a network controller was not desirable. The communication rate dropped considerably under high inter-group message transfers because of the overloaded network controller. To overcome this problem, the cluster structure was modified and separate network controllers were used for each group. In addition, an extra multiport memory was used to interconnect the network controllers. With this modification, the performance of a cluster was significantly improved and overloading of the network controllers was considerably reduced. The structure of a network of clusters was also improved to accommodate the modified cluster structure, and other measures were implemented to reduce the load of network controllers. The improved structure can be used for medium to large-scale systems.

Statement of Originality

This work contains no material which has been accepted for the award of any degree or diploma in any university or other tertiary institution and, to the best of my knowledge and belief, contains no material previously published or written by another person, except where due reference has been made in the text.

The author hereby consents to this copy of the thesis, when deposited in the University Library, being available for loan or photocopying.

Nasser Asgari

February 2003

Acknowledgments

I would like to express my gratitude to Professor Neil Burgess, my supervisor, for his continual help and support. His guidance, criticism, and encouragement were valuable sources of motivation for me in undertaking this research.

My sincere thanks to Mr Michael Liebelt, my second supervisor who was always available for discussion at any stage. He was the one to rely on when everything else failed.

I would like to acknowledge the efforts of my wife Shiva and my daughters Kimiya and Hasti for providing me a suitable environment to work on this research. I am thankful of their companionship, support, and understanding when much of my time was spent on studying.

I would also like to thank the Ministry of Culture and Higher Education of Iran for providing me the opportunity and support to undertake this research.

To Shiva, Kimiya, and Hasti

The loves of my life.

Contents

List of Figures

List of Tables

List of Publications

[1] Asgari N and Burgess N, **"Multiport Memory for High-Speed Interprocessor Communication in MultiCom"**, *the International Journal of Science and Technology, Scientia Iranica*, Sharif University of Technology, Tehran, Iran, vol. 8, no. 4, pages 322-31, October 2001.

[2] Asgari N and Burgess N, **"The Structure and Design of MultiCom, a Small Multiprocessor with Multiport Memory Based Communication"**, in *Proceedings of the Third Australasian Computer Architecture Conference (ACAC'98),* Perth, Australia, pages 15-24, Springer-Verlag, February 1998.

[3] Asgari N and Burgess N, **"Memory Management and Dynamic Allocation of MultiCom"**, in *Proceedings of the Forth Annual International CSI Computer Conference (CSICC'98),* Tehran, Iran, pages 72-81, January 1999.

[4] Asgari N and Burgess N, **"Interprocessor Communication in Tree Structured Multiprocessors Using Multiport Memory"**, in *Proceedings of the Forth Australasian Computer Architecture Conference (ACAC'99),* Auckland, NZ, pages 161-72, Springer-Verlag, January 1999.

CHAPTER 1

Introduction

1

This thesis makes the case for the use of multiport memories for interprocessor communication in multiprocessors. In this introductory chapter, the background and the motivation for the research undertaken are presented briefly and the objective and scope of the study are outlined. In addition, the contributions made by the study are put forward. Finally, the structure of the thesis including a short explanation of the contents of each chapter is presented.

1 Background and motivation

Interprocessor communication is one of the major activities in a multiprocessor system. There is a frequent demand for interaction and exchange of data among nodes and a high communication bandwidth is required. Interprocessor communication can be regarded as the dominant component affecting performance in multiprocessors. An efficient communication scheme requires high bandwidth and reliability with minimal cost and software/hardware overheads.

Two major communication structures for multiprocessors are shared memory and message passing. In shared memory systems, processors have access to a common memory and communication is performed implicitly by memory load and store instructions. One processor can write a message in the shared memory and other nodes can receive the message by reading it. Sharing a conventional single-port memory among nodes using a bus is one method to realize this concept. In this method, the nodes need to take turn in using the shared memory and only one node can use it at a time. Because of the limited bus bandwidth, this method is only useful for small number of processors. Examples of this structure can be found in [Tabak 90], [Culler+ 98], and [Patterson+ 98].

In an interleaved memory structure, various blocks of memory are shared among several nodes using different techniques. Nodes can access different memory banks simultaneously; however, only one access per bank is allowed. Multiple bus or crossbar switch can provide non-blocking connection of nodes to memory banks at high cost. Multistage networks reduce the cost, but full connection of processors to the memory banks may not be possible because of the blocking nature of the network. Memory interleaving is useful for medium-size systems and it has been the basis of many multiprocessors. Several examples can be found in [Tabak 90] and [Gajski+ 83].

In message passing systems, the nodes are interconnected by a communication network. Nodes communicate with each other by explicit send and receive commands. A low to medium number of nodes can be interconnected using topologies such as ring or mesh. For larger systems, hypercube is a better topology and some massively parallel processors have used this structure to interconnect up to 2^{16} nodes [Hennessy+ 94]. In

these topologies communication is performed through serial links. The required wiring for connecting nodes in a network of this size is a challenging task. In addition, the slow nature of a serial link that sends data bit by bit is a limiting factor to achieve high communication rates.

Message passing through dedicated parallel links has been also investigated by researchers. For example, [Tuazon +85] suggested the use of first-in, first-out (FIFO) buffers between two nodes. The transmitting node had to write the message into the FIFO, and the receiving node could retrieve it from there. Similarly, [Su+ 92] proposed the use of FIFO RAMs to act as a communication buffers between nodes.

Developments in dual-port memory structure have initiated new methods for interprocessor communication. Dual-port memories allow two devices to have independent and simultaneous access to the memory cells. The nodes connected to a dual-port memory can communicate in both directions using two separate ports. As the overhead of bus is eliminated, higher communication rate can be achieved. Several structures have been proposed for interconnecting a limited number of nodes in [Jagadish+ 89], [Khan+ 94], and [Campbell+ 96]. With only two ports to access the memory, this approach is restricted to small systems.

Multiport memory offers a better structure for interprocessor communication. Several nodes can share a multiport memory using independent ports and they can communicate directly by writing and reading the shared memory concurrently. This structure is much simpler than other shared memory structures such as multiple bus or multistage networks and it does not have the overhead and the delay associated with these structures. The limiting factor is the small size and low number of ports on multiport memories. This restriction makes the design of communication structures for large systems very challenging. The structure proposed in this study is based on limited number of ports, and the shared memory created by multiport memory is used as a link for message passing. Unlike other shared memory systems, the shared memory in this structure is exclusively used for communication purpose and as explained later, a small memory size is adequate for this purpose.

As explained in Chapter 3, the concept of using multiport memory for interprocessor communication has not been deeply explored by researchers and only very few structures for basic communication have been proposed in [Handy 90] and [Varshneya+ 94]. No evaluation of the proposed structures has been performed. Hence, this is an open research area and the challenging nature of it, together with the expected benefits and outcomes are some of the incentives to undertake this research.

2 Objectives and scope of the study

The main objective of this study is to demonstrate that interprocessor communication in multiprocessors can be performed efficiently through multiport memories. Suitable structures can be designed in which multiport memory with limited port count and capacity can be used as a link for message passing.

This study covers the following:

- Designing a basic structure for using multiport memory for interprocessor communication with a limited number of ports and capacity
- Developing strategies for the management of multiport memory
- Developing the required communication protocols
- Designing structures to connect a large number of nodes in a network
- Developing support facilities for multiport memories

3 Contributions

This study explores the possibility of using multiport memory for interprocessor communication. Contributions claimed by the author are the following:

- Communication structure
- Multiport memory management
- Communication protocols
- Support circuits for multiport memory

Each one will be discussed briefly in the following sections.

3.1 Communication structure

The concept of using dual-port memory for interprocessor communication has been explored by some researchers and successful outcomes have been reported. However, the use of multiport memory for this purpose has not been examined in depth and apart from a couple of proposed structures as presented in Chapter 3, there has been no significant work in this area. Furthermore, no work has been carried out to evaluate the proposed structures.

This study proposes a novel structure for interprocessor communication using multiport memory and supports the proposed structure by evaluating it with a hardware prototype and a simulation model. The hardware prototype tests the functionality and effectiveness of the structure under real conditions, and the simulator provides further in-depth tests of the expanded structure for conditions that cannot be tested easily on real systems. The hardware prototype was also used to calibrate the simulator to produce more reliable results when expanded. After several revisions of the structure based on the results obtained from the simulation model, the final structure presented in Chapter 7 was devised for medium to large-scale systems.

Compared to a system using dual-port memories, this structure offers a considerable increase in performance, and a remarkable decrease in the cost of the system as presented in Chapter 5. In addition, it enjoys a much simpler design and reduced number of links. It also shows much better results compared to bus-based systems or systems using serial links.

3.2 Multiport memory management

The proposed structure relies on conflict-free flow of data through multiport memory. As each node is connected to a multiport memory through an individual port, the possibility of a conflict between nodes is very high. The memory management of the multiport memory requires hardware support facilities or software-driven mechanisms for removing the conflicts or avoiding them. Because of the lack of hardware supports

on multiport memories, the memory management developed in this study was based on software methods. In particular, two newly devised algorithms for implementing a lock mechanism for multiport memory were successfully implemented and tested on the hardware model in Chapter 5. These locks were used in dynamic allocation of the memory buffers to the requesting nodes.

3.3 Communication protocols

In order to effectively control the communications in the hardware prototype, a basic communication protocol was developed and tested on MultiCom in Chapter 5. The simulator designed in Chapter 6 was used to evaluate the scaling of the communication structure in several stages and the communication protocol was gradually modified to support the requirements of the expanded system. More improvements were applied to the communication protocol to meet the requirements of the improved communication structure in Chapter 7.

3.4 Support circuits for multiport memory

As discussed in Chapter 5, support circuits such as semaphore logic facilitate the use of dual-port memories. If similar facilities are not available on multiport memories, software approaches need to be developed for memory management and overhead will increase. This reduces the usefulness of multiport memory in a system design. In the Appendix, several new support circuits for multiport memories are proposed and designed. In particular, the new circuit designed for multiport semaphore logic considerably simplifies the management of multiport memory and makes the use of this type of memory more convenient. The designed circuits are practical and have been tested by hardware design tools.

4 Thesis structure

This thesis is divided into eight chapters and one Appendix. Each chapter is discussed very briefly in the subsequent sections.

4.1 Background

Chapter 1 presents a brief introduction and the motivation for the research undertaken. The objective and scope of the study are discussed, the contributions claimed by the author are presented, and the structure of the thesis is explained.

4.2 Interprocessor communication

Chapter 2 presents the literature review for interprocessor communication. Shared memory and message passing as two major communication architectures in multiprocessors are explained and different approaches for designing each method are presented. For shared memory, single-bus and interleaved memories are discussed and the use of multiple bus, crossbar switch or multistage networks for connecting several nodes to memory banks is illustrated. For message passing systems, the use of serial links in topologies such as ring, mesh or hypercube is discussed and other structures that can create a dynamic link between two nodes on demand are reviewed. In addition, the use of parallel links for message passing is explained and generating such links using FIFOs or dual-port memory is discussed.

4.3 Multiport memory for interprocessor communication

Chapter 3 describes the research proposal for this study. In the literature review presented at the beginning of this chapter, the achievements in multiport memory cell design are explored and some of the applications of multiport memories in system design are reviewed. Then, a structure for interprocessor communication using multiport memories is proposed and the expansion of the structure to cover more nodes in a network is presented. The proposed structure is based on multiport memories with limited number of ports and small capacity in which shared memory is used as a link for message passing.

4.4 Hardware model

Chapter 4 discusses the structure and implementation of MultiCom, a small multiprocessor designed as a hardware prototype to verify the efficiency of the proposed communication scheme on small scale. The node processors and the 4-port

memory used in this design are explained and a basic memory management method called static allocation is introduced. Finally, the performance of this system is compared to serial systems.

4.5 Memory management

Chapter 5 introduces dynamic allocation as an advanced memory management scheme and discusses the required communication protocols. Dynamic allocation can provide a better memory utilization, but requires a sophisticated lock mechanism to eliminate shared memory conflicts. Two new software locks devised for the control of multiport memory are explained and the efficiency of the communication in MultiCom is measured and compared with other methods.

4.6 Simulation model

Chapter 6 presents the design of a simulation model for the evaluation of larger systems. First, a model for MultiCom is generated and its performance is matched to that of MultiCom. Then, the simulation model is expanded to include more nodes in a cluster and the performance of the system is evaluated. The expansion of the model to encompass more clusters in a network is also explained and the communication bottleneck detected by the simulation model is discussed.

4.7 Improved communication structure

Chapter 7 describes the modifications required for the communication structure. Based on the results obtained from the simulation model, a modified structure for nodes connected in a cluster is presented and an improved structure for connecting several nodes in a network is explained. Several issues on scaling the communication scheme are also discussed.

4.8 Further directions

Chapter 8 is the conclusion. The steps undertaken in designing, evaluating, and improving the structure are described and the achievements are discussed. In addition,

several possibilities to improve the structure and further directions in pursuing this study are presented.

4.9 Structure of multiport memory

The Appendix presents the evolution of single-port memory cells to dual-port and multiport memory cells. The control logic used on dual-port memory chips is explained and several issues in the design of multiport memory chips are discussed. In addition, new circuits for the control of multiport memory are proposed and designed.

CHAPTER 2

Interprocessor Communication

2

Interprocessor communication in multiprocessors is a very important task and requires a high bandwidth. In the literature review presented here, first, shared memory and message passing as two major communication structures for multiprocessors are explained and different methods to realize each method are presented. In the shared memory structure, nodes can communicate with memory load and store instructions and different techniques such as single bus or interleaved memory can be used to create it. In message passing, nodes communicate with explicit send and receive commands using an interconnection network, which is generally based on serial links and can be organized using different topologies. Next, the use of dual-port memory as a communication medium between two nodes is discussed and different communication structures based on dual-port memories are explored. Finally, shared memory using a multiport memory is introduced and some of the unresolved issues in designing large-scale multiport memories are presented.

1 Introduction

Multiprocessors enhance the capability and performance of computer systems by using parallelism. Several interconnected processors can process different parts of a workload in parallel and finish it faster. Workload sharing demands close cooperation and frequent exchange of data between processors. Hence, interprocessor communication is a very important task in multiprocessors. In [Almasi+ 89], a parallel computer is defined as "a collection of processing elements that communicate and cooperate to solve large problems fast". In [Culler+ 98], parallel architecture is viewed as "the extension of conventional computer architecture to address issues of communication and cooperation among processing elements". Both of these definitions highlight the importance of communication in multiprocessors. For a high performance multiprocessor, the communication structure should provide a high bandwidth.

The communication architecture of the majority of multiprocessors falls within the following categories:

- single address space or shared-memory
- message passing

In shared-memory architectures, processors share a single memory address space and communicate through the shared memory. In message passing systems, processors exchange data and other information by sending and receiving messages to each other. Both are discussed briefly in the subsequent sections.

2 Shared memory

This class of multiprocessors uses a single memory address space that is shared among processors. Processors still can have private or local memories, but part of their memory space is mapped to a common physical memory that is accessible by all processors. Communication is performed implicitly through shared memory using conventional load and store instructions. Shared memory can be implemented in many ways and two popular methods for implementing it using a single bus and interleaved memory are discussed here.

2.1 Shared memory with a single bus

In this method, a conventional single-port memory is shared among several processors by a single bus, which has a fixed bandwidth. Implementing this bus is straightforward and processors can be added to the bus or removed from it if required, although the bus speed will be limited if the bus is long and/or many devices are connected to it. Because of the nature of the shared bus, only one request for the memory can be handled at a time and other requests have to wait for their turn. A control circuit such as arbitration logic is required to resolve simultaneous requests to memory. Under heavy bus demand, the limited bandwidth of the bus can create a bus bottleneck and consequently, this structure is only applicable when the number of processors is relatively low. The maximum number of processors on a bus depends on the bus bandwidth and the traffic per processor. Figure 2.1 illustrates the overall structure of a single-bus shared memory.

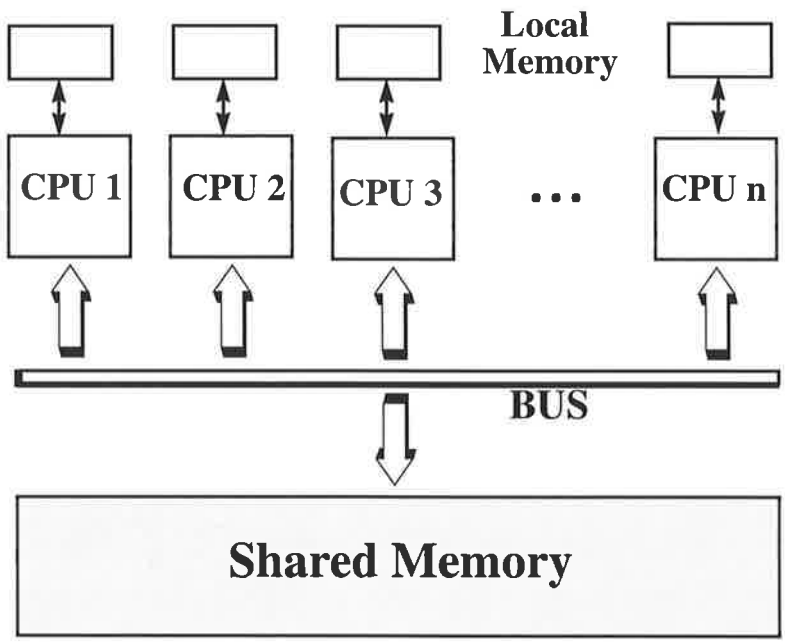

Figure 2.1 Simple shared memory

A conventional single-port memory can be used as a shared memory by using a bus. Only one request on the shared memory can be processed at any time.

As a consequence of working in parallel, processors frequently work on shared data and need to coordinate their access to shared variables. As only one processor should be able to modify a shared variable, synchronization mechanisms such as locks or semaphores should be implemented to avoid memory conflicts. Semaphores will be discussed in detail in Chapter 5.

Shared memory structure has a long history, dating at least to precursors of mainframes in the early 1960s, and today it has a role in almost every segment of the computer industry. An example of earlier shared memory systems is IBM System 370. Supports for multiprocessor configurations including atomic memory operations and interprocessor interrupts were the key extensions in the evolution of the 360 architecture to System 370 [Culler+ 98].

Commercial examples of single-bus architecture are ELEXSI System from ELEXSI Corporation which features up to 12 processors connected to a single bus system called "Gigabus" [Tabak 90], and Compaq ProLiant 5000 from Compaq Corporation connecting four Pentium Pro processors [Patterson+ 98].

2.2 Interleaved shared memory

Memory interleaving is a method used for allowing several concurrent accesses to memory. In this method, the memory is divided into blocks or banks and an interconnection network is used to connect memory banks to the processors. Several processors can access different banks simultaneously; however, only one connection per bank is possible at a time. If more than one access to a particular bank is requested, only one of them can proceed while the others have to wait. Figure 2.2 demonstrates an interleaved memory structure.

Interconnection networks can be implemented using multiple busses, crossbar switches, or multistage networks. Each one will be discussed briefly.

2.2.1 Multiple-bus shared memory

In multiple-bus structures, processors and memory banks are connected to all of the available buses as shown in Figure 2.3. There are several redundant paths from each

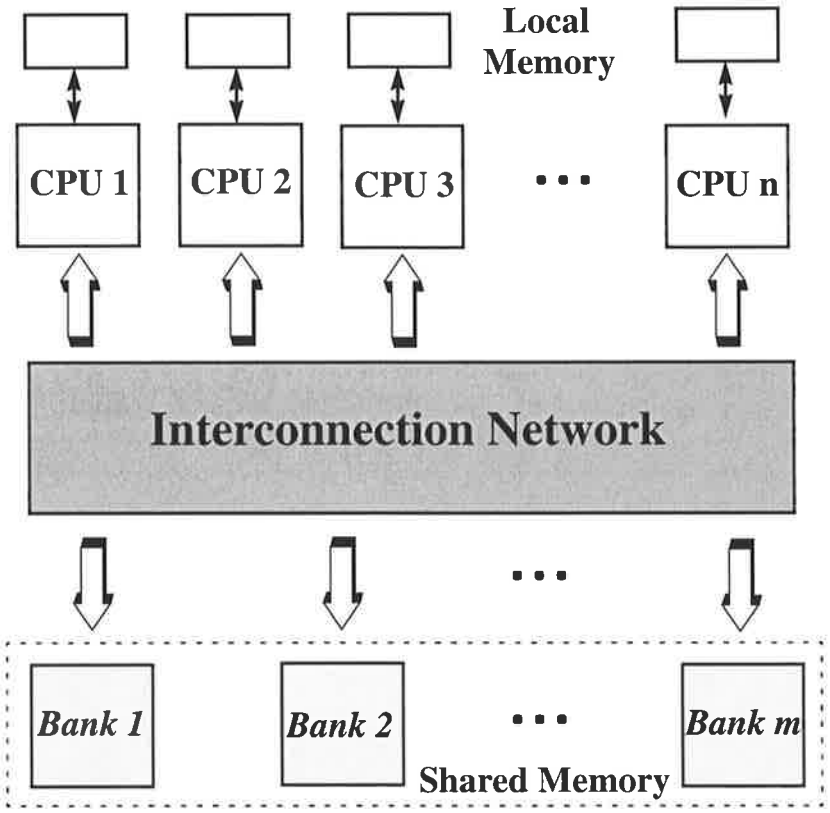

Figure 2.2 Interleaved memory

The shared memory is composed of several memory banks and they are connected to different processors using an interconnection network.

processor to a memory module. Each processor can use any of the available buses to access a memory bank. Each bus can be used with only one processor at a time to access a memory bank, but several processors can access different banks simultaneously using separate buses. The cost and complexity of multiple-bus is very high. In addition, a complex bus arbiter is required to control the bus traffic. [Mudge+ 87] analyses multiple-bus structures in detail.

2.2.2 Crossbar switch

A crossbar switch can also provide full connection of memory banks and processors. As shown in Figure 2.4, in a matrix of switches, processors are connected to the rows and memory banks to the columns. Crosspoint switches are placed in the intersection of

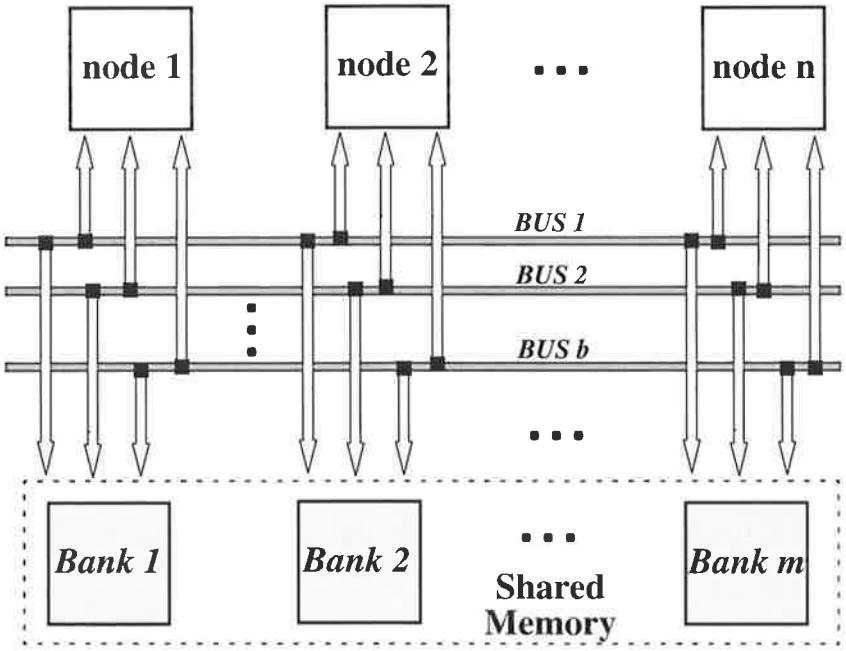

Figure 2.3 Multiple-bus shared memory

Processors and memory banks are connected to all buses. Each bank can be accessed using any bus. There are several redundant paths for each connection.

rows and columns and they can route each processor to a memory module. Crossbar switches scale well, but their cost is also high. For example, for N processors and N memory modules, N^2 crosspoint switches are required. Moreover, for accessing memory bits in parallel, each bit should be routed by a separate switch. A crossbar switch is a nonblocking network and a connection between a processor and a memory module does not block the access of other processors to other memory modules. The structure in Figure 2.4 can be regarded as a multiple-bus system in which processors are connected to all buses, but memory banks are connected to only one bus. Hence, there is only one path from any processor to any memory bank.

An example of a commercial system using crossbar switches is IP-1 from International Parallel Machines Incorporation in which eight 64-bit processors were connected to 8 memory modules through an 8x8 crossbar switch. The Alliant System from Alliant Computer Systems Corporation also used a crossbar switch in its structure [Tabak 90].

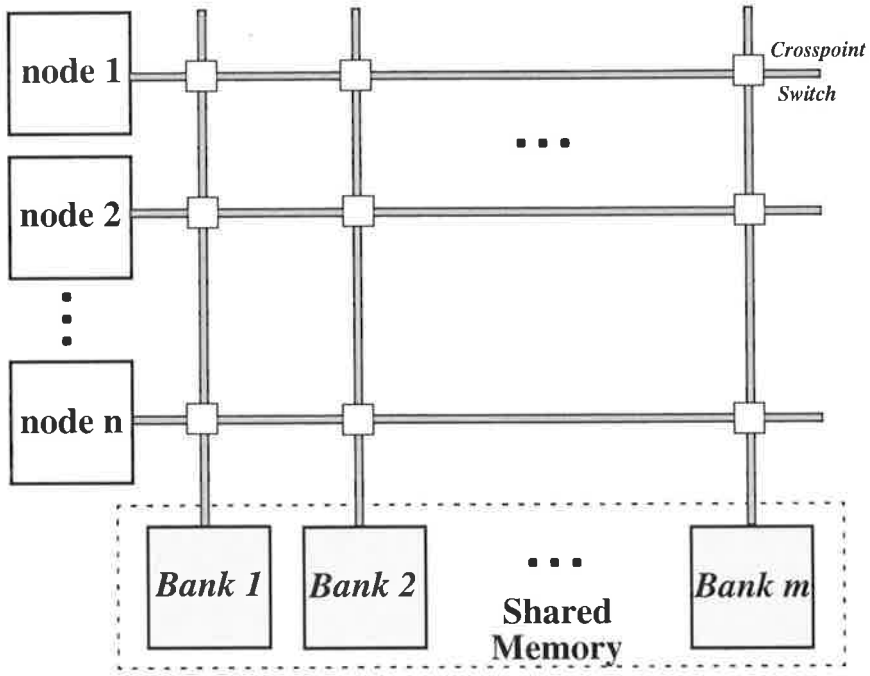

Figure 2.4 Crossbar switch

Each memory bank is connected to a column line. Nodes can be routed to a memory bank by activating the appropriate crosspoint switch.

2.2.3 Multistage network

A multistage network can reduce the cost of a switching network to (N/2) $\log_2$N. As shown in Figure 2.5, instead of directly routing a processor to a memory module, several single-stage switches are activated in series to make the connections. Many structures for the design of multistage networks have been proposed and investigated in the literature. Examples are the omega, banyan, baseline, and delta networks. These are blocking networks and an existing connection may result in conflicts in the use of network and may block other connections. [Feng 81] presents a survey of interconnection networks in detail.

The circuit required to build the interconnection network can be large, especially if the number of nodes and memory blocks is high. In addition, as the circuit expands, the delay associated with it increases.

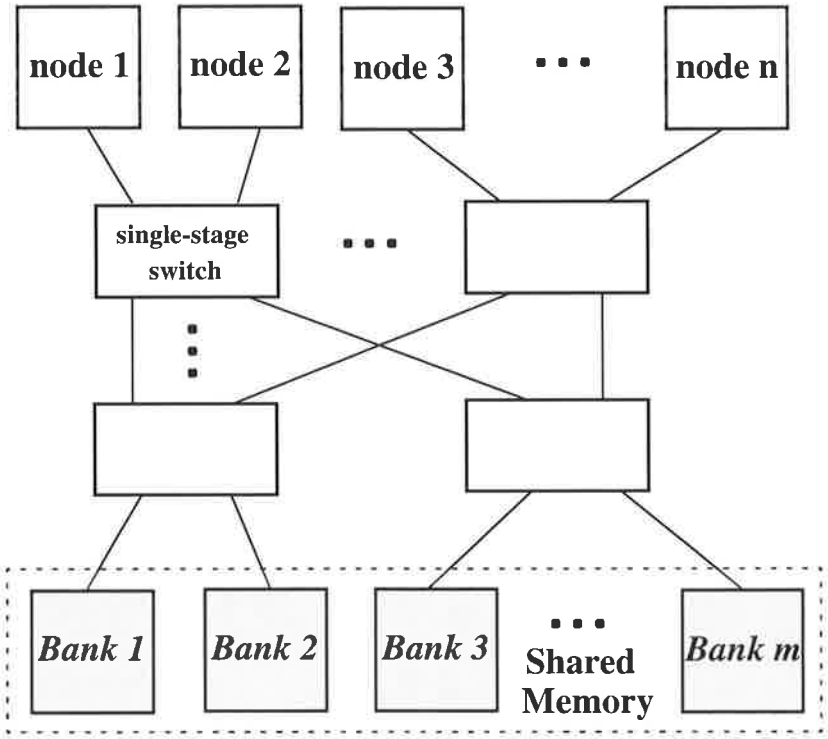

Figure 2.5 Multistage network

The nodes are connected to different memory banks using a multistage network. This structure is a blocking network. For example, routing node 1 to bank 1, and node 2 to bank 2 at the same time is not possible.

An example of a research machine using multistage networks is the "CEDAR" a large-scale multiprocessor built in University of Illinois. It used a special switching network called "Global Network" which provided redundant paths between processors and memory modules for conflict avoidance and fault tolerance. It was designed to scale up to 1024 processors [Gajski+ 83]. A commercial example of multistage network is the BBN Butterfly from the BBN Advanced Computers, which could scale up to 256 processors, and the shared memory was accessible by all processors through the system's logarithmic, packet-switched communication network, the "Butterfly Switch" [Tabak 90].

3 Message passing

In this class of communication structure for multiprocessors, several nodes each comprising a processor, an I/O system and private memory are interconnected by a network. Communication between nodes is performed with explicit send and receive commands. If a message is large, it is divided into several packets and each one is sent independently. After receiving all the packets, the receiver can reassemble the original message. Message passing is usually slower than shared memory, but it avoids memory contention problems and scales very well.

Two general mechanisms for message passing are packet switching and circuit switching. The packet switching method works in a store-and-forward manner, analogous to the mail service. Each node stores the received message and then forwards it to the next node. The minimum message latency depends on the number of hops and the message length. In the circuit switching mechanism, a path from source to destination is initially established and remains connected until the message is transmitted in full [Gaughan+ 93]. This method is similar to a telephone switching system.

Nodes of a message-passing system can be interconnected using several topologies as explained in the next section.

3.1 Network topologies

The straightforward way to interconnect a number of nodes is to create a fully-connected network in which a dedicated communication link is used between any two nodes. The links are normally bidirectional and data can flow in either direction. Between the high cost/performance of this network and the low cost/performance of a bus, there are several other networks that constitute a wide range of trade-offs in cost/performance. Network costs include the number of switches, the number of links on a switch to connect to the network, the width or the number of bits per link, and the length of the links on the physical machine [Patterson+ 98]. Network topologies can be grouped into static and dynamic categories, depending on the type of links used in the network. These are explained in the subsequent sections.

3.1.1 Static topologies

In a static topology, links between two processors are passive and dedicated buses cannot be reconfigured for direct connection to other processors [Feng 81]. Some of the topologies in this category are discussed below.

In a ring topology as shown in Figure 2.6, each node is connected to two adjacent nodes and several simultaneous transfers are possible. A message sent to a nonadjacent node will require extra hops to go through intermediate nodes. The maximum number of hops for N nodes is N/2. As the average message delay and message traffic density increase with the number of nodes on the ring, this topology is only useful for small systems.

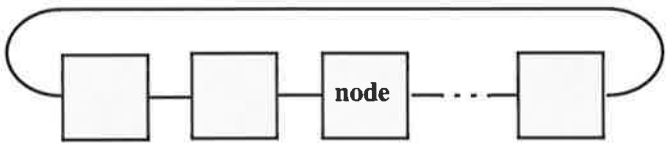

Figure 2.6 Ring topology

In the ring topology, each node is connected to two adjacent nodes and the end nodes are connected together.

In a mesh or grid topology as illustrated in Figure 2.7, the nodes are arranged in a two-dimensional grid and each node is connected to the adjacent nodes. A node will have a maximum of four direct links to adjacent neighbours. For N nodes, the communication of two nodes located on two opposite corners will take $2(\sqrt{N} -1)$ hops. This is 30 hops for N=256.

Nodes can be also arranged in a three-dimensional grid. In this topology, each node will be connected to a maximum of six adjacent nodes.

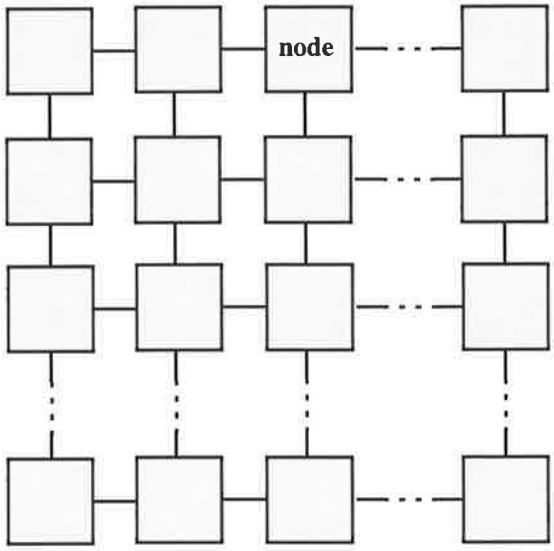

Figure 2.7 Mesh or grid topology

The nodes are arranged in a two-dimensional grid. Each node in the middle is connected to four adjacent nodes, but the corner nodes and boundary nodes are connected to two or three nodes respectively.

In a ring-mesh topology, a combination of ring and mesh is used. The corner and boundary nodes of a mesh structure are interconnected using several rings as shown in Figure 2.8. For N processors arranged in a ring-mesh array of $\sqrt{N} \times \sqrt{N}$, the longest path between any two processors requires $\sqrt{N}$ hops. This is 16 hops for N=256.

A three-dimensional grid can be also converted to a ring-mesh topology by connecting the boundary nodes using several rings. This will reduce the number of maximum hops required in the network.

A network of Transputers is a good example of the ring-mesh topology. Transputers can be used as the main elements of a multiprocessor system and there are four serial ports on each chip [Inmos 88]. Several Transputers can be interconnected using this topology and the serial ports on each processor are fully utilized. A 4-cube topology as explained below is also suitable for a network of 16 Transputers.

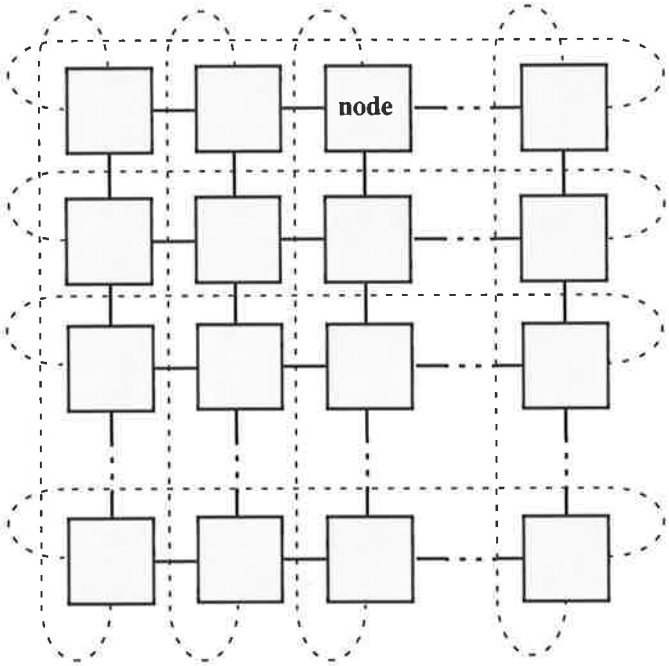

Figure 2.8 Ring-mesh topology

A combination of ring and mesh is used in this topology. Each node is connected to four other nodes.

For a larger N, hypercube is a better topology where the number of links per node and the number of hops are optimized. For $N=2^n$ nodes, each node should have $n=\log_2 N$ links and the required hops to communicate from any node to another is the Hamming distance between the node numbers [Bhuyan+ 84]. The longest distance is $\log_2 N$, which is nine for $N=2^9=512$. The number of total links is $(N/2)\log_2 N$, which is 2304 links for 512 nodes. This hypercube is called an n-cube.

Figure 2.9 shows a 5-cube configuration with all of its 80 links. Communication between node X and node Y requires five hops. The arrows in the figure show one possible path for this communication. Figure 2.10 shows a 6-cube, and Figure 2.11 illustrates a 9-cube configuration. Only some of the links are shown in these figures.

The design of a microprocessor-based hypercube system is explained in [Hayes+ 86]. The Intel iPSC (personal supercomputer) and iPSC/2 are examples of commercial hypercube systems. They offer 5-cube, 6-cube, and 7-cube options. The iPSC uses

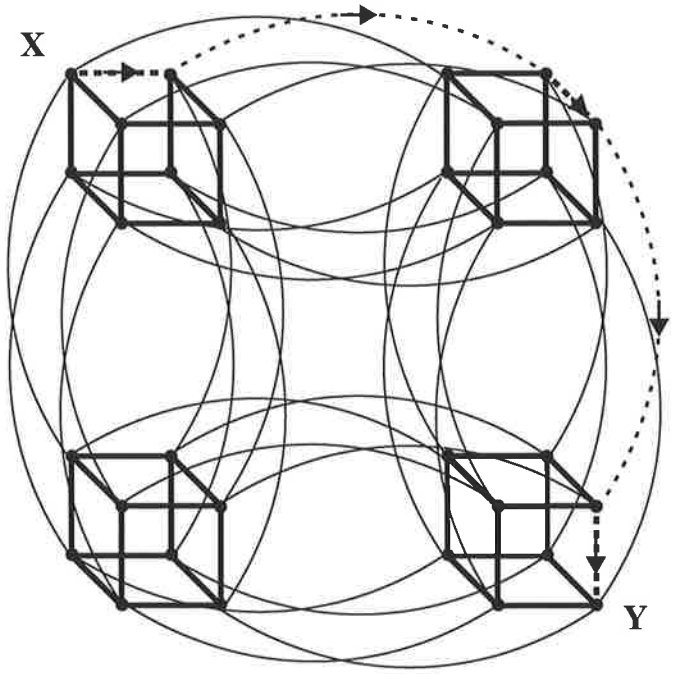

Figure 2.9 32-node hypercube (5-cube) with all of its links

Each node in this hypercube is connected to its four neighbours through serial links. The dotted arrow shows one possible way of connecting node X to node Y.

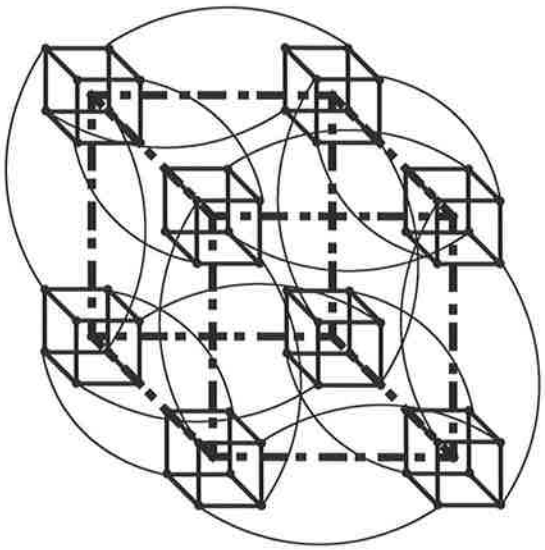

Figure 2.10 64-node hypercube (6-cube)

In a 64-node hypercube, there is a 3-cube in each corner of a cube. Only some of the links are shown.

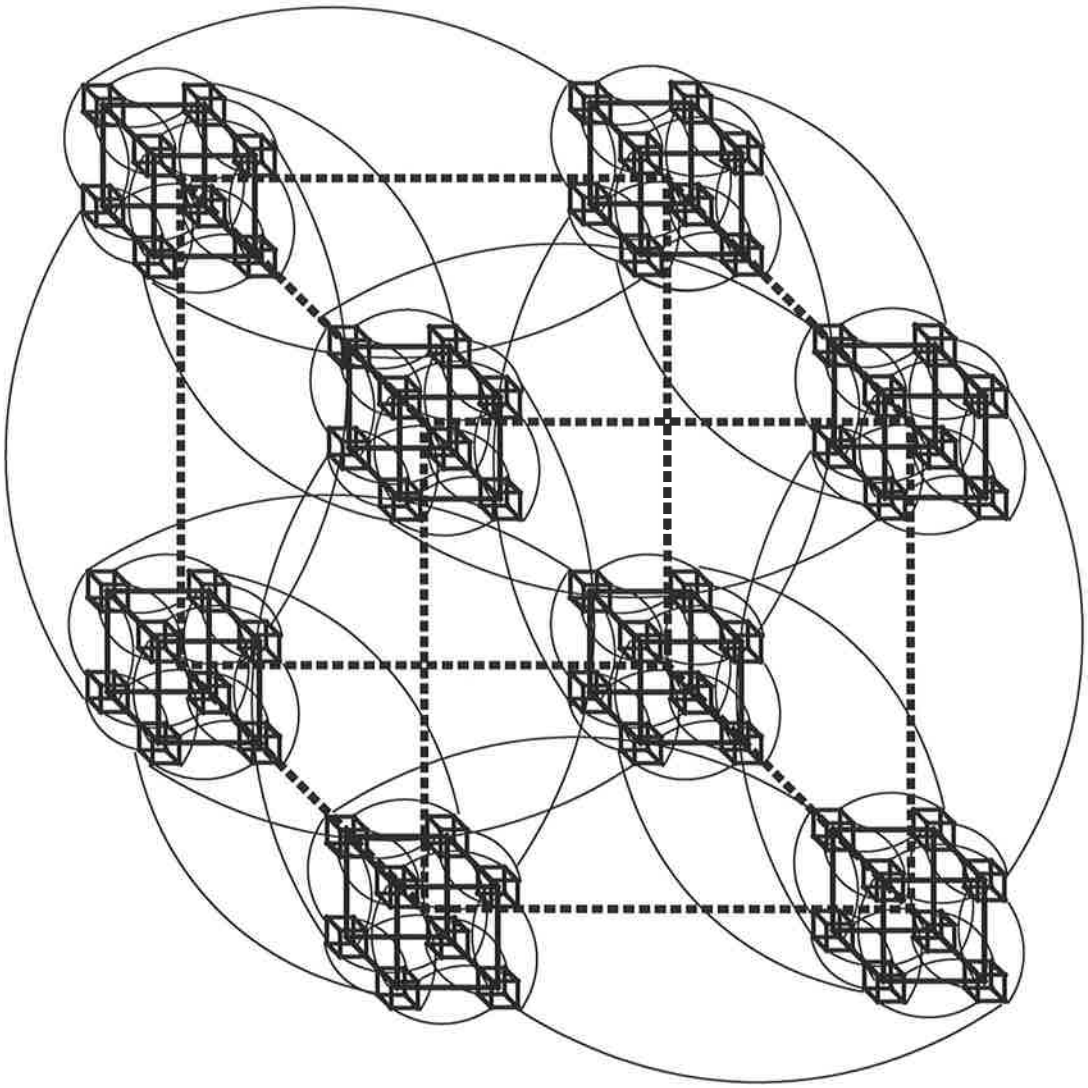

Figure 2.11 512 node hypercube (9-cube)

A 9-cube is composed of a cube with a 6-cube in each corner. Only some of the links are shown.

80286 processors with 80287 coprocessors and iPSC/2 is the 80386 version [Tabak 90]. Another example is the CM-2 supercomputer with 2^{16} processors. It is a massively parallel machine from the Thinking Machines Corporation and uses a 12-cube configuration for its interprocessor communication. Each node of the hypercube is a cluster of 2^4=16 processors. There are also other dedicated links for communication inside each cluster [Hennessy+ 94].

Other examples in static category are the star, tree, systolic array, and chordal ring topologies [Feng 81].

3.1.2 Dynamic topologies

In contrast to static topologies, the links in a dynamic topology are not fixed and can be reconfigured. This can be done by setting the network's active switching elements [Feng 81]. Hence, when required, a dynamic link can be established between two nodes for direct communication.

Dynamic links can be created using an interconnection network based on crossbar switches, multistage switches, or ATM (Asynchronous Transfer Mode) switches. The structure of an interconnection network using a crossbar switch is similar to the one shown in Figure 2.4 with the exception that the memory modules are replaced with nodes 1 to n. This implies that each node will have one input link and one output link connected to the network. Hence, a direct path can be created between any two nodes by setting the appropriate crosspoint switch. Similarly, by replacing the memory modules with nodes, the multistage switch shown in Figure 2.5 can route a path for the communication of two nodes. As stated earlier, this is a blocking network, but it costs less than a crossbar switch network.

Some of the static topologies shown in previous section can be constructed using dynamic links. For example, in the ring topology shown in Figure 2.12, each node is connected to the ring using a switch [Patterson+ 98]. The switch is capable of isolating the node from the ring, or connecting it the right or left side. With some restrictions, several communications can be performed simultaneously. For example, the two nodes on the right of Figure 2.12 can communicate with each other while the two nodes on the left are also communicating.

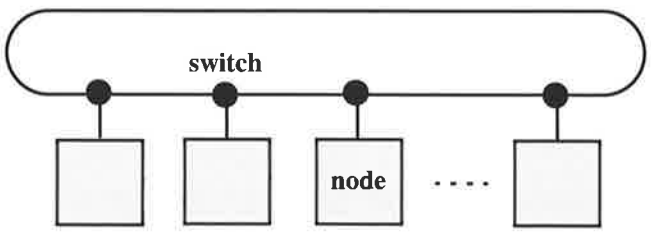

Figure 2.12 Ring topology with dynamic links

Each node is connected to the ring using a switch. Limited simultaneous communications are possible.

An example of a commercial computer using dynamic links is the CM-5, a supercomputer from the Thinking Machine Corporation, which is designed using an advanced structure called "fat-tree" for its interprocessor communication [Hennessy+ 94]. Another example is IBM SP-2, which is a scalable parallel machine constructed essentially out of complete RS6000 workstations. The interconnection network is a butterfly-like structure, constructed by cascading 8 x 8 crossbar switches [Culler+ 98].

3.2 Interconnection methods

An interconnection network can be implemented using serial links or parallel links. Each one will be explained briefly in the subsequent sections.

3.2.1 Serial links

Communication using serial links is a well-established method where the message is transmitted serially from one processor to another, one bit at a time. Serial communication can be performed point-to-point, or on a network using multistage switches. In the simplest form, the link is composed of a pair of wires that runs between two nodes. The major disadvantage of this link is its low data rate, which is the result of the serial nature of the communication that sends data bit by bit at usually low data rates. Its advantage is that it significantly reduces the number of wires used in the network. Another advantage is that as serial communication is normally performed using independent modules, the main processor is not heavily engaged in the communication and is free to perform other tasks while the communication is running in the background.

3.2.2 Parallel links

Parallel links are capable of transferring several bits of data at the same time. For a data transfer, the sender writes to the link and the receiver reads from the link. A simple parallel link can be created by connecting an I/O port of a node to that of another node. This method requires extensive handshaking between the nodes and each unit of data needs to be signalled and acknowledged before sending others. The data bus of a memory module can be also used as a parallel link for data transfer. [Tuazon +85] suggested the use of first-in, first-out (FIFO) buffers as parallel links between two

nodes. The transmitting node had to write the message into the FIFO, and the receiving node could retrieve it from there. Similarly, [Su+ 92] proposed the use of FIFO RAMs to act as communication buffers between nodes. [Culler+ 98] explains an early system in which eight nodes were connected in a 3-cube structure and there were two sets of FIFOs between two neighbours for sending and receiving. A total of 24 FIFOs were used in this structure. Because of the limited size of FIFOs, a message sent to a FIFO should be picked up by the receiver before sending another message.

The availability of dual-port memory in the second half of the 1980's provided a better solution than FIFOs because of its ability to work in both directions and many structures were proposed on this basis. As the discussion of dual-port memory is vital to the communication structure proposed in this thesis, it is covered in a separate section.

3.2.3 Dual-port memory

As shown in Figure 2.13, a true dual-port memory has two sets of address, data, and control lines. Each memory location can be accessed through either of the ports.

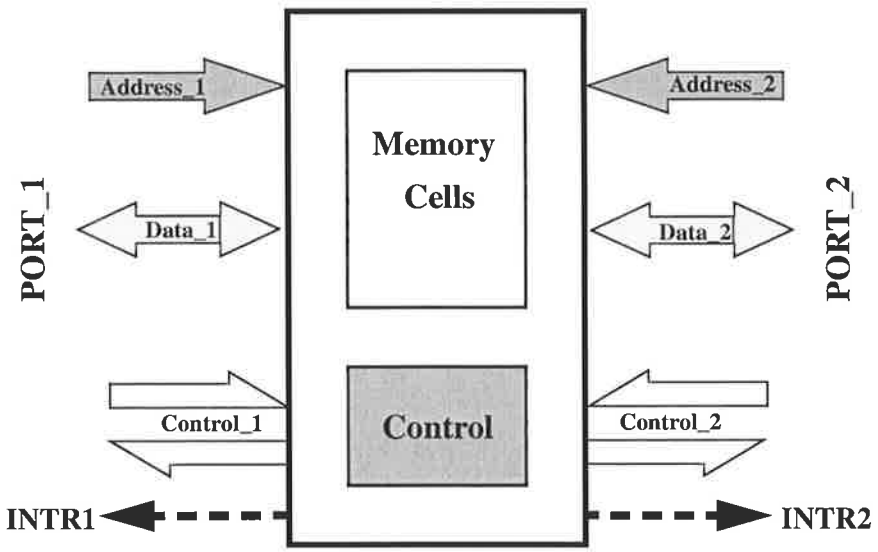

Figure 2.13 Dual-port memory

A dual-port memory has two sets of address, data, and control buses. The memory cells can be accessed from either port. The chip may contain signalling facilities between the ports, like an interrupt line that alerts one side when data has been written by the other.

Simultaneous read and writes are allowed, provided that they are performed on different addresses. In order to achieve data integrity, two concurrent writes to a memory location should be prevented. Similar prevention methods should be applied to a concurrent read and write because this may produce changing data at the output of the read operation.

A typical dual-port memory has a size of 16 to 64 KBytes with the facility of preventing undesired operations, as well as a mechanism to send an interrupt from the processor connected to one of the ports to the processor connected to the other port. The interrupt mechanism works by simply writing to a specific memory location. [Cypress 96].

Dual-port memory can be used as a register file inside a processor where two sets of data are supplied to the ALU (arithmetic logic unit). By the use of simple latches at the ALU inputs, the output of the ALU could be written back to the register file [Elliot+ 89]. There are several other applications for dual-port memories such as digital video cameras, data acquisition systems, displaying data on a monitor, and communication between processors. The latter application is one of the major applications of the dual-port memories [Wyland 88] [Pryce 89]. There is a preliminary report on the use of dual-port memory for a cluster of four Transputers and a network controller in [Khan+ 93]. The next report from the same team discusses the detail of their design [Khan+ 94]. The architecture used in this design is illustrated in Figure 2.14, in which there is a separate

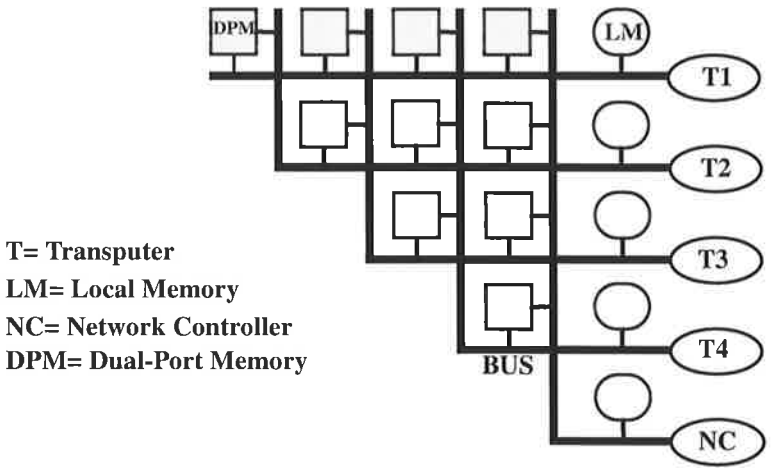

Figure 2.14 Restricted Shared Memory (RSM) architecture for communication through dual-port memories

In this structure, a cluster of four Transputers and a network controller are interconnected using dual-port memories.

link through a dual-port memory from any node to the network controller or the other nodes. As seen in the figure, ten memory blocks were used in this architecture to interconnect four nodes and a network controller.

Another work is cited in [Jagadish+ 89], in which dual-port memories were used to interconnect eight nodes and a network controller. As shown in Figure 2.15, each node in this design is located in a corner of a cube and has a separate link to three of its neighbours as well as another link to a network controller. Only the dual-port memories associated with node X and node Y are shown in this figure. The total number of dual-port memory blocks required for this structure is 20. Even with this number of memory blocks, the link between some nodes, such as nodes X and Y, must be established through the other nodes or the network controller as shown by the arrows.

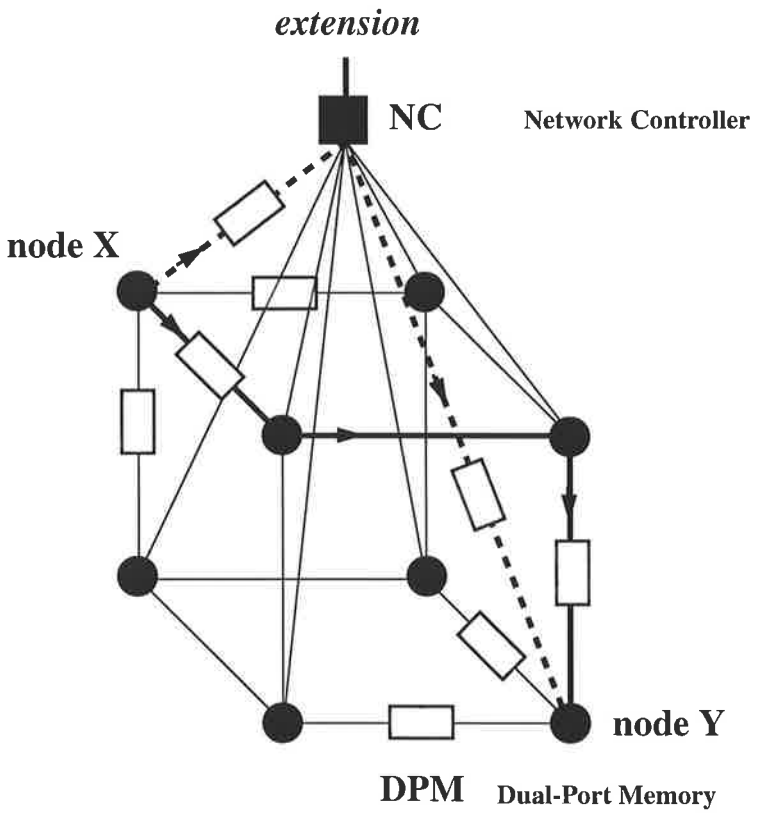

Figure 2.15 Using DPM to connect nodes in a cube

The nodes are located in the corners of a cube and there is a dual-port memory between any two nodes. A network controller can facilitate the communication between non-adjacent nodes. Only DPMs connected to nodes X and Y are shown. Arrows show two possible paths between X and Y.

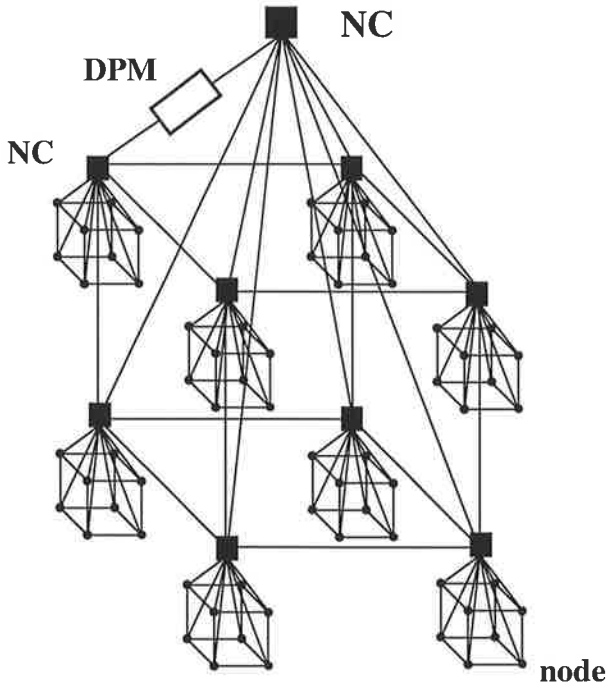

Figure 2.16 Extending the structure using DPMs to 64 nodes

In this structure, the network controllers of each cluster are interconnected using DPMs in a similar manner as nodes of a cluster. Only one DPM is shown, but the structure requires 180 DPMs

The authors have proposed that the structure could be extended to higher order cubes such as 2^6=64 nodes (6-cube) as shown in Figure 2.16. The number of required memory blocks would be 180.

Both the structure presented in this work and the RSM discussed earlier combine the use of shared memory with message passing. In fact, similar to the structures using FIFOs described earlier, shared memory is used as a link between nodes for message passing.

The RSM architecture presented in [Khan+ 94] is similar to the structure used in [Jagadish+ 89]. As shown in Figure 2.17, the structure of Figure 2.14 can be viewed as a hypercube of order 2 in which four nodes are placed in corners of a square and each one shares a DPM with each adjacent node. In addition, the diagonal nodes are

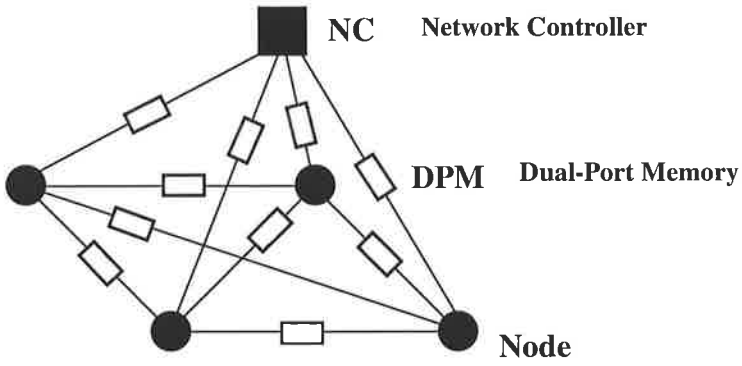

Figure 2.17 Another view of RSM architecture

RSM structure can be viewed as a fully connected network for four nodes using DPMs. In addition, each node has a link to a network controller.

interconnected using other DPMs and each node has a link to a network controller. With the exception of the diagonal links, this structure is similar to the top part of the structure shown in Figure 2.15.

With the contributions from one of the authors of [Jagadish+ 89], the continuation of this research has been presented in [Campbell+ 96]. It was reported that a multiprocessor called COMPS (COmmon memory Message Passing System) was being developed in Curtin University of Technology in Perth, Australia and used a dual-port memory as the communication link between two nodes. A five-node prototype system was reported as being in its final stage and near completion. The results of this system will be reviewed in Chapter 4.

4 Convergence

Shared address space and message passing are two clearly distinct architectures. However, the evolution of hardware and software has gradually blurred the boundary and substantial convergence has taken place [Culler+ 98]. Some of these are explained below.

At the user level, most shared memory machines also support send/receive operations used in message passing through shared buffer storage. In addition, a shared virtual

address space can be established on a message passing system. A group of processes can have a region of shared address space, but each process has access to its local pages. When a non-local page is addressed, a page fault occurs and the operating system initiates a message passing transaction to transfer the missing page and map it to the user address space [Culler+ 98].

As explained in the previous sections, shared memory can be used as links for message passing. Furthermore, with the advances in the design of scalable interconnect networks, several machines, which may be shared memory in their own right, are interconnected to operate as a parallel machine on individual large problems or as many individual machines on a multiprogramming load [Culler+ 98]. Such systems can utilize the advantages of both architectures.

The Stanford DASH multiprocessor (abbreviated for Directory Architecture for Shared Memory) was designed to investigate the scalability issues for shared-memory multiprocessors. As in message-passing machines, the main memory in DASH is distributed among the processing nodes and a scalable interconnection network is used to connect the nodes together. Unlike message-passing machines, however, the processing nodes share a single global address space. The DASH architecture thus combines the scalability of message-passing machines with the ease of programming associated with single address space machines [Lenoski+ 91].

The FLASH multiprocessor (abbreviated for Flexible Architecture for Shared Memory) also developed in Stanford supports distributed shared memory and message passing while minimizing both hardware and software overhead. Each node in FLASH contains a microprocessor, a portion of the machine's global memory, a port to the interconnection network, an I/O interface, and a custom node controller called "MAGIC". The MAGIC chip handles all communication both within the node and among the nodes, using hardwired data paths [Kuskin+ 94].

The underlying machine structures for message passing and shared address space have converged toward a common organization, represented by a collection of complete computers, expanded by a "communication assist" connecting each node to a scalable

communication network. Thus, it is natural to consider supporting aspects of both in a common framework [Culler+ 98].

5 Shared memory using multiport memory

Multiport memories provide several ports for accessing the memory cells and each node of a multiprocessor can be connected to one of the available ports exclusively. All the nodes can simultaneously read from different memory locations. They can even read from the same memory location concurrently. In contrast to reads, simultaneous memory writes are only possible if different memory locations are used. If two or more concurrent writes to a memory cell are requested, only one of them should proceed.

The use of multiport memory as in Figure 2.18 can improve the performance of a shared-memory system significantly. This is because concurrent memory access is possible without the overhead of the bus system or the interconnection network. However, the main problem is that the available multiport memories have very few ports (maximum 4 ports at the time of conducting this research) and small capacities (2-4 KBytes for 4-ports) [ICmaster 99]. This small range is not sufficient to build a shared memory suitable for a relatively large number of nodes. There are several issues in the design of a large-scale multiport memory that need to be resolved before producing multiport memories with large capacity and more ports. Some of these issues are the complexity of the memory cells, the huge number of connections inside the chip, large pinout required by the chip, and a preventing mechanism for avoiding concurrent writes, or even concurrent read-write on the same memory cell.

The small port count and limited capacity of multiport memories make their use as a shared memory a challenging task. As explained in Chapter 3, this area has not been explored in depth, and only very few structures for interconnecting nodes in small systems have been proposed. Furthermore, no work has been reported previously in evaluating a communication scheme based on multiport memories and no attempt has been made to design the required memory management and associated communication protocol. Hence, the openness of the area, the challenging nature of it, and the expected advantages over other methods make it an excellent research area.

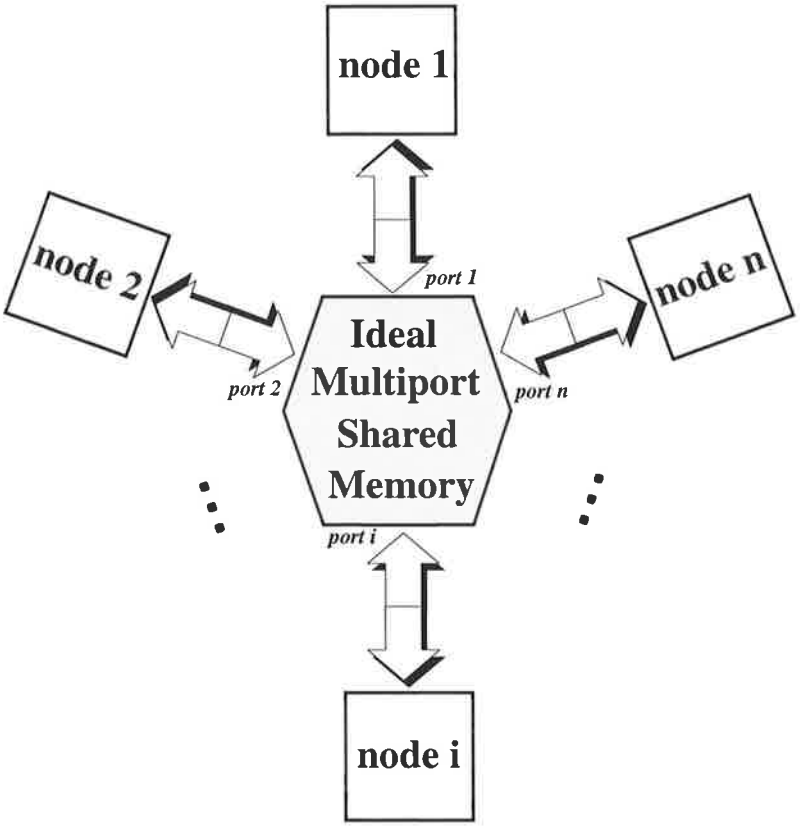

Figure 2.18 Multiport shared memory

Each node can be connected to one of the ports of an ideal multiport memory. They can communicate with each other through the shared memory.

As multiport memory is the focus of this thesis, its use in interprocessor communication will be discussed in depth in Chapter 3. The proposed structure is based on multiport memories with limited size and port count. A small shared memory is created for interconnecting a small number of nodes. The memory is very small to be used as a normal shared memory, but it is large enough for use as a communication medium for message passing only. The scaling of the structure is performed by a network of multiport memories in which data is exchanged among the nodes using message passing. The structure of multiport memory cells and new circuits to resolve some of the issues in designing multiport memories are presented in the Appendix.

6 Conclusion

Interprocessor communication is a vital task for sharing data among the nodes in multiprocessors. Two major architectural techniques for sharing data in multiprocessors are shared memory and message passing. In shared memory systems nodes have access to a common memory using different techniques such as a single bus or interleaved memories. The communication among nodes is implicitly performed by load and store instructions.

A conventional single-port memory can serve as a shared memory using a single bus. This structure is only useful for a small number of nodes. Interleaved shared memory can provide higher memory bandwidth than a single-bus system and can be designed using multiple-bus, crossbar switch, or multistage networks.

In message passing systems, the nodes are connected by an interconnection network and they can share data by sending and receiving messages. Generally, there are explicit send and receive commands for communication. The interconnection network can use different topologies such as ring, mesh, or hypercube, and nodes can be connected using either conventional serial links or dedicated parallel links. In the latter case, a FIFO can be placed between two nodes and messages can be transferred in one direction by writing to and reading from it.

The advent of dual-port memory introduced new methods for interprocessor communication. A dual-port memory shared between two nodes can be used as a communication medium and nodes can communicate simultaneously in both directions using load and store instructions. Using appropriate structures, several nodes can be interconnected with dual-port memories.

Multiport memory provides more ports for concurrent access of memory. Ideally, it can be used as a communication medium for several nodes with very low overhead. However, the small number of ports and the limited capacity of the multiport memory make it less attractive. In the next chapter, a new structure for interprocessor communication based on multiport memory with limited port-count and small capacity will be presented in which shared memory is used as a link for message passing.

CHAPTER 3

Multiport Memory for Interprocessor Communication

3

Multiport memories can facilitate the interprocessor communication in multiprocessors. Communication can be performed in parallel streams by accessing a multiport shared memory through independent ports. This method is capable of increasing the performance and reducing the size, cost, and the number of required components and interconnections. In this chapter, first, the advances in multiport cell design are explored and its applications in system design are presented. Then, an interprocessor communication scheme based on multiport memory is proposed in which shared memory is used as a link for message passing. For small systems, the nodes can communicate directly in a group structure. Groups can be connected in a cluster, and a network controller would handle the inter-group messages. For expanded systems, the network controllers of various clusters can be interconnected by sharing a multiport memory at the top of the hierarchy. Finally, several issues related to multiport memories are discussed.

1 Multiport memory: background and previous work

Multiport memory offers new communication methods for multiprocessors and can provide high-speed communication using parallel streams. In a small-scale system using this method, each node can use a separate port to access a multiport memory shared among the nodes. One node can write data into the shared memory from one port, and other nodes can read it from other ports. Nodes can communicate in parallel and achieve high performance, provided that they do not interfere with each other. This is achievable by proper memory management schemes. The limiting factor in scaling this structure is the limited number of ports and the small size of true multiport memories. Hence, special structures need to be designed for building larger systems.

A multiport memory cell is similar to a dual-port memory cell, but there are more than two ports to access the cell. The block diagram of a 4-port memory is shown in Figure 4.2 on page 57. The internal circuit and operation of multiport memory are explained in detail in the Appendix.

The design of multiport memory and its applications are active research fields. Several structures for memory cells have been proposed and tested, and it has been utilized in many applications. More specifically, with its parallel paths, it is widely used in the design of the datapath for high performance processors. Using it for interprocessor communication for large systems is a challenging area that has not been explored much before, and is the objective of this study. In the following subsections, the advances in multiport memory cell design are explained and different structures proposed for building it are presented. In addition, several applications based on multiport memory including very few structures for interprocessor communication are demonstrated.

1.1 Multiport memory design

A multiport memory cell can be achieved by extending the structure of the dual-port memory cell, which in turn is derived from the single-port memory cell. The operation of single-port, dual-port, and multiport memories are explained in detail in the Appendix. Several variations in multiport memory design have been investigated by

researchers and different structures have been proposed and tested. Some of these will be mentioned in this section.

Earlier versions of multiport memory had very small capacity and limited number of ports. However, this was useful enough to enable datapath designers to group individual registers into a register file, which could be implemented by a multiport memory. One of the earliest use of multiport memory as a register file has been reported in [Dedrick 84]. It discusses the design of LFR08, an eight 8-bit register file from Logic Devices Incorporation with two read ports, two write ports, and one bidirectional port. The paper shows that the use of this chip in a bit-slice architecture processor like Am2903 is superior to the traditional approach. If used in pipeline processors, the reconfiguration of registers can be simplified and data routing through the system can be controlled by software.

[Maly+ 91] discusses the design of a memory chip for a 24-port global register file. A chip with eight read ports and eight write ports with capacity of 256x2 bits was fabricated as a base unit. In order to achieve the required 24 ports (8 write and 16 read ports), two chips were used and the write ports of the chips were connected together, but the read ports remained isolated. With this configuration, any write operation was performed on both of the chips, so that the data in the chips was the same. As each chip could be accessed by eight independent read ports, the number of effective read ports was increased to 16. The required word size could be created by connecting several memory chips in parallel. The base unit was fabricated using 2 μm CMOS technology.

In [Silburt+ 93], a family of modular memories was designed based on synchronous self-timed architecture. For a 0.8 μm BiCMOS process, nominal access time was 5.5 ns for 64-Kbit blocks of 1, 2 and 4-port SRAMs. The cell for 4-port SRAM was three times bigger than the cell for a single-port RAM. In [Lai+ 94], a new design methodology for SRAM cell has been proposed that utilizes fewer bit-lines to perform read/write access at lower cost. [Nii+ 95] reports a new proposal for cell design that contributes to the operation at high speed and low voltage. A 3-port test chip was fabricated in 0.5 μm CMOS SOG (Sea-Of-Gates) and could operate with the access

time of 4.8 ns at 3.3 volts. [Izumikawa+ 96] describes a sense circuit that can be applied to a single-ended multiport SRAM to accelerate the memory access 3.2 times.

[Zhi+ 96] discusses a compact cell design for multiport register file with one write port and three read ports for implementation in 0.8 μm CMOS technology. [Chin+ 96] describes a 3-port register file fabricated in 0.6 μm BiCMOS technology using a 3.3V power supply. The pin-to-pin access time was measured as 1.3 ns. [Franch+ 97] also reports a 3-port register file fabricated in 0.25 μm CMOS technology with an access time of 640 ps.

Other researchers have tried to automate the design of multiport memory to the requirements of the user. [Hana+ 89] reports a multiport RAM compiler that can generate user definable memories. Up to four read and two write ports with a capacity of up to 128 words and word length of 64 bits can be designed. The compiler places and routes various cells comprising the RAM, and generates a floor plan and a layout for fabrication of the chip. A more powerful compiler is reported in [Shinohara+ 91], which is more flexible in layout and port organization, and can generate faster chips. The compiler is capable of designing read, write, or read-write ports with word lengths of up to 72 bits, and memory size of up to 32 Kbits 3-port, or 16 Kbits 6-port.

The trend in multiport memory research is to design faster memory cells that occupy less space and can be integrated more densely. In addition, successful attempts have been made to use higher number of ports.

1.2 Innovative structures for multiport memory

Rather than designing true multiport memory, some researchers have used innovative structures to build pseudo multiport memory with the efficiency of true multiport memory. A few examples are given below:

[Endo+ 91] describes a pipelined time-sharing access (PTA) technique that can be used for the construction of high-density multiport memories with a large number of ports. In this technique, an N-port memory can be designed with N/2-port memory cells resulting in a smaller chip size and wider operating margins. A 4-port memory with the

capacity of 8 KBytes was designed and fabricated in 0.8 μm n-well CMOS technology using PTA technique and had a cycle time of 16 ns. The chip area was only 1.2 times larger than the equivalent dual-port memory.

A new approach for designing 4-port memories is presented in [Hirano+ 98]. The multiport memory is called Shared DRAM (SHDRAM) and it uses four DRAM *mats*. The data written to the sense amplifier of a mat is broadcast to the sense amplifiers of all the mats by using special fast broadcast buses, and is written to the corresponding memory cells of each mat. Hence, all the mats have identical data that can be accessed by four separate ports. An 8-Kbit test chip was fabricated and tested successfully using this technique.

[Landsberg+ 93] describes the modelling and design of a 6-port CMOS static RAM. The cell is relatively large, but it can swing the bit-line very quickly. The memory system is a true multiport memory and can be read and written independently and simultaneously subject to consistency constraints. The model can be extended to characterize general multiport memories.

1.3 Application of multiport memory

Researchers have utilized multiport memory to develop innovative structures for many applications. The main use of multiport memory has been in the design of high performance processors. It has been also used in the design of special purpose systems. The other application area is the communication among processors, which is the focus of this study. Some of these applications are explained below:

1.3.1 Processor design

Multiport memory plays an important role in the datapath of most high performance processors. Designers use on-chip multiport memory to group the processor registers into a register file. Various functional units within the processor can access different registers simultaneously using separate read/write ports and high throughput is achievable.

Register files has been an integral part of almost every RISC or superscalar machine. [Bakoglu+ 90] describes the hardware overview of the IBM RISC System/6000 processor, and [McGeady 90] discusses the structure of i960CA superscalar processor from Intel using a 6-port register file.

[Asato+ 95] discusses the design of a register file with 10 read ports and four write ports and size of 116x64 bits. The access time of the register file is 3.8 ns and it was used in a four-issue V9 SPARC-architecture superscalar processor operating at 154 MHz.

Multiport memory has been also used in the design of very long instruction word (VLIW) processors. Examples can be found in [Labrousse+ 90] and [Nakamura+ 96]. Moreover, a data-flow CPU has been designed using an on-chip 3-port *smart memory.* Beside conventional read/write operation, the smart memory had content addressability and support for branch prediction and exception handling [Uvieghara+ 90].

In order to establish a connection from a register to a functional unit in different control steps, registers should be assigned properly to the memory ports and interconnect minimization becomes more important. [Ahmad+ 93] presents a design methodology for datapath synthesis using on-chip multiport memories. The proposed technique can be applied to scheduled algorithms to reduce the design space. This method can group variables into a minimum number of multiport memories depending on the available ports and the access requirements of the variables. The method also can minimize the interconnection hardware such as buses, multiplexers, or tri-state buffers. Similar techniques can be found in [Lee+ 95] and [Mandal+ 96].

Researchers have worked on efficient memory design techniques to fulfill high bandwidth demand for the memory of fast processors. As explained in previous sections, some researchers have designed faster cells with several read/write ports. Others have developed special testing models to test the embedded multiport memory using built-in self-tests (BIST). These models take into account the complex couplings resulting from simultaneous access of memory cells to ensure very high fault coverage. Examples can be found in [Castro+ 92], [Matsumura 95], and [Yuejian+ 97].

1.3.2 Special purpose systems

Before the availability of true multiport memory, researchers have designed systems in which multiport memory was generated with innovative structures. [Shibayama+ 87] reports a knowledge-based machine that was implemented using a memory system called multiport page-memory (MPPM). [Strohman+ 89] describes the control system for the Cornell Electron Storage Ring (CESR) that used a 16-port memory and could be accessed by many computers. The multiport memory was built with conventional DRAM, 16 FIFO buffers, and semaphore registers. [Litazie+89] explains the concept of serial multiport memory and proposes a structure for designing a multiprocessor using this concept. The serial multiport memory used a conventional memory that was connected to several shift registers. Each node was connected to one shift register using a high-speed serial link. [Mzoughi+ 93] reports the implementation of this multiprocessor and discusses several design issues.

After the release of the first commercial true 4-port memory by IDT (Integrated Device Technology), researchers have proposed innovated structures for the design of special purpose systems. [Handy 90] proposed several architectures such as a pipelined FFT processor, an array processor, and a multiported image memory. He claimed that considerable speed increase could be achieved by the new architectures. [Nanduri 91] explains how multiport memory can improve system bandwidth, data flow rate, and system speed requirements. Various examples were presented to illustrate the potential benefits of these devices in alleviating the bottlenecks and ensuring that a balance was attained in the system. [Lin+ 96] describes the design of a matrix multiplication engine for graphics and DSP applications using 4-port memories.

Multiport memory has been used in the design of multiprocessors for artificial intelligence applications. In [DeMara+ 91], the design of a parallel architecture called Semantic Network Array Processor (SNAP) is explained in which 4-port memories were used for *marker passing* in the clusters. [DeMara+ 93] reports the implementation of the first generation SNAP-1 system and evaluates its performance. [Zhang+ 95] discusses the scalability of a parallel signal processing system using shared multiport memory for neural networks.

Moreover, some specific purpose multiprocessors have been designed using multiport memory. For example, Aladdin is a distributed memory multiprocessor designed for automatic target recognition and radar processing applications. The *multimode* memory used in its design is an 8-port image memory with a variety of input and output scan patterns. It is capable of simultaneously inputting 16-bit data to four ports while outputting 16-bit data from other ports [Lum+ 92].

ATM switch design is another area in which researchers have recently focused on the use of multiport memory. Novel switch architectures based on using multiport memory as buffers have been proposed. Several switches were fabricated on this basis and tested successfully with high efficiency and speed [ElGuibaly+ 96] [Kornaros+ 97].

1.3.3 Communication

Several communication structures have been proposed and implemented using dual-port memories. Some examples were explained the in the previous chapter. However, only very few structures for communication using multiport memories have been proposed. [Handy 90] explains one of the earliest proposals. As shown in Figure 3.1, the nodes and 4-port memories are arranged in a two dimensional array. Each node is connected to four 4-port memories, and each memory is connected to four nodes. Neighbouring nodes can communicate directly, but non-neighbouring nodes should involve intermediate nodes for their communication. This architecture was termed a *Computing Fabric Hypercube.*

An equivalent architecture has been also proposed in [Varshneya+ 94]. This architecture is almost identical to the previous one except that the boundary of the structure is all nodes. The dotted square in Figure 3.1 illustrates this structure. Moreover, a similar architecture based on 6-port memories was also proposed. In this structure, six nodes arranged as a hexagon share a 6-port memory and several hexagons can be connected together to create a two dimensional grid similar to Figure 3.1.

The above communication structures based on multiport memory were only on the proposal stage. Hence, the detail of communication was not explored and no performance evaluation was carried out in either of the proposals.

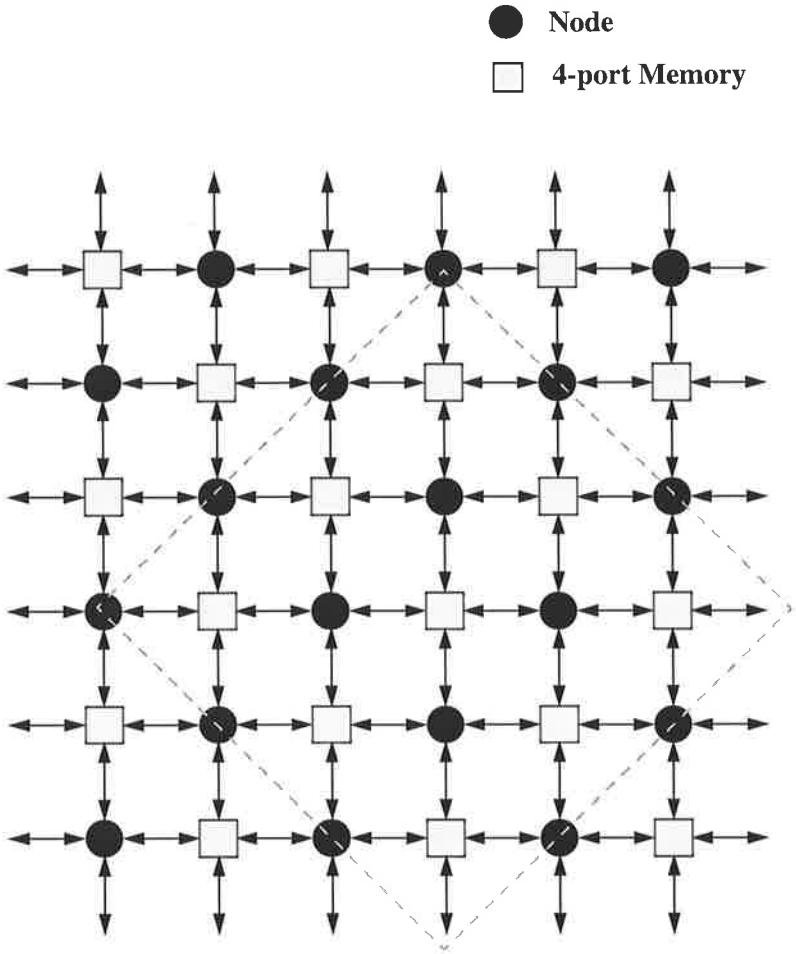

Figure 3.1 Communication in a grid with 4-port memory

In a two dimensional grid, each 4-port memory is surrounded by four nodes and each node by four memories. Adjacent nodes can communicate directly through 4-port memory, and non-adjacent nodes should communicate using intermediate nodes. The dashed square shows a similar proposal in which the boundary is all nodes.

2 Proposed structure for interprocessor communication

In this section, a new structure for interprocessor communication using multiport memory is proposed. In this structure, shared memory is used as a medium for message passing between the nodes. The proposal is presented in three stages: communication of nodes in a group, communication in a cluster, and communication in a network. Each stage is subsequently discussed in detail.

2.1 Communication in a group

Multiport memory can be used to interconnect several nodes in a group. As shown in Figure 3.2, each node is connected to a separate port of a multiport memory. The nodes can communicate with each other directly in two steps:

1. The transmitting node writes the message in the shared memory.
2. The receiving node reads the message from the memory.

Data transfer between a node and its memory can be performed in 16, 32 bits or more depending on the width of the node's data bus. Because nodes can read from or write to the memory using a single instruction, the rate of memory data transfer can be very high and the nodes can communicate with each other very efficiently. Moreover, by using appropriate memory management, several communications can be performed simultaneously.

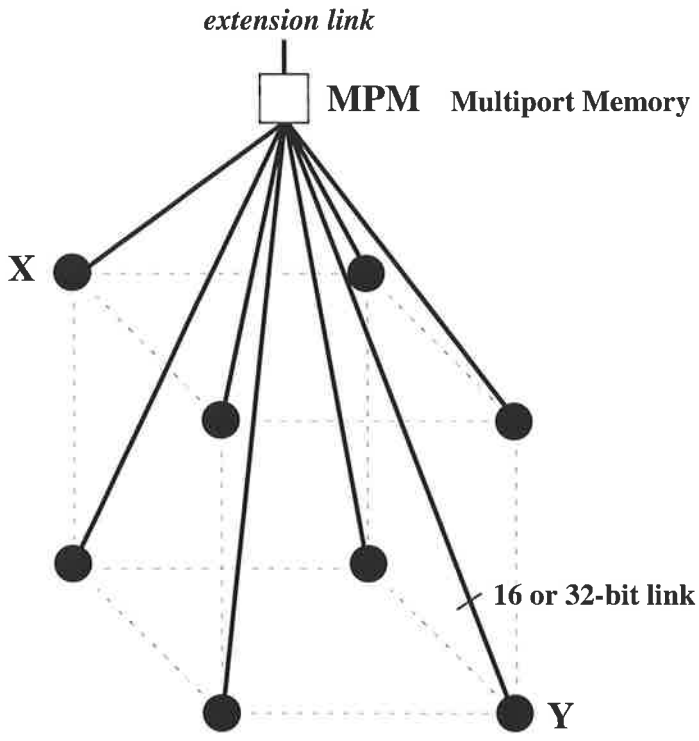

Figure 3.2 Communication with multiport memory in a group

In a group, the nodes are connected to a common memory through a separate port and they can communicate directly. As an example of communication using this structure, node X writes the message to the memory and node Y reads it. The extension link is used to expand the structure.

This structure is in fact a shared memory structure and can be compared to a system using interleaved memory. Its design is much simpler than the multiple-bus, crossbar switch, or multistage networks used in the interleaved memory systems and it does not have the delay and overhead of such networks. However, this structure cannot be used to create a true shared memory system because of the limited capacity of the multiport memories. As mentioned before and will be explained later in more detail, this structure uses multiport memory as a link for message passing and a small shared memory is adequate for this purpose. Unlike other shared memory systems, the small shared memory in this structure is exclusively used for communication purpose.

Compared to communication structures using dual-port memories discussed in the previous chapter, this structure can increase the throughput and reduce the size, cost, and the number of required components and interconnections as explained in Chapter 5.

The number of available ports in multiport memories is another limiting factor in expanding the system and only a few nodes can be grouped together using this structure. If more nodes are to be connected, another structure should be used in which different groups are interconnected using extension links. The expansion of the structure to cover more nodes is explained in two stages in the subsequent sections.

2.2 Communication in a cluster

The number of ports on the available multiport memories is very limited and the memory capacity is small. This is because increasing the port count increases the size of memory cell as well as the access time. In addition, with more ports, the complexity of the internal wiring increases significantly and the area available for memory cells is reduced. With advances in technology and achievements in cell size reduction, it can be expected that memories with more ports will become available in the future. However, as explained in the discussion section, the number of ports would be still limited and the size of the memory would be small. This suggests that the number of nodes connected in a group will always be a limiting factor and suitable structures are required to overcome the restriction of limited port count.

The structure of Figure 3.3 is proposed for the interconnection of a cluster of groups of nodes. In this structure, the groups are connected to a network controller (NC) to form a cluster. The network controller is a special processor that has permission to access the shared memory of different groups. Its main task is to get a message from the sender in one group and pass it to the appropriate receiver in another group. The nodes in different groups communicate through the NC in two hops. In the first hop, the transmitter sends the message to the NC. In the final hop, the NC reads the message and transfers it to the shared memory of the group in which the receiving node is located.

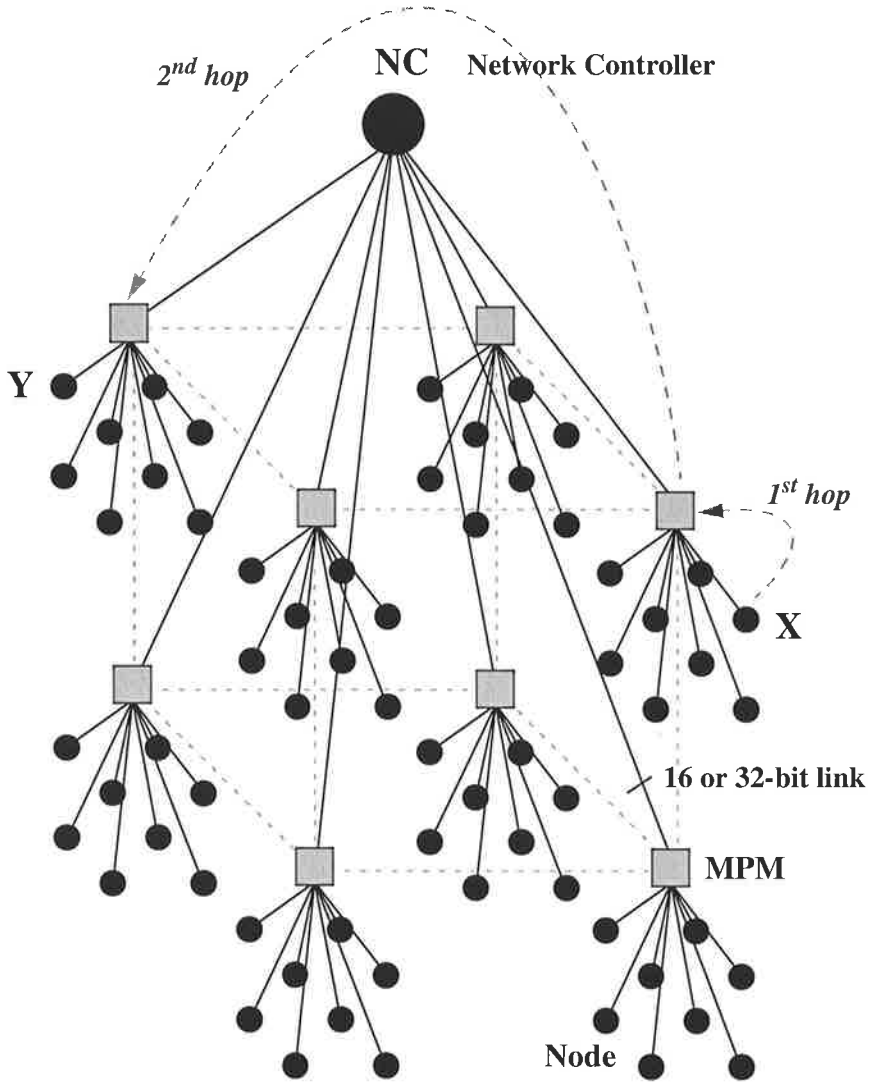

Figure 3.3 Communication in a cluster

In a cluster, several groups are interconnected by a network controller. The nodes within each group can communicate directly. If the receiver is in another group, the communication takes place in two hops. First, the transmitter (X) sends the message to the NC through the MPM of its group. Then the NC delivers the message to the receiver (Y) by the way of the MPM in receiver group.

The message can be collected by the receiver from the relevant shared memory. This process takes longer than a direct communication within a group.

Several local shared memories have been used in a cluster structure and as the mechanism of data transfer demonstrates, they are used as links for message passing.

If several nodes need to concurrently use the NC for inter-group communication, the NC will be overloaded. In this case, the system performance will drop because of delays in delivering the messages by the NC. Hence, the number of groups in a cluster should be limited. It is anticipated that under heavy communication traffic, this structure might be inefficient, especially if several groups are connected to the NC, or the request for inter-group message transfer is high. Based on the results of simulations described in Chapter 6, modified versions of the cluster structure will be presented in Chapter 7.

2.3 Communication in a network

A cluster can accommodate a limited number of groups. In order to interconnect a large number of nodes, the structure of Figure 3.4 is proposed. In this structure, the NCs are interconnected using an extra multiport memory in the upper level of the hierarchy. As explained in the previous section, the nodes communicate directly within groups and use the NC of the cluster for message passing to the other groups in the same cluster. If the receiver is in another cluster, the message is handled by two network controllers in three hops. In the first hop, the transmitting node sends the message to the NC of the cluster. In the second hop, the NC sends the message to the NC of the receiver cluster by transferring it to the multiport memory at the top level of the hierarchy. In the third hop, the NC of the receiver cluster transfers the message to the multiport memory of the group where the receiving node is located. The message can be collected by the receiving node from this memory.

This introductory structure is useful for light traffic conditions. This is because the NCs have limited communication capacity, and if they have to handle several messages, they will be overloaded, resulting in delays in message delivery and inevitable performance loss. After reviewing the results of simulations, an improved communication structure will be introduced in Chapter 7.

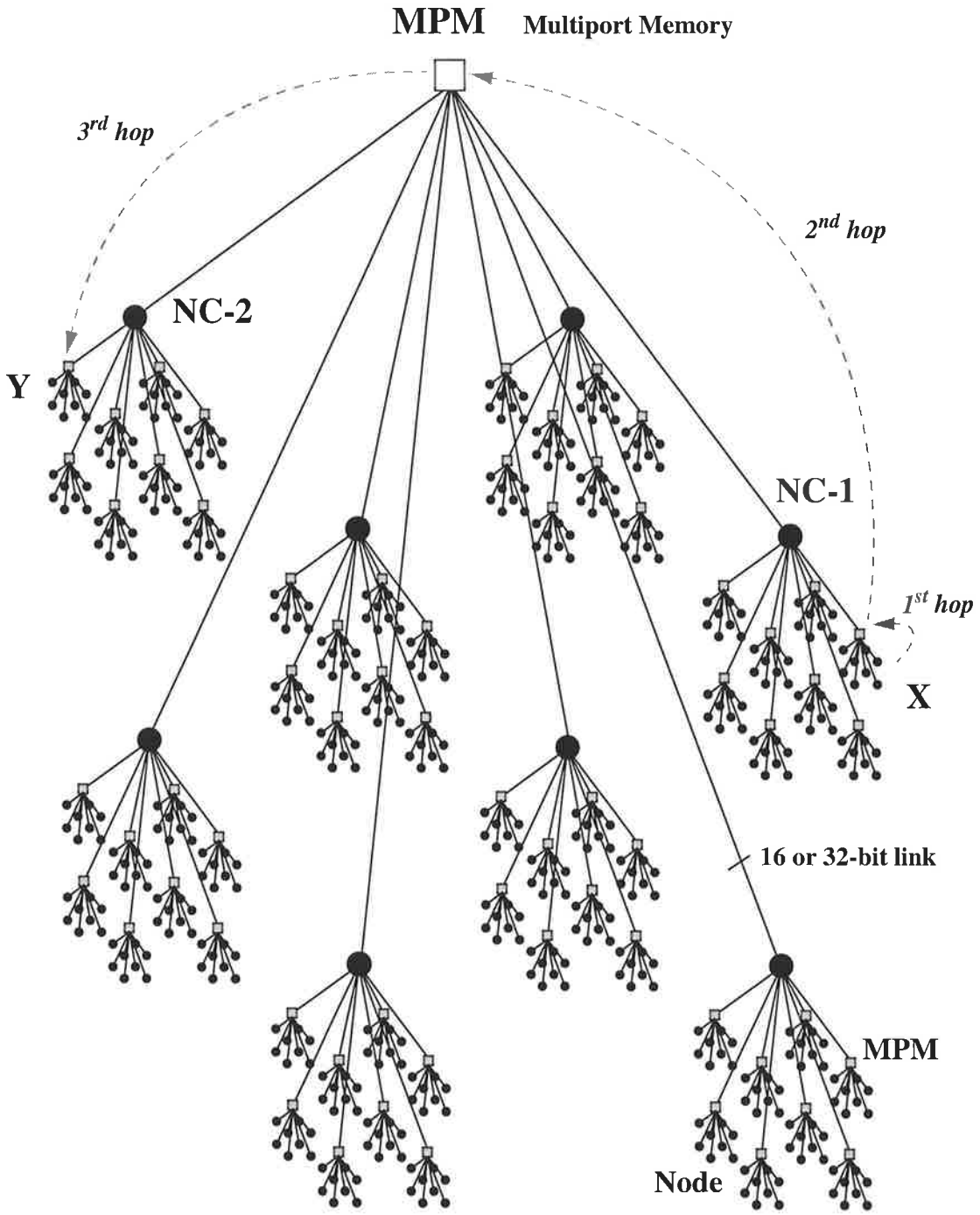

Figure 3.4 Communication in a network

In the proposed structure for a network, the clusters are interconnected by sharing a multiport memory at the top of the hierarchy. An inter-cluster message from node X to node Y should be sent in three hops. First X sends the message to NC-1. Then NC-1 transfers the message to the NC-2 located in the receiver cluster. Finally, NC-2 sends the message to Y by the way of the group MPM.

3 Verifying the structure

Multiport memory has not been widely used for communication in large scale. As explained in previous sections, only few structures have been proposed in this regard and no evaluation on these structures has been reported.

The proposed structure in this thesis was verified in two stages. As a hardware model, a prototype system was designed and constructed to check the feasibility of the structure and to evaluate its performance. Based on the results of this prototype, a simulation model was created to further assess the prototype itself and its expansion to larger systems. Both models are briefly explained here, and they will be analysed in depth in separate chapters.

3.1 MultiCom, a hardware prototype

In first stage of evaluating the proposed structure, a small multiprocessor called MultiCom was designed and built. In this prototype, four nodes were interconnected using 4-port memories as the communication medium. Chapter 4 describes the hardware design and implementation of MultiCom and discusses its programming. Chapter 5 discusses the memory management required for MultiCom and explains different memory allocation schemes used to control the shared memory. The communication protocols and the achieved results are also presented in Chapter 5.

3.2 Simulation model

MultiCom was a valuable device to characterize the behaviour of the nodes and multiport memory used in the proposed structure. A software simulation model was constructed on the basis of MultiCom. As described later in Chapter 6, initially the model of MultiCom was created and its timing was fine-tuned to produce the same results of MultiCom. Then the model was gradually expanded to simulate larger groups, a cluster of groups, and several clusters in a large network. The simulation model revealed that some modifications were necessary. Hence, the structure was improved and re-evaluated in Chapter 7.

4 Discussion

As multiport memory is the main component of the proposed structure, its availability with large port count is discussed here and the requirement on the pinout is explained. In addition, the effect of overloading a network controller in the proposed structure is discussed.

4.1 Availability of multiport memory

The structure of multiport memory is explained in detail in the Appendix. The major problems in designing a large multiport memory are the large size of memory cells, and more importantly, the extent of internal wiring. If an extra port is added, more switches must be used in the memory cell, and the cell size will increase. In addition, increasing the port count increases the number of required internal buses, and distributing them inside the chip increases the amount of internal wiring. This requires the use of large connection matrices for routing vertical and horizontal lines. The connection matrices occupy a large portion of the chip area and reduce the size of the area available to memory cells that are already bigger in size. Hence, for a given chip area, increasing the port count considerably decreases the capacity of the memory.

The feasibility of true multiport memories has been discussed in [Forsell 94]. The paper concluded that true multiport memories were feasible and anticipated that they would be available with larger port counts and greater capacity in the future. On the other hand, the literature review presented at the beginning of this chapter indicates that multiport memory is an active research field and many structures for cell design and building multiport memories have been proposed. Successful prototypes with as many as 16-ports (8 write, 8 read) have been fabricated, and faster cells with access time up to 640 ps for three ports have been designed [Franch+ 97]. In addition, several compilers are available for automatic design of small multiport memories.

For some time, the largest multiport memory commercially available had four ports and the capacity of 2 KBytes [IDT 95]. During the design of MultiCom, a newer version with the size of 4 KBytes was released [IDT 96]. In fact, MultiCom was implemented with the engineering samples of this product. The fastest 4-port memory available at the

time of design of MultiCom had an access time of 20 ns. IDT released a dual-port memory with a capacity of 32Kx36 bits and access time of 4.2 ns (equivalent to the speed of 133 MHz) in 1999 [IDT 99]. It has been claimed that this product was 33% faster than any other 36-bit offering. IDT has recently upgraded its 4-port memory to 64Kx18 bits and access time of 5 ns [IDT 02].

The simulation model presented in Chapter 6 confirms that the proposed structure does not require a large multiport memory. The capacity of 4 to 8Kx16 bits is adequate for efficient operation of the system. As explained later, even a smaller capacity will be adequate if a wider data path is used. Hence, the structure can be implemented using 9-port memories, even with a small capacity. In the light of VLSI technology and with the trend discussed earlier, the availability of large multiport memories is not far away. In this author's opinion, proposing suitable applications for large multiport memories will boost their availability and increase their commercial production.

4.2 Pinout of multiport memory

In the proposed structure, groups and clusters require a memory chip with nine ports. As explained in section 4.2 in Chapter 7, even the improved structure can be built using 8-port memories. As there are separate address, data and control buses for each port, the number of pins can increase considerably for large memories. Simple calculation shows that memory pinout is not a major problem. As shown in Figure 4.2 on page 57, in the 4-port memory used in MultiCom with a capacity of 4 KBytes, each port requires 23 lines (12 address lines, 8 data lines, and 3 control lines). With several power and ground lines the chip is available with 108 pins. With similar calculation, a 9-port memory with the same capacity would require around 240 pins, a 16-bit version around 320 pins, and a 32-bit version around 480 pins. These requirements can be met by current packaging technology.

Traditional packaging technology uses wires for interconnection between the die and the substrate, but advanced packaging technologies use different approaches. For example, the *flip chip* packaging technology used by Xilinx on its high-performance FPGA chips utilizes conductive bumps that are placed directly on the area array pads of the die surface. This technology offers excellent thermal performance, higher frequency

switching, and higher I/O density. The packaging FF1517 (Flip-chip, Fine-pitch) used by Xilinx accommodates 1517 pins in a 40x40 mm package with 1.0 mm pitch size [Xilinx].

As far as pinout is concerned, advanced packaging technologies such as flip chip makes it possible to package a 4Kx64, 9-port memory with around 850 pins, and a 128-bit version with around 1500 pins. Although not required, one approach to reduce the number of pins is to multiplex address and data buses, or low and high data buses at the expense of slightly increased access time. In addition, memories with narrow data widths can be used in parallel to achieve wider data widths. In this regard, using innovative packaging technologies such as the ones offered by [DensePac] can be very beneficial.

4.3 Overloading of network controller

The structure of a cluster relies heavily on the NC for inter-group communication. Increasing the number of messages between the groups can overload the NC and reduce the throughput. This point became more obvious after analysing the results of simulations in Chapter 6. The structure of a cluster will be modified to reduce the overloading effect of the NC in Chapter 7. Similar modifications should be applied to the network structure as it uses the same cluster structure. It is worth mentioning that the structures presented in this chapter are based on the original plan proposed at the start of this research. The modified and final structure will be presented in Chapter 7.

5 Conclusion

In this chapter, a structure for interprocessor communication using multiport memory was proposed. In this structure, the nodes can communicate by writing and reading the message from a shared memory. As the number of ports available on multiport memories is very limited, the structure should be arranged in a hierarchy for large systems. A limited number of nodes can be connected directly to a multiport memory to create a group; a network controller is used to interconnect different nodes in a cluster, and several NCs are interconnected with a multiport memory to create links between

the clusters. Several small shared memories have been distributed in the network and they have been used as links for message passing. The evaluation of the structure will be carried out with a hardware prototype and a simulator as explained in the subsequent chapters.

Overall, as communication in the proposed structure is performed in parallel streams with memory access instructions, it is expected that the system will achieve a high performance.

CHAPTER 4

MultiCom: A Hardware Model

4

MultiCom is a small MIMD multiprocessor that has been designed to check the efficiency of interprocessor communication through multiport memories. In this prototype, four nodes are interconnected using 4-port memories and the communication is performed by writing and reading the memory in parallel streams. This chapter discusses the structure and implementation of MultiCom and explains the node processors, 4-port memory, interface logic, and programming. For the memory management of the multiport memory, a primary method called static allocation is introduced briefly, and the results are compared to serial communication. More details of static allocation, and more advanced memory management schemes are presented in the next chapter.

1 Hardware design

The first step taken to verify the validity of the proposed structure for interprocessor communication was to build a prototype system with the nodes interconnected by multiport memories. In this step, a small experimental system called MultiCom was designed and constructed, and the communication scheme was tested on it. The performance of this system was used to evaluate the effectiveness of the structure, and the outcomes were used in modelling and designing larger systems.

MultiCom was based on four DSP processors from TI (Texas Instruments) as nodes, and two 4-port memories from IDT as shared memory. The nodes could communicate with each other through the shared memory simply by writing the information into the memory from the transmitter side, and reading the memory from the receiver side. With suitable memory management, all the nodes could read from or write to the shared memory concurrently without interfering with each other. This section briefly describes the nodes and the 4-port memory of MultiCom, and explains the block diagram of the system and the interface circuit.

1.1 Nodes

The nodes of MultiCom were TMS320C50 DSP processors from TI. The main reason for choosing them as nodes was their availability in small and cheap modules called DSK (DSP Starter Kit). Each module was supplied with a TI assembler and debugger to run on a PC connected as a host. Figure 4.1 shows the layout of DSK. Each module consists of the following features [TI 96]:

- TMS320C50 16-bit integer DSP processor with 50 ns cycle time (20 MHz)
- modified Harvard architecture with separate banks for program memory, data memory, and input/output
- 4-stage pipeline with effective cycle of one or two per instruction
- 10Kx16 bits on-chip RAM, and 32 KBytes on-board boot ROM
- one 16-bit timer and four external interrupts
- one full duplex serial port (5 Mbps) and one TDM (Time Division Multiplexing) serial port useful for the multiprocessing environment

- analog interface chip with 14 bit A/D and D/A
- TI Assembler and debugger with RS232 link to PC

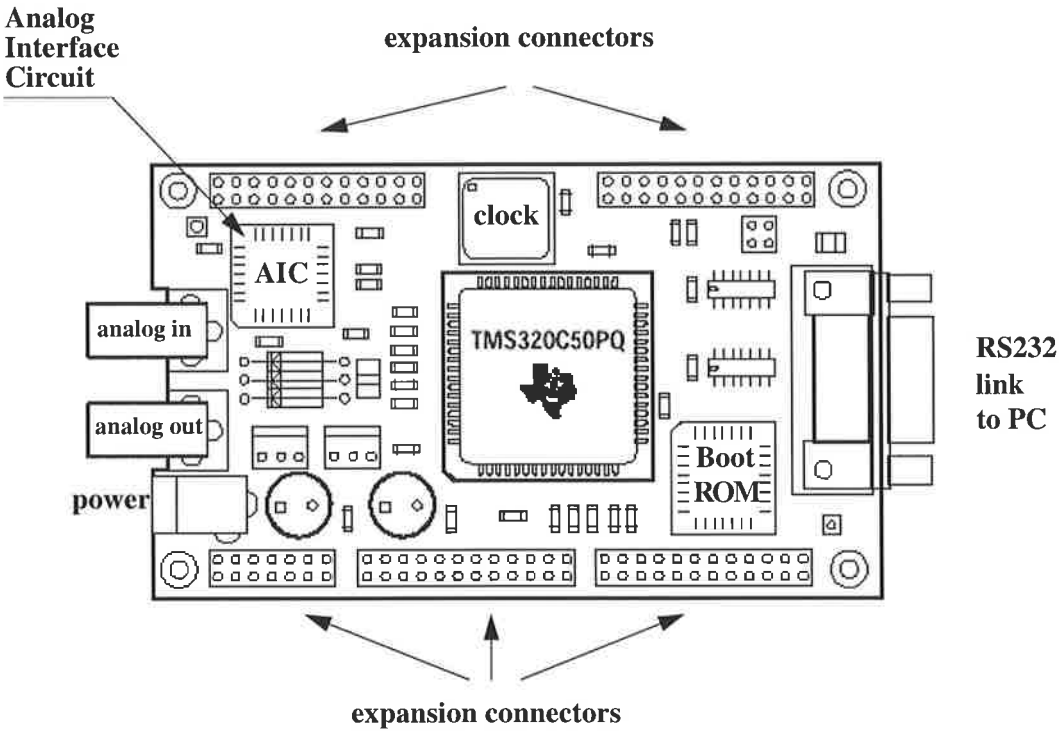

Figure 4.1 DSK as nodes of MultiCom

TMS320C50 DSK with a 16-bit integer DSP processor is used as nodes of MultiCom. It can be connected to a COM port of a PC using RS232 link

TMS320C50 is based on the modified Harvard architecture. Internally it has two separate buses for the program and data memories and can access both of them simultaneously in different stages of its 4-deep pipeline. For example, fetching a new instruction and reading an operand for another instruction can be performed on the same cycle. However, the two buses are combined to create a single external bus for accessing external program and data memories, or I/O ports.

For external memory, the write operation uses one extra cycle. This allows a smooth transition between write and any adjacent bus operation. Hence, for an external memory connected with zero wait states, a data read takes one cycle; however, a data write

requires two cycles [TI 97]. This information will be used in the calculation of the communication bandwidth for MultiCom in the next chapter.

1.2 Four-port memory

The 4-port memory used in MultiCom was IDT7054 from IDT with an access time of 35 ns and 108-pin packaging. As shown in Figure 4.2, the memory cells are organized in a 4-Kbyte array. There are four independent ports with separate control, address, and data lines. Each port is capable of performing independent and asynchronous access to read from or write to any location in the memory. It is the user's responsibility to ensure data integrity when accessing the same memory location from different ports [IDT 96].

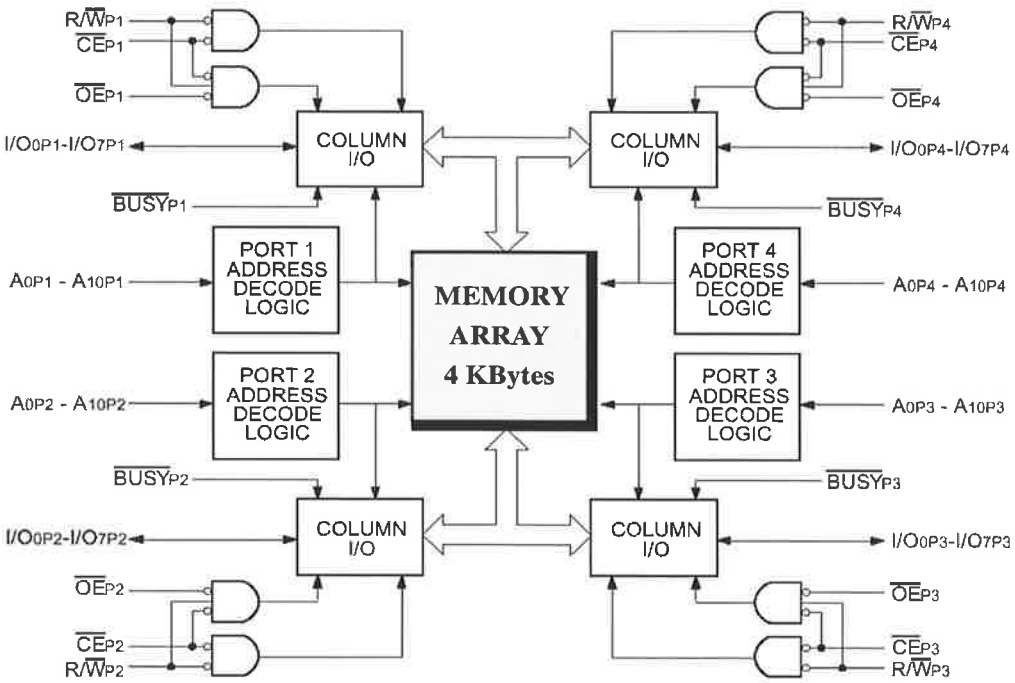

Figure 4.2 IDT7054 four-port RAM

The memory array can be accessed through four independent ports. No extra logic is implemented on the chip to facilitate the shared memory operations.

In general, there are two restrictions in using this chip [Stodieck 96]:

1. Simultaneous writes to the same location must be avoided.
2. Simultaneous read from a location that is being written could return incorrect data.

For a reliable memory operation, these limitations must be considered in the design of any system. As explained later, these issues were addressed in the memory management of MultiCom by using static or dynamic memory allocation.

1.3 Interface and hardware block diagram

The hardware block diagram of MultiCom is shown in Figure 4.3. The four nodes of this system were interconnected through 4-port memories with total capacity of 4Kx16

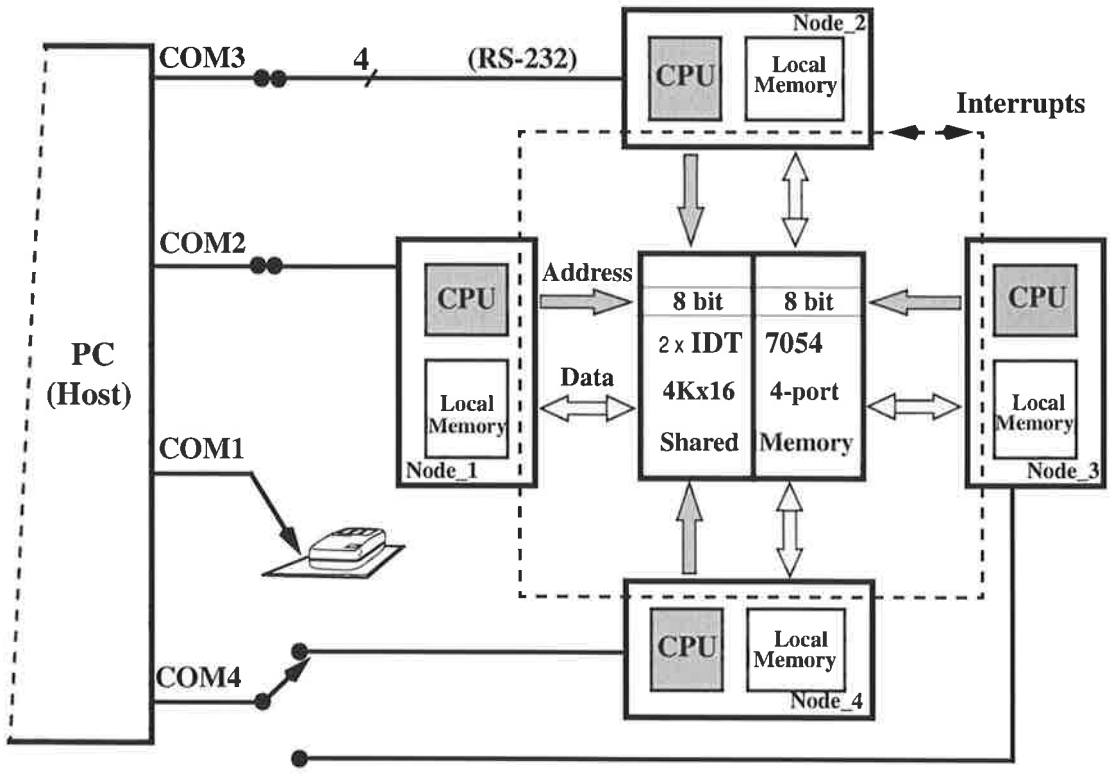

Figure 4.3 The block diagram of MultiCom

In MultiCom, two 4-port memory chips were used to interconnect four nodes. The host was a PC connected to the nodes through the available COM ports.

bits. This memory size was generated by connecting two IDT7054 chips in parallel. The interface circuit incorporated the required logic to connect the 4-port memory to each node. An interrupt bus was also created for signalling among the nodes. Each node could generate three interrupts for the other three nodes, and receive three interrupts from them. As explained later, the interrupts were used as part of the required handshaking for communication through the 4-port memory. Figure 4.4 shows the board designed for MultiCom.

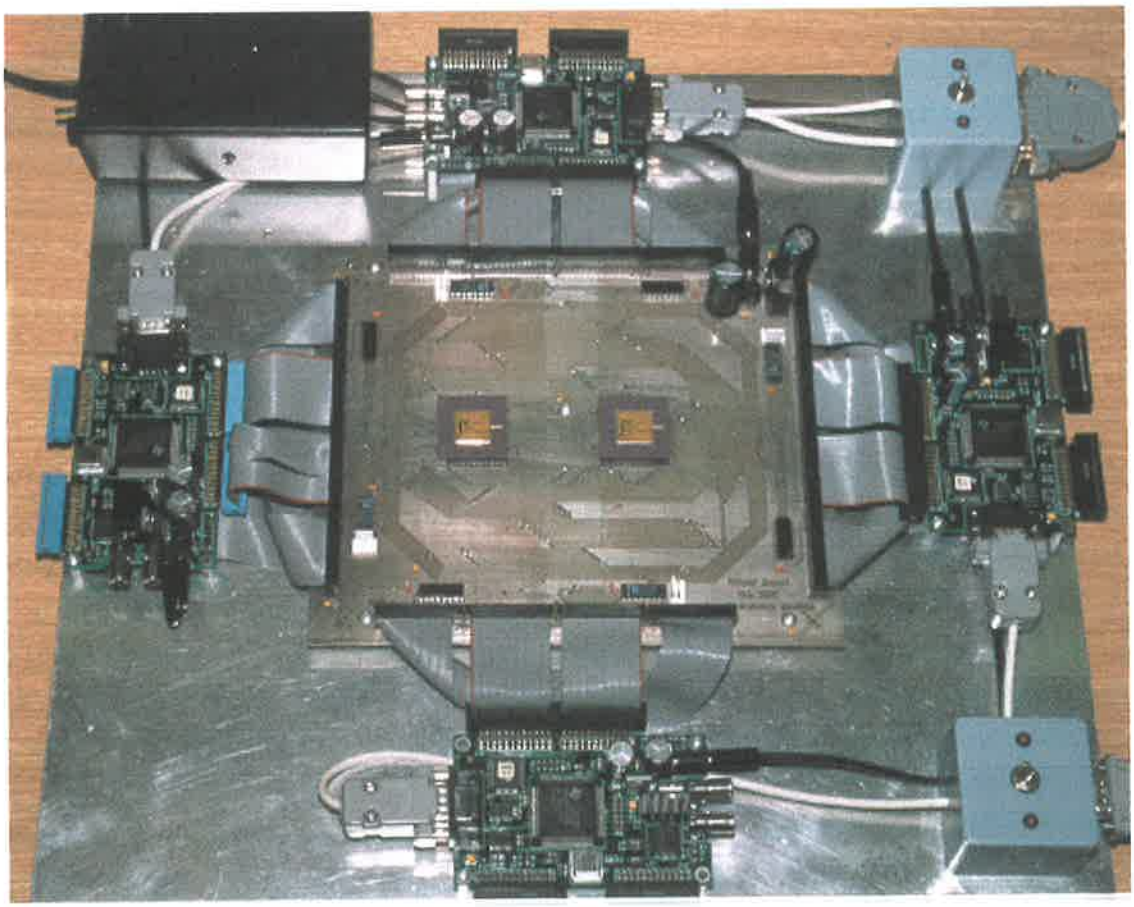

Figure 4.4 The board designed for MultiCom

Two 4-port memories were connected in parallel to achieve a 4Kx16 bits shared memory. Four nodes of MultiCom are shown on each side and each one is connected to one port of the multiport memory.

A PC was used as a host for all the nodes and each node was connected to one of its COM ports. The host could program the nodes through RS-232 link, and it could control or monitor the activities of the nodes using the debugger program. More details are given in the next section.

2 Programming

The programming language for MultiCom was the assembly language for the TMS320C5X series. The assembler program was supplied as part of the DSK module. A debugger was also available to run and test the programs in assembly language.

For programming the nodes, first an assembly language program was written for all the nodes in general. Then a program written in C was used to produce the local variables and codes for each node. Finally, the assembler was invoked to generate an individual executable file for each node.

The executable codes were downloaded to each node through the serial links of the host. For each node, a debugger program was running on a separate window. This window was used to control the nodes and transfer the results back to the host.

The on-chip timer of each node was used to measure the time spent on communication of the node from the beginning until all the nodes finished. This measurement was the basis of all the results. It will be explained in more detail later.

2.1 Synchronization

In order to get a correct time measurement, all the nodes must be synchronized to start at the same time. The synchronization was easily performed with the aid of the shared memory using the algorithm shown in Figure 4.5.

In this algorithm, one of the nodes is considered as the master node. Every other node registers its presence by writing a unique ID in the shared memory when activated. The master raises the sync flag when all the nodes are present. The details are as follows:

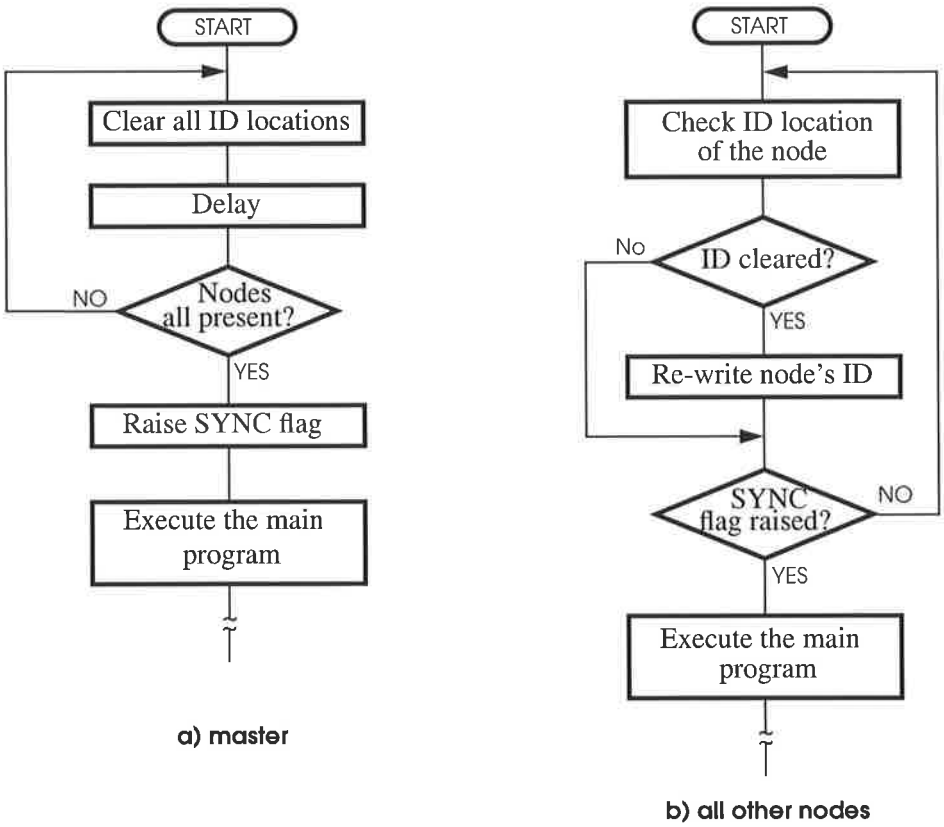

Figure 4.5 Synchronization algorithm

a) The master node checks all the nodes and raises the SYNC flag if all are ready. **b)** Other nodes refresh their IDs in the shared memory and check the SYNC flag continuously.

- The master clears all the IDs and enters a delay routine to give the nodes time to register their presence. After the delay, it checks the IDs. If all the nodes are ready, the master activates the sync flag; otherwise, it clears the IDs and starts over again.
- When activated, each node writes its ID in the shared memory and monitors the sync flag in the synchronization loop. If the sync flag is not raised, the node checks its ID. If it has been cleared by the master, the node writes it again.
- Once all the nodes have registered their IDs, the master raises the sync flag and starts its main program. Other nodes that continuously monitor the sync flag also start their main program.

The presence check of the nodes in this algorithm was performed dynamically as a static presence check was not enough in practice. In some cases, before starting all the nodes, a node that was already started was stopped for different reasons such as performing a check or re-programming. In these conditions, which might happen several times during a test, the static presence check could generate a false sync; however, the dynamic check provided a correct sync. The order in which the master and the nodes were activated, or the number of times they had been stopped and started again was not important in the dynamic check.

2.2 Test program

In order to test the structure under heavy communication traffic, a simple program based on all-to-all communication was tested on MultiCom. The all-to-all test program creates contention on memory links and makes it possible to test the system performance under worst-case traffic conditions. Because of the 16-bit node processors in MultiCom, the word length is defined as 16 bits throughout this chapter and most of other chapters. As shown in Figure 4.6, each node should send a message of 1792 words (700 hexadecimal) to the other three nodes through the shared memory. If a message is large compared to the shared-memory buffer size, it is sent in several packets.

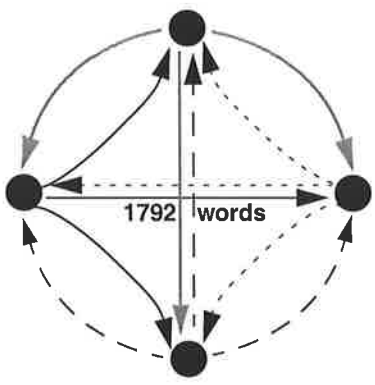

Figure 4.6 All-to-all test program for MultiCom

Each node sends 1792 words to the other nodes. This generates heavy traffic on the shared memory.

At the start up, each node fills its transmit buffers with the data to be sent to the other nodes and then enters the synchronization loop. After a successful synchronization, each node starts its timer and executes the main loop.

In the main loop, each node checks the transmit buffers. If there is data to be sent, it checks the status of the receiving node. If the node is ready to receive, the transmitter allocates a buffer for transmission and writes the size of the packet in the buffer. Then it transfers the data in the packet to the buffer and sends an interrupt signal to the receiving node.

Upon receiving the interrupt, the following steps are performed in the receiver:

- The transmitter is identified.
- The receiver refers to the appropriate buffer in the shared memory and reads the data count, which is the packet size.
- The data is transferred to the local memory and is appended to the previous data received from the same transmitter if any.
- A flag in shared memory is activated to signal the transmitter that the receiver is ready to receive more data from the same transmitter.

The handshaking between the transmitter and receiver is performed with the aid of an interrupt from the transmitter side, and a flag in the shared memory from the receiver side.

When all transmissions are complete, each node acknowledges the end of transmission by writing a word in the shared memory and waits for the other nodes to finish. After the entire communication is completed, each node stops its timer and writes the timer value in the shared memory. The largest of the timer values written by the nodes is chosen as the time spent on the overall communication.

As is apparent from the algorithm, there are several overheads in the communication among the nodes, and each node spends some time in the management of the transmission and reception of the data including checking the transmit buffers, checking the status of the recipient nodes, allocation of a buffer for transmission, and interrupt

overhead. In programming the system, every attempt was made to reduce the overhead to a minimum.

3 Buffer allocation

Each node should be able to read from or write to the multiport memory independently. Because of the concurrent activities of the nodes on the shared memory, they can easily interfere with each other. A reliable communication protocol must consider the restrictions in using multiport memory, and eliminate the conflicts among the nodes. This can be achieved by the use of a proper memory management, and is the subject of the next chapter. In order to complete the discussion of MultiCom, a primary method called static allocation is briefly presented here. More detailed explanation and advanced allocation methods are discussed in the next chapter.

In static allocation, the shared memory is divided among all the possible transmitters and receivers, and a dedicated buffer is pre-allocated for each transmission. There are four nodes in MultiCom and each node can send to three other nodes. Hence, 12 buffers in the shared memory are required. With the available memory of 4Kx16 bits, the maximum size of each buffer is 336 words. The leftover words are reserved for administrative purposes such as the ready signal from the receiver to the transmitter. The 1792-word message in the test program is sent in (1792/336=) 6 packets.

Static allocation removes the possibility of write-write and write-read conflicts, because each active buffer is either written by a node, or read by another node. There is no situation where more than one node can attempt to write into the same memory location, or one node writes while the other node reads the same location simultaneously.

4 Results and discussion

Static allocation was tested successfully on the system. Apart from other outcomes, it confirmed that interprocessor communication with multiport memories was feasible

and could offer a low-overhead communication. As shown in Figure 5.14 on page 102, the best communication rate obtained was 45.5 MBytes/s, which showed 15% overhead compared to the peak communication rate of 53.4 MBytes/s. The peak communication rate is the bandwidth for the overall system, in which all the nodes communicate through the memory with the maximum possible rate without any overhead (Refer to "Details of data transfer" on page 100). The rate of 45.5 MBytes/s was obtained for the buffer size of 336 words, which was the maximum buffer size for the shared memory of 4K words. Reducing the buffer size would increase the overhead and drop the performance.

The results of static allocation will be discussed in more detail in the next chapter.

4.1 Comparison of results

In this section, the results of MultiCom are compared to a system that uses serial links for communication of its nodes. Other detailed comparisons will be performed in the next chapter.

To get a basis for comparison, it is assumed that the nodes of MultiCom are interconnected using serial links in a 2-cube structure as shown in Figure 4.7. In MultiCom, the TMS320C50 processors operate with a 20-MHz clock. The two on-chip full-duplex synchronous serial ports operate at the maximum rate of 5 Mbps. Synchronous links require the clock signal to be sent along the serial bits. In addition, another signal for frame synchronization is also needed. This requires more connections between the processors. On the other hand, the clock signal is not required for asynchronous links and synchronization is achieved by sandwiching the serial data between a start bit and one or more stop bits. Commonly, a parity bit is also sent for error checking. In this method, the number of connections between processors is minimum, but throughput is reduced because of the extra time required to send start and stop bits. Using the on-chip serial ports for this hypothetical system will reduce its throughput significantly because of the low transfer rate of 5 Mbps. In order to achieve a more realistic result, it is assumed that the system is using two external asynchronous serial ports operating at 20 Mbps, which is comparable to the processor speed.

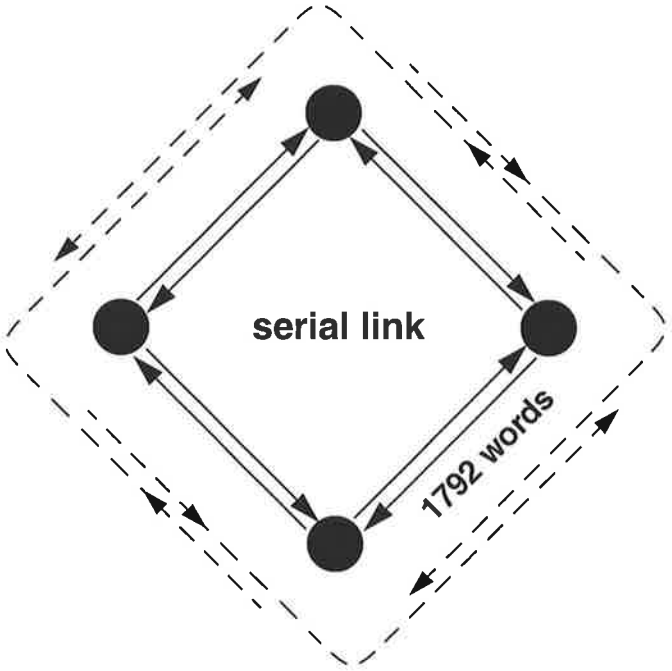

Figure 4.7 Serial communication of four nodes in 2-cube structure

Assuming no overhead, in the first time slot each node communicates with two neighbouring nodes, as shown with solid arrows. In the second time slot, each node communicates with non-neighbouring nodes, as shown with dotted arrows. A total of 12x1792 words should be transferred.

Overall, the minimum time for all-to-all communication is double that of the time spent in sending from one node to another.

It is worth mentioning that a hypercube of order 2 with serial links is not a very suitable structure for four nodes, and in practice, other structures can be used for this small system. This hypothetical system was used only to get a basis for comparison.

Similar to the all-to-all test program of Figure 4.6, each node should send 1792 words to all other nodes. As each of four nodes sends a message to other three nodes, this adds up to 4x3x1792 words in total . In order to obtain the maximum performance for this system, the following assumptions are made:

- Links are full duplex and messages are sent or received on separate lines.
- The intermediate nodes do not store the data. They pass it directly to the next node.

- An optimum communication protocol is used so that no node is idle at any time, and no message is waiting because of a line being used for another transmission, or a free line cannot be used because it has been blocked by other communications.
- There is no overhead in sending or receiving and only the time spent on the actual transmission is considered.
- The communication is asynchronous. For each byte, a total of 11 bits comprising of 8 data bits, a start bit, a stop bit, and a parity bit is sent.

It is very unlikely that all of the above conditions are met for a realistic system. Hence, the performance of a practical system would be lower to some extent.

The following steps can be taken to deliver all the messages in this system:

1. In the first time slot, each node communicates with the two neighbouring nodes to send and receive the required data at the same time. All the communication to the neighbouring nodes can take place in this slot as shown by the solid arrows in the figure.
2. In the second time slot, each node communicates with a non-neighbouring node through an intermediate node. The data is sent to the neighbouring node and is immediately passed on to the destination. As the dotted arrows in the figure show, all the remaining communication can be performed in this time slot.

Using these steps, the overall communication would only take two time slots. Considering the speed of serial ports (20 Mbps, equivalent of 0.05 μs/bit), sending 11 bits for each byte will take 0.55 μs. Hence, sending 1792x2 bytes in each time slot would take 1971.2 μs (i.e. 1792x2x0.55 μs). The size of the total message transferred between the nodes is 12x1792x2 bytes and the effective communication rate can be calculated as 10.9 MBytes/s (i.e. 12x1792x2 bytes/2x1971.2 μs).

Comparing this result to the 45.5 MBytes/s of static allocation shows that the performance has increased more than 4.2 times. Note that as explained before, in practice the increase in performance could be even higher, as no overhead has been considered in serial communication.

It is worth mentioning that the number of links used for each node in MultiCom is 16 bits (size of the data bus). On the other hand, the number of links connected to each node for asynchronous serial communication is 4 (two full-duplex serial links).

MultiCom can be also compared to a bus-based system and a system using dual-port memories. This will be performed in the next chapter where timing of MultiCom is discussed in detail.

5 Conclusion

This chapter has discussed how MultiCom, a hardware prototype, was designed and implemented as the first step to evaluate the performance of the proposed structure. The nodes of MultiCom were selected from off-the-shelf DSP processors and they were interconnected using 4-port memories. Conflicts between the nodes on the shared memory were removed by using static allocation as an initial allocation method. MultiCom proved to have very efficient performance in communication achieving 45.5 MBytes/s. This rate was very close to the peak communication rate and the overhead was less than 15%. Compared to a system using 20 Mbps asynchronous serial links with no overhead, it showed at least a 4.2-fold speed improvement. The improvement could be even higher if realistic overheads encountered in practical systems were also considered. Overall, the design of MultiCom proved that interconnecting systems with multiport memories is feasible and it can provide high performance. Other benefits such as reducing the size, cost, and internal wiring will be discussed later. This structure will be elaborated upon in the subsequent chapters.

CHAPTER 5

Memory Management and Communication Protocol

As the communication of nodes in MultiCom is performed through multiport memory, it is essential to manage the shared memory in a way that the nodes do not interfere with each other. For MultiCom, a range of memory management strategies are possible. In static allocation, as introduced in Chapter 4, each transmitter uses a pre-allocated buffer to send a message to a receiver. An alternative is dynamic allocation, in which a free buffer can be allocated to any communication on demand. Better memory utilization is expected for dynamic allocation; however, a lock mechanism is required to eliminate the shared memory conflicts. In this chapter, first a detailed discussion of static allocation is presented. Then dynamic allocation is explained, and two newly devised software locks are introduced. Multicasting/broadcasting and communication protocols are discussed later. Finally, the results obtained from MultiCom are presented and compared to other systems interconnected with serial links, dual-port memories, or bus-based systems.

1 Memory management

Since shared memory is used as a communication medium, the nodes need to perform several activities on the memory as part of transmitting or receiving data. In order to preserve data integrity and to achieve reliable communication, the nodes must not interfere with each other. Hence, proper memory management and a suitable communication protocol must be implemented.

Allocation of buffers in the shared memory to different data transmissions is an important issue in the memory management. A proper allocation method must prevent a node from using a buffer that is already in use by another node unless both reading. Allocation of buffers can be performed in two ways: in static allocation, the nodes use a pre-allocated buffer for transmission to each node; in dynamic allocation, the buffers can be allocated to any transmission on request. Each method is discussed separately, and its advantages and disadvantages are highlighted.

1.1 Static allocation

In static allocation, all of the possible combinations of transmitters and receivers are determined and a buffer is assigned to each combination. For N nodes connected to a multiport memory, each node can transmit to N-1 nodes and N(N-1) buffers are required in total. Hence, MultiCom would require 12 buffers, and for the shared memory of 4K words, the maximum buffer size would be 336 words. Figure 5.1 shows the layout of memory under static allocation in which a dedicated buffer is available for each transmission. The outcomes of static allocation are presented in the result section.

The benefit of static allocation is that a node simply refers to an address table to find the location of the buffer pre-assigned to the desired transmission. This procedure requires very small overhead. Moreover, the allocation method ensures that other nodes will not interfere while the node is writing into the buffer. This is because each buffer is assigned to a unique transmission only, and the receiver will read the buffer after the message is completely written. Hence, both of the restrictions for using 4-port memories as mentioned on page 58 have been addressed in this method.

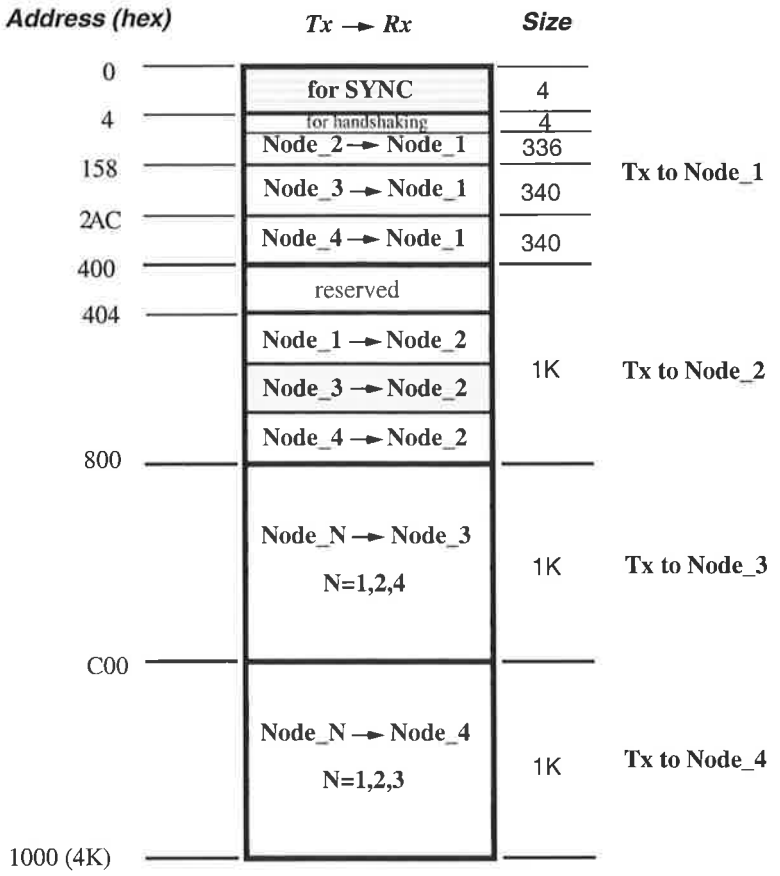

Figure 5.1 Memory map of MultiCom for static allocation

The 4 Kword-memory is divided into 12 buffers of 336 words and each buffer is pre-allocated to a specific transmission. Memory is not utilized efficiently and part of it is idle at a time, but there is no conflict on the memory access.

On the other hand, as only some of the nodes are actively communicating at a time, only a fraction of the shared memory is concurrently used for communication. In other words, after the data in a buffer is delivered to the receiver, the buffer remains idle until used again by the same transmitter and receiver. Hence, the valuable shared memory is not utilized efficiently in static allocation and part of it remains idle at a time. This is one of the disadvantages of static allocation. Another disadvantage is that it does not scale properly when the number of nodes increases. This point will be discussed in the next chapter.

A more sophisticated allocation method can use the memory more efficiently. Instead of creating a buffer for every possible communication, fewer non-allocated buffers can be generated and the allocation process can assign each of them to a transmission when requested. Hence, the buffers are assigned only to the active communications and the number of required buffers is reduced. Consequently, larger buffers can be created resulting in reduced overhead and increased performance. This method is called dynamic allocation and is the subject of the next section.

1.2 Dynamic allocation

In dynamic allocation, the shared memory is divided into a number of buffers that are not dedicated to any specific communication link. The list of the buffers is stored in a table called the "buffer allocation table". Each buffer has an entry in the table that defines whether the buffer is free or in use. More information such as the size and the start address of the buffers can be also stored in this table. Any node requiring a buffer for transmission refers to this table and if a free buffer is available, the node allocates it to its transmission. Figure 5.2 illustrates the memory layout under dynamic allocation.

As there is no central controller on the shared memory to perform the buffer allocation, each node is responsible for the required allocations. If two or more nodes perform the allocation process simultaneously, there is a potential for conflict among the nodes, which can cause data loss. For example, if two nodes simultaneously allocate the same buffer to their transmission without knowing the activity of the other node, some data will be overwritten and data loss is inevitable. Hence, there must be a mechanism to control the allocation process so that it can be performed exclusively. In order to achieve this goal, a lock mechanism implemented in hardware or software must be used and each node must possess the lock exclusively before allocating a buffer. This prevents the others from doing a similar task until the lock is released.

It is worth mentioning that the lock is only used in critical activities such as buffer allocation, where there is a possibility for conflicts among the nodes. Other tasks such as writing into or reading from a pre-allocated buffer do not require the possession of the lock. Therefore, nodes can work independently on unlocked areas of the shared memory, in parallel with the other memory activities.

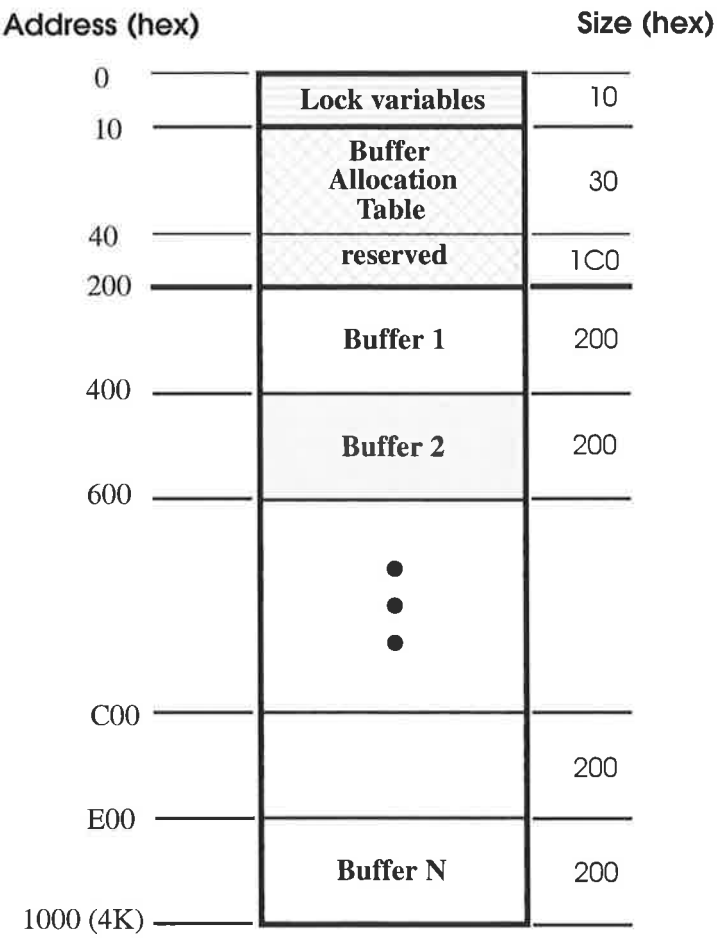

Figure 5.2 A typical memory layout for dynamic allocation

Memory is divided into several buffers and each can be allocated to any transmission on demand. Memory can be utilized more efficiently, but a lock mechanism is required for buffer allocation.

The advantage of dynamic allocation over static allocation is the use of fewer but larger buffers. This is achieved by combining the active buffers with the idle ones. In general, larger buffers can reduce the communication overhead and increase the performance, provided that sufficient buffers are available. The drawback of dynamic allocation is the use of a lock that introduces a serial mechanism into the system and requires extra overhead.

1.3 Multicast / broadcast

One benefit of using dynamic allocation is that if the same message is to be sent to two or more nodes, multicasting or broadcasting can be used instead of sending the message

to each node individually. In this method, once the message is copied into a buffer, it can be sent to more than one node without rewriting. Each involved node can receive the message from the same buffer simultaneously and in parallel with the other nodes. The buffer must be released by the last node.

In multicasting, multiple attempts for getting the lock, allocating a buffer, and transferring data to the buffer are reduced to one attempt only. Hence, communication overhead drops and system performance rises. This method will be discussed in more detail later.

2 Semaphore signalling

In assigning a buffer using dynamic allocation, there is a possibility that nodes could interfere with one another. The use of a lock mechanism or semaphore signalling can remove the conflict and produce mutually exclusive access to the shared memory. Each node is required to possess the semaphore before performing a sensitive task on the memory such as buffer allocation. This will preserve data integrity.

Semaphore signalling can be implemented in either hardware or software. With a hardware semaphore, if a node possesses the lock, the other nodes attempting to get it will receive a denial until the lock is released. With software semaphores, after requesting the lock, the nodes are required to check regularly a dedicated memory location for their turn to use the memory. Different algorithms can be used to create a software semaphore. Both methods are discussed in subsequent sections.

2.1 Hardware semaphore

Hardware semaphores have been implemented in some dual-port memories for a long time. In general, semaphore latches are independent from the memory locations on the chip. The control of semaphore requests can be handled using a standard write followed by a read instruction. There is no requirement to lockout the other processor to access the semaphore between the write and read. In some products, as many as eight semaphore latches have been implemented in one memory chip [Cypress 96].

Hardware semaphores can be used to implement a token controlled scheme allowing the port in possession of the token to have exclusive access to a block of shared memory. The port releases the token after finishing its critical section. A request for possessing the token from another port will be denied if the token is held by the other port, but the request will be registered to take affect after the token is released.

Possession of the token is indicated by the state of a semaphore latch formed by two cross-coupled NAND gates as shown in Figure 5.3. Only one port can set the semaphore latch at a time. An extra input latch on each port is used to hold the request for setting or clearing the semaphore latch. Output latches are also used to prevent the output data from changing during a read operation.

The semaphore latch is accessed through the address and data buses similar to accessing a memory location. The semaphore enable line should be activated in this cycle instead of the memory chip select. The latch is accessed using the lower data bus line, and if more than one semaphore latch is implemented, the lower address bus lines can select one of them.

The semaphore latch in Figure 5.3 is active high. A node can request the control of the semaphore by writing "1" into the port. This request is stored in the port input latch and held until the same node clears it by writing "0" into the input latch. If the semaphore latch is free, the node will immediately gain the control of the latch. On the other hand, if the semaphore is controlled by the other port, the request will be denied. If the request remains pending in the input latch, the requesting node will be granted the control only after the other port has released the semaphore by writing "0" into its input latch.

The read of the semaphore will indicate if a request for controlling it was successful. A readout of "1" indicates that the port controls the semaphore and "0" indicates a denial. As the state of the semaphore latch could change during a read operation, an output latch is used to prevent propagation of the change to the output line. In this case, the next read will show the updated state of the semaphore. The node receiving the denial should either repeatedly check the status of the semaphore until control is granted, or write "0" to clear the request and try another time [Baumann 96].

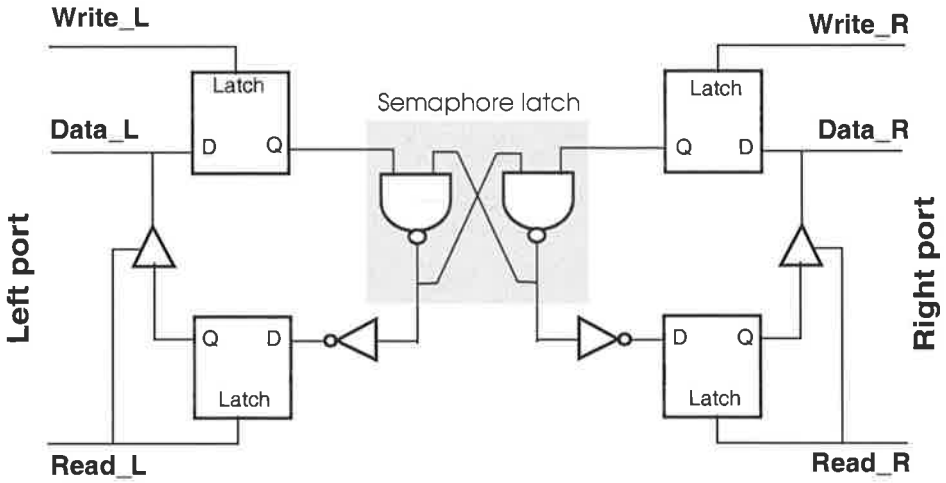

Figure 5.3 Semaphore latch cell

Only one port can possess the semaphore at a time. Each port can request the semaphore by writing "1" into the input latch. Reading "1" on the output indicates a success and "0" a denial.

If the semaphore is free and both ports attempt to request it at the same time, semaphore arbitration logic guarantees that only one side gains the control [Cypress 96].

As IDT7054, the 4-port memory used in MultiCom, was a very new product at the time of system design, no semaphore logic was implemented on this chip. The structure of the semaphore as explained above applies only to two ports and needs a major change for implementation on more ports. The semaphore latch should be modified to have more inputs, and other parameters such as the order in which the nodes apply for the semaphore should be taken into account. In addition, a priority scheme should be set up for pending requests or for requests applied at the same time. Consequently, the overall logic can become complicated. On the other hand, the absence of semaphore logic in the IDT7054 could have commercial reasons such as releasing the chip to market as fast as possible. A new design for a multiport semaphore is explained in the Appendix.

The conclusion is that hardware semaphores are currently unavailable on multiport memories. There is no semaphore latch on multiport memories with more that two ports and the control of these memories is entirely left to the system designer. The only remaining solution is the use of software semaphores as discussed in the next section.

2.2 Software semaphore

A hardware semaphore is a convenient way to control a shared memory. In its absence, software semaphores should be used. Software semaphores can be implemented using dedicated memory locations to hold the semaphores and are extensively used as a mechanism to provide synchronization and concurrency for different processes [Ben-Ari 82] [Stallings 98]. In a general semaphore, the number of waiting processes is stored in a variable and the ID of each requesting process is stored in a queue. A binary semaphore can only have values of 0 or 1. In a system using a simple binary semaphore stored in a reserved memory location, each node attempts to gain control of the semaphore by using an indivisible test and set instruction. The test instruction checks if the semaphore is set by other nodes. If the semaphore is free (cleared), the node sets the semaphore and gains exclusive control of a block of memory associated with that semaphore. If the semaphore is not free, the set instruction is aborted and the node should reapply again later.

Software approaches to mutual exclusion and semaphores can be implemented for concurrent processes that execute on a single processor or a multiprocessor machine with shared memory. These approaches rely on some elementary mutual exclusion mechanism at the memory access level [Lamport 91]. That is, simultaneous accesses to the same memory location in the shared memory are serialized by some sort of memory arbiter, although the order of access granting is not specified ahead of time. With this mechanism in place, accessing a memory location excludes any other access to the same location simultaneously [Stallings 98].

Checking a semaphore and changing its value must be an indivisible atomic action [Tanenbaum 01]. The system must guarantee that once a semaphore operation has started, no other process can access the semaphore until the operation has completed. According to [Tanenbaum 01], "this atomicity is absolutely essential to solving synchronization problems and avoiding race conditions."

The software semaphores discussed in [Ben-Ari 82], [Stallings 98], and many other operating systems textbooks cannot be directly implemented for multiport memories. The reason is that on a system using multiport memories, nodes can freely access the

shared memory from different ports concurrently. Unless there is a hardware memory arbitration circuit, or a software mechanism to enforce mutual exclusion, the algorithms cannot be successful. Multiport memory has not been discussed in these references and new methods need to be found to address this issue.

Hardware memory arbitration has been implemented in some dual-port memories to achieve mutual exclusion. [Wyland 88] discusses an address arbitration circuit for dual-port memories. It consists of common address detection logic and a cross-coupled arbitration latch. If the same memory location is accessed from both sides simultaneously, this logic provides a busy signal to the address that arrived last and inhibits write operation for the port receiving the busy signal. It also makes a decision in favour of one port or the other when both addresses arrive at the same time. A busy line is available in most of dual-port memory products [IDT] and is explained in more detail in the Appendix. It can be used to extend the memory cycle for the operation performed by the losing processor until the winning processor finishes its access.

As stated earlier, the IDT7054 has no extra circuit to resolve simultaneous access to a common memory location. The IDT7052, the 2K version of this memory, has a BUSY line for each port. This input line has a very limited functionality and is different from the busy line explained in the previous paragraph. If activated by external logic, it will block the write to the addressed location from the pertinent port [IDT 95]. The system designer should devise the required external logic to detect the address match from different ports and resolve the conflict in favour of one of them by providing busy signals to the others. A simplified version of this external circuit may search for only one specific location such as address zero, where the semaphore flag could be kept. If a BUSY line were available in IDT7054, it could facilitate the implementation of software semaphores; however, the extra address line required to expand the chip from 2K to 4K has replaced the BUSY line in the pinout. Hence, system designers should rely on software methods to implement a lock for IDT7054.

The only method that has been implemented to control 4-port memories with no hardware arbitration, is TOKEN passing [Mick 96]. Token passing and other methods attempted in this study are discussed in subsequent sections.

2.2.1 TOKEN passing

In this method, every node has a unique ID. The owner or the master is the node whose ID matches the TOKEN that resides in a dedicated memory location. The master can use the locked part of the memory exclusively for performing critical tasks such as buffer allocation. When finished, the master passes the token to another node in a prescribed order by writing the ID of the node into the token.

At the start, one of the nodes is the master by default. To determine who is the master, each node should read the token regularly and compare it to its ID. The node successful in finding a match can go ahead to use the memory as the master.

After finishing with the lock, the master passes the token to another node in a circular manner. While the owner is writing the new master's ID in the token, if other nodes attempt to read the token, there is a possibility of data corruption for the read operation. Hence, the new master must verify its success by multiple read / compares [Mick 96].

Token passing is very easy to implement, and it is very effective as long as all the nodes are active in the token passing operation. On the other hand, a node that no longer needs the token, or is performing tasks not requiring the token, would still be included in the token passing cycle. This demands that all of the nodes, regardless of their interest in becoming a master, should regularly check the token to use it, or at least to pass it to the others. This requirement is an extra burden for the nodes and failure to do it on time may result in long delays for the other nodes waiting for the token.

Moreover, as explained in Section 4.2 on page 101, in the node processors used in MultiCom, transferring packets of data to or from allocated buffers were performed by specific instructions in repeat mode. Once started, the repeat mode generates a firmware loop and transfers the entire packet before stopping. This increases the transfer rate considerably, but it is not possible to perform other tasks such as token checking while the transfer is in place.

For the reasons discussed above, in spite of its simplicity, token passing was not suitable for MultiCom and other methods were devised for implementing a software lock. These methods are explained subsequently.

2.2.2 Waiting list

With a waiting list, similar to the token passing, each node accesses the memory using a separate port and the master is the node with a matching ID to the number in OWNER. Each node requesting to become the master registers its ID at the end of a waiting list in the shared memory. The registered node should regularly monitor the list to upgrade its position in the queue. If the upper request is cleared because of a move in the queue, the node should upgrade its turn by moving its ID to the cleared location. This process is repeated until the node reaches the highest position and writes its ID in OWNER indicating that it is the master. Figure 5.4 demonstrates this algorithm.

The move in the queue is initiated when the master clears its ID from OWNER. If there is a node waiting at top of the queue, the node writes its ID in OWNER and becomes the new master. It also clears its ID from top of the queue. This in turn enables any requesting node to advance in the queue by moving its request to a higher position.

This method is acceptable in principle; however, its implementation on MultiCom showed that there were several potentials for write-write, and write-read conflicts as discussed below:

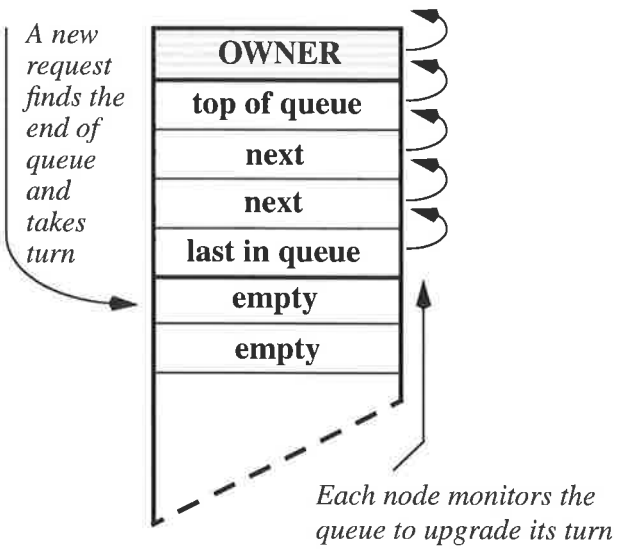

Figure 5.4 Waiting list

Each node registers at the end of the queue and upgrades its position if the queue is moved up. The node with its ID in OWNER is the master and clears its ID when finished.

- More than one node may attempt to register at the end of the queue at the same time. The conflict can be compensated for by verifying the write operation and starting again if not successful.
- In the upgrading process, each node moves to a higher position and clears the lower one. Before all the nodes finish the upgrading, a new request may be inserted in a freed location. To avoid this, two different codes should be used to clear the requests, one code for end of the queue where there is no other request, and another code for between the other requests. Even with this approach, a lucky node may find a free location for registering its ID while there is another request underneath. This case may happen if registering of a request for a node at the end of the queue coincides with the release of the last request by another node. Normally this situation causes no problem because the new request is upgraded within a short time. However, if a third node attempts to register in the queue before this request is upgraded, it may register in a place that is not the end of the queue. The results of several tests using this method showed that the probability of this situation was very low; however, it was observed in long run under heavy demand for the lock.
- In the upgrading process, several write-read conflicts are possible. The sensitive ones are overcome by repeated reads and compares. There are also a few possibilities for write-write conflicts as discussed above.
- The algorithm is very sensitive to the processor and memory timing. If either one is changed, the algorithm should be adjusted for the new condition.
- All the registered nodes should monitor the queue continuously for an upgrade in their position. Delay in upgrading could hold the other nodes registered in the lower parts of the queue.

For the reasons discussed above, this algorithm was not successful. Applying a few modifications may result in a successful algorithm. For example, if the upgrade process is performed entirely by the master or even by the node at the top of the queue, most of the memory conflicts can be removed. The latter is a better alternative, because no provision is required when no master is available. The algorithm may work as follows:

- The nodes register at the end of the queue as discussed previously.
- After the master releases the lock, the node on top of the queue writes its ID in OWNER and upgrades the positions of the other nodes if any.

- Other nodes only monitor the top of the queue. The node the ID of which reaches this point will be responsible for the upgrade.

The algorithm would be successful even if the queue is empty and there is no master. In this case, any node registering at top of the queue would become the master. There are still some remaining conflicts in this algorithm, but they can be overcome by repeated verifying.

The author came across this idea only at the time of writing the thesis. Therefore, the modified algorithm was not tested on MultiCom.

2.2.3 Lock with BUSY

This new lock scheme was designed in an attempt to exclude the uninvolved nodes from the token passing scheme. In this method, only the nodes registered for the lock are considered in the TOKEN passing cycle. As shown in Figure 5.5, each requesting node can register in a dedicated location if the BUSY flag is cleared. Otherwise, the node should keep checking the flag until it is cleared.

The master is the node the ID of which matches the number in OWNER. After finishing the allocation process, the master determines the next owner by checking all the registered nodes in a circular priority in which the current owner is in the lowest position. Before determining the new owner, the master raises the BUSY flag to prevent the other requests from registering. This is essential because it prohibits write-read conflicts in a sensitive situation. A short delay is also inserted to settle any request that might be in progress.

In the process of determining the new master, if there is no request, this simple algorithm may fail. This is because if a new request is made later, none of the nodes will be responsible for determining the master. Therefore, a flag called *No_owner* is added to the algorithm. When this flag is set, the requesting node takes the responsibility of determining the owner. Sometimes there might be more than one node registered. They all compete to be the master, and the node having the highest circular priority is the winner. The algorithm is shown in the flow chart of Figure 5.6 and is explained in more detail in Table 5.1.

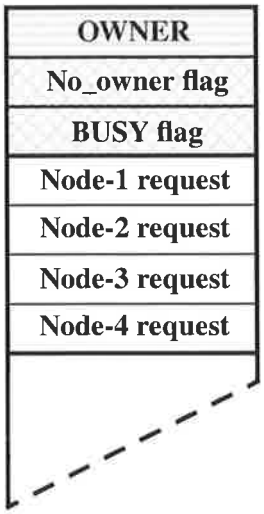

Figure 5.5 Lock with BUSY

The nodes register in a dedicated location if BUSY is not set. The current owner determines the next owner using circular priority. If there is no owner, the requesting node is responsible for determining the owner.

This algorithm was successful in sharing the lock among the nodes of MultiCom. There is no starvation for any of the nodes, because after clearing the busy flag, there is plenty of time for all the waiting nodes to register their request before busy goes high again. Moreover, circular priority ensures that all the registered nodes will receive the lock. Although the overhead of the algorithm is slightly high, the benefit is that a registered node is free to perform other tasks while its request is processed by other nodes. The only remaining write-write conflict is a possible setting of the BUSY flag by more than one node simultaneously. This situation may happen if there is no current owner, and two or more nodes register at the same time. All of the nodes activate the BUSY flag to block the new requests, and start to determine the new master. As they all write a "1" in the BUSY flag, the conflict has no drawback. The shaded box in Figure 5.6 shows the location of this possible conflict. In this case, the node with the highest circular priority will be the master.

For a reduced overhead, the algorithm can be modified as explained in the next section.

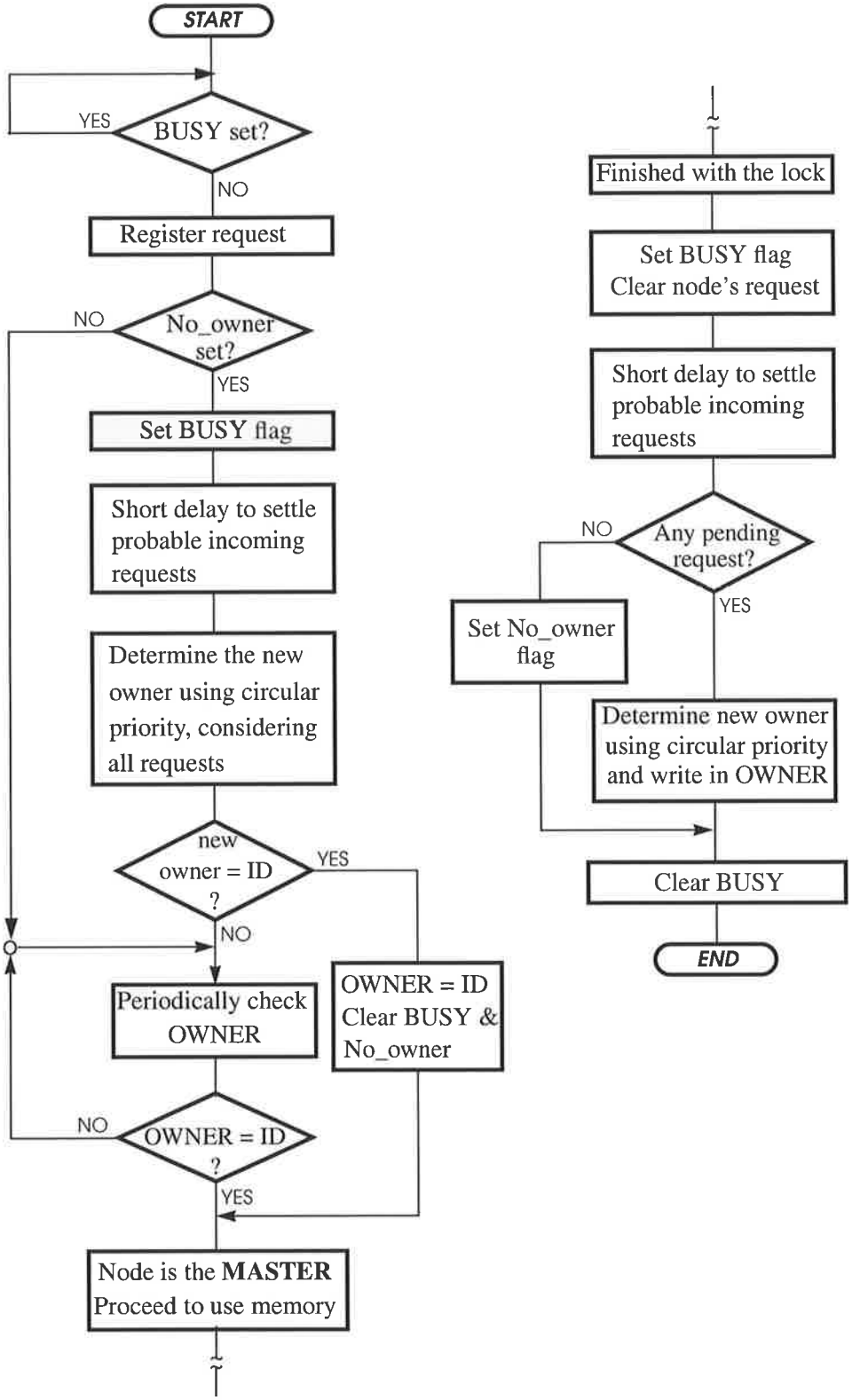

Figure 5.6 Flow chart of the lock with BUSY

If there is no owner, the requesting nodes should determine which one is the master. Otherwise, the current master determines the next owner after finishing with the lock.

Table 5.1 Algorithm for lock with BUSY

- Each node registers its request if the BUSY flag is cleared. Otherwise, it waits until the flag is cleared.
- If there is a current owner, the requesting node only should check the OWNER regularly. When the ID of the node matches the OWNER, the node is the master.
- If there is no owner, the requesting node sets the BUSY flag to block new requests from registering. As more than one node may register simultaneously in this situation, a short delay is inserted to make sure that any other possible requests in action are registered. Considering all the requests, each registered node determines the owner in a circular priority where the previous owner has the lowest rank. The node the ID of which matches the determined owner is the new master. It writes its ID in the OWNER, clears the No_owner and BUSY flags, and can use the locked part of the shared memory. The losing nodes, if any, perform regular checking of the OWNER to determine their turn to obtain the lock.
- When the owner finishes with the lock, it sets the BUSY flag to disable incoming requests, clears its own request, and determines the next owner using the circular priority. It writes the ID of the new owner, if any, into the OWNER; otherwise, it sets the No_owner flag. It clears the BUSY flag in the end.

2.2.4 Fast lock

This lock has a structure similar to the previous one. The main difference is that each node is responsible for acquiring the lock rather than receiving it through the current master. It was devised in an attempt to eliminate the No_owner flag in the previous algorithm. Figure 5.7 shows the memory structure used for the fast lock.

If BUSY flag is not set, the requesting node sets it to block the other requests and registers in a dedicated location in memory. In some conditions, more than one node may register. This is because as the BUSY flag changes from set to clear, all the waiting nodes register almost simultaneously before the flag is raised again. Each registered

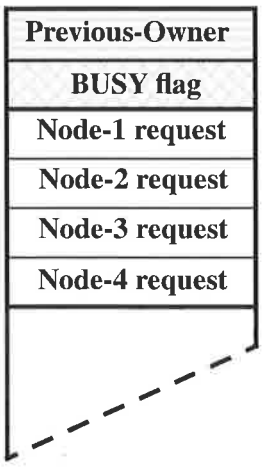

Figure 5.7 The fast lock

All registered nodes determine the owner and the winning one can proceed as the master. Others keep checking their turn to become the master using circular priority.

node determines the next owner using circular priority where the previous owner has the lowest priority. The winning node in finding a match with its ID is the new master. After finishing with the lock, the master writes its ID in the Previous-Owner and clears its request. Consequently, another node having a pending request can win to become the new master. If there are no more requests, the master clears the BUSY flag to enable waiting nodes to register new requests. Figure 5.8 shows the flow chart of this lock and Table 5.2 illustrates its detail.

Fast lock is simple and very efficient. The associated overhead is also low. The drawback is that the registered nodes should continuously check for their turn to become the master. Moreover, registering a new request for the lock may take longer than the previous method, because the BUSY flag remains active until all the registered nodes have finished with the lock. Similar to the previous lock, there is no starvation in this lock, as nodes have time to register, and after registering, they will certainly receive the lock.

Dynamic allocation was implemented on MultiCom using both the "lock with BUSY" and the "fast lock" as software semaphores. Both algorithms proved to be efficient in

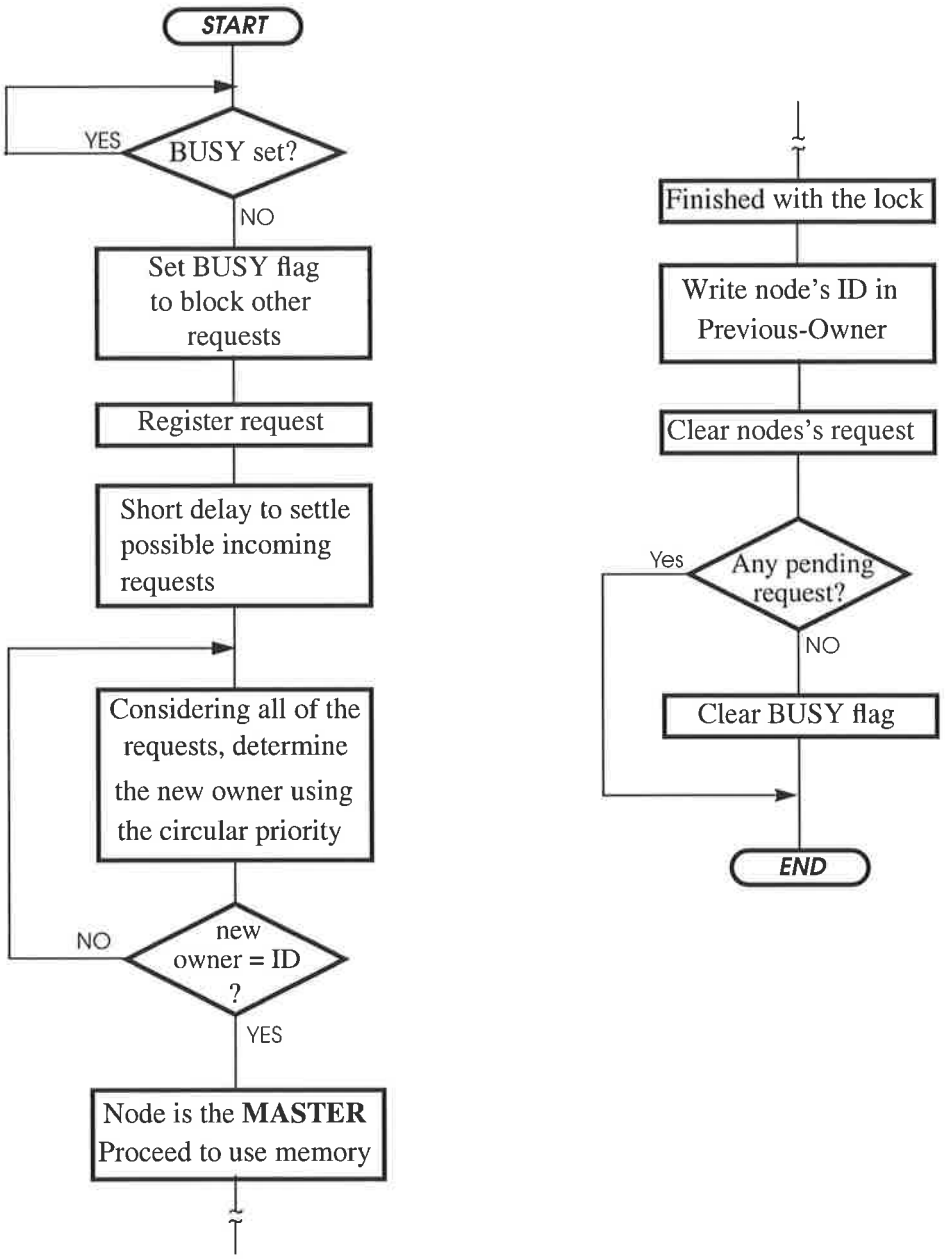

Figure 5.8 Flow chart of the fast lock

After registering, each node determines the next owner using circular priority and the winning node is the master. After finishing with the lock, the master clears its request and writes its ID in the Previous-Owner. Other nodes, if any, determine the new owner.

Table 5.2 Algorithm for fast lock

- If not BUSY, each requesting node raises BUSY to block the other requests and registers in a dedicated location. Under heavy demand for the lock, usually more than one request can be registered.
- After a short delay to make sure that all requests in progress are recorded, each registered node determines the next owner based on the circular priority with the previous owner having the lowest rank. The winning node can proceed as the master, while the losing nodes keep determining the owner repeatedly until it matches the ID of the node.
- After finishing with the lock, the current owner writes its ID in the - Owner and removes its request. If no other request is pending, the owner also clears BUSY to enable registering of new requests.
- By changing the Previous-Owner and removing the relevant request by the current owner, one of the other registered nodes, if any, can win to be the new master.

assigning the lock to the requesting nodes with no conflicts. It is worth mentioning that these algorithms have not appeared anywhere in the technical literature and are new lock schemes. The outcomes of dynamic allocation are presented in the result section.

Buffer allocation is part of a larger algorithm called "communication protocol" that controls the overall communication. It is discussed in the next section.

3 Communication protocol

A proper communication protocol facilitates the communication task in a system. A simple and basic protocol was used to test the static allocation on MultiCom. The basic protocol was gradually upgraded to include the extra features required for complicated algorithms such as dynamic allocation or broadcasting. For evaluation of larger systems using the simulation model in the next chapter, the communication protocol was also modified and upgraded. The basic communication protocol is explained in the next

section, and the required modifications for other algorithms are discussed in the relevant sections.

3.1 Basic communication protocol

The basic communication protocol was tested on MultiCom for static allocation. In the all-to-all test program, as discussed in page 62, depending on whether a node is a transmitter or a receiver (or both), two different activities are performed in the node. They are explained below:

Transmitter:

Each node sets up the transmit buffers with the messages to be sent to other nodes. Each message can be sent in several packets if it is larger than the buffer size. Within a loop, each node checks the transmit buffers, and if there is data to be sent, it checks the appropriate receiver. If the receiver is ready, a packet of data is copied into a pre-allocated buffer in the shared memory and an interrupt is generated for the receiver. This process is repeated several times until all the messages in transmit buffers are sent out to the appropriate receivers. Figure 5.9-a shows the activities of a transmitter in the basic protocol.

Receiver:

In a receiver, interrupts coming from different transmitters are connected to separate interrupt lines. The receiver identifies the transmitter by the received interrupt and refers to the appropriate buffer in the shared memory. Then it transfers the packet of data from this buffer to a local buffer and appends it to the previous packets of the same message received from the sender, if any. When transfer is complete, the receiver raises the receiver ready flag to acknowledge that the data is received and the receiver is ready again to get more data from the same transmitter. After updating the necessary variables, it terminates the interrupt service routine. Figure 5.9-b illustrates the receiver activities in the basic protocol.

Note that the shaded parts of Figure 5.9 depend on the protocol in use, whereas the unshaded parts are general. Only the shaded parts will be explained for other protocols.

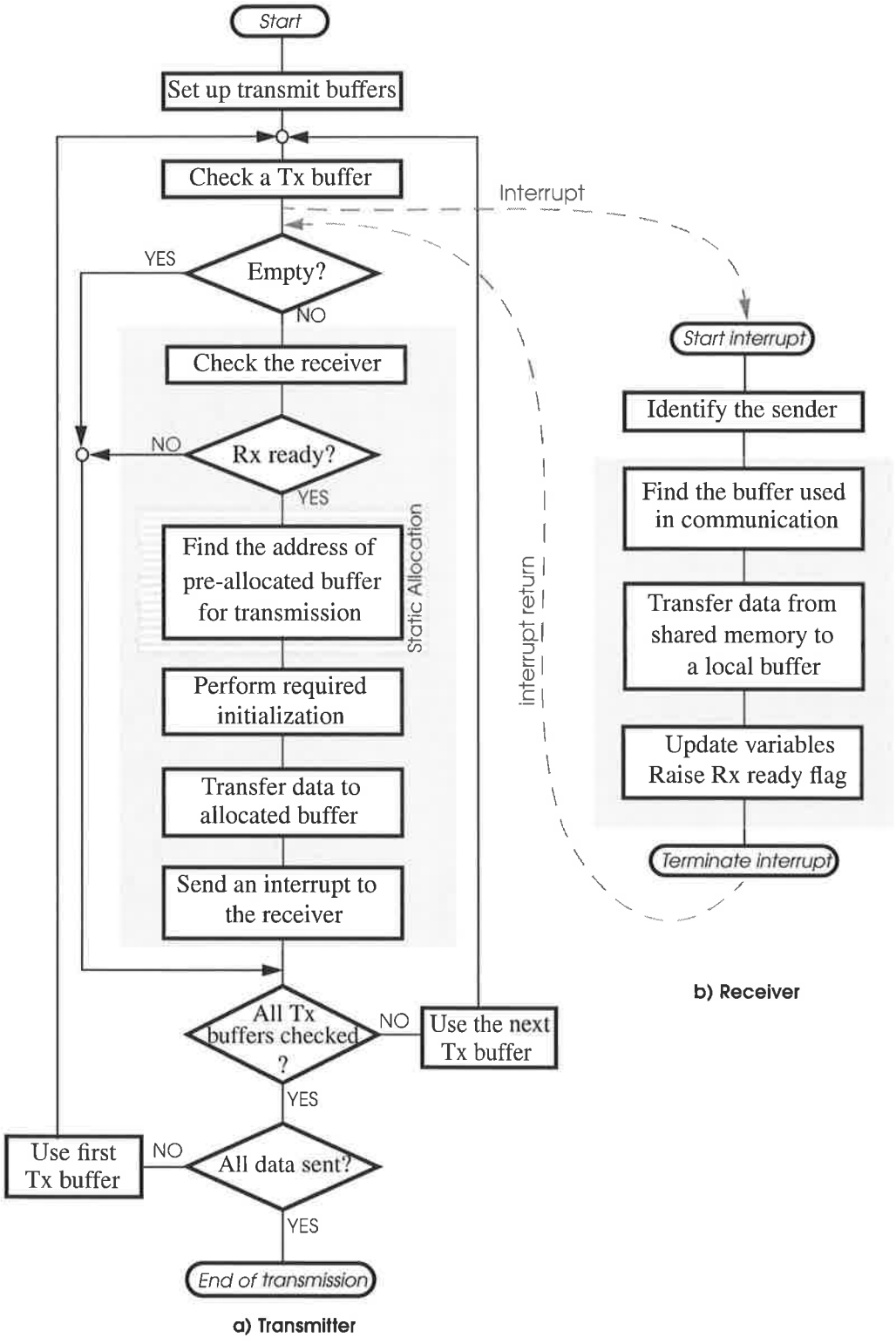

Figure 5.9 Basic communication protocol

The figure shows the tasks performed in each node **a)** as a transmitter, **b)** as a receiver. The receiver is activated by receiving an interrupt from other nodes at any time. The unshaded parts are general for all-to-all communication, but the shaded parts depend on the protocol in use.

Table 5.3 summarizes the communication protocol under static allocation.

Table 5.3 Basic communication protocol

In transmitter:
1. In the transmission loop, each transmit buffer is checked. If there is data to be sent, step 2 is performed; otherwise, the next buffer is checked. After checking all the buffers, the loop is executed again until data in all the buffers is transmitted. 2. The status of the receiving node is checked. If it is not ready, indicating that the previous transmission is still in progress, the transmission loop is resumed. 3. The address of the pre-allocated buffer is determined and a packet of data with proper size is generated. Proper size is the minimum of the buffer size, and the size of remaining data in the transmit buffer. 4. The packet of data is transferred to the buffer in the shared memory. The internal variables are updated. 5. An interrupt is generated for the receiving node. The transmission loop is resumed.
In receiver:
1. Depending on the received interrupt, the identity of the transmitter is known to the receiver. Therefore, upon receiving an interrupt, the appropriate buffer address in the shared memory is determined and the size of the packet is obtained. The local buffer to transfer the received data is also identified. 2. The packet is transferred from the shared memory into the local buffer and is appended to the previous packets of the same message, if any. 3. The receiver ready flag is raised to indicate that the data is received and the receiver is ready again for more data from the same transmitter. Internal variables are updated and the interrupt service routine is terminated.

3.2 Protocol for dynamic allocation

The general activities in the dynamic allocation protocol are similar to the basic protocol, but the memory management is very different. Before describing the protocol, a closer look at dynamic allocation is needed to identify the required components. Figure 5.10 shows the memory layout and the essential elements for dynamic allocation. Detailed explanations are given below.

Lock variables:

As explained in previous sections, OWNER, No_owner, BUSY, and node requests are the variables used for the lock.

Buffer address table:

Buffer address table lists the start address of the buffers in the shared memory. The address of a buffer can be obtained by using the buffer number as an index to this table. The number of available buffers and the buffer size is determined by the size of available memory.

Buffer allocation table:

Each buffer has a 16-bit entry in the buffer allocation table. Bit-15 shows the status of the associated buffer. "0" indicates a free buffer and "1" an allocated buffer. The remaining bits are reserved for multicasting/broadcasting and will be explained later.

Receiver mailbox:

Each of the possible receivers that can be connected to a transmitting node has a 16-bit mailbox entry dedicated for that transmitter. As there are four nodes each transmitting to three other nodes, there are 12 mailboxes. Bit-15 of each mailbox shows the status of the receiver associated with it. "0" is used for ready and "1" for busy. Other bits are used to hold the number of the buffer allocated for the transmission. If a receiver is ready, the transmitter writes the allocated buffer number in the mailbox of the receiver and sets bit-15 to mark it a busy receiver. It also sends an interrupt to the receiver to signal that a message is ready to be collected. Upon receiving the interrupt, the receiver refers to the appropriate mailbox to identify the number of the buffer in use. After

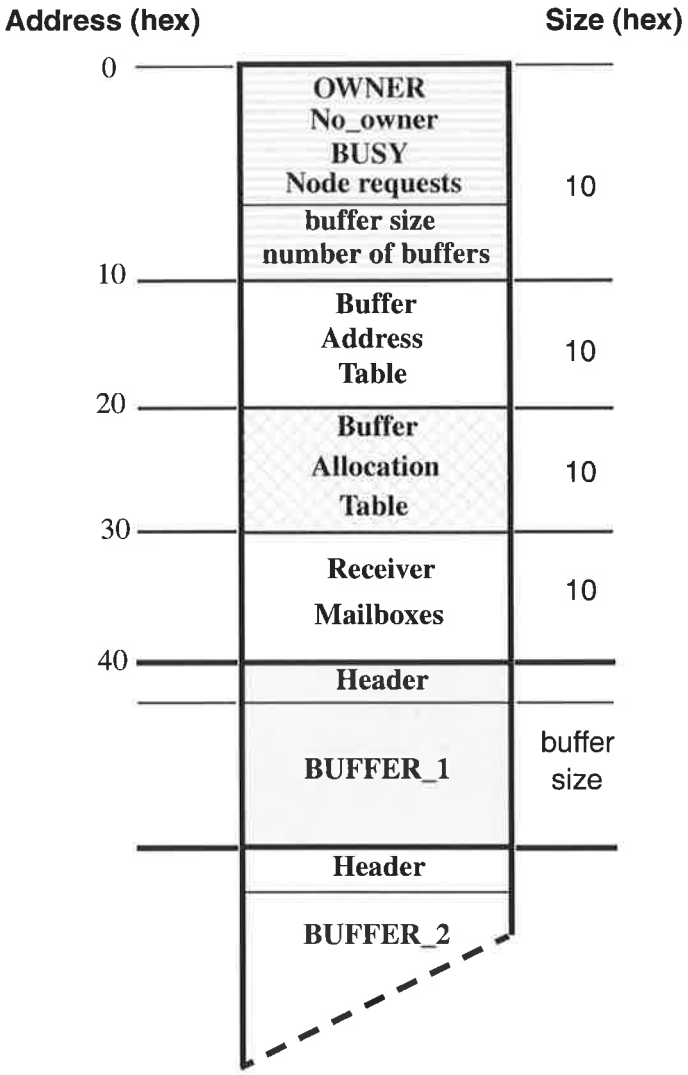

Figure 5.10 Memory map of MultiCom for dynamic allocation

The components used for dynamic allocation in shared memory are lock variables, buffer address table, buffer allocation table, receiver mailbox, and several buffers.

removing data from the buffer, the receiver clears bit-15 in its mailbox to indicate that the node is ready to receive more data from the same transmitter.

Buffers:

Each buffer consists of a header and data. In the simplest form, the header contains the size of data in the buffer. Other information such as the sender's ID, the receiver's ID, and the relative order of the packet in the entire message can be included in the header.

The protocol is shown in Figure 5.11. More detail can be found in Table 5.4.

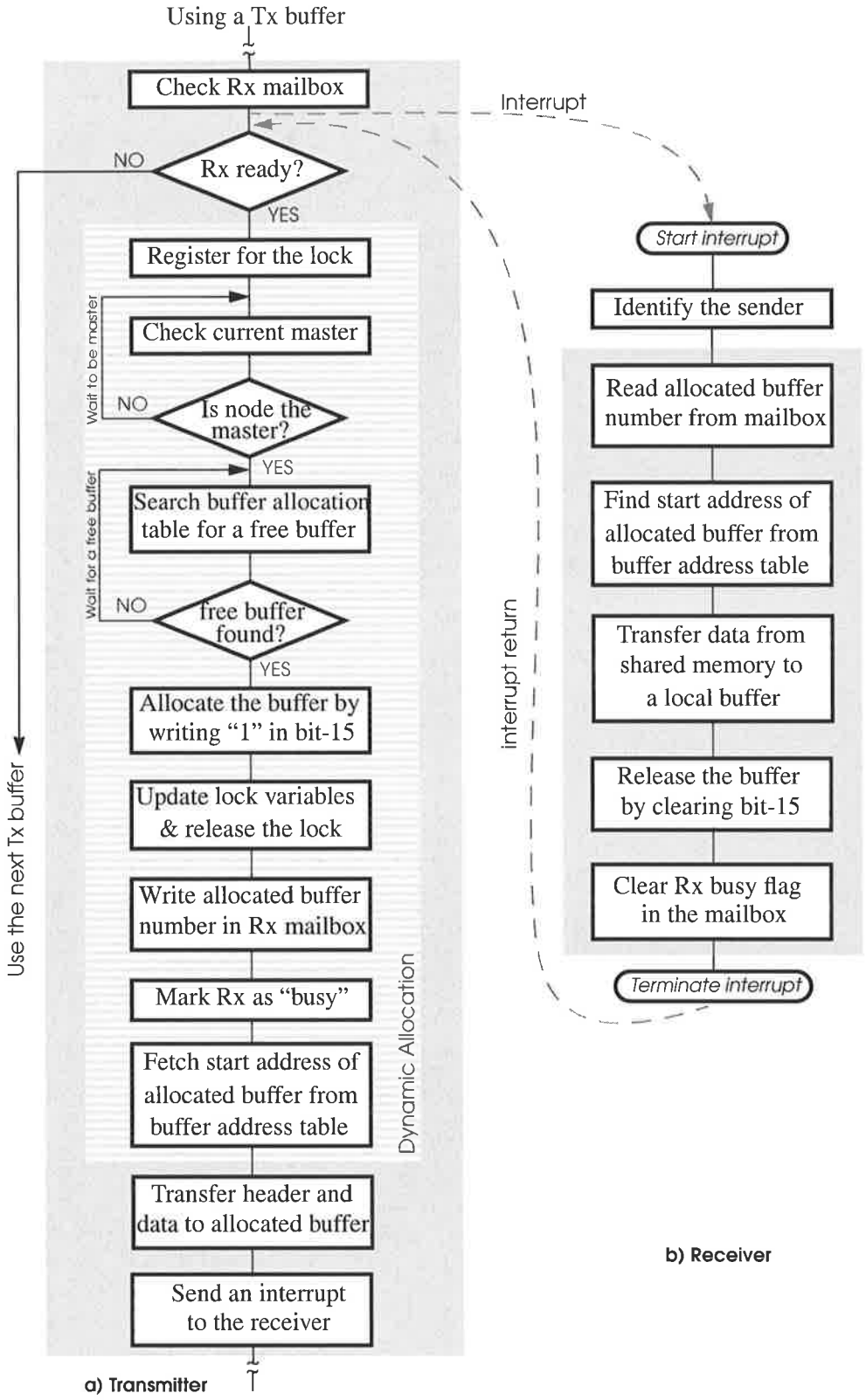

Figure 5.11 Communication protocol for dynamic allocation

Only the shaded parts of the protocol are shown. **a)** The transmitter applies for the lock. After getting the lock, the transmitter allocates a buffer, writes the buffer number in the mailbox of Rx, releases the lock, and transfers data to the buffer. **b)** The receiver gets the allocated buffer number from its mailbox, reads data, releases the buffer, and clears its busy flag in the mailbox.

Table 5.4 Protocol for dynamic allocation

In transmitter:

1. The status of the Rx node in the mailbox is checked. If it is not ready, indicating that the previous transmission is still in progress, the transmission loop is resumed.
2. A buffer is allocated for transmission. To do this:
   - The lock is obtained by applying and waiting to become the master.
   - The file allocation table is searched and a free buffer is allocated. If no free buffer is available, the search is performed repeatedly until one buffer is freed by other nodes.
   - The lock is released.
3. The packet is transmitted as follows:
   - The allocated buffer number is written into the receiver mailbox and the Rx is marked as busy.
   - The start address of the buffer is obtained from the address table.
   - The header and data are written into the buffer.
   - The variables are updated.
4. An interrupt is generated for the receiver. The transmission loop is resumed.

In receiver:

1. Depending on the received interrupt, the identity of the transmitter is known to the receiver. Therefore, after receiving an interrupt:
   - The local buffer to hold the data is determined.
   - The buffer number used in transmission is read from the receiver mailbox and its address is obtained from the buffer address table.
   - The size of the packet is obtained from the buffer header.
2. Data is transferred from the shared memory into the local buffer.
3. The buffer is released and the busy flag in the mailbox is cleared to indicate that the packet is received and the receiver is ready to get more data from the same transmitter. Variables are updated and the interrupt service routine is terminated.

Note that in the last step of the protocol as shown in Figure 5.11-b, the receiver releases the buffer without using the lock. The reason for this will be explained in the discussion section.

3.3 Protocol for multicast / broadcast

Multicast and broadcast can be implemented by applying minor changes to the protocol. Each entry in the allocation table is capable of holding broadcast information. A modified entry incorporates the following information:

- Bit-15 is for buffer status ("0" for free, "1" for allocated).
- Bit-14 is for transmission type ("0" for normal, "1" for multicast/broadcast).
- Bits 3-0 are node-bits for the receivers 4 to 1 in bitmap form. A "1" in each bit position indicates that the corresponding node will receive the data.

The structure of the buffer allocation table is shown in Figure 5.12. The format and the protocol were modified in a way that the added overhead to the transmissions not using multicasting was reduced to a minimum.

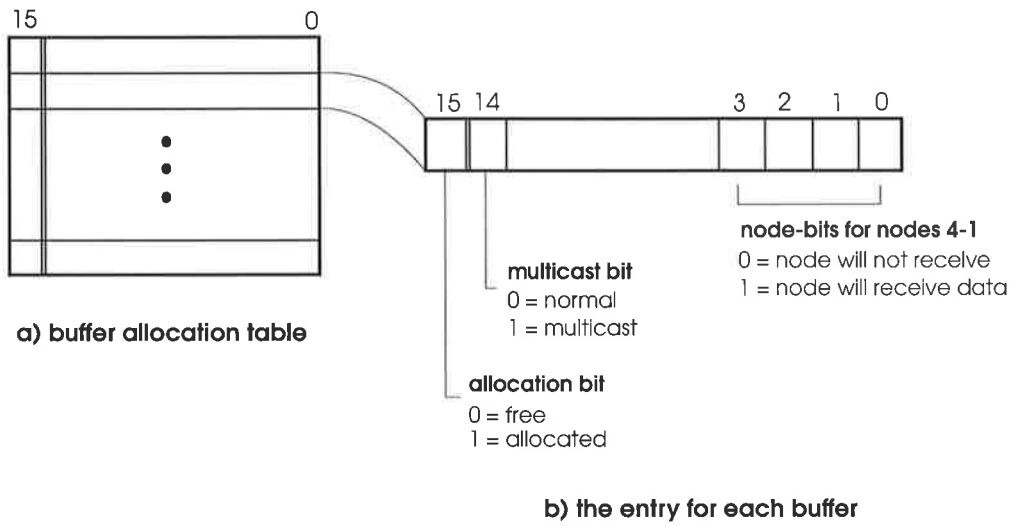

Figure 5.12 The structure of buffer allocation table

a) Each buffer has an entry in the allocation table. Bit-15 is used to allocate the buffer. **b)** Other bits of the buffer entry are used in multicasting/broadcasting. Bit-14 shows normal or multicast transmission, and bits 3-0 define which nodes should receive the data.

The transmitter checks the status of all the receivers. If all are ready, it allocates a buffer, writes the broadcast information in the buffer entry in the allocation table, and transfers data to the buffer. Then it writes the buffer number into the mailbox of each receiving node and marks them as busy. In the end, it generates an interrupt for each of the receivers.

The receiver reads the data from the buffer and clears its node-bit from the corresponding buffer entry in the allocation table. If the buffer entry shows that no more nodes are left to receive the data, the receiver also releases the buffer as the last node.

Figure 5.13 shows the communication protocol under multicasting/broadcasting.

Because of the structure of the buffer allocation table, there is a possibility of write-write conflict on the shared memory. This may happen if two or more nodes try to remove their node-bits from the buffer entry at the same time. In removing a node-bit, other information in the buffer entry must be preserved. Therefore, any programming method should first read the entry, mask the corresponding bit, and write it back. Even with the instruction set of the TMS320C50 that makes it possible to do this task using only one instruction, the memory read and write take place at two different clock cycles. Therefore, if two or more nodes perform this step at the same time, one of them, which is normally the lagging one, may overwrite the data written by the other nodes. Consequently, removing the node-bit may not take effect properly and the buffer may remain allocated.

This conflict is unlikely to happen and it was observed only in long runs on MultiCom. However, a reliable system requires the removal of the conflict. Two approaches can be considered in this regard:

1. After removing the node-bit from the buffer entry, each node checks the entry and if unsuccessful, tries again.
2. The node-bit is removed by the aid of the lock.

The first approach is very simple and there is not much overhead involved. However, because at least two nodes may attempt to write different data in the same location simultaneously, undesired information could be written in the target location, which

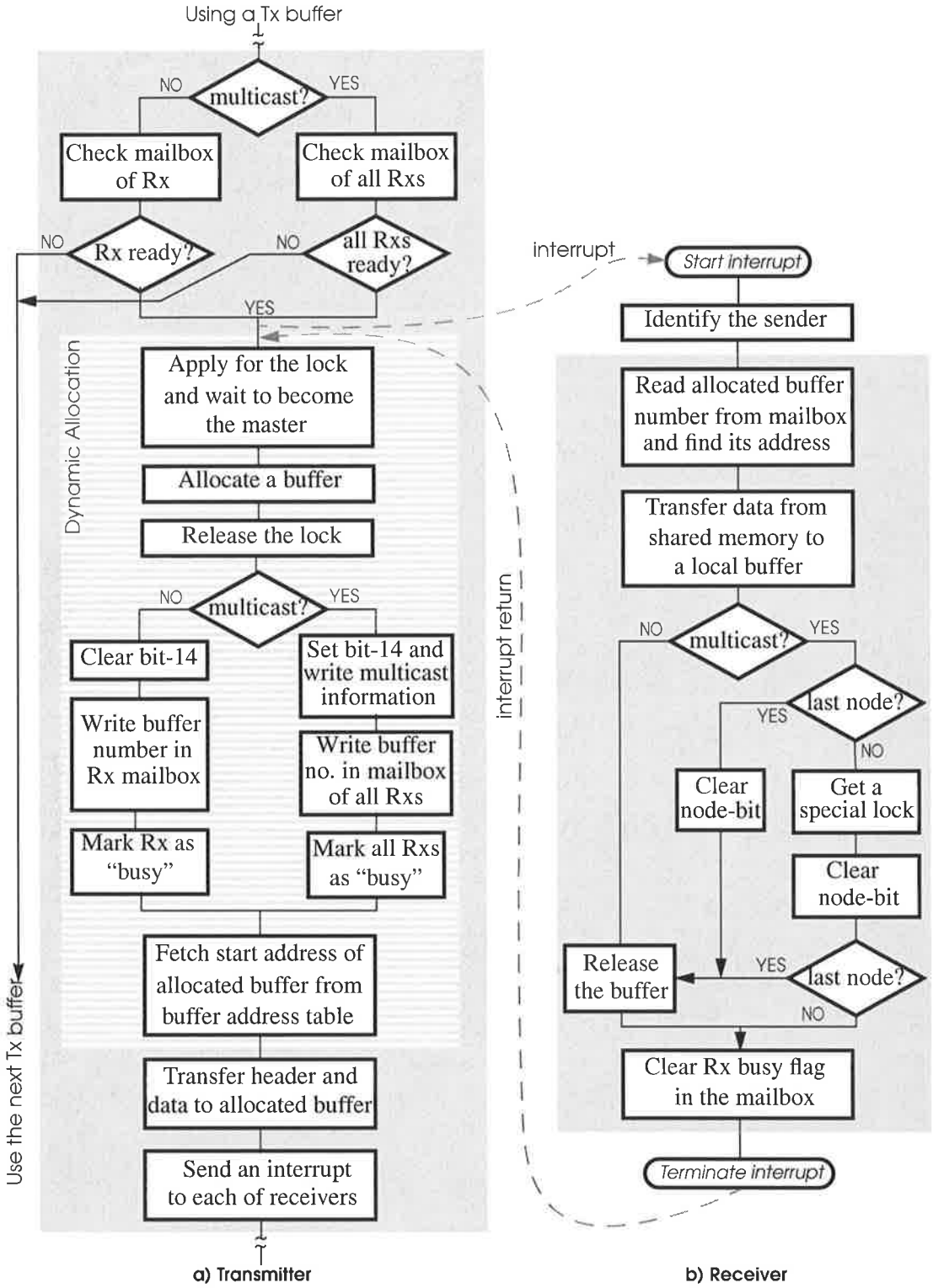

Figure 5.13 Communication protocol using multicasting/broadcasting

a) In case of a multicast/broadcast, transmission starts only if all of the receiving nodes are ready. After obtaining the lock and allocating a buffer, the transmitter writes the information of the receivers in the buffer entry, and the buffer number in the mailbox of each receiver. Then it transfers data to the buffer and sends an interrupt to each receiver. **b)** In the receiver, first data is collected from the buffer. Then, if there is no other node to receive the data, the receiver releases the buffer, otherwise, it only removes its node-bit using a special lock.

could be neither of the two nor even a combination of them. This may create a deadlock and is not acceptable for a reliable system. It is worth mentioning that no such situation was observed in MultiCom even in the long runs.

The method of using a lock to remove node-bits is better suited for a reliable system. There are two options in this case:

1. Use of the existing lock.
2. Use of a special lock as explained below:
   - Removing the node-bits is performed on an allocated buffer entry, but the allocation process searches for a free buffer. Therefore, there is no conflict between the two, even if both are performed at the same time. Hence, a separate lock can be used for this purpose.
   - In order to minimize the overhead, a simplified version of the fast lock can be used. In this version, a fixed priority for the nodes rather than circular priority can be implemented.
   - If other nodes have already removed their node-bits from the buffer entry, the last node does not require a lock to remove its node-bit.

Using the existing lock increases the overhead, and the system may end up in a deadlock as explained in the discussion section, but the special lock has a reasonably small overhead. Therefore, the latter was used for removing the node-bits in multicasting/broadcasting.

It should be restated that the programming was carried out with the minimum added overhead to the communication without multicasting/broadcasting.

Considering the fact that broadcasting can reduce the communication overhead in the system by a large extent, the small overhead of removing the node-bits is negligible. The system throughput is expected to rise considerably under this protocol.

The results obtained from MultiCom under different protocols are presented and discussed in the following section.

4 Results from MultiCom

Using suitable protocols, static allocation, dynamic allocations, and multicasting/ broadcasting were tested on MultiCom and the performance of the system was evaluated. The results are presented in this section. The outcomes are also compared to other systems using serial links, bus-based architecture, and dual-port memories. Before discussing the results, the mechanism of data transfer using the nodes of MultiCom is described so that the results can be analysed thoroughly.

4.1 Details of data transfer

In order to interpret the achieved results better, the mechanism of data transfer is explained in more detail. The transfer of data from a local buffer to a buffer in the shared memory and vice versa is performed with block transfer instructions. The TMS320C50 has a series of instructions to perform this task. One of them is BLDD standing for "Block Load from Data memory to Data memory". This instruction can be repeated with the RPT instruction to create the transfer loop. Considering the effect of the 4-stage pipeline of the processor, BLDD in repeat mode is optimised to act as a single cycle instruction. Therefore, the data transfer from shared memory to local memory takes only one cycle per a 2-byte word. However, the reverse transfer that writes to the external memory includes an internal wait cycle and takes two cycles per word. As explained in the discussion of the nodes in Chapter 4, this extra cycle is introduced to allow a smooth transition between write and any adjacent bus operation [TI 97]. Hence, the overall transfer of a 2-byte word from one node to another takes place in three cycles; two for writing by the transmitter, and one for reading by the receiver. This number puts an upper limit for the communication bandwidth of the system.

With the 50 ns cycle time of the processor, the communication bandwidth through the memory for each node is 13.3 MBytes/s (i.e. 2 bytes in 150 ns). For four processors in the system, and assuming that all of them are communicating without any overhead, the peak communication rate or the bandwidth for the overall system would be 53.4 MBytes/s. In practice, the following factors, as explained in different protocols, degrade the peak rate:

- Finding the transmit buffers and checking the readiness of the receiver.
- Allocating a buffer in the shared memory (especially in dynamic allocation, which requires applying for the lock, acquiring it, and searching for a buffer).
- Initialising the counter and address pointers, initiating the repeat mode, and generating an interrupt for the receiver.
- Latency in acknowledging the interrupt in the receiver, and the overhead of identifying the transmitter, initialising and initiating a block transfer.
- The busy status of the receivers, which adds extra delay to the communication.
- The finite and limited buffer size in the shared memory, which causes a long message to be sent in several packets rather than one.

The buffer size in the shared memory is a very important factor in the communication rate. As there is a fixed overhead in sending a packet of any size, by using bigger buffers, a long message can be sent in fewer large packets and the overall overhead will be reduced, resulting in a higher throughput.

With the total communication rate as the focus of the experiment, MultiCom was tested under static allocation, dynamic allocation, and broadcasting. The results are presented in the next section.

4.2 Results of static allocation

The effect of the buffer size on the communication rate and the associated communication overhead using static allocation are shown in Figure 5.14.

The figure shows that by increasing the buffer size, the communication rate increases and approaches the peak communication rate. For static allocation, the maximum buffer size for the shared memory of 4K words was 336 words. The maximum communication rate of 45.5 MBytes/s was achieved for this buffer size, showing overhead of 15%. Considering the unavoidable overheads in the system as discussed above, 15% is reasonable and shows the effectiveness of the protocol under heavy traffic. As explained in the next chapter, by increasing the memory size, the communication rate can increase and approach the peak rate, because bigger buffers can be used.

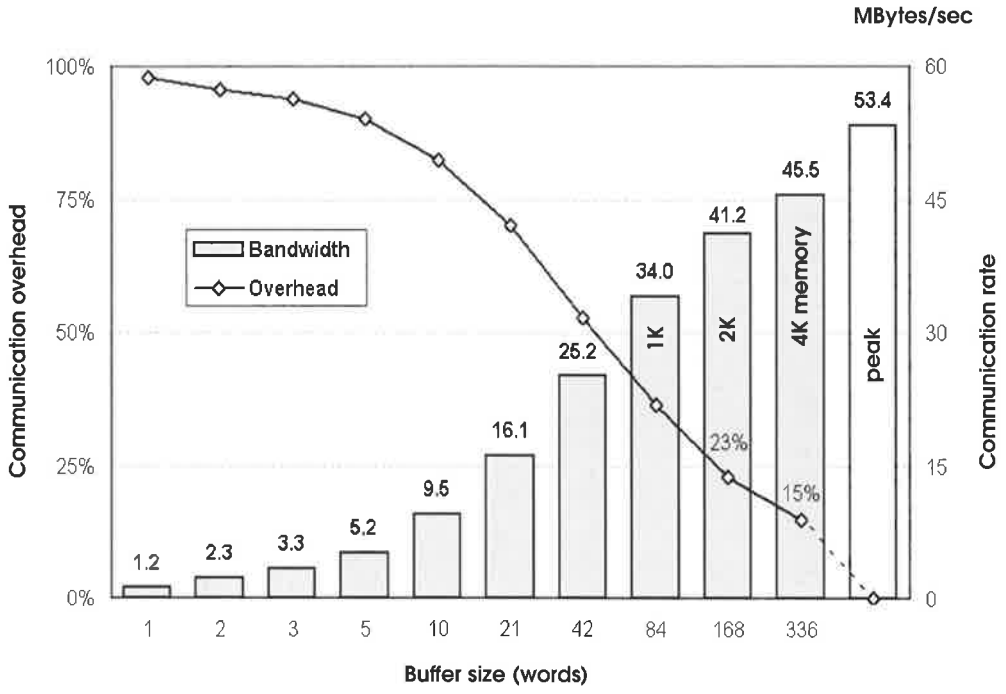

Figure 5.14 Effect of buffer size on communication rate and overhead in static allocation

Increasing the buffer size increases the communication rate and reduces the overhead. The maximum rate of 45.5 MBytes/s was achieved for the buffer size of 336 words. With 15% overhead, the result is close to the peak rate.

The figure also shows that by reducing the memory size to 2K words, which reduces the buffer size reduces to 168 words, the communication rate is reduced by only 9%. Hence, an acceptable performance can be achieved even with a small shared memory. Increasing the memory size beyond its current size of 4K words will only improve the performance slightly and may not justify the additional expense.

This is an important outcome of the experiment indicating that the structure does not require a large shared memory [Asgari+ 01]. This outcome will be verified by the simulation model, and will be discussed in more detail in the next chapter.

Latency is defined as the time it takes for a packet to reach from a sender to a receiver. The latency for the static allocation was 2.4 μs for a 4-byte packet and 12 μs for a 128-byte packet.

4.3 Results of dynamic allocation

With the performance of static allocation being reasonably close to the peak communication rate, there is not much to gain in applying dynamic allocation. However, there are two reasons why dynamic allocation is more useful:

1. Dynamic allocation scales better than static allocation. As explained in Chapter 7, the ultimate goal is to design a communication structure using 9-port memories. In static allocation, an *N*-port memory should be divided into *N(N-1)* buffers. Therefore, the buffer size would be very small for large *N*. Moreover, as the number of ports increases, the internal connections inside the memory chip increase considerably resulting in less space for the memory cells. Hence, a 9-port memory is expected to have a small capacity and dividing it into 72 buffers will yield even a smaller buffer size. Hence, the performance loss will be substantial. Considering the fact that not all of the buffers are simultaneously used, a better memory management such as dynamic allocation is required to make efficient use of small shared memory size.
2. Multicast and broadcast, which potentially increase the system performance considerably, can be implemented easily in dynamic allocation as explained before. However, in static allocation, each buffer is pre-allocated to one transmission only and it cannot be shared with the other nodes. It might be possible to change the structure of static allocation to incorporate multicast and broadcast, but it would require a lock similar to the one used in dynamic allocation and will lose its simplicity, speed, and efficiency.

Dynamic allocation adds extra overhead to the system, mainly because of the lock mechanism and the buffer allocation process. However, its capability for easy implementation of multicasting/broadcasting, as well as its better scalability makes it more attractive than static allocation. In Chapter 6, the difference between the scalability of these allocation methods will be demonstrated.

Figure 5.15 shows the communication rate for different buffer size and buffer counts. During each test, the number of allocations performed on each buffer was also recorded. An interesting result was achieved from these figures; they showed that for four nodes under heavy communication traffic, no more than six buffers were required

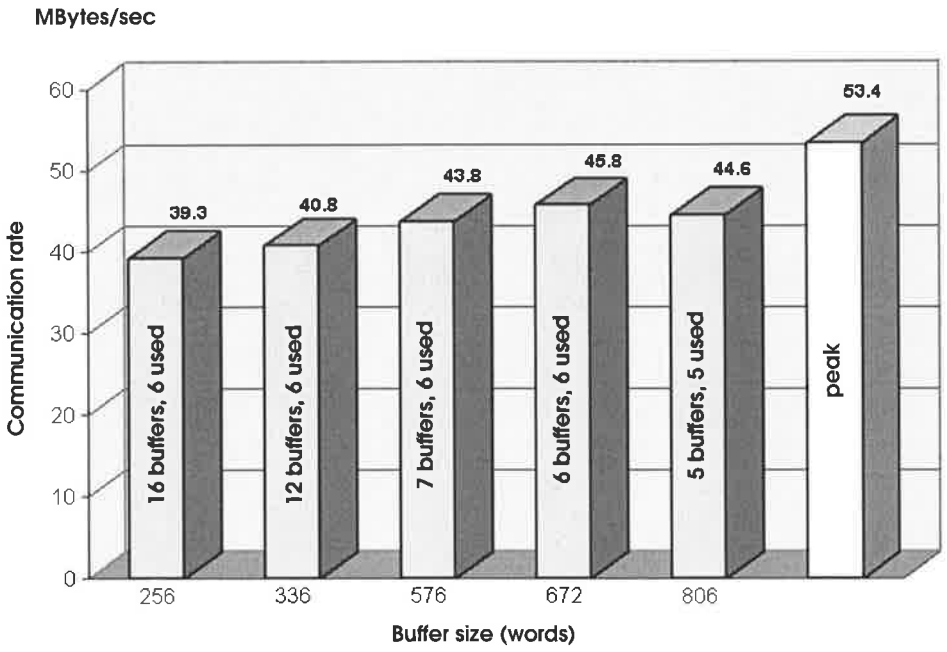

Figure 5.15 Communication rate for dynamic allocation

In dynamic allocation, increasing the buffer size increased the communication rate, but no more than six buffers were used. The best result of 45.8 MBytes/s was achieved for the buffer size of 672 words. Increasing the buffer size to 806 words dropped the performance because only five such buffers could be created with memory size of 4K words.

(In a rare case, the seventh buffer was used only once). Therefore, for four nodes, only half of the buffers used in static allocation were used. The dependency between the number of nodes and the buffers in use will be checked by the simulation model in the next chapter.

The highest value for the communication rate was 45.8 MBytes/s, achieved for six buffers of 672 words. As expected, the increase in performance is not significant compared to the 45.5 MBytes/s achieved for static allocation; however, it is expected that this method can perform much better if a large number of ports and small memories are to be exploited. Furthermore, as shown in the next section, better performance can be achieved by using dynamic allocation for multicasting/broadcasting. It should be noted that the results of Figure 5.15 were obtained for a system in which multicast/ broadcast was also checked. Slightly higher performance could be achieved if the overhead of checking for multicast/broadcast was removed.

For five buffers of 806 words, although the buffer size was increased, the performance dropped. This is because one less buffer could be created for the limited memory size of 4K words. Hence, some nodes had to wait for a buffer to be released. Increasing the number of buffers to seven dropped the performance because of the reduced buffer size. The extra buffer in this case was rarely used. The rate obtained for the buffer size of 336 words was 40.8 MBytes/s compared to 45.5 MBytes/s of the static allocation for the same buffer size. This reveals the extra overhead imposed on the system by using the lock mechanism and the allocation process.

Moreover, the latency of the dynamic allocation was slightly higher than the static allocation, obviously for the same reasons stated before.

Dynamic allocation enjoys the benefits of better memory management, but it suffers from the overhead imposed by the serial nature of the lock mechanism, and the software implementation of the lock. Better performance could be expected if a hardware lock similar to the semaphore logic were available for multiport memories. Moreover, improving the algorithm for the software lock could enhance the performance slightly.

4.4 Results of multicasting/broadcasting

One of the major advantages of the dynamic allocation is the easy implementation of multicasting/broadcasting. A message is written into the shared memory buffer once, but it is received by more than one node. This results in a considerable saving in time by avoiding duplicate buffer writes. In addition, multiple attempts to apply for the lock and to allocate a buffer are reduced to one attempt only and the overhead decreases. Hence, messages are transferred much faster and the communication rate increases.

Figure 5.16 shows the communication rate for broadcasting. The best result was 68.2 MBytes/s, and shows about 50% increase in the performance of static or dynamic allocations. Owing to the fact that messages can be delivered much faster because of fewer writes and reduced overhead, the achieved communication rate is 30% better than the peak rate for node-to-node communication. The maximum number of buffers involved in communication did not exceed four.

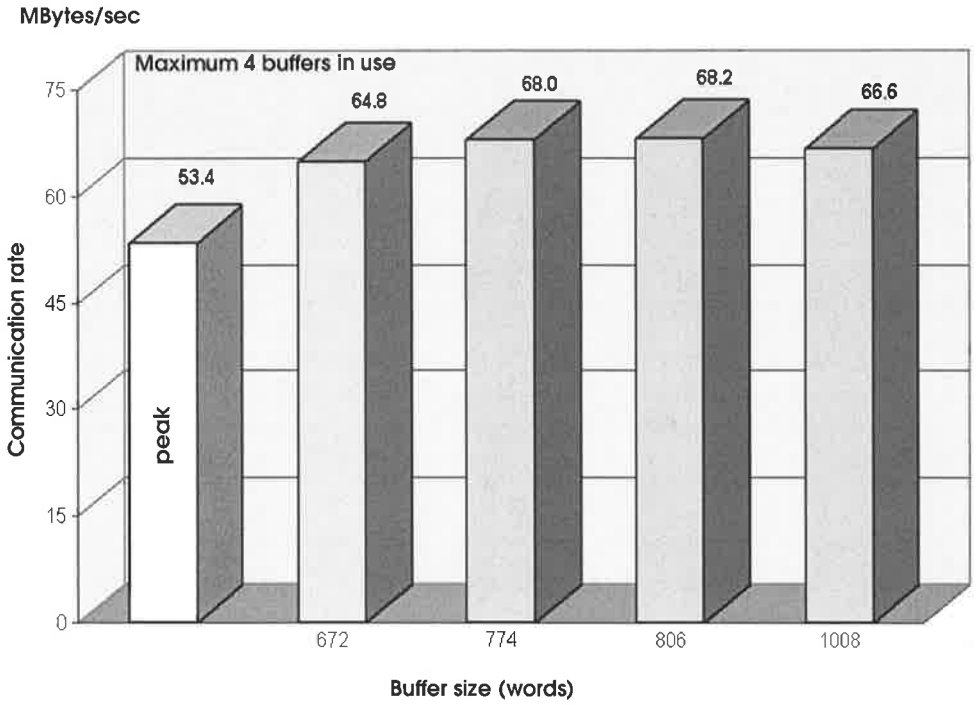

Figure 5.16 Results of broadcasting

Broadcasting can increase the performance significantly because of fewer writes and reduced communication overhead. The best result of 68.2 MBytes/s is 50% better than the results of static or dynamic allocations. The realistic performance is 64.8 MBytes/s achieved for the same buffer size used in dynamic allocation.

For a buffer size of 1008 words, the communication rate shows a drop. This is because the buffer size is comparable to the message size of 1792 words and in sending the message in two packets, part of the buffer is wasted. If the message size were increased, the performance would be much better; however, because of memory limitation of the nodes of MultiCom, it could not be tested. This fact was discovered by the simulation model presented in the next chapter.

Although 68.2 MBytes/s is the best result achieved, the realistic result is 64.8 MBytes/s obtained for the buffer size of 672 words. The reason is that in a system, most of the messages are transferred between two nodes, and only a few may be capable of being broadcast. Hence, multicasting/broadcasting should be considered in the context of the normal operations of the system. As the maximum available buffer size for dynamic

allocation is 672 words, only the performance of the broadcasting for this buffer size should be considered as the realistic throughput. Compared to static or dynamic allocations, the performance increase is 41%.

These results confirm that dynamic allocation performs better than static allocation because of the ability to implement multicasting/broadcasting, and better scalability as shown in the next chapter.

4.5 Comparison of results

The results from MultiCom show that both static and dynamic allocation can achieve a reasonably good performance close to the peak communication rate. Furthermore, dynamic allocation shows its advantage by the use of broadcasting that can reduce the overhead considerably and achieve a higher communication rate.

As discussed in Chapter3, all the communication structures based on multiport memory have not gone beyond the proposal stage and their performance have not been investigated. This makes it difficult to compare the outcome of this research to the previous work. However, in the following sections, the results of MultiCom are compared to a bus-based system and other systems using dual-port memories or serial links.

4.5.1 Comparison to a bus-based system

On a bus-based system using a conventional single-port memory as shown in the structure of Figure 2.1 on page 12, the processors need to take turn for using the bus and the communication rate is limited by the memory bandwidth and the speed of processors. The realistic communication rate would be significantly lower than the memory bandwidth. This is because each word of a message requires two transactions on the shared memory. The transmitter should write the word in the shared memory and the receiver should read from it. Hence, for a transfer of a word under heavy traffic, the delay of the arbitration logic and the waiting time of the processors for the busy bus should be taken into account. In addition, the time spent on local memory transactions to fetch or store the word should be also considered. Moreover, if the system is upgraded to more nodes, the communication rate will not improve at all, and in fact, it

will be further downgraded, as there would be more demands to access the shared memory and increased waiting period for the nodes.

To illustrate these facts, the results of a research to interconnect four Inmos T800 Transputers through a bus-based shared memory are discussed here. [Boianov+ 91] reports a network of 20-MHz T800 Transputers interconnected with 64 KBytes of single-port memory (with 45 ns access time) through memory arbitration logic. If the bus was used by one of the nodes, other nodes requesting a memory access were forced into a wait state by the arbitration logic. When the bus was released, the arbitration logic terminated the wait state for the node with the highest priority enabling the node to use the bus for its memory access. For a reduced communication overhead, the memory management of this system used predefined memory buffers similar to the static allocation discussed earlier. The communication rate for 2 or 3 nodes was 3.36 MBytes/s; however, adding the 4th node reduced the communication rate by 25% to 3.31 MBytes/s. This was because of the increased waiting period for the nodes to perform a memory access.

MultiCom outperforms this bus-based system by a factor of 14. The main reason for this large difference is that in the bus-based system the nodes have to share the limited bandwidth of the bus. In a heavy demand for memory, such as all-to-all communication, most of the times the nodes are forced into wait states and the throughput of the system is reduced considerably.

4.5.2 Comparison to a system using dual-port memory

The result of a system using dual-port memories for communication is discussed below. [Campbell+ 96] reports the design of COMPS, which stands for "COmmon memory Message Passing System". This system was designed by using Intel 486DX2/66 CPUs as node processors (with 66 MHz processor clock and 33 MHz bus clock), and IDT7006 dual-port memories (16 KBytes, 45 ns) as shared memory. The memory chips were used in pairs to achieve a word length of 16 bits. A cluster of five nodes was reported to be near completion in [Campbell+ 96]. Under light traffic conditions and for a buffer size of 1000x16 bits, the measured memory access rate to send a packet of data to the dual-port memory was quoted to be around 2 MBytes/s. The same rate applied for

receiving data from the dual-port memory. Consequently, each node had a communication rate of 1 MBytes/s. The aggregate communication rate was not mentioned; however, for comparison, an upper bound can be estimated for a four-node system. In order to have a full connection between four nodes, this system would require six blocks of dual-port memory between the nodes, and each node would need to be connected to three different memory blocks. Under these conditions, the upper bound communication rate would be 4 MBytes/s [Asgari+ 98]. It should be mentioned that COMPS was designed to work under an operating system with slow bus protocol and it used slower memory than MultiCom (45 ns compared to 35 ns of MultiCom).

The results of MultiCom show a considerable improvement over the system using dual-port memories in terms of both performance and cost. The communication rate is increased more than 11 times and the number of blocks of memory is reduced six times (one 4-port memory compared to six dual-port memories, which is about 75% reduction in cost). The increase in performance could be even higher if the exact value of the communication rate rather than an upper bound was available. Moreover, broadcasting shows an increase of at least 16 times in performance compared to this system.

It is worth mentioning that although broadcasting is possible in structures using dual-port memories, it will not have similar benefits as the systems using multiport memories.

4.5.3 Comparison to serial links

The results of a system using serial links with comparable capabilities to MultiCom are also included in this comparison. As discussed in detail in the previous chapter, in this hypothetical system, four nodes are connected in a 2-cube structure using 20 Mbps links. The intermediate nodes pass the data to the destination node without storing it. It is assumed that there is no overhead in the communication and the optimum approach in delivering the messages is selected. Under these conditions, the upper bound for the aggregate communication rate would be 10.9 MBytes/s. Compared to the upper bound performance of this system, MultiCom performs at least 4.2 times better.

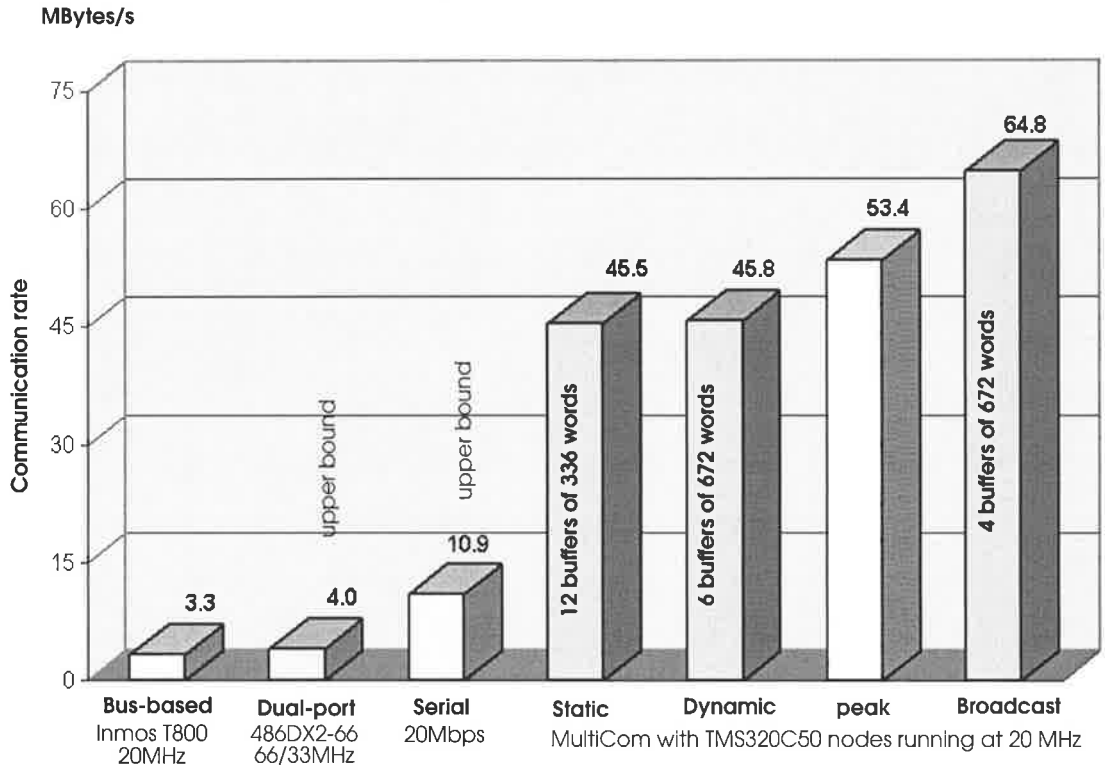

Figure 5.17 Comparison of results

MultiCom outperforms certain bus-based systems and systems using dual-port memory or serial links by a considerable amount. The performance is more than 4.2 times compared to a serial system and 11 times better than a system using dual-port memory, and 14 times better than a bus-based system. Broadcasting has the highest performance in MultiCom.

Figure 5.17 summarizes the results obtained form MultiCom and compares them to a bus-based system and other systems using dual-port memories or serial links.

5 Discussion

In this section possible ways of improving the performance of MultiCom such as using wider datapaths or faster nodes are discussed. In addition, the structure of the allocation table is discussed and the advantages of using this structure over other methods are presented.

5.1 Improving the performance

The performance of MultiCom can be improved in two ways as explained in the following sections:

5.1.1 Increasing the speed of nodes

The performance of MultiCom was tested with 20 MHz node-processors. Increasing the speed of the nodes can enhance the performance of the system. In upgrading the processor speed, the speed of memory or its access time should be also considered. If a linear performance increase is required, the speed of memory should be upgraded at the same rate of the processor speed. If the memory speed cannot follow the rate of increase of the processor clock, arrangements should be made for the memory to be accessed in two or more cycles. In this case, the performance will depend mostly on the memory access time. Nevertheless, performance boost is still expected as tasks other than data transfer can be performed at a higher clock rate, reducing the communication overhead.

In 2002, IDT introduced IDT70V5388, a four-port memory with capacity of 64Kx18 bits and speed of 200 MHz [IDT 02]. The performance of MultiCom can be increased significantly by using high-speed processors and faster memories such as IDT70V5388.

5.1.2 Use of wider datapath

MultiCom was built as a prototype to evaluate the performance of a communication scheme using multiport memory. It used 16-bit processors and the required 16-bit memory was created by two 8-bit memories connected in parallel. There is no restriction in using a wider datapath. For example, if a 32-bit processor is selected, the required memory can be built with four 8-bit memories in parallel, or with two 18-bit newer version of the 4-port memory. Similar arrangements can be made for 64-bit nodes.

Using a wider datapath can enhance the communication bandwidth linearly because transferring a fixed amount of data to or from memory will require less time. The practical communication rate will be slightly less than this, because the communication overhead that arises mainly from the preparation of the nodes for transmission or

reception does not depend very much on the width of the datapath and remains virtually the same.

A wider datapath can compensate for the use of smaller multiport memories in larger systems. This is because horizontal expansion of memory can yield higher performance increase than vertical expansion. For example, MultiCom upgraded to a 4Kx32-bit shared memory can achieve much higher rates than a system upgraded to an 8Kx16-bit memory. Although both use the same number of memory cells, the throughput will be almost doubled in the former, where as in the latter the throughput will increase only slightly (Refer to Figure 6.1 on page 123). This point is especially important in bigger systems in which a larger number of ports for the multiport memory is required. As explained before, multiport memories with larger port counts will probably have very limited capacity; however, the small capacity can be balanced by the use of a wider datapath.

Wider datapaths combined with higher port counts will increase the routing complexity of the PCB design. However, with current multi-layer PCB design technology, a careful design can overcome the routing problems. The designer should also keep the PCB trace lengths balanced to avoid signal skews.

Wider and faster multiport memories are supported by the leading memory manufacturing companies. For example, IDT70V3579 is a dual-port memory module from IDT with a capacity of 32Kx36, speed of 4.2 ns/133 MHz in a 208-pin packaging [IDT 99]. Moreover, as explained before, IDT has released its new version of 4-port memory with the capacity of 64Kx18 bits and speed of 200 MHz in a 256-pin packaging [IDT 02].

5.2 Structure of the allocation table

As explained earlier in Section 3.3, the allocation table uses one entry per buffer, and bit 15 indicates if the buffer is allocated or free. An alternative approach would be to use only one word to hold the status of all the buffers as a bitmap. Although this is a very common practice, it may end up in performance loss by adding extra overhead to the system. The reasons are discussed below:

In the last step of the protocol in Figure 5.11-b, the receiver releases the buffer without using the lock. The reason lies in the structure of the allocation table as shown in Figure 5.12. A free buffer is sought in the allocation process, whereas an allocated buffer is freed in the releasing process. As each buffer has a separate entry, there is no conflict between allocating and releasing, even if performed simultaneously. Hence, there is no need to acquire the lock for releasing a buffer.

On the other hand, if the bitmap approach were used to keep the status of the buffers, the use of the lock for releasing a buffer would be inevitable. This is because two nodes allocating and releasing two distinct buffers at the same time would write different information in the same location; hence, there is a potential for a write-write conflict. This type of conflict could easily crash the entire communication in the system because of the major dependency of the communication on the allocation table.

Using the lock to release the buffer would require the receiver to apply for the lock and wait to acquire it. This process increases the overhead of the system and reduces the performance. Moreover, it could interfere with the other activities of the receiving node such as transmitting to other nodes. The reason is that at the end of an interrupt service routine where the receiving node should free the buffer, the node has no information about its previous attempt to register for the lock (for transmission purpose) and the progress made so far. If the node applies a second time, it may destroy the previous request or interfere with it. It is possible to solve this problem by modifying the protocol to check previous attempts and share it for releasing the buffer as well, but this means unwanted overhead, extra performance loss, as well as a prolonged interrupt service routine.

Moreover, the bitmap structure for the allocation table could end up in a deadlock. In order to clarify this statement, imagine that a transmitting node acquires the lock to allocate a buffer for its transmission, but no free buffer is available. Although the critical section of a node when using a lock should be short and free of loops, there is no point in releasing the lock for others to use. This is because the lock is used exclusively to allocate a buffer. If another node gets the lock, it will also search for a free buffer, which is not available yet. While holding the lock, the node keeps trying until a buffer is

released. Meanwhile, another node finishes receiving the data and needs to release the buffer. It registers to get the lock. This creates a deadlock because there is no buffer to allocate, and no lock to release an allocated buffer. The deadlock can be removed by adding extra steps to the protocol such as releasing the lock when no buffer is available and applying again later; however, adding extra steps increases the overhead and decreases the performance.

Another benefit of using a separate entry for each buffer is that it facilitates the implementation of multicasting/broadcasting. As explained in Figure 5.12 on page 96, the unused part of the buffer entry is used to identify the receiving nodes. If a bitmap structure for the allocation table were used, a separate table would be required to hold the broadcast information.

Furthermore, there is no extra overhead in searching the current structure of the allocation table for a free buffer compared to searching the bitmap structure. In fact, with the instruction set of the processor in use, the current structure is searched even faster.

6 Conclusion

Memory management is an important issue for the interprocessor communication of MultiCom and it is implemented using both static and dynamic allocations. Static allocation is easy to implement and the associated overhead is very low. However, only a portion of the valuable shared memory is actively used in communication at a time, and the rest remains idle. On the other hand, dynamic allocation can use the entire capacity of the memory in the communication, but its structure is much more complicated and requires a lock mechanism, which has a serial nature. The lock mechanism can be easily implemented with hardware semaphores; however, the memory used in MultiCom did not have a semaphore latch. Hence, the only possible way to implement a lock was to use software semaphores.

The waiting list algorithm was tested on MultiCom to implement a software semaphore. In addition, two newly devised algorithms based on a modified TOKEN passing scheme

were designed and tested on the system. Both of the algorithms proved to perform effectively and could overcome all the memory conflicts.

For the communication protocol, first a basic protocol was developed and tested for static allocation. Then the software semaphore was added and the protocol was modified to support dynamic allocation. As an advantage of dynamic allocation, multicasting/broadcasting was also implemented by applying the required modifications to the protocol. The structure of the buffer allocation table was designed in a way that it could facilitate the implementation of multicasting/broadcasting and the lock was not required for releasing buffers.

The aggregate communication rate in the entire system was measured in several experiments. The buffer size of the shared memory was a very important factor in this regard. For static allocation, the best communication rate was 45.5 MBytes/s. For dynamic allocation, the best rate was 45.8 MBytes/s. This rate was achieved in the presence of the extra overhead for checking multicasting/broadcasting in the protocol. Both of the results were very close to the peak communication rate in which all of the nodes could communicate with zero overhead. Considering the amount of overhead available in the protocols, the overhead of 15% is very reasonable. Multicasting/ broadcasting could achieve the rate of 64.8 MBytes/s, which was 41% better than the node-to-node communication rate using static or dynamic allocations.

The capability of dynamic allocation in implementing multicast/broadcast as well as its better scaling characteristics proves that dynamic allocation is a better memory management technique compared to static allocation. This outcome will be discussed in more detail in the next chapter.

MultiCom outperformed a bus-based system in its class by a factor of 14. Compared to a system using dual-port memory for interprocessor communication, the performance was boosted 11 fold, and the cost was reduced by using one shared memory. If multicasting were also considered, the boost in performance would be more than 16. MultiCom performed 4.2 times better in comparison with a hypothetical system using serial links.

The performance of the system can be greatly enhanced by using faster node processors and wider datapaths. The communication bandwidth of MultiCom can be upgraded to over 1 GBytes/s by using 32-bit processors running at 200MHz and fast memories.

Overall, the system shows a significant increase in performance over comparable multiprocessor architectures with serial links, dual-port memories, or bus-based systems. It also shows a significant cost reduction and a much simpler design compared to dual-port memory schemes.

CHAPTER 6

Simulation Modelling

Scaling of MultiCom cannot be easily performed within the limited resources of a research laboratory. Hence, the best way to evaluate larger systems would be the use of a simulator. In this chapter, the simulation model of MultiCom is introduced and is gradually expanded to cover a larger network. First the extension of the model to cover a large group is presented and scaling of the system under both static and dynamic allocations is verified. Then the expansion of the model to encompass a cluster of groups of nodes interconnected by a network controller is demonstrated and performance of the system is evaluated. A proper communication protocol was implemented in each simulation stage. The overloading of network controller is also explored and the cause of a communication bottleneck is investigated. Finally, the model for a larger network and the associated problems are discussed.

1 Simulation model

As a new framework for interprocessor communication, MultiCom showed that the communication between the nodes could be performed through a multiport memory with high throughput. Small systems are easy to implement and design; however, moderate or large-scale systems cannot be implemented within the limited resources of a research laboratory. Moreover, the design of larger systems requires significant financial support and well-organized teamwork. Hence, the only possible way to evaluate the structure on larger systems was the use of a simulation model.

MultiCom provided sufficient background to build a simulator. Observing the behaviour of nodes and their interaction with multiport memory in a working system was very valuable in building the structure of the model. As the first step in designing the model, it was decided to simulate MultiCom. This could be useful in two ways:

1. The basic structure of the model would be generated and verified by a working prototype.
2. Further expansion of the model would be relatively reliable, as it would be based on a verified model.

The simulation model created in this stage was modified to cover larger systems in three stages [Asgari+ 99b]. In the subsequent sections, the structure of the model and the simulation stages are explained and the achieved results are discussed.

2 Structure of simulation model

The simulation model incorporated several modules. The main modules were the node module, multiport memory module, and network controller module. Related modules such as group module and cluster module were derived from these main modules. Other minor modules were also used in the simulator. Because of the modular features of the model, an object-oriented approach was used and the simulations were carried out in C++. Only important modules are explained in the subsequent sections.

2.1 Node module

The simulation model of the nodes was based on the simulation of TMS320C50 processors. Critical tasks such as applying and acquiring the lock that were subject to memory contentions were simulated on a cycle-by-cycle basis. For non-critical tasks, instead of simulating every instruction, an abstract form was used whereby a group of instructions that performed a specific task was simulated as a single command that required several cycles. Checking receivers, allocating a buffer, initialising for transmission, transmitting data, generating interrupts, and various steps in receiving data in an interrupt service routine are examples of the available commands. Based on the characteristics of the TMS320C50 node processors and the observation of their behaviour in MultiCom, the timing of the commands and the number of cycles required for each one were defined in a configuration file. The activities of the nodes under the applied protocol were defined in a program file using these commands.

The clock rate of the nodes was fixed at 20 MHz, which was the same rate used in MultiCom. In fact, the clock rate does not affect the simulation model, but it defines the period of the simulation clock used in calculating the communication rate.

After defining the node module, a group module was created by instantiating the required number of nodes and integrating them with a multiport memory module, which is explained in the next section. A cluster could be generated by connecting several groups to a network controller module as explained later.

2.2 Multiport memory module

The module for multiport memory was designed by considering the structure and behaviour of IDT5054, which is the 4-port memory used in MultiCom. The module could be linked to the required number of node modules with each node capable of reading from or writing to the memory cells independently.

Apart from the normal activities of this module as a shared memory, a sub-module was also defined. All the signalling among the nodes could be performed through this sub-module. For example, generating an interrupt for a node was recorded in the sub-module and every node had to check it routinely to detect the incoming interrupts.

In both the memory module and the sub-module, provisions for simultaneous access of the nodes to critical data such as lock variables were provided. Lock variables could be accessed by more than one node at a time, and modifying their contents needed to take place only after termination of all the accesses in progress. By implementing this method, the write-write and write-read conflicts that may have occurred in queuing or acquiring the lock and in interrupt generation or reception were taken into account.

2.3 Network controller module

Similar to the nodes, network controllers were based on the TMS320C50 processors; however, the two modules were not identical. One of the differences between a node and a network controller is that a node only accesses one multiport memory, but a network controller needs to access more than one multiport memory in order to link different groups or clusters. As shown in the structure of Figure 3.4 on page 48, for linking eight groups of a cluster, each network controller is connected to eight different multiport memories. It is also connected to another multiport memory for linking its cluster to other clusters. The other difference is that the activities of a network controller acting as a link between nodes are different from the activities of a normal node working as a transmitter or a receiver in the network. Hence, a network controller required a separate program file and new commands were created for programming its tasks.

Consequently, the model of a network controller module was very similar to that of the node module with the provision of accessing several multiport memory modules arranged as a memory array. In addition, new commands were defined for its programming. With the capabilities of C++, it was possible to derive the node and network controller modules from the same class; however, for various reasons, a separate module was defined for each one.

2.4 Log file

The log file was not really a module. It was an output file generated by the simulation program and every activity in the network was recorded in it. As the simulation clock advanced, the activity and status of all the nodes and network controllers were recorded

in this file. These records were listed in chronological order and the status of the network at any stage could be checked by reviewing the recorded events. The log file was a valuable tool in debugging the simulation program, especially at the development phase, and was used for checking the validity of the communication between the nodes. In addition, conditions such as an insufficient number of buffers or an overloaded network controller could be detected by examining the records of the file.

3 Simulation stages

The simulations were carried out in four stages starting from the simulation of MultiCom to the simulation of the entire network of Figure 3.4. The stages were as follows:

- Simulation of MultiCom
- Simulation of a large group with more nodes
- Simulation of a cluster with several groups interconnected by a network controller
- Simulation of the entire network

Each stage will be discussed separately and the results will be presented in separate sections.

3.1 Simulation of MultiCom

Simulation of MultiCom was a good starting point in developing the framework for the simulation model. In this stage, the basic structure of the model was implemented according to the characteristics of the nodes and the multiport memory of MultiCom. The results were compared to the results of MultiCom, and the model was fine-tuned to produce identical results. This was performed to ensure that the simulation results would be more reliable in the expanded system.

The all-to-all test program used in MultiCom was used to test the model. As explained earlier, each node sent a message of 1792 words to all other nodes through the shared

memory and heavy traffic was generated. For testing buffer sizes bigger than the buffer size of MultiCom, the message size was also increased to maintain the heavy traffic.

In verifying the simulation model, every attempt was made to ensure that the model worked under the same conditions as in MultiCom. After checking the results obtained from the model, the timing of commands in the configuration file was adjusted so that the simulation results matched the results of MultiCom.

Both static and dynamic allocations were implemented on the simulation model. Each case is discussed in a separate section.

3.1.1 Static allocation

The simulation model was tested under static allocation and the results are plotted in Figure 6.1. The X-axis is almost logarithmic as, with few exceptions, for each entry the buffer size and the memory size are double the previous entry. (In fact, the buffer size is divided by two in decreasing X direction and rounded.) The shaded bars on the graph are almost identical with the results obtained from MultiCom in Chapter 5. The striped bars show that if the buffer size (hence the memory size) is increased, performance of the system will approach the peak rate and the overhead will drop to 3%. As explained earlier, the peak rate was calculated from the maximum rate that the nodes could write to and read from the multiport memory without any overhead.

As stated in the previous chapter, a large shared memory was not required for this communication scheme. Figure 6.1 confirms this statement. For the memory size of 4K words and under heavy traffic, the performance is close to the peak rate with 15% overhead. By increasing the shared memory size to 8K words, overhead drops to 8%. Additional increase of memory to 16K and 24K words drops the overhead to 4% and 3% respectively and the performance rise is not significant. A further memory increase has no effect and the overhead stays at a minimum value of 3%. It can be concluded that for the all-to-all test program, the system is able to operate with a shared memory as small as 4 or 8K words and can achieve near peak performance with reasonable overhead. Additional increase of memory will not boost the performance significantly. As explained in the next section, this is even more obvious in dynamic allocation.

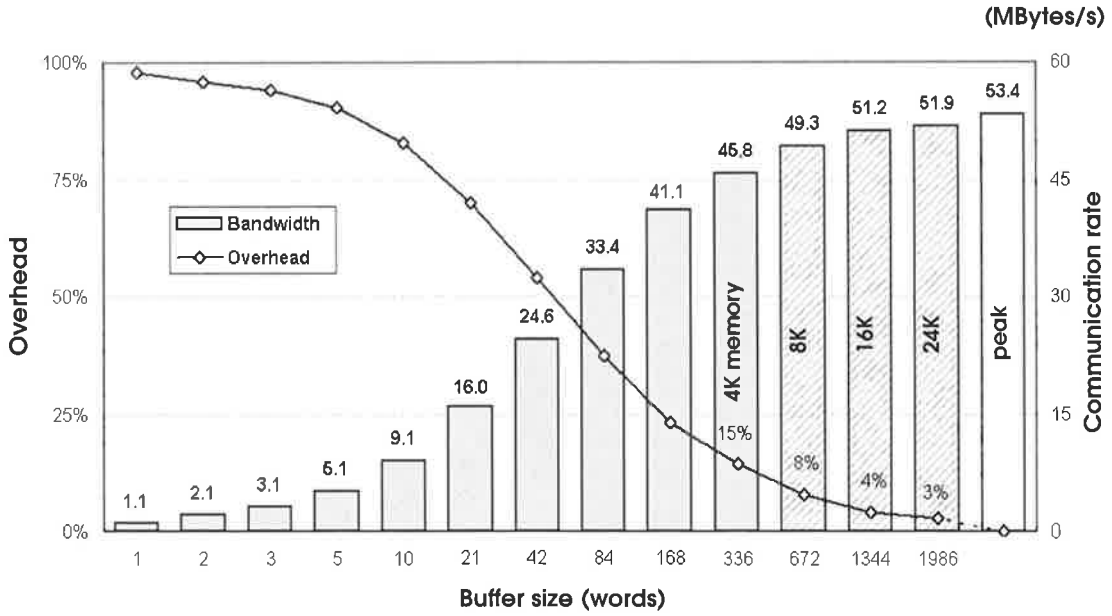

Figure 6.1 Simulation results for static allocation

The shaded bars match the results of MultiCom. As shown by the striped bars, increasing the buffer size increases the performance and drops the overhead, but the change is not very significant. Static allocation performs very well for small systems.

Moreover, performance gain should be compared to the cost of increasing the memory size. For example, increasing the memory size from 8K to 16K words will only increase the performance by 4%, which is too small to justify the associated increase in the cost of using bigger memories.

3.1.2 Dynamic allocation

The simulation model of MultiCom was tested with dynamic allocation. The results are shown in Figure 6.2. Similar to the static allocation, the shaded bars match the results of MultiCom. The striped bars were obtained by increasing the buffer size (hence the memory size). Increasing the memory size up to a point increases the performance slightly; however, for the memory sizes beyond 12K words, performance is fixed at 50 MBytes/s. The reason is that the software lock and the associated allocation process both have a serial nature. They impose extra overhead on the data transfer and limit the communication rate.

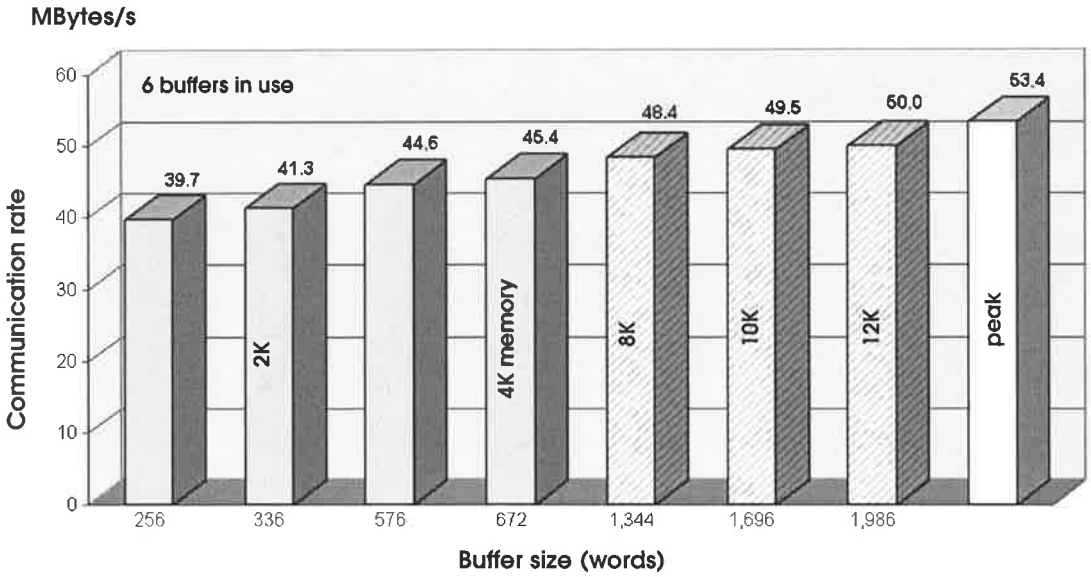

Figure 6.2 Simulation results for dynamic allocation

The shaded bars are almost identical to the results of MultiCom. By increasing the buffer size, the performance does not increase considerably because of the overhead of the lock mechanism and the buffer allocation process.

For the all-to-all test program, the results strongly confirm the low memory requirement of the communication structure. The figure shows that by increasing the memory size beyond 4K words, the gain in performance is very small. A memory size of 4K words is adequate for MultiCom and the negligible boost in performance achieved by doubling the memory size cannot be justified. The outcome is particularly significant because dynamic allocation is the memory management better suited for large systems. This point will be explained later in this chapter.

For a four-port system, the effect of memory size on both allocations can be better explained on a graph using linear axes. Unlike Figure 6.1 or Figure 6.2, the X-axis representing the memory size in Figure 6.3 is linear. Both static and dynamic allocations perform more or less the same. The figure shows that performance will not increase considerably if memory size is increased beyond 4 or 8K words. The saturation level is 51.9 MBytes/s for static allocation and 50.0 MBytes/s for dynamic allocation.

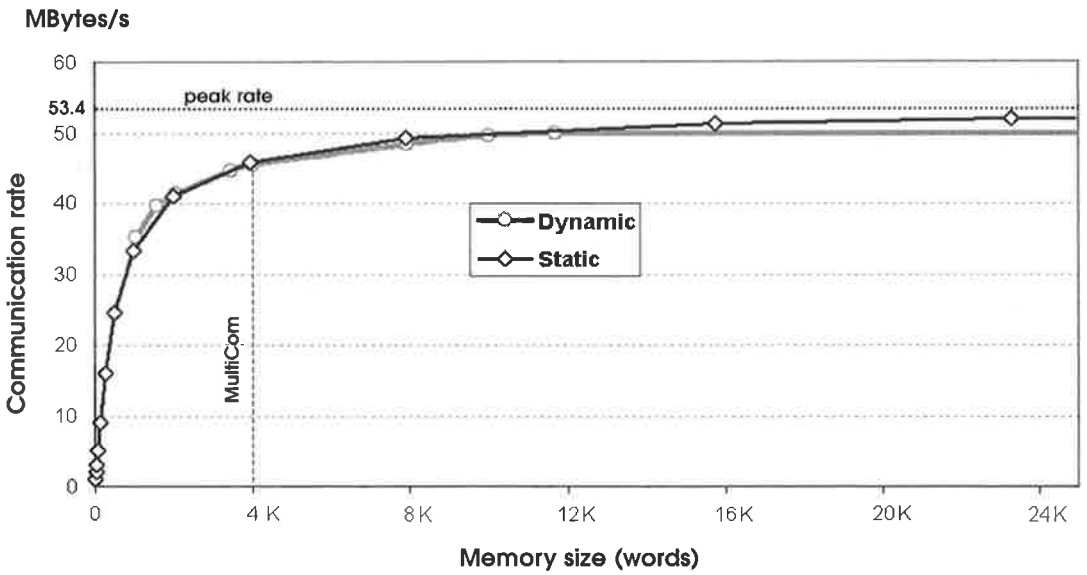

Figure 6.3 Effect of memory size in static and dynamic allocations

For four nodes, increasing the memory beyond 4 or 8K words does not affect the performance significantly. The 4K-word memory used in MultiCom is the best compromise in terms of cost/performance.

The slight difference is mainly because of the lock mechanism used in dynamic allocation.

The simulation model of MultiCom developed in this stage was expanded to cover more nodes as explained in the next section. As the model was verified by a physical system, the results of simulation for larger systems would be more reliable.

3.2 Simulation of a larger group

The simulation model was expanded to cover a larger group with more nodes connected directly to a multiport memory. A proper communication protocol as discussed in the previous chapter was implemented in this model. In this stage of the simulation, performance of the communication in a group was evaluated and the maximum number of buffers required in dynamic allocation was explored. In addition, the efficiencies of static and dynamic allocations in larger systems were compared.

Figure 6.4 displays the communication rates for different node counts under both allocations. With the fixed buffer size of 672 words in dynamic allocation, the size of shared memory was increased to accommodate all the required buffers. The figure shows that by increasing the number of nodes and ports, higher communication rates can be obtained and the dependency is almost linear. As an example, for eight nodes, the rate of 90.9 MBytes/s was achieved, which is double the rate for four nodes. The size of 8-port memory was 8K words compared to the 4K words of 4-port memory.

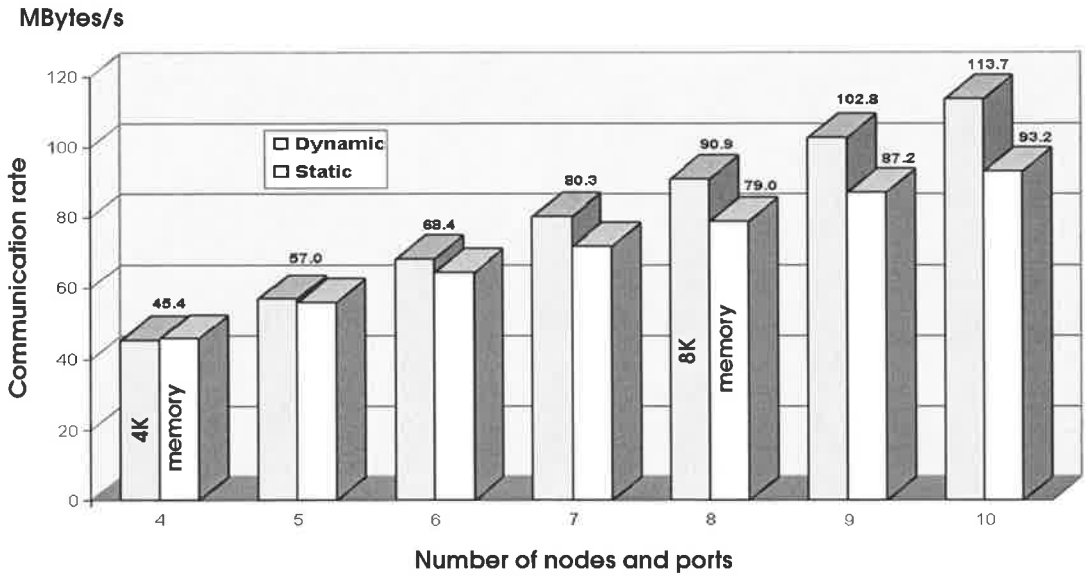

Figure 6.4 Scaling in a group

In dynamic allocation, by adding more nodes and increasing the memory size linearly, the communication rate increases linearly. However, with the same situation, static allocation does not scale very well and the performance drops for large node counts.

The figure also shows the performance of static allocation for increasing numbers of nodes. In each case, the memory size was equal to the size used in dynamic allocation. For N nodes, N(N-1) buffers were required and the buffer size was determined by dividing the memory size by the number of required buffers. The results show that by increasing the number of nodes the communication rate increases; however, its performance is lower than dynamic allocation. In fact, dynamic allocation scales linearly but static allocation does not.

For dynamic allocation, a test program was used to explore the dependency of the required number of buffers on the number of nodes. In this test, enough buffers were available and the buffer size was fixed at 672 words. The number of allocations performed on each buffer was recorded during the test run. The number of buffers used in the communication was obtained from the recorded data. As shown in Figure 6.5, for N nodes, the number of buffers actively used in the communication was approximately 1.6N. If more buffers were available, they would not be used [Asgari+ 99b].

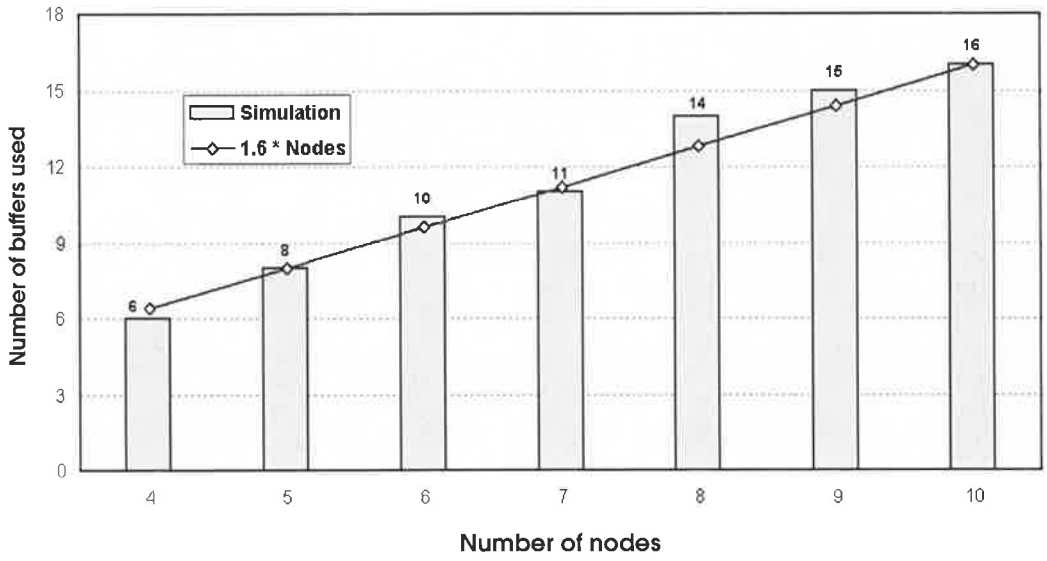

Figure 6.5 Number of buffers used in dynamic allocation

The maximum number of nodes required in dynamic allocation is approximately 1.6 times the number of nodes. If more buffers are available, they will not be used.

The difference between performance of static and dynamic allocations was explored further on the simulator. With the size of multiport memory fixed at 4K words, performance of the system for different numbers of nodes was determined under both allocations. As anticipated, static allocation quickly falls behind the dynamic allocation. The reason is that the memory is divided into N(N-1) buffers in static allocation, while a maximum of 1.6N buffers is required for dynamic allocation. Hence, if static allocation is used in a system with fixed memory size, increasing the number of nodes does not increase the communication rate considerably as the buffer size is reduced by N^2. On the other hand, under similar conditions for dynamic allocation, increasing the

number of ports increases the communication rate, as the buffer size is only reduced by 1.6N. Figure 6.6 compares the communication rates for static and dynamic allocations for the fixed memory size of 4K words. The figure shows that performance of dynamic allocation rises as the number of nodes increases. Since the memory size is fixed at 4K words, the communication rate does not increase linearly as in Figure 6.4. On the other hand, performance of static allocation flattens at around eight nodes, and even drops slightly for 10 nodes, achieving only 67% of the performance of dynamic allocation.

The results obtained in this stage of simulation confirm two important features of the communication structure:

1. Dynamic allocation performs better than static allocation.
2. The structure requires only a small amount of shared memory for interprocessor communication.

It is worth mentioning that unlike other shared memory systems, in this structure, the shared memory is only used for message transfer, and a small memory size is adequate for this purpose.

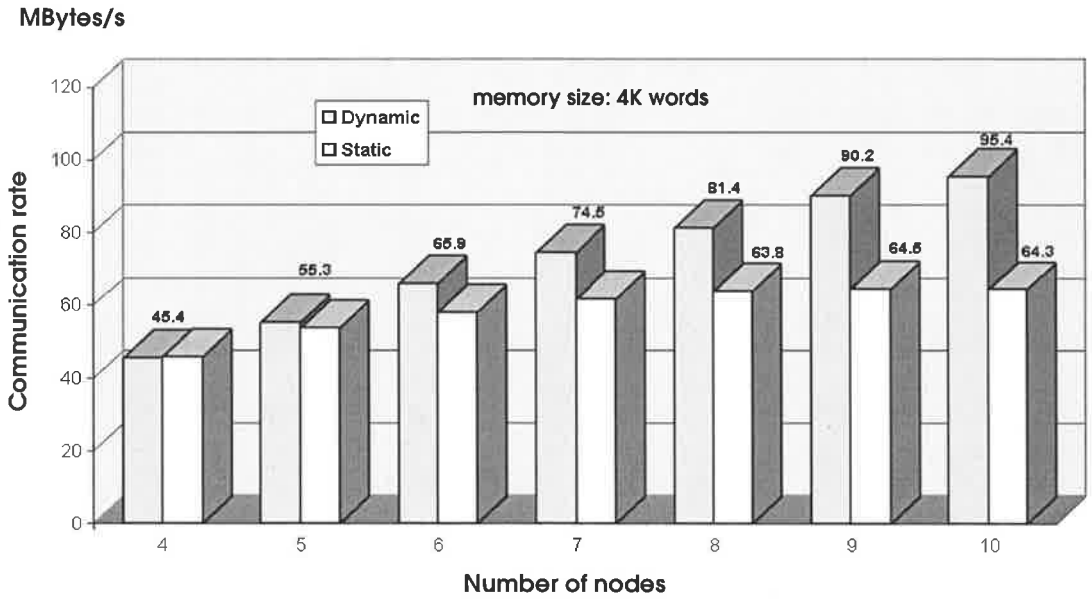

Figure 6.6 Static and dynamic allocations with fixed memory size

For a fixed shared memory size of 4K words, increasing the number of nodes in dynamic allocation increases the communication rate considerably, but in static allocation, the communication rate saturates. This result confirms that the memory management of dynamic allocation is better than static allocation.

Because of the better performance of dynamic allocation and its capability for implementing multicasting/broadcasting, only this method was used throughout the rest of the simulations.

3.3 Simulation of a cluster of groups

In this stage, the simulation model was further expanded to encompass a cluster of groups. Several groups can be interconnected in a cluster using a network controller (NC). As shown in Figure 3.3 on page 46, a network controller is connected to the multiport memory of each group through an extra port and acts as an intermediate node for inter-group communication. Before transmitting data, if a node discovers that the receiver is in another group, it sends the message to the NC. The data received by the NC can be transferred to an internal buffer, or it can be kept in the multiport memory. When the end receiver is ready, the NC transfers the message to the appropriate multiport memory, where the receiver can collect it.

The simulation model was based on a structure using eight groups, each comprising of eight nodes. Each node was programmed to send a message of 1792 words to each of the other seven nodes using dynamic allocation. The receiving nodes could be in the same group or in other groups. The buffer size was 672 words and 14 buffers were available in each 9-port memory module.

There are two types of communication in this model. One type is the direct communication within a group, which is fast and takes very short time. The other type is the inter-group communication that takes longer to finish because it must go through the NC and requires two hops. As there is only one NC with limited communication capacity, if too many inter-group messages are to be sent, the NC can be overloaded and long delays will be expected. Several different simulations were performed to determine the effects of sending messages within groups, and between groups of a cluster.

Considering these points, the communication rate is no longer constant in this model. The reason is that the nodes that communicate within a group finish early and become idle, while some nodes are still communicating with the NC. In fact, there are two

different communication rates in this case. The first one is the communication rate between the nodes within a group, plus the rate between the NC and the end receivers. The second one is merely the rate of communication between the NC and some nodes as the end receivers. In order to measure these rates, the simulator was programmed to compute the total data communicated in the entire system within fixed time intervals. By dividing this result by the duration of the interval, the communication rate in the interval was calculated. It is worth mentioning that only the data received by the end receivers was included in the total communication measurements, and the intermediate communications such as transferring data from a node to the NC were not taken into account.

The interval size was set to 4000 simulation clocks, equivalent to 200 μs. The simulation model was programmed to generate an output file in which the time, the cumulative communicated data, and the communication rate in each interval were recorded. The data in this file can be plotted versus time to achieve the graphs of the total communication and the communication rate. A closer look reveals that the communication rate is in fact the slope of the total communication and each one can be derived from the other one using integration or differentiation. However, as each graph carries different visual information, both the total communication and the communication rate are included in the figures.

3.3.1 Communication protocol in a cluster

The communication protocol needed some modifications to include the requirements of a cluster. The protocol for the nodes required minor changes, but a new section was added for the tasks of a network controller. The modified protocol for the nodes was derived from the communication protocol for dynamic allocation as shown in Figure 6.7. As a transmitter, the node checks the end receiver. If it is in the same group, the node proceeds as before, otherwise, it sets the NC as the receiver of the message. A new header, as explained in the next section, was used to identify the original sender and the end receiver for the messages distributed by the NC. As a receiver, each node checks the source of received interrupt and if it is from the NC, it refers to the header of the message to identify the original sender. Only the outline of the protocol is shown in this figure and the details are omitted.

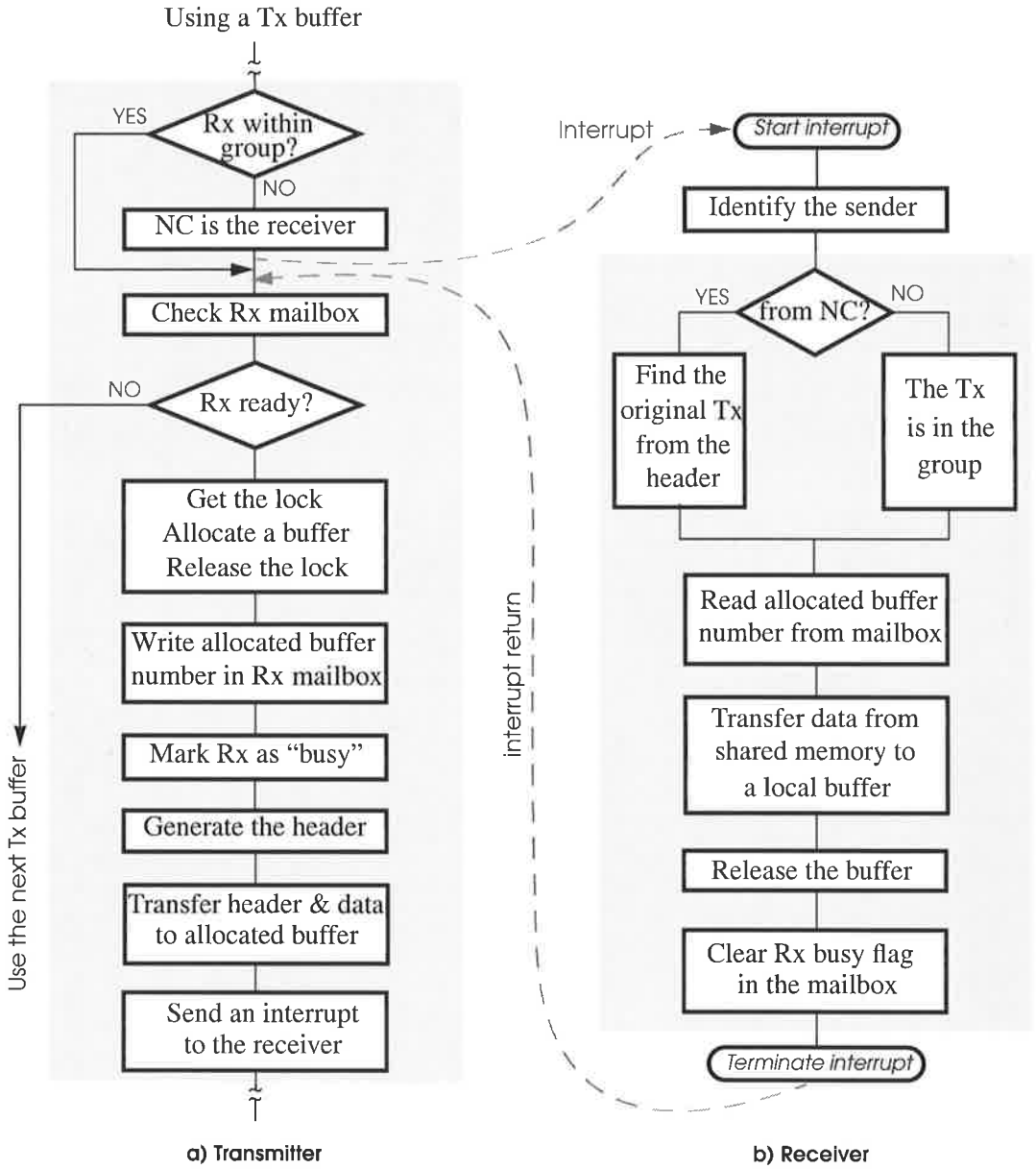

Figure 6.7 Communication protocol for a cluster

a) A transmitter node sends the message to the NC if the receiver is in another group. **b)** A receiver node finds the original sender from the header if the message has been distributed by the NC.

A simplified version of the tasks of a network controller is illustrated in Figure 6.8. After receiving an interrupt, the NC determines the sender and refers to the appropriate buffer in the shared memory of the sender. It creates a transmit buffer and transfers the header information along with other useful data into the buffer. As a transmitter, if there

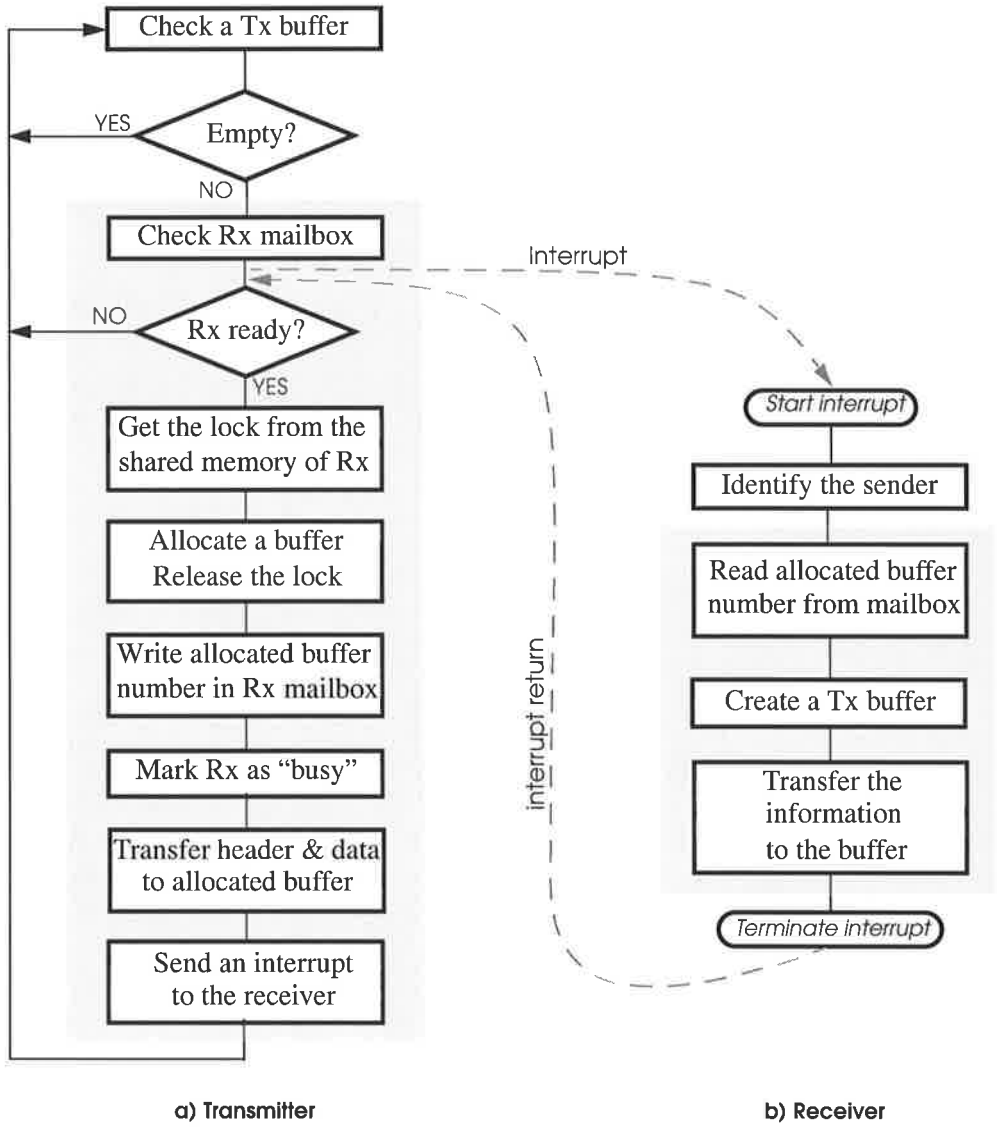

Figure 6.8 Communication protocol for the NC of a cluster

a) As a transmitter, the NC gets a buffer from the shared memory of the end receiver and transfers the message to it. **b)** As a receiver, the NC creates a buffer for the incoming message and transfers the information to the buffer.

is an active transmit buffer, the NC checks the end receiver. If it is ready, the NC applies to get the lock from the shared memory of the receiver. After obtaining the lock, the NC allocates a buffer, transfers the header and the data to the allocated buffer, and sends an interrupt to the receiver.

3.3.2 Header of a packet

The ID of each node represents its group and node numbers. The header of a packet contains the following information:

- sender's ID
- receiver's ID
- size of data in the packet
- a flag for the last packet of the message

If the message is long, it is sent in several packets. A flag indicates that the current packet is the last packet of the message. Each receiver checks the header and generates an error message if it finds any error.

3.3.3 Results

The model of a cluster was tested under the conditions explained in the previous sections. Figure 6.9 shows three different cases. In case (a), all the nodes were communicating within their groups. In total, 1.61 MBytes was transmitted in around 2.2 ms and the average communication rate was 0.73 GBytes/s. This rate is the slope of the dashed line in the figure. Each 8-node group contributed 91 MBytes/s to this rate, consistent with the results of Figure 6.4. The graph of the communication rate as the slope of the total communication shows a rate of 0.78 GBytes/s. This rate is slightly higher than the average communication rate, because it excludes the initial delay in the start of the communication. In case (b), one of the receivers of only one node in each group was in another group and the message was delivered by the NC. Case (c) was similar to (b) with two receivers of a node located in other groups. In both cases, the majority of the messages were transmitted within the groups with the rate of 0.77 and 0.76 GBytes/s respectively, slightly less than the rate in case (a). The inter-group messages took longer to go through the NC for another hop. When the communication within the groups terminated, the communication rate dropped considerably indicating that the remaining communication had to be handled by the NC. In case (c), it took longer for the NC to distribute the inter-group messages because of its limited communication bandwidth. For the transfer of larger inter-group messages in a cluster,

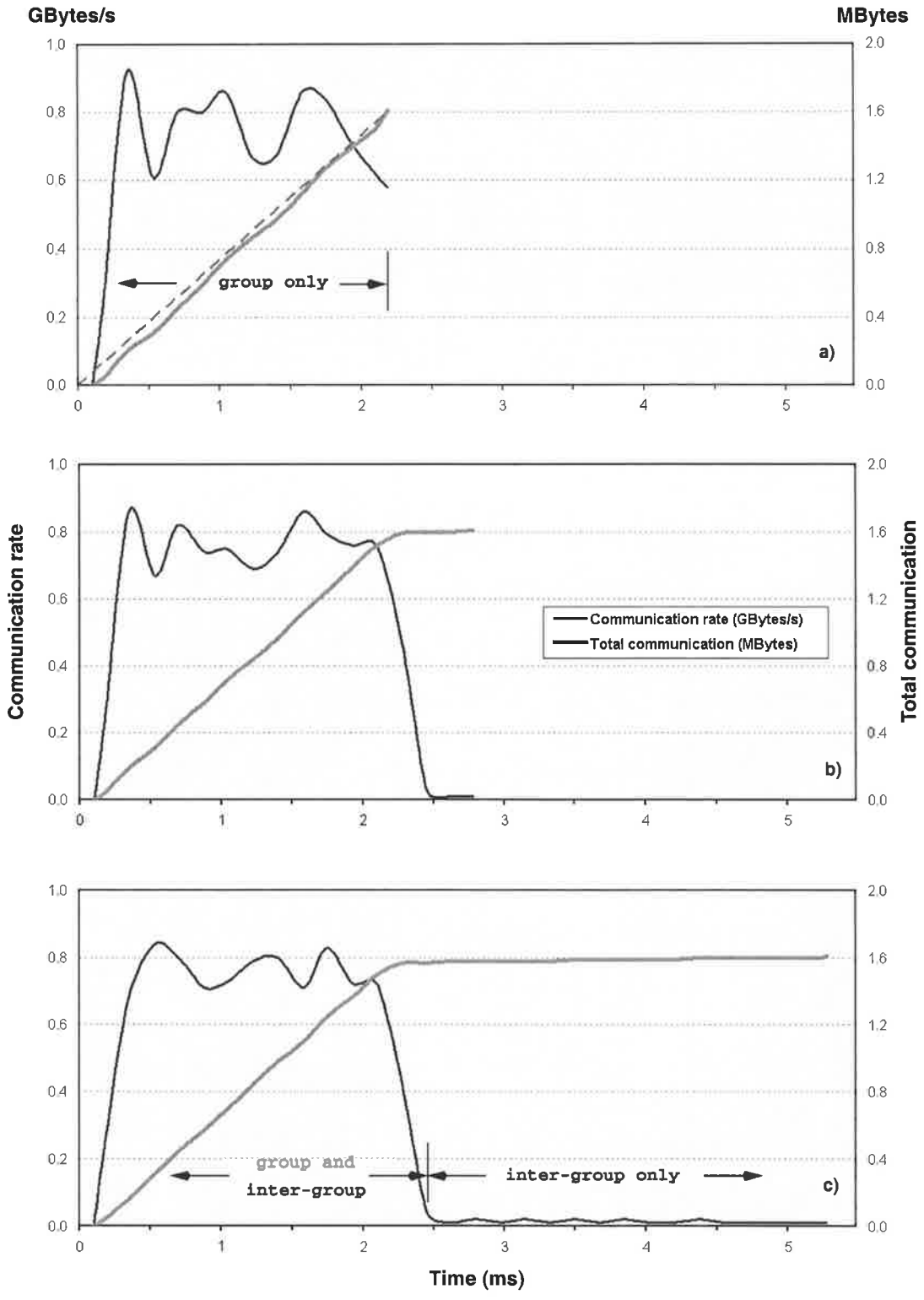

Figure 6.9 Communication of groups in a cluster

a) Communication within groups only.

b) One of the receivers of a node in each group was in another group.

c) Two of the receivers of a node in each group were in other groups. It took longer for the inter-group messages to be transmitted.

the only available NC can be overloaded and longer delays will be expected, resulting in performance loss.

It is worth mentioning that the graphs show only the effective communication. This denotes that only the messages delivered to the end receivers were considered and intermediate messages received by the NC were excluded from the measurements. This point can be observed from the initial rise of the communication rate graph. In case (c), the initial rise is slightly flattened compared to case (a) because of sending some of the messages to the NC.

The results show that the structure of a cluster could suffer from a communication bottleneck if the NC is overloaded. This is caused by using only one NC that has a limited transfer capacity. In order to investigate overloading of the NC further, two other tests were conducted on the model. The results are plotted in Figure 6.10. In case (a), four nodes in each group had a receiver in other groups. Under this condition, the load of the NC was increased to $1/14^{th}$ of the total messages. The graph of total communication clearly displays two different slopes. One is 0.72 GBytes/s showing the rate of delivery of messages by both the nodes and the NC. The other one is 11. 5 MBytes/s, which shows the rate of delivery by the NC only. This is consistent with the communication bandwidth of the NC, which like the other nodes is 13.3 MBytes/s. The test shows that the NC is working effectively, but the bandwidth it provides is not enough for large inter-group transfers.

In case (b), the NC was further overloaded by doubling its load. Every node in each group had a receiver in a different group. The NC was responsible for delivering one out of seven messages of a node. The messages within the group were delivered at the rate of 0.56 GBytes/s. The rate had dropped considerably compared to case (a) partly because a portion of the available bandwidth of the nodes was used to transmit to the NC and had not been included in the measurements. The other reason is the effect of an overloaded NC and will be explained in the discussion section. It can be also observed that the communication of nodes within the group has taken slightly longer than case (a). This is also caused by an overloaded NC and will be explained later.

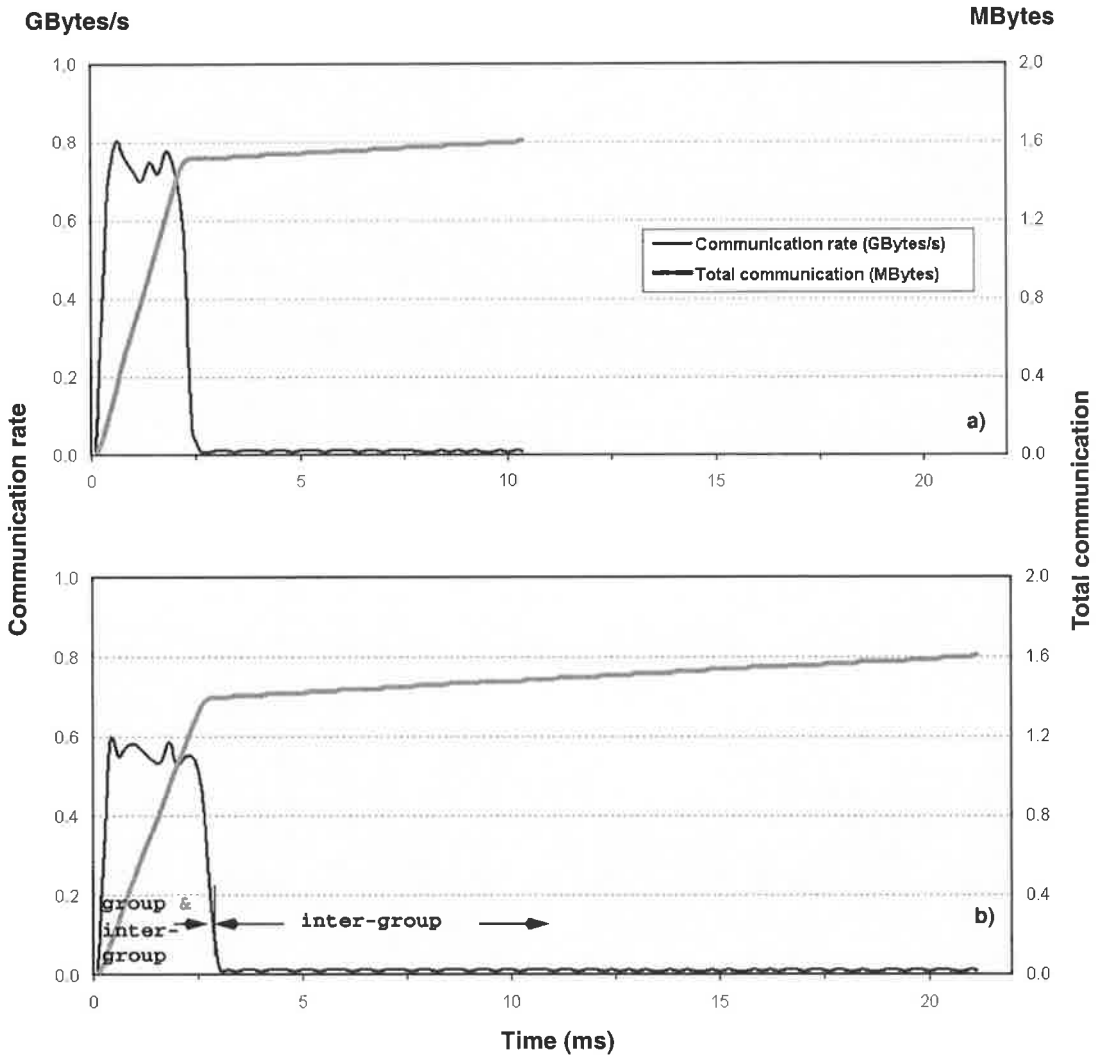

Figure 6.10 Overloading effect of the NC in a cluster

a) Four nodes in each group had a receiver in other groups. In this case, $1/14^{th}$ of the total messages were delivered by the NC.

b) Every node in each group had a receiver in other groups. In this case, $1/7^{th}$ of the messages were delivered by the NC. The effect of an overloaded NC can be observed as rate drop, longer time to deliver the messages within groups, and very long time to deliver the inter-group messages.

The simulation results show that the structure is only suitable for a light inter-group traffic. Heavy traffic will overload the NC and performance will drop considerably. In order to get an acceptable performance, the structure needs to be improved and the bottleneck must be removed. An improved structure will be presented and evaluated in the next chapter.

3.4 Simulation model for the entire network

In the network structure of Figure 3.4 on page 48, the clusters were interconnected by sharing a multiport memory among the network controllers. For this structure, each node is defined by its cluster number, group number, and node number. A message can be sent directly to a node within a group. If the receiver is not in the same group, the message should be sent to the NC. The NC checks the cluster number of the receiver. If it is in the same cluster, the NC sends the message to the appropriate group; otherwise, it sends the message to the NC of the cluster where the end receiver is located.

In this structure, both inter-group and inter-cluster messages should be handled by the network controllers. The simulation model of a cluster in the previous section indicated that a network controller could be overloaded by large inter-group messages. Interconnecting clusters using the same network controllers used for connecting the groups increases the load of the network controllers, resulting in a poor performance for the network. Although this stage of simulation was implemented and preliminary results were presented in [Asgari+ 99b], it will not be discussed here until the structure of a cluster as its sub-model is improved. The improved communication structure is presented in the next chapter.

4 Discussion

The simulation model for MultiCom was a credible and simple start for a complex model. The results of this model showed that increasing the memory size could increase performance of the system. For very large memories, performance of static allocation was slightly better than dynamic allocation because of the lower overhead. However, the increase of memory size beyond 4 or 8K words was not advantageous in terms of cost/performance.

The results of group communication and the effect of overloaded NC are discussed in the following sections.

4.1 Group structure

The expansion of the model to cover a larger group showed that in the range of up to 10 nodes dictated by the limited number of ports on multiport memories, a system can scale linearly under dynamic allocation; however, performance drops if static allocation is used. Given that dynamic allocation is also capable of performing multicasting/ broadcasting, it was selected as the memory management for complex models. The results also confirmed the small shared memory requirement of the structure and showed that for an 8-node group, good performance was achievable even with a memory size as small as 4K words.

The structure of a group can achieve higher performance compared to serial links used in hypercubes. With similar assumptions and calculations as stated in Chapter 4, the all-to-all communication for an 8-node system interconnected using a hypercube of order-3, would require at least five time slots if no overhead is assumed. For this system, a calculated aggregate communication rate of 20.4 MBytes/s (i.e. 56x1792x2 bytes in 5x1971.2 μs) could be achieved. If the system were implemented using multiport memories, the practical communication rate would be 91 MBytes/s, showing a performance increase of 4.5 times. If a practical rate rather than the calculated rate for serial communication were considered, performance boost would be even higher.

The structure also shows a large reduction in the number of memory elements and a significant increase in performance compared to a system using dual-port memories. If the cluster of eight nodes were to be implemented with dual-port memories similar to the system explained in [Campbell+ 96], it would require 20 modules of dual-port memory, instead of one multiport memory. Even with this many DPMs, the communication between some nodes would not be direct and would require the use of at least one intermediate node [Asgari+ 01]. This would decrease the performance of the DPM system considerably compared to a system communicating in one hop using one multiport memory.

4.2 Overloaded network controller

The simulation model of a cluster showed that the NC could be overloaded if the number of inter-group messages was high. An overloaded NC will result in longer delays in delivering the messages to the end receivers. In addition, it can reduce the communication rate within the groups by using most of the resources in the system. This effect can be observed in Figure 6.10-b. Compared to case (a), there were fewer messages to be transferred within each group, but the group communication was performed at a lower rate and was completed in a longer time. The lower rate was partly because the nodes had to send some of the messages to the NC, which was not included in the total communication until delivered to the end receivers. With fewer inter-group messages, the group communication was expected to finish sooner, or at least at the same time of case (a); however, it took longer. The cause for this delay was not very clear until the log file was examined. The records of the log file revealed that most of the nodes had allocated a buffer to send a packet of data to the NC, and had transferred the data to the buffer, but the messages were not collected by the NC for some time. As the NC could not handle all of the requests within a short time because of its limited communication bandwidth, the allocated buffers were not released. Hence, there were fewer buffers available for group messages and it took longer for them to be transmitted.

In order to verify this point, another experiment with the same conditions but with more buffers was tested. The results showed that the communication within groups was completed within the time and almost at the same rate of case (a). This experiment showed that an overloaded network could create a bottleneck in the communication by delaying the delivery of the inter-group messages. It could also prolong other activities in the network by not releasing some of the available buffers. An improved structure for a cluster is required to overcome this bottleneck.

Moreover, inter-cluster messages in a network increase the load of NCs. The performance of a network can be highly degraded because of overloaded NCs. A new network structure based on the improved cluster structure will be introduced in the next chapter.

5 Conclusion

In this chapter, a simulation model was developed to evaluate the efficiency of the proposed communication structure on larger systems. Initially, the model of MultiCom was created and its functionality and timing were verified against the results from MultiCom. The model confirmed that the structure does not require a large shared memory for its operation, because shared memory is only used for communication purposes.

Expanding the model to higher numbers of nodes showed that the system could scale linearly if dynamic allocation were used. On the other hand, static allocation was suitable for small systems, but its performance dropped for expanded systems. The main reason was that in static allocation the number of required buffers increased as N^2, whereas in dynamic allocation it increased as 1.6N. Considering the capability of implementing multicasting/broadcasting, and in spite of its lock mechanism, dynamic allocation proved to be a better solution for the memory management of larger systems.

A system of eight nodes connected in a group can perform at least 4.5 times better than a hypothetical system interconnected in an 8-cube structure by serial links running at comparable speed. A wider datapath could improve the communication bandwidth considerably. In addition, increasing the speed of the nodes and the multiport memories would also increase the performance.

Compared to a system using dual-port memories, the 8-node system requires only one multiport memory instead of 20 blocks of dual-port memories. This is a significant reduction in the cost and complexity of the system, and a considerable increase in performance because of direct communication.

A cluster is a group of nodes interconnected using a network controller. The inter-group messages should be transmitted in two hops through the NC. The results obtained from the simulator showed that the current cluster structure was only useful if the number of inter-group communications was low. Because of its limited communication bandwidth, a network controller could be overloaded as the number of inter-group messages was increased. An overloaded NC would create a communication bottleneck that could

reduce the performance significantly. The cluster structure will be modified in the next chapter and similar improvement will be applied to the network structure.

Overall, communication using multiport memory is very efficient for small systems. The structure is simple to implement and requires a very small shared memory. It scales linearly achieving a substantial increase in performance. It also enjoys a big reduction in the system cost by minimizing the number of required components. The efficiency of medium and large systems will be discussed after evaluating the improved structure in the next chapter.

CHAPTER 7

Improved Communication Structure

Evaluation of the simulation model in the previous chapter showed that if the number of messages transferred between groups was high, the NC of a cluster could be overloaded. In this chapter, first an improved structure for a cluster is presented which can reduce the overloading considerably. Then, a modified network structure for using the improved clusters is presented. Evaluation of the modified network shows that the configuration is not suitable yet, as inter-cluster messages can still overload the NCs. Hence, an improved network structure is proposed that can handle the inter-cluster messages with similar efficiency of the communication within clusters. Finally, the effect of an extra hop required for some inter-cluster messages is explored and scaling of the system is discussed.

1 Improved cluster structure

In the cluster model used in the previous chapter, all of the inter-group messages in a cluster were handled by a single NC. As the NC had limited communication capacity, it could be easily overloaded. One solution to reduce overloading would be to distribute the load among several NCs rather than exhausting a single NC under heavy traffic. Based on this idea, an improved structure for a cluster is presented in Figure 7.1. In this structure, each group is connected to a group-NC and several groups are interconnected using a cluster-MPM. If the receiver is in a different group, the message is sent to the group-NC. Then, it is transferred to the NC of the receiver group through the cluster-MPM, and finally it is delivered to the end receiver.

In this structure, an inter-group message requires three hops. This may increase the latency of an individual message; however, the transmission of several inter-group messages would be much faster. In order to reduce latency and to increase the cluster

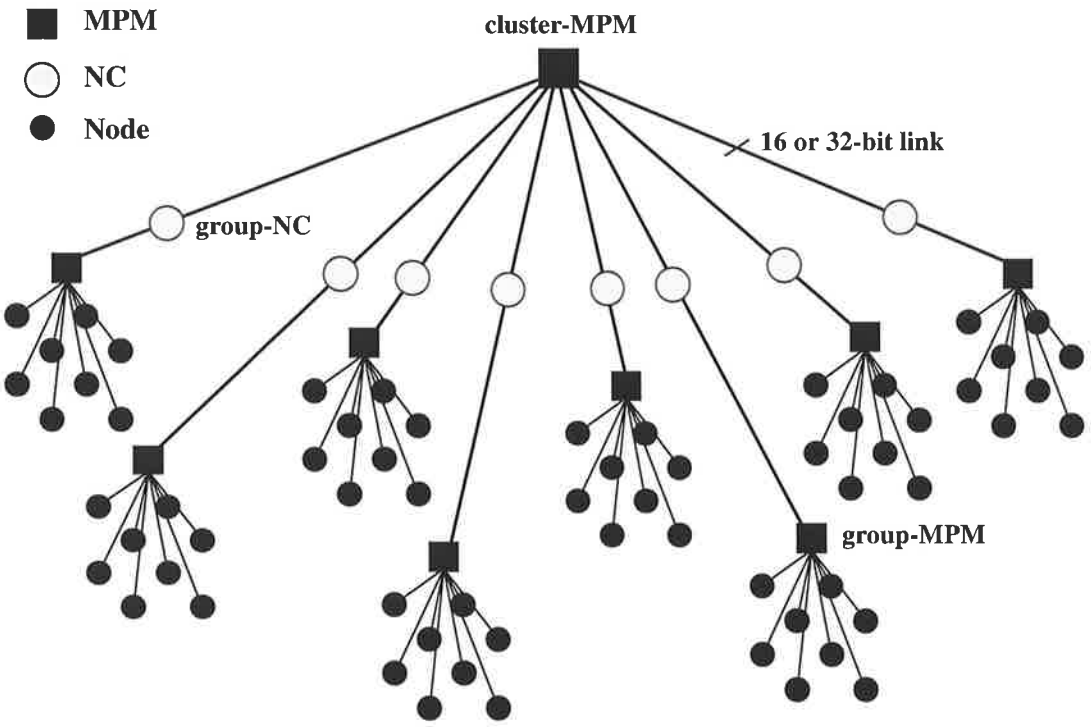

Figure 7.1 Improved cluster structure

The communication load of a cluster is distributed over several NCs. In this structure, an inter-group message is sent to the group-NC by the transmitter node. Then, it is transferred to the NC of the receiver group through the cluster-MPM for delivery to the end receiver.

performance, the NCs were upgraded to operate at twice the speed of node processors. As a network controller mainly handles memory transfers, a cut-down processor specifically designed for network requirements could be used for this purpose. The memories connected to NCs should be capable of working at the increased speed.

The structure was implemented in the simulator and was tested under different loading conditions. Figure 7.2 illustrates the performance of the system under the same conditions of Figure 6.10. In case (a), four nodes in each group had a receiver in another

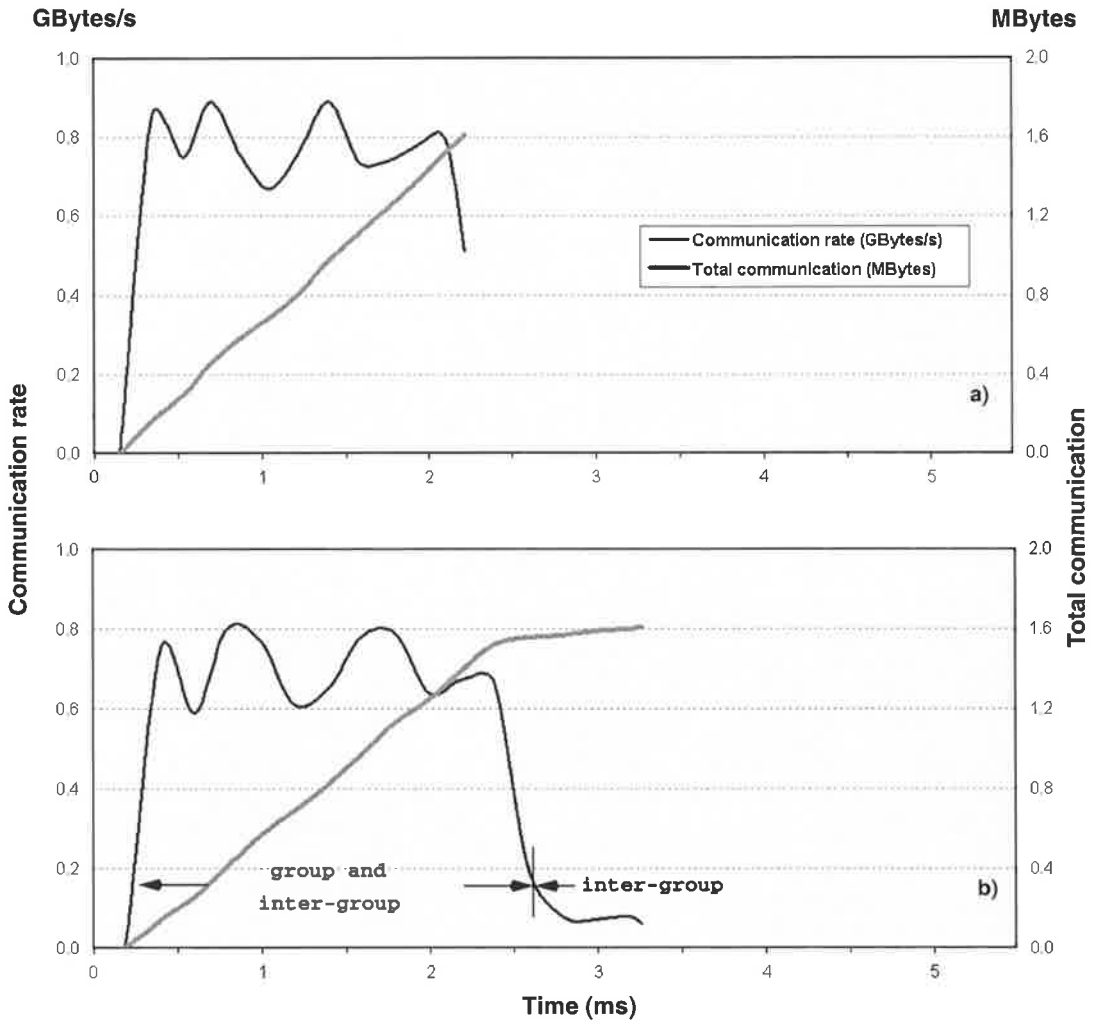

Figure 7.2 Communication of groups in improved cluster

a) Four nodes in each group had a receiver in another group. In this case, $1/14^{th}$ of the total messages were delivered by the NCs.

b) Every node in each group had a receiver in another group. In this case, $1/7^{th}$ of the messages were inter-group type.

In contrast to Figure 6.10 on page 136, the NCs were not overloaded in the modified cluster and the messages were delivered much faster.

group, requiring $1/14^{th}$ of the total messages to be delivered by NCs. As shown in the figure, the communication was performed in almost the same time of the communication within groups only (refer to Figure 6.9-a on page 134). In addition, unlike Figure 6.10-a on page 136, the inter-group communication did not increase the overall communication time. In case (b), each node in every group had a receiver in another group, requiring $1/7^{th}$ of the messages to be delivered by the NCs. The communication was performed six to seven times faster than Figure 6.10-b, and no overloading effect was observed.

These results show that the improved cluster structure can considerably reduce overloading of NCs. For further evaluation of this structure, the model was tested with more inter-group transfers and Figure 7.3 presents the results. In case (a), the number of inter-group messages was doubled and $2/7^{th}$ of the total messages were handled by the NCs. In case (b) and (c) the load of the NCs was increased to $3/7^{th}$ and $4/7^{th}$ of the total messages respectively. The results show that in all cases, the transfers were performed in a reasonable amount of time, without excessive overloading.

As previously explained in Chapter 6, the graph of the total communication shows two different slopes. One is around 650 to 720 MBytes/s, which is mostly the message delivery rate by nodes. The other one is 75 MBytes/s, which is the rate of message delivery by NCs only. The NC delivery rate shows an improvement of six to seven times compared to a structure using a single NC.

These results show a simple but effective way for estimating the time required for the transfer of total messages. In a similar communication pattern, the messages can be divided into two categories: within groups and inter-groups. For each category, the transfer time can be calculated using the appropriate rate for the category. The time required for the total communication is approximately equal to the longer calculated time. The validity of this method was verified by all the cases shown in Figure 7.2, Figure 7.3, and some other cases not shown.

It can be concluded that the performance of a cluster depends on two types of transfers. If most of the messages are transferred within groups, the system will show high

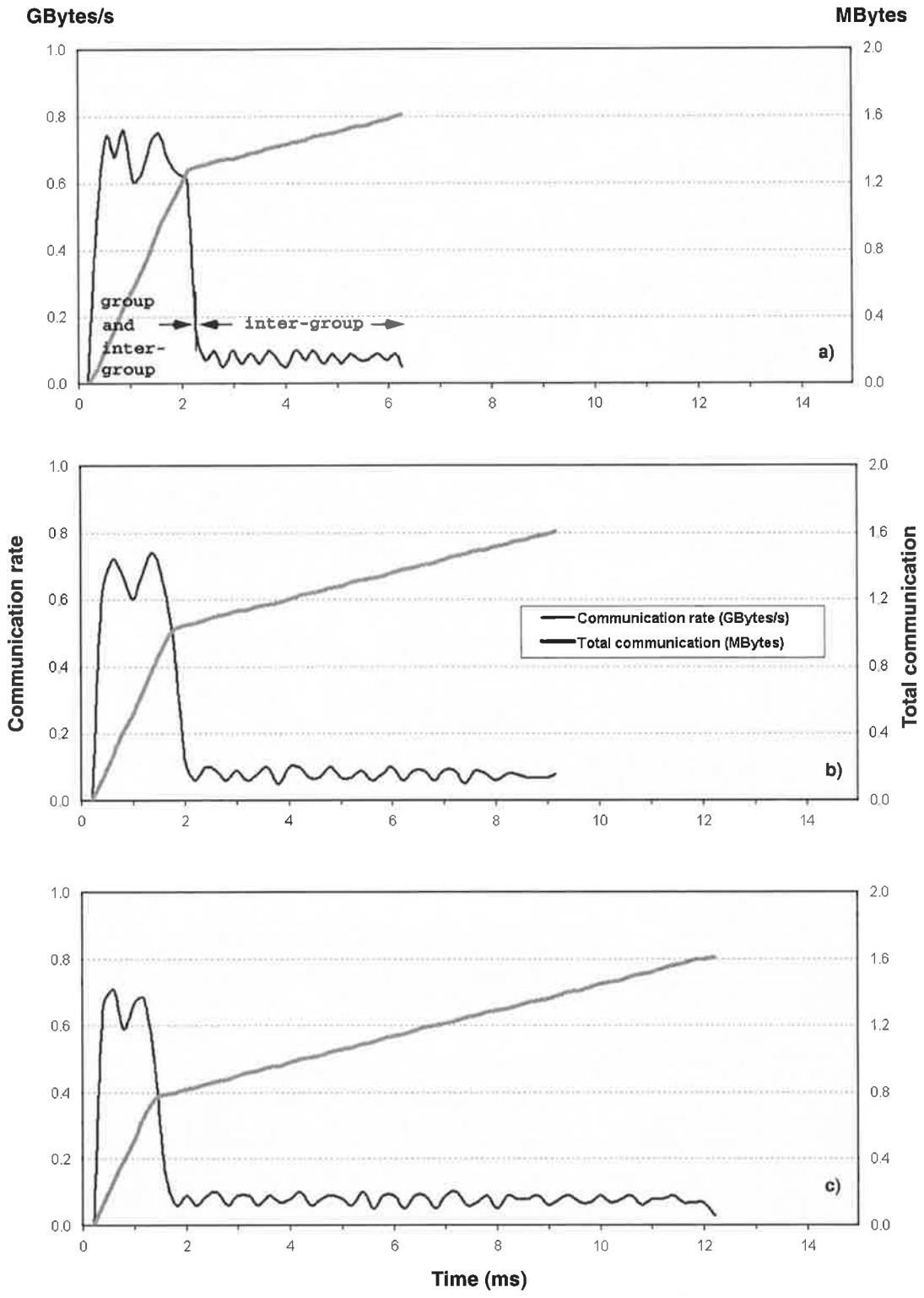

Figure 7.3 More inter-group communications in improved cluster

a) Every node in each group had two receivers in other groups. Inter-group messages were $2/7^{th}$ of the total messages.

In **b)** and **c)** $3/7^{th}$ and $4/7^{th}$ of the total messages were handled by NCs, respectively.

throughput because of the direct communication between nodes. On the other hand, if most of the communication is of inter-group type, the communication will have lower throughput, as these messages will be handled by the NCs and will require three hops.

2 Modified network structure

For interconnection of clusters in a network, each cluster requires at least one extra link. In the structure of the improved cluster, no extension link is available for further

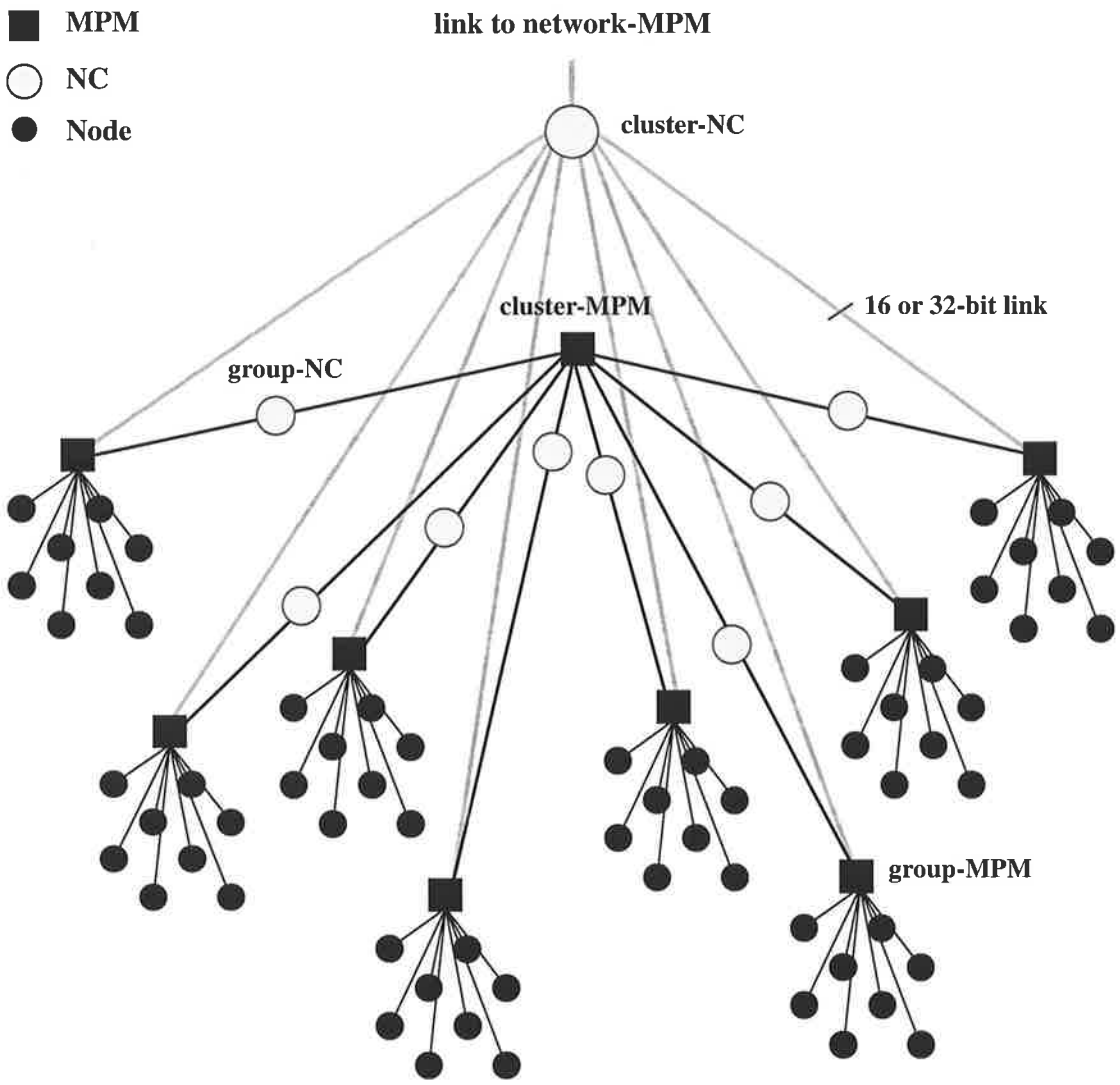

Figure 7.4 Modified cluster for a network

Adding an extra NC to the cluster structure can provide the required link for the interconnection of the clusters in a network. To achieve the modified network, this structure should replace the cluster structure in Figure 3.4 on page 48.

expansion. One possible method for creating such a link is to add an extra NC to the cluster structure. As shown in Figure 7.4, a cluster-NC is connected directly to the MPM of each group. It uses an extra link for connection to other clusters. In a network structure as shown in Figure 3.4 on page 48, the clusters are interconnected through a network-MPM that is shared among different cluster-NCs. The modified network structure can be obtained by substituting the old clusters with the modified ones.

In the modified network structure, the inter-group messages are handled by the group-NCs. Similarly, inter-cluster messages are handled by the cluster-NCs, as explained in Chapter 3. The advantage of this structure over the previous structure of Figure 3.4 is that in the new structure the cluster-NCs only handle the communication between the clusters and they are not overloaded by the inter-group communication. Although the performance of the new structure will improve very much, the limited bandwidth of the cluster-NCs can be still a problem and overloading of NCs may be observed if the number of inter-cluster messages is high. The drawback of the structure is the need for an extra port on the memory, which adds up to 10 ports in total.

The simulation model was upgraded to simulate the modified network with the new cluster structure. Figure 7.5 illustrates the system performance under inter-cluster communication. In case (a), two nodes in each group had a receiver in another cluster. The aggregate communication rate within the groups was 6 GBytes/s, which was 0.7 GBytes/s for each of the eight available clusters. However, the transfer of inter-cluster messages took longer to go through the cluster-NCs. In case (b), the load of cluster-NCs was doubled to handle $1/14^{th}$ of the messages. Consequently, the time spent on communication was doubled. In both cases, the communication rate for inter-cluster messages was around 70 MBytes/s, which was very low for a network of this size. The overloading of the very few available cluster-NCs was the main cause of performance loss. These results show that the model still requires more modifications.

The model was also tested in presence of a large number of inter-group messages. In addition to the same conditions of Figure 7.5, three out of seven messages in each group were directed to other groups within the same cluster. As there was very little interaction between the inter-cluster and inter-group messages, the results of Figure 7.6

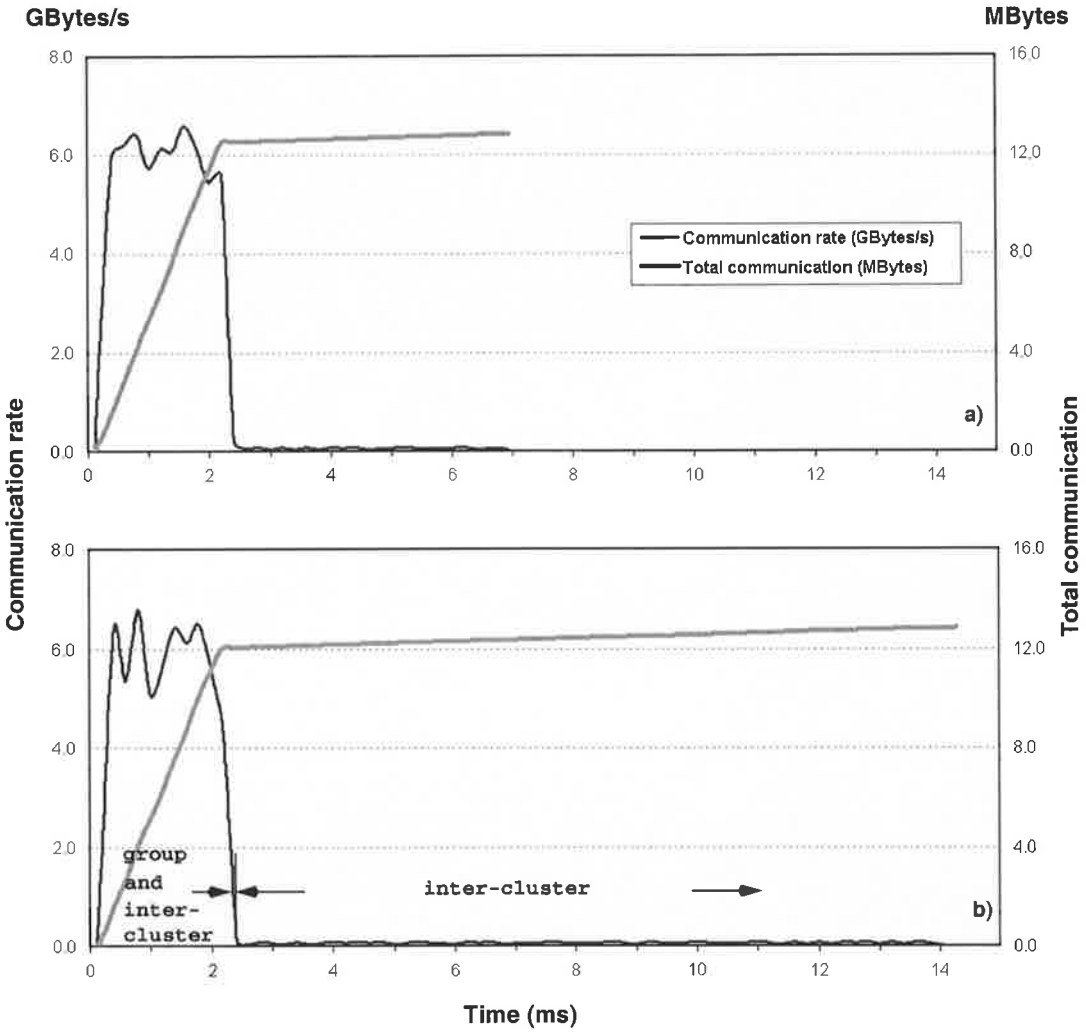

Figure 7.5 Inter-cluster communications in modified network

a) Two of the nodes in each group had a receiver in other clusters.

b) The load in a) was doubled and $1/14^{th}$ of the total messages were transferred between the clusters using the cluster-NCs.

were in fact the combination of the results of Figure 7.5 and Figure 7.3-b. The communication rate had three components in this Figure. The highest one was mainly the rate of message transfer within groups observed at the beginning of the communication combined with the inter-group and inter-cluster rates. After this initial high rate, the combination of inter-group and inter-cluster rates added up to around 670 MBytes/s. Although not very clear in the graph of case (a), inter-cluster communication finished within 6.8 ms and the rate dropped to 600 MBytes/s. After this point, which is shown on the graph with a circle, the rate was merely the inter-group communication rate achieved by scaling the network to eight clusters of 75 MBytes/s. In case (b), inter-

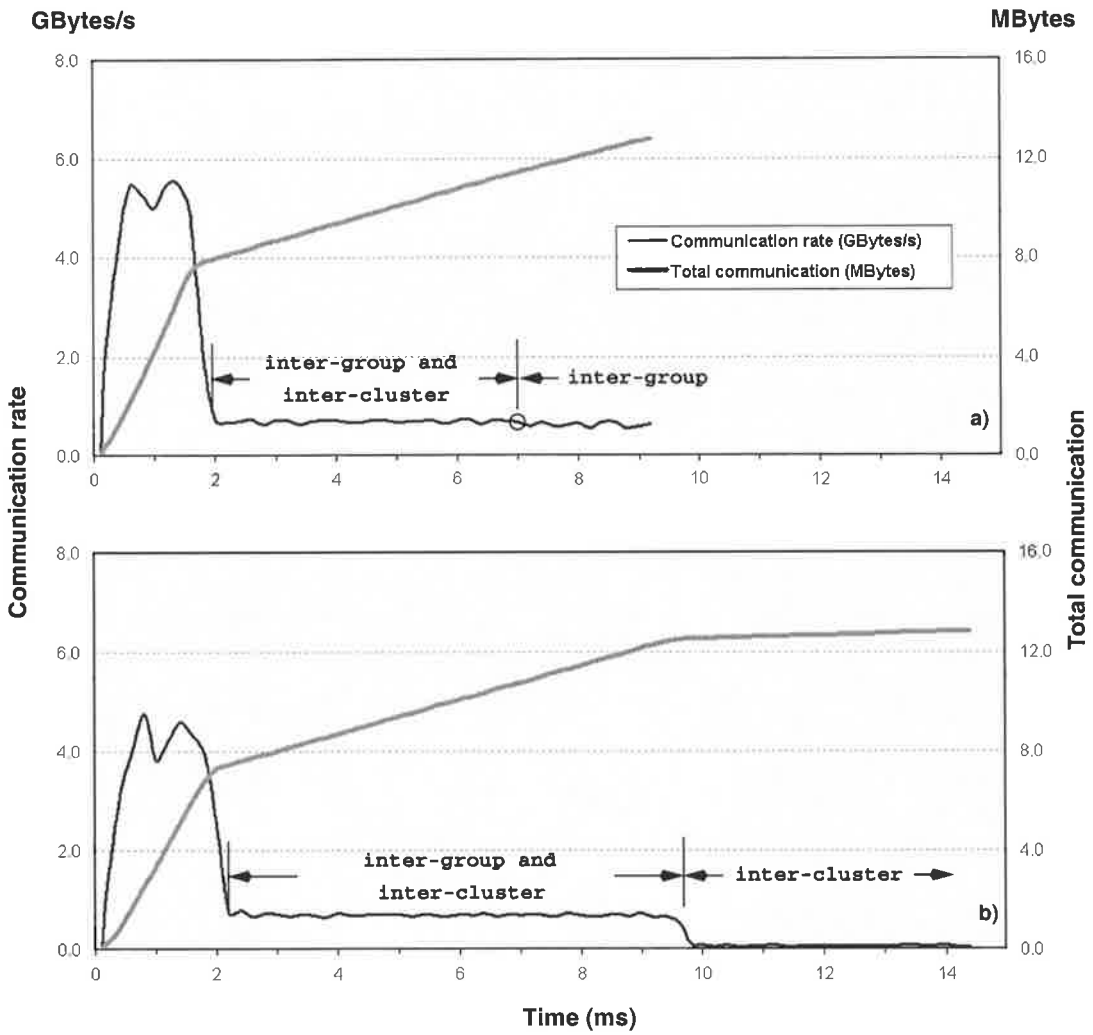

Figure 7.6 Inter-cluster and inter-group communications in modified network

a) 3/7th of the messages were inter-group and 1/28th were inter-cluster types. Both cluster-NCs and group-NCs were used for message delivery.

b) Same as a) with inter-cluster messages doubled.

group transfer was the same, but inter-cluster transfer was doubled. In this case, inter-group transfers finished within 9.4 ms (the same time it took for inter-group transfer in case-a) and the remaining communication was between clusters with a rate of 70 MBytes/s. An initial rate of 6 GBytes/s was expected for both (a) and (b); however, only 5.2 and 4.3 GBytes/s were achieved respectively. Similar to the case explained in section 4.2 in Chapter 6, this was the result of overloaded cluster-NCs and was detected by reviewing the records of the simulation log file. Inter-cluster messages that were sent to cluster-NCs were not picked up from the shared memory at a rate comparable to the

transmit rate because of the limited communication bandwidth of the cluster-NCs. Hence, communication within groups was conducted with fewer available buffers and the rate was reduced. Increasing the number of buffers fixed this problem and the expected rate was achieved.

These results confirm that in the modified network, inter-group communication is performed with acceptable speed, while a large number of inter-cluster messages can create a bottleneck. This demands more improvements to the model. In the next section, the network structure will be further improved to achieve the desired performance.

3 Improved network structure

The performance of the modified network was not satisfactory as large inter-cluster communication could degrade it significantly. An improved structure with more links for interconnecting the clusters is presented in Figure 7.7. In this structure, the cluster-NCs are removed and group-NCs in different clusters (simply called NCs) are interconnected using several MPMs.

As shown in the Figure, each NC is connected to a group-MPM, a cluster-MPM and a network-MPM. Each network-MPM interconnects the NCs connected to similar groups in different clusters. If the receiver is in another cluster, the node sends the message to the NC of the group. The NC passes it to the linked NC in the destination cluster. If the receiver node has the same group number as the sender, the NC delivers the message directly, otherwise, it sends the message to the NC of the destination group for delivery to the end receiver. Hence, an inter-cluster message may be transferred with three or four hops depending on the group number of the sender and the receiver nodes.

The improved network structure can be viewed as the expanded version of the original structure in Figure 3.4 on page 48. The cluster-MPM has been duplicated several times; the clusters have been substituted by improved clusters, and they have been linked by cluster-MPMs. In fact, by considering only one of the cluster-MPMs and ignoring the others, the structure of Figure 3.4 can be obtained. In the resultant network, the clusters

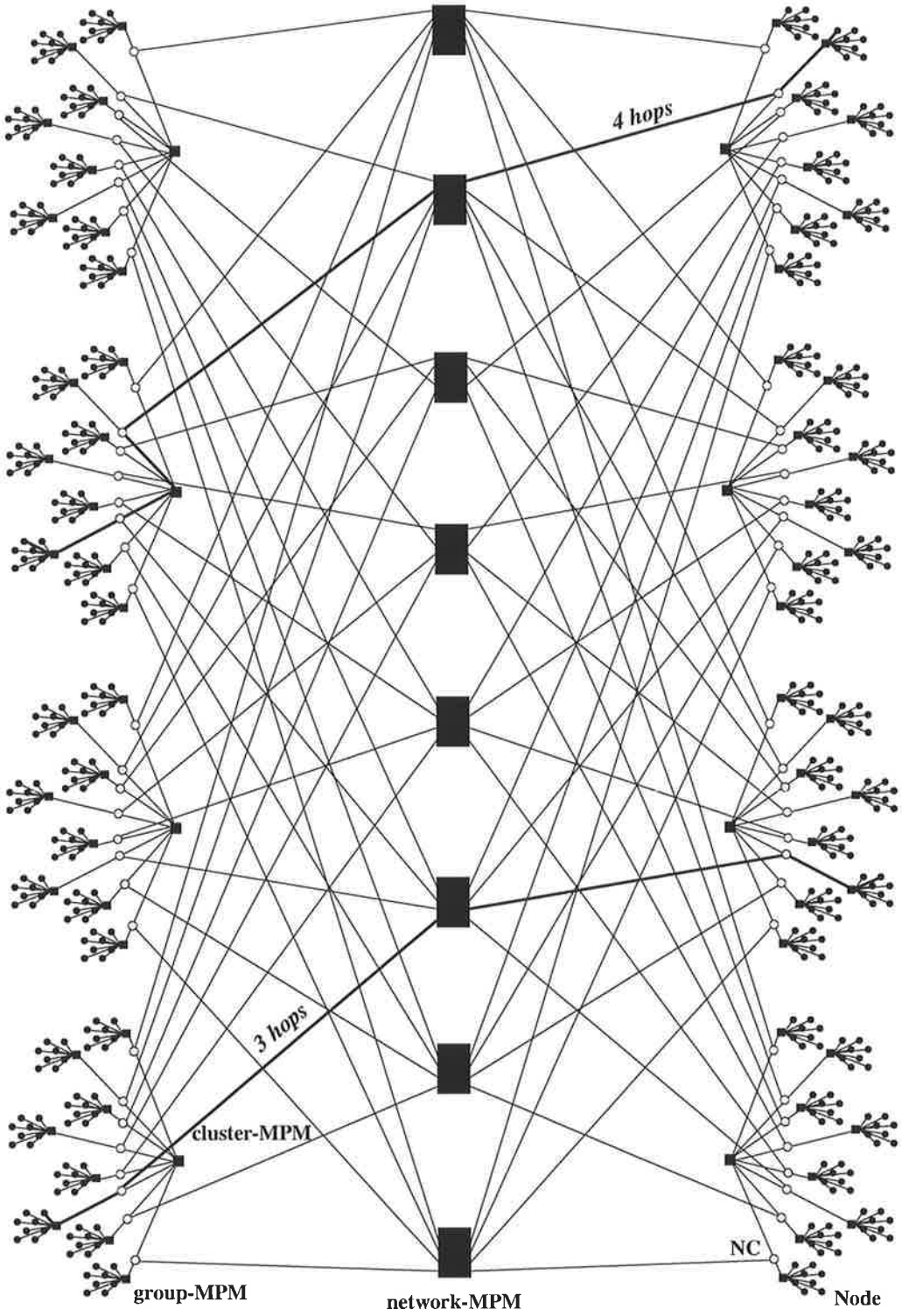

Figure 7.7 Improved network structure

The NCs in different groups are interconnected with several network-MPMs. Inter-cluster messages can be delivered in three or four hops. The highlighted paths show two different types of inter-cluster communication.

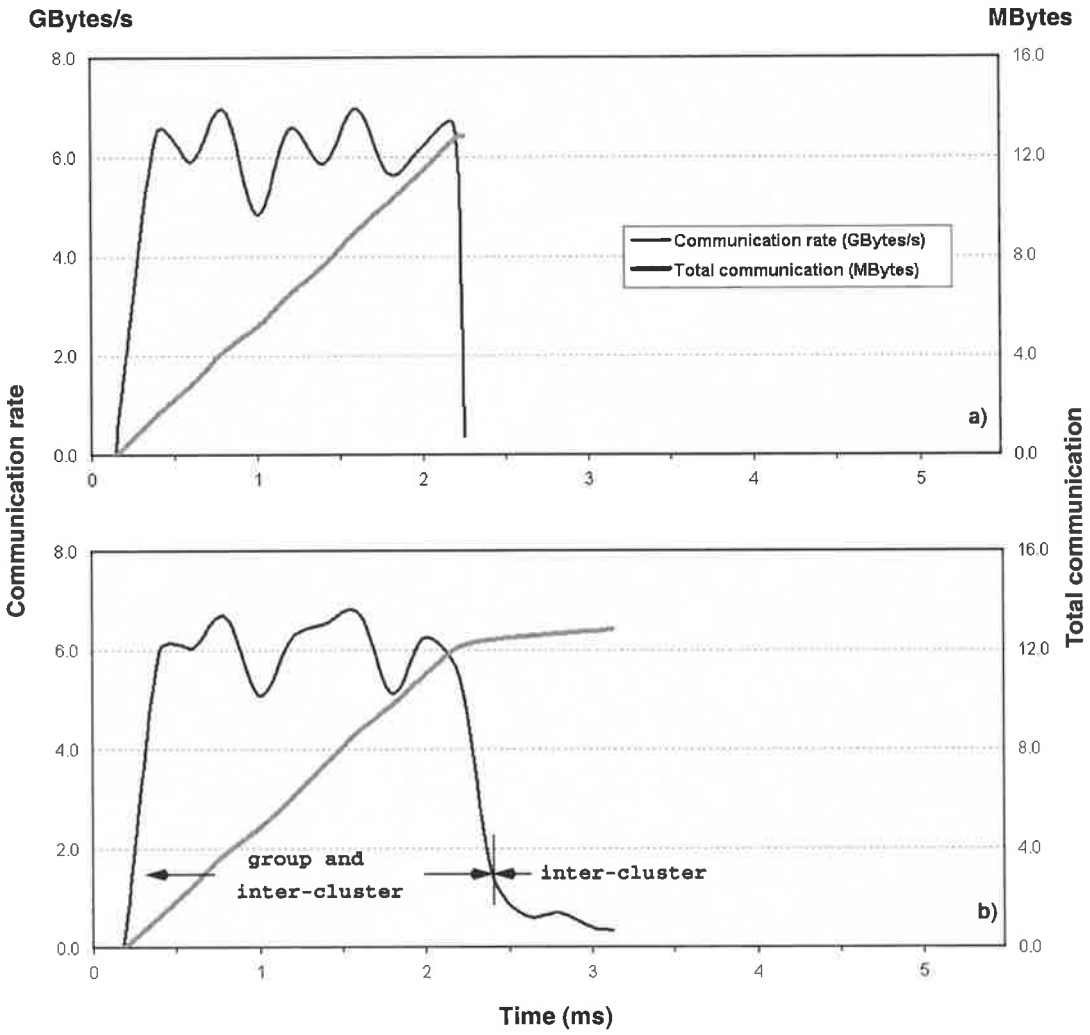

Figure 7.8 Inter-cluster communications in improved network

a) 1/14th of total messages were transferred between clusters and 3/4 of them required four hops.

b) 1/7th of the total messages were inter-cluster type.

are interconnected with more links and the communication load is distributed among several NCs.

The simulation model was modified to accommodate the required changes in the structure. At first, besides communication within groups, only inter-cluster communication was tested on the model and the results are displayed in Figure 7.8. In case (a), 1/14th of total messages were inter-cluster type with 75% requiring four hops. The condition was similar to the case in Figure 7.5-b, but it finished much faster and required almost the same time of communication within groups only. In case (b), the

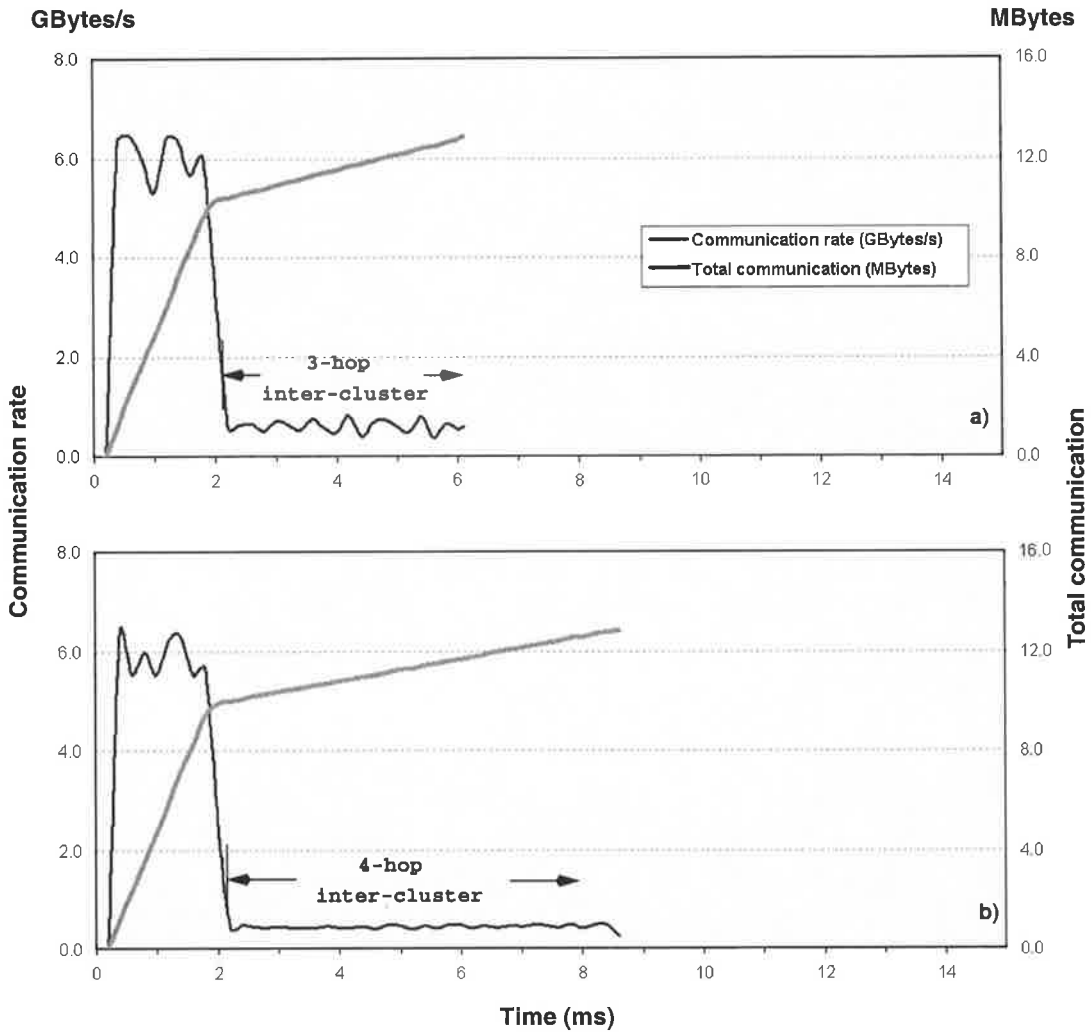

Figure 7.9 Effect of extra hop for inter-cluster communications in improved network

In both cases, $2/7^{th}$ of the messages were inter-cluster type. They required **a)** three hops **b)** four hops.

load was doubled to $1/7^{th}$ of the total messages and all were delivered in a short time. These results show that overloading of NCs has been reduced significantly.

In another test, the effect of the extra hop on the performance was investigated. Figure 7.9 illustrates the performance of the system with $2/7^{th}$ of the total messages as inter-cluster type. In case (a), all of these messages required three hops. The rate for inter-cluster communication was around 0.6 GBytes/s. In case (b), all of the messages required four hops and the rate dropped to 0.45 GBytes/s as the consequence of using

an extra hop. These results show that the improved network can handle inter-cluster communication much better than the modified network.

Figure 7.10 illustrates the results of a test in which besides communication within groups, a mixture of inter-group and inter-cluster messages, each consisting of $1/7^{th}$ of the total messages were used. In case (a), three hops were required for inter-cluster messages. Both inter-group and inter-cluster messages were delivered at the rate of 0.6 GBytes/s, which can be seen after the initial high rate for the communication within groups. In this case, the system performance was similar to the case in Figure 7.9-a, and

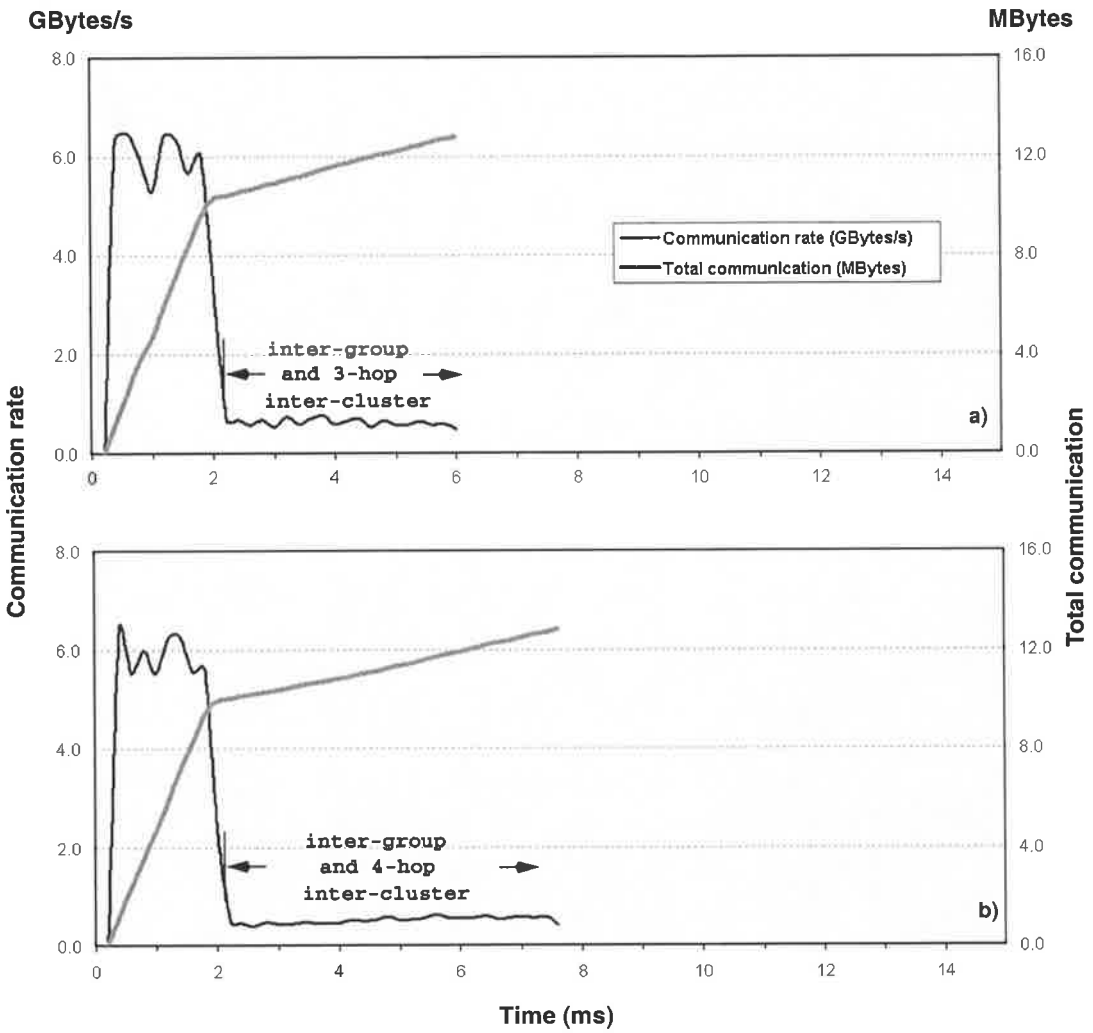

Figure 7.10 Inter-cluster and inter-group communications in improved network

$1/7^{th}$ of the total messages were inter-cluster type and a similar portion was inter-group type. Inter-cluster massages required **a)** three hops, **b)** four hops.

it shows that inter-group and 3-hop inter-cluster messages had been treated almost equally, because the same NCs handle both types. In case (b), inter-cluster messages required four hops and the communication rate dropped to 0.45 GBytes/s for the duration of this type of communication. The rate was increased to 0.6 GBytes/s after the inter-cluster communication terminated.

As can be seen from this test, communication of messages using NCs is controlled by two rates. Inter-group communication can be performed in three hops with a rate of 0.6 GBytes/s. This rate is achieved by scaling the structure to eight clusters, each having a rate of 75 MBytes/s. Likewise, 3-hop inter-cluster communication can be performed at the same rate of inter-group communication because the mechanism for both types is very similar. On the other hand, 4-hop inter-cluster communication is performed with the reduced rate of 0.45 GBytes/s because of the extra hop. As both inter-group and inter-cluster messages are handled by the same NCs, in the worst case, a mixture of both type of messages will require the sum of the time spent on each communication individually.

For further evaluation of the system performance with mixed types of communication, more tests were conducted and Figure 7.11 shows the results. In case (a), $3/7^{th}$ of the total messages were inter-cluster type, all requiring three hops. As expected, the communication rate was around 0.6 GBytes/s. In case (b), the configuration was changed to $1/7^{th}$ inter-cluster and $2/7^{th}$ inter-group messages. A similar performance was achieved and the communication rate and time were almost identical. Case (c) was similar to case (b) with the number of hops of inter-cluster messages increased to four. The extra hop dropped the communication rate to 0.45 GBytes/s. When inter-cluster communication was terminated, the rate of 0.6 GBytes/s could be achieved for inter-group communication.

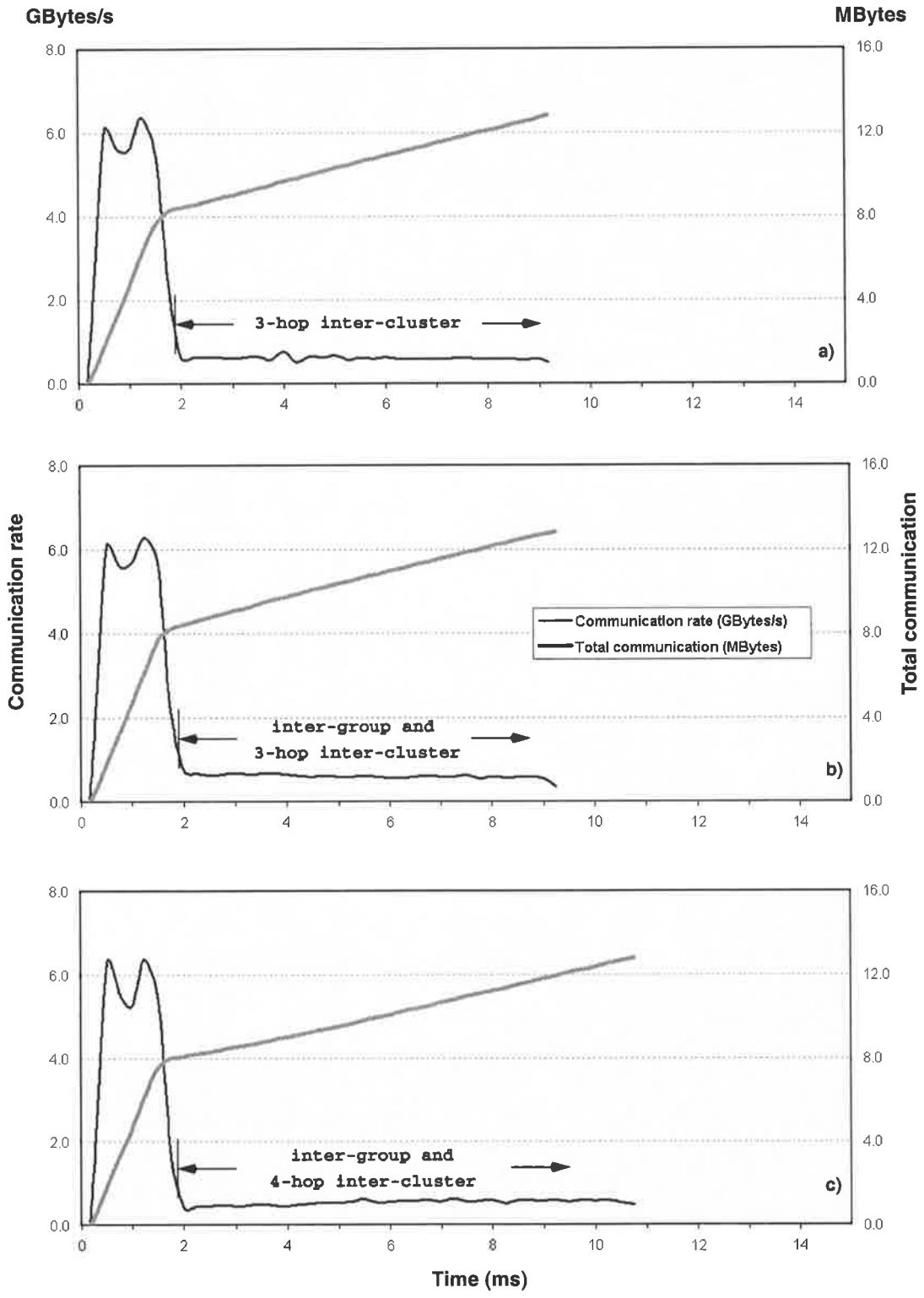

Figure 7.11 Mixed message types for improved network

a) $3/7^{th}$ of total messages were inter-cluster type requiring three hops. **b)** $1/7^{th}$ of messages were 3-hop inter-cluster type and $2/7^{th}$ were inter-group type. **c)** Same as (b) with four hops for inter-cluster messages.

4 Discussion

4.1 Scaling of the structure

Communication within groups has the highest performance in this structure as the nodes can directly communicate in one hop. For a limited number of nodes, the group communication is scalable. The limiting factor is the number of available ports on multiport memories. For a group of eight nodes running at 20 MHz, the communication rate is about 90 MBytes/s.

For more nodes, a cluster in which groups of nodes are interconnected with NCs can be used. An inter-group message is handled by two NCs and it is transferred in three hops. Hence, it cannot be transmitted as fast as the communication within groups. The communication rate for a cluster of eight groups (64 nodes) is 75 MBytes/s. However, the communication rate within groups can rise up to 700 MBytes/s.

A network of 512 nodes can be organized as the structure of Figure 7.7, in which eight clusters are interconnected using eight multiport memories. The communication within groups has the highest throughput and can rise up to 6 GBytes/s. The inter-group communication has the second highest rate and can rise up to 600 MBytes/s. Inter-cluster communication has the lowest performance. Depending on the location of sender and receiver in the structure, it can be performed with the same rate of inter-group communication (600 MBytes/s) in three hops, or with the reduced rate of 450 MBytes/s in four hops. Scaling of the structure beyond 512 nodes requires more changes and is not beneficial because of the increasing number of hops.

4.2 Use of 8-port memories

The structure of an 8-node group discussed so far is based on 9-port memories, which includes an additional port for a network controller. The idea of using eight nodes in a group originated from the structure of a hypercube of order-3 for interconnecting eight nodes. Another reason for choosing 8-node groups was the ease of using a 3-bit binary code to address each node. This reasoning also applies for eight groups in a cluster and eight clusters in a network. In the growth of multiport memories, a more realistic

expectation is to look for 8-port memories rather than 9-port. Hence, in communication structure using multiport memories, it is more sensible to use 8-port memories rather than 9 or 10 ports.

One of the advantages of the improved network structure is that for a network with eight clusters, each including eight groups, the network MPMs and cluster MPMs require only eight ports. This means that if only the number of nodes in each group is reduced to seven, all the multiport memories in use will be 8-ported. With this modification, the number of total nodes in the network is reduced from 512 to 448, but a more realistic network is proposed. Of course, a smaller number of ports can still be used; however, reducing the number of ports reduces the parallelism in the structure, which is the main contributor to performance rise.

The network structure based on 7-node groups has not been evaluated by the simulator. However, it is expected that the higher communication rate related to the communication within groups will be reduced by a factor of 1/8 or 12.5%. The rate of communication for inter-group and inter-cluster messages should not be affected.

4.3 Practical issues

The improved network structure presented in Figure 7.7 was evaluated by simulation and practical issues were not addressed. The results illustrate the system performance under perfect conditions. Obviously, implementing this structure in practice will face several practical issues. One of the problems is the physical location of the nodes, network controllers, and multiport memories. Physical location of these components affects the length of the links that connect them. As data is transmitted in parallel bits with speeds comparable to processor clock, the length of the links need to be kept to a minimum. This in turn puts a constraint on the physical size of a large system that may not be easily achievable and will require a meticulous design.

The critical component in the network structure is the network controller. Each network controller is connect to three MPMs, one of which, the network-MPM may not be in the close vicinity. Proper buffering of the network controllers buses, especially the one connected to network-MPM can alleviate the lengthy bus-issue discussed earlier.

Wider data paths combined with higher port counts will increase the routing complexity of the PCB design. However, with current multi-layer PCB design technology, a careful design can overcome the routing problems. The designer should also keep the PCB trace lengths balanced to avoid signal skews.

The network structure encompasses many cluster structures that are interconnected by several network-MPMs. A cluster structure also contains several group structures. It would be very beneficial to highly optimize the physical structure of a group and use it to generate an optimum cluster structure. With this approach, the remaining problem would be the interconnection of several clusters through network-MPMs. Using appropriate buffering on the network controller as discussed before will considerably reduce the complexity of the connections.

In implementing any type of structure, several small or large practical issues need to be considered and resolved. The experience of the technical team plays an important role in this process. The structure proposed in this study was not intended to address all the practical issues related to a complex structure such as the one presented in Figure 7.7. Careful consideration of the practical issues need to be performed at the time of implementing the system and proper solutions need to be applied.

5 Conclusion

The limited bandwidth of the single NC in a cluster could create a communication bottleneck as discussed in the previous chapter. In the improved cluster, the bottleneck was removed by distributing the load among several fast NCs. In the new structure, an inter-group message requires three hops, as an extra hop is used to transfer the message between NCs. Even with this extra hop, throughput was greatly enhanced and overloading of NCs was reduced considerably.

The network structure was modified to utilize the improved cluster structure. In each cluster, an extra NC was used to handle inter-cluster communication. Inter-cluster NCs were interconnected by a cluster-MPM. The communication within groups and inter-group communication scaled very well; however, cluster-NCs could be overloaded if

the number of inter-cluster communications was increased. This demanded extra improvements to the model.

The network structure was further improved by removing the cluster-NCs and interconnecting the group-NCs through different paths generated by several MPMs. In this structure, overloading of cluster-NCs is reduced and inter-cluster communication takes three or four hops depending on the location of the sender and receiver. A 3-hop message can be transferred at the same rate of an inter-group message, as they both use a similar mechanism for transmission. On the other hand, the communication rate for 4-hop messages is reduced by 25% as the result of the extra hop. In the worst case communication when all NCs are engaged in message handling, the communication time would be equal to the sum of the time spent on inter-cluster and inter-group communications.

CHAPTER 8

Conclusion and Further Directions

This study has explored the possibility of a new scheme for interprocessor communication using multiport memories. A novel structure for this type of communication was proposed and evaluated by a hardware prototype and a simulation model. In this concluding chapter, the steps involved in designing, evaluating, and improving the structure are presented briefly and the achievements are discussed. Finally, several possibilities for improving the system and further directions in pursuing this study are presented.

1 Conclusions

The intended focus of this study was to improve the performance of interprocessor communication in multiprocessors. A novel communication scheme based on passing information through multiport memories was proposed and the structure was evaluated by a hardware prototype and simulation modelling.

In shared memory systems, several processors are connected to a common memory and communication is performed by using memory load and store instructions. In message passing systems, several nodes are connected by an interconnection network and communicate by sending and receiving messages. In the proposed structure, several nodes share a local shared memory in a group and communicate through independent ports of shared memory without the overhead and delay of bus or interconnection network. A cluster of groups can be created in which differnt groups are interconnected by network controllers and other multiport memories. Nodes can communicate by sending and receiving messages through multiport memories. In fact, multiport memory is used as a link for message passing. Unlike message passing systems in which communication is normally performed serially, in this structure a message is passed through wider data path of the shared memory. In contrast to other shared memory systems, the common memory in this structure is only used for message passing and small capacity is adequate for this purpose.

In order to evaluate the structure, a small system called MultiCom was built as a prototype, which had four nodes and a four-port memory. The memory management of the system had to be designed in a way that it could prevent the nodes from interfering with each other. By using static allocation, the memory was divided into several buffers and each buffer was allocated to the communication of two specific nodes. No interference was expected in this method as a separate buffer was assigned for the activities of each node. However, the idle buffers that were not used in a communication at a time could waste the memory and reduce the efficiency of the communication scheme. On the other hand, dynamic allocation could use fewer but larger buffers that could be assigned to any communication on demand, and full memory utilization was

thus possible. This approach required a lock mechanism for allocating the buffers and higher overheads were involved.

As no hardware lock was available on the multiport memories, the lock had to be implemented entirely in software. New algorithms for the lock mechanism were devised and were successfully tested on MultiCom. A primary communication protocol for both allocations was established and it was improved gradually to accommodate the required modifications deemed necessary in the process of scaling the structure.

The performances of static and dynamic allocations were almost the same on MultiCom. For 20-MHz node processors, the system could achieve an aggregate communication rate of 45.5 MBytes/s. This is at least more than 4.2 times faster than a hypothetical system using serial communication running at comparable speed, 11 times faster than a system communicating through dual-port memories, and 14 times better than a bus-based system. In addition, the cost of the system was reduced considerably compared to a system using dual-port memories as a result of reducing the number of required chips.

The main advantage of dynamic allocation over static allocation in MultiCom was that multicasting/broadcasting could be performed with dynamic allocation with higher efficiency. This increased the communication rate to 64.8 MBytes/s.

In order to evaluate larger systems, a simulation model was created. To make the model more accurate and reliable, at the first stage, a model of MultiCom was created and tested successfully. Evaluation of the model revealed that increasing the memory size of MultiCom beyond the memory size of 4 or 8K words did not improve the performance very much. If fact, doubling the existing memory size increased the communication rate by 7%, and further doubling increased the rate by only 2.5%. Doubling the memory size increases the cost considerably and the gain in performance is very little to justify the additional expense.

This stage of simulation confirmed an important feature of the proposed communication scheme: For the all-to-all test program, the structure only requires very small shared memory for communication. This is a very important outcome, especially

considering the fact that the capacity of the future multiport memories with large number of ports would be very small. Even with a small memory size, it would be still possible to achieve a high performance using this structure. Unlike other shared memory systems, the small shared memory in this structure is exclusively used for communication purpose. In fact, shared memory is used as a link for message passing.

The model of MultiCom was expanded to cover more nodes. This stage of simulation revealed that dynamic allocation could scale linearly, but static allocation could not. The reason was that for a fixed memory capacity, the buffer size in static allocation was reduced by N^2, but in dynamic allocation, it was reduced by 1.6N. Considering the advantage of using multicast/broadcast, dynamic allocation was used as the memory management of the larger systems in the simulation model.

The simulation model was expanded to cover several groups in a cluster, which were interconnected by a network controller. The results of this step showed that using only one NC for a cluster could easily overload the NC because of its limited communication capacity. To achieve a higher performance, an improvement in the structure was necessary.

The cluster structure was modified by introducing a separate NC for each group in the cluster. In addition, the NCs were also interconnected using an extra multiport memory. Although this modification increased the number of hops for inter-group communication from two to three, the performance of the system was significantly improved and overloading of the NCs was reduced considerably.

The simulation model was expanded again to cover several clusters in a network. Because of the modified cluster structure, the original proposal also had to be modified to accommodate the new cluster structure. For this purpose, a cluster-NC was added to the cluster structure for handling communication between clusters. The cluster-NCs from different clusters were interconnected by a network multiport memory. Evaluation of this structure revealed that it was not efficient and the cluster-NCs could be overloaded in moderate or large inter-cluster communications.

To overcome the shortcoming of this structure, a major change in the proposed structure was necessary and several parallel paths were required to distribute the communication load among the NCs. In order to achieve the improved network structure, the cluster-NCs were removed and the group-NCs with identical group numbers in different clusters were interconnected using a multiport memory (refer to Figure 7.7 on page 152). This structure created several data paths for communication between clusters, and depending on the position of the transmitting and receiving nodes in the network, a message could be delivered in three or four hops.

The simulation model showed that the new improvements reduced the load of NCs considerably. Three communication rates were measured in the improved network. A rate of 6 GBytes/s could be achieved for communication within groups. A rate of 600 MBytes/s was also achievable for inter-group communication. The inter-cluster messages were transferred at a rate of 450 or 600 MBytes/s, depending on whether four or three hops were needed.

The overall conclusion is that the structure is very efficient for small-scale networks, it is good for medium to large-scale structures up to 512 nodes, but it is not recommended for very large networks, because of increased number of hops.

2 Further research

There are several possibilities to improve and extend the results of this study. They range from the use of hardware supports for multiport memories to the use of other structures for connecting the nodes and multiport memories. Some of these possibilities are discussed here as the recommendation for further research.

2.1 Hardware support for multiport memory

Dynamic allocation proved to be a better memory management strategy for the proposed communication scheme. The major disadvantage of this method is the overhead of the lock mechanism and the allocation process, as both have a serial nature.

The use of hardware support can reduce the overhead considerably. A few options are discussed in the following sections.

2.1.1 Multiport semaphore logic

As discussed in Chapter 5, semaphore logic can simplify the required lock mechanism for the multiport memory. Unfortunately, at the time of this study, semaphore logic was only available on dual-port memories. Hence, in the design of MultiCom and the simulation model, a purely software-based lock was used. Some of the problems in designing the semaphore logic for multiport memories are discussed in the Appendix and a new semaphore logic based on fixed priority is proposed and tested by the hardware design tools for satisfactory operation. The use of this logic or similar ones in multiport memory will lead to a much faster lock mechanism. This can reduce the overhead and increase the performance.

2.1.2 Centralized control

Using hardware support for the tasks that are time consuming in the software can also reduce the overhead of the lock mechanism. For example, buffer allocation can be also performed with a controller rather than by each node processor. In addition, other control circuitry such as semaphore logic and interrupt logic (as proposed in the Appendix) can be integrated into the controller chip to achieve a centralized control. This chip can greatly simplify the management of the multiport memory. A simple approach for designing this chip can be as follows:

1. A very small size multiport memory with the same number of external ports as the main multiport memory plus an extra port for the access of the controller is required. In general, the word size of this control memory can be smaller than the main multiport memory.
2. A fast but simple custom-designed controller should be connected to the control memory through the extra port. It should be capable of reading and writing the control memory and performing simple calculations and tests. As there is no interaction between the controller and the main multiport memory, no extra port for the main memory is required.

3. Each node can request a service from the controller by writing a command to a specific location assigned exclusively to that node. For example, a node can request a buffer, and the controller can return the allocated buffer number to the requesting node. Note that no lock mechanism is required in this structure, as only the controller performs the sensitive tasks. In addition, the nodes do not interfere with each other in any way.
4. The interrupt generation logic as proposed in the Appendix can be also integrated into this chip as an independent logic. Although the controller mentioned above is also capable of controlling the interrupt logic, using an independent circuit for the interrupt logic within the same chip is beneficial, because the response time for buffer allocation will not be affected.
5. Any other required logic can be integrated into this chip as an independent logic, or as an additional task for the controller.

The use of this kind of controller external to the main multiport memory has the advantage that the memory chips can be designed to be simpler and bigger, because no extra logic is added to the chip. Furthermore, in horizontal or vertical expansion of the memory, which is performed by connecting several chips in series or in parallel, there will be no duplicate control circuitry which would normally be left idle. In addition, the control chip can be used with any kind of multiport memory, as there is no interaction between the main memory and the controller. The only requirement is the compatibility of the port counts.

2.2 Use of DMA for data transfer

In this communication structure, a shared memory created by multiport memories is only used as a communication medium for message transfer. Unlike other shared memory systems, the memory is not used as a shared area in general. As shown earlier, the structure requires only a small shared memory for message transfer. Otherwise, as it is not practical to create very large memory size using multiport memories, the structure would not be feasible.

In this approach, each node assembles a message in its local memory and sends it to another node in several packets through the shared memory. Hence, data transfer

between local and shared memory or vice versa is a common task for a node and improving it will improve the communication rate.

In both the hardware prototype and the simulation model, all data transfers were performed by the node processors. It is possible to increase the speed of data transfer by using Direct Memory Access (DMA) controllers. In this approach, a node should apply for the lock and allocate a buffer in the shared memory. It should also initialise a DMA channel for the appropriate transfer. At this stage, the DMA controller can take over and perform the required data transfer. This approach will have the following benefits:

1. In general, the overall data transfer rate is higher with DMA, because everything is controlled by hardware and no software loop is required.
2. If the DMA transfer is transparent to the processor, it is possible for the processor to perform other tasks like applying for another lock, acknowledging an interrupt, or programming another DMA channel. The higher transfer rate and the freedom of the processor to perform other tasks can boost the performance. If the DMA transfer is not transparent, the node can be forced into an inactive state while DMA is efficiently performing the data transfer.
3. If the memory of the processor including the shared memory is partitioned and located in different memory banks, more than one DMA transfer can be active at a time, provided that adequate hardware support for accessing simultaneous banks is incorporated in the system design. The performance boost in this approach can be very high.

2.3 Using different communication structure

Other structures for interconnecting nodes using multiport memory can provide different performance. A mesh structure as discussed in the following subsection can be used for medium-scale systems. In addition, a method that virtually increases the number of ports of a multiport memory is also presented.

2.3.1 Mesh structure

Other structures can be also used for connecting nodes through multiport memory. Figure 8.1 shows a new structure using a two-dimensional mesh. Each node is

connected to two 8-port memories (excluding boundary nodes). The nodes sharing the same multiport memory can communicate directly, but other nodes should use intermediate nodes for their communication. The structure can be expanded in both X and Y directions and is useful for low to medium size networks. This mesh structure is the extension of the structure presented in [Varshneya+ 94] for 6-port memories. Based on the future availability and organization of new multiport memory chips, it is feasible to develop a variety of such meshes.

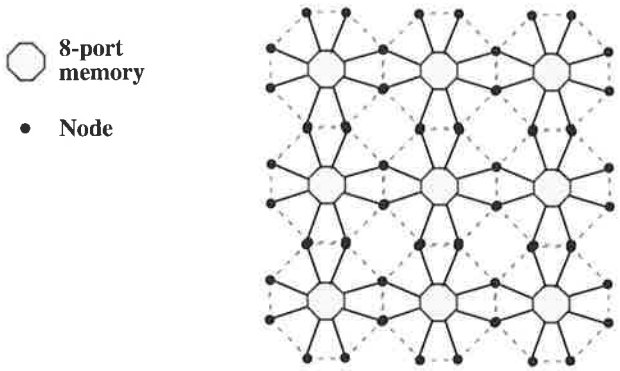

Figure 8.1 Mesh structure

Each node can access two 8-port memory modules. Some nodes can communicate directly. Others should use intermediate nodes for data passing.

The effectiveness of this structure needs further investigation; however, it is anticipated that it would only be useful for medium-scale systems, because of the increasing number of intermediate nodes involved in the communication of non-neighbouring nodes as the system expands.

2.3.2 Increasing the port count of multiport memory

The simulation model showed that increasing the number of ports of a multiport memory could increase the performance; however, the availability of multiport memories with higher port counts still remains a question. With some performance degradation, the bus system can be combined with multiport memory to increase the

number of port counts. Several nodes can be connected to each memory port using a bus to have a shared access to the memory. Although in principle, more than two nodes can be connected to each bus, no more than two nodes is recommended for this application. Figure 8.2 shows an example in which eight nodes are connected to a 4-port memory using a bus at each port. This method virtually doubles the port count, but it reduces the performance. It can be used as a compromise for the unavailability of higher port counts on multiport memories. Further investigation is required to analyse the impact of sharing a port among more than one node under different traffic conditions.

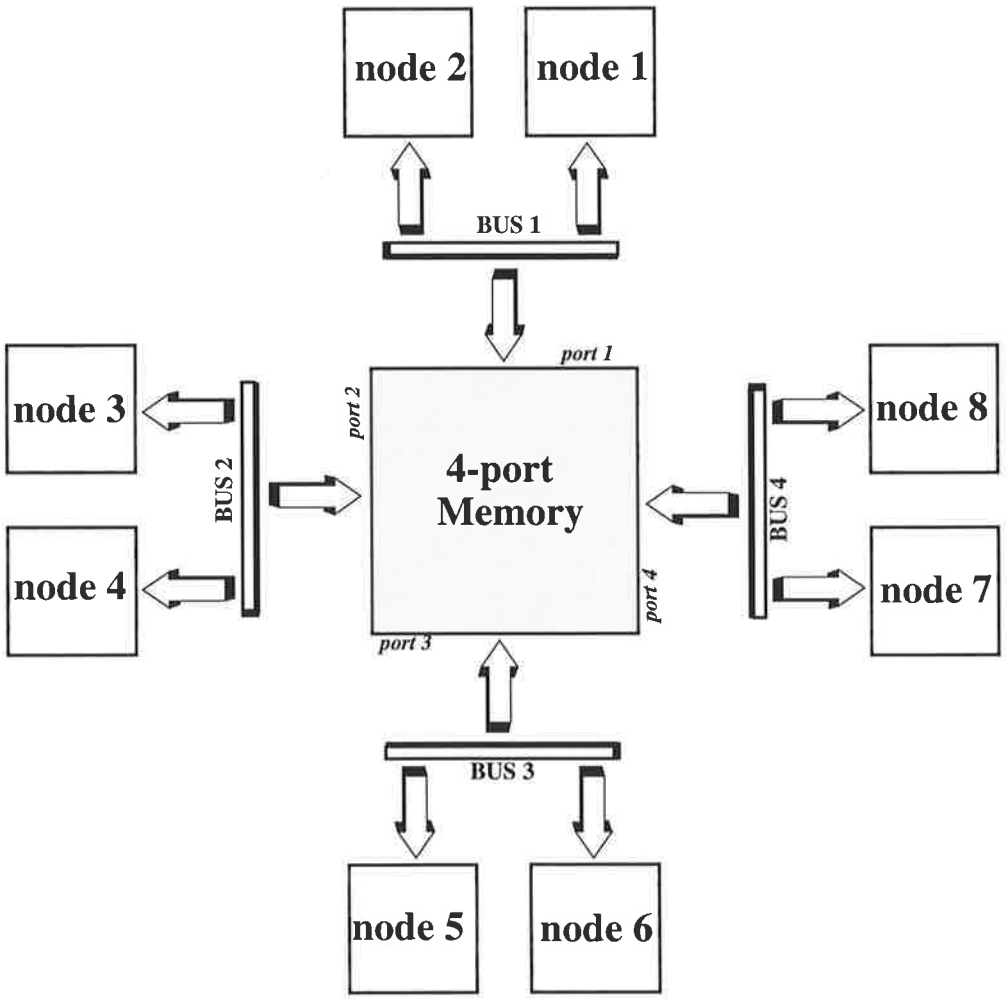

Figure 8.2 Increasing port count of multiport memory

By using a bus for each port, more than one node can be connected to shared memory. This virtually doubles the port count.

APPENDIX

Structure of Multiport Memory

For an efficient design with multiport memories, it is very beneficial for design engineers to understand the structure and architectural features of this kind of memory. In this Appendix, a typical structure for single-port memory is presented and is gradually expanded to dual-port and multiport memories. In addition, the control logic commonly used in dual-port memories to handle the simultaneous access conflicts are presented. These materials are from [Mick 96], [Wyland 88] and [Baumann 96]. In the discussion section that follows, several issues in designing and using multiport memories are discussed and new circuits for multiport semaphore logic and interrupt logic that can facilitate the use of multiport memories are proposed.

A

1 Structure of single-port memory

Figure A.1 shows a typical single-port four-transistor static RAM cell. This architecture is commonly used by most static RAM manufacturers because it offers high density, good speed, and low power.

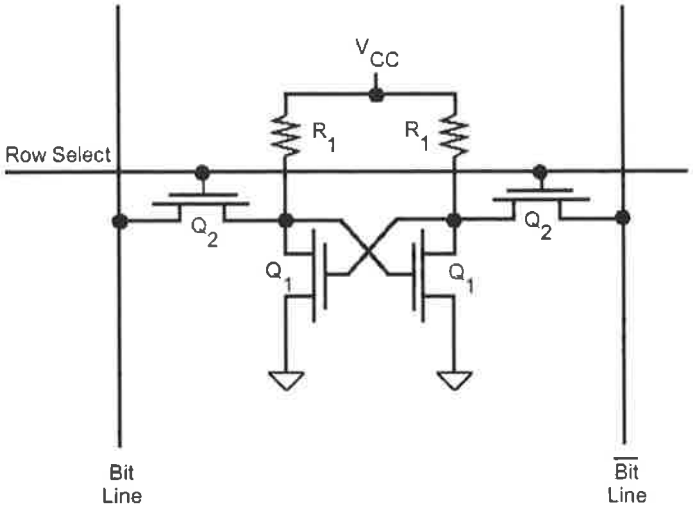

Figure A.1 Typical four-transistor SRAM cell

The cross-coupled inverters form the basis of the memory cell. When one is ON the other is OFF. Q2 transistors activated by row select connect the cell outputs to complementary bit-lines used for reading or writing.

The memory cell consists of two N-channel transistors (Q1) and two resistors (R1) that are connected to form two cross-coupled inverters. This gives a regenerative action such that when one transistor is ON, the other is OFF. Two additional N-channel transistors (Q2), usually called pass transistors are connected between the cell outputs and the bit-lines. The gates of the pass transistors are connected to the row select line. When a particular row of cells in the RAM is addressed, these two transistors are turned on. Depending on the current state of the cell, one of the bit-lines is driven high and the other one low.

A simplified structure of a 16x1 bits RAM organized in four rows and four columns is shown in Figure A.2. The row address decoder selects only one row. In each column,

one cell is selected and its outputs appear on the corresponding bit-lines. The bit-lines of each column are connected to the inputs of a differential sense amplifier by means of N-channel switches called data multiplexers. These switches are controlled by the column address decoder and only one column is selected. Hence, the sense amplifier is connected only to the one cell located at the intersection of the selected row and the selected column. The sense amplifier detects whether the state of the cell is logic one or logic zero depending on the relative polarity of the two bit-lines. In a read operation, the sense amplifier drives the Data-Out pin accordingly.

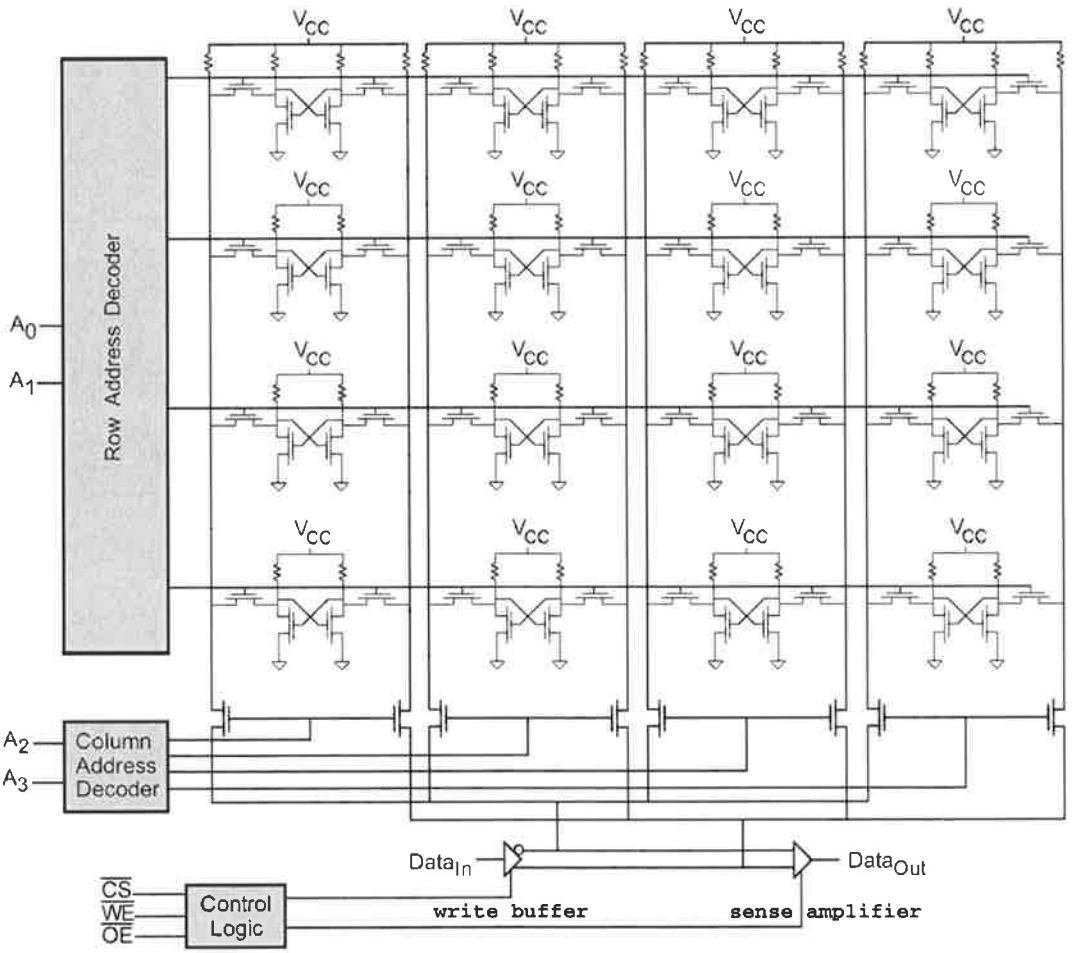

Figure A.2 The structure of 16x1 bits RAM

Only one cell is selected at the intersection of a row and a column. For READ, the differential sense amplifier derives the Data-Out line according to the state of the two bit-lines connected to its inputs. For WRITE, the write buffer overpowers the two bit-lines and sets the selected cell according to the state of the Data-In line.

For a write operation, a cell is selected at the intersection of a row and a column and its output appears on the active bit-lines. However, the write buffer that is driven by the Data-In line differentially drives one bit-line high and the other one low as determined by the logic state of the data input. The output of the write buffer is more powerful than the inverter transistors in the RAM cell (Q1 pair in Figure A.1) and it easily overpowers them if it is necessary to flip the static RAM bit.

A variation on the standard four-transistor static RAM cell is the six-transistor static RAM cell. In this cell, the two pull-up resistors (R1 pair in Figure A.1) have been replaced by two P-channel transistors. The operation of such a six-transistor cell is identical to the four-transistor cell described above. The difference between the two approaches is that the physical size of the cell with the P-channel transistors is larger than the cell with the resistors. The standby power is lower for the six-transistor cell because there is ordinarily no power being dissipated; in a four-transistor cell, one of the pull-up resistors is always dissipating power since one transistor of the cell is always ON. The six-transistor cell has higher radiation hardened characteristics than the four-transistor cell because the voltage swing in the cell is larger. This is because the internal node in the cell that is high is pulled to the Vcc rail by the P-channel transistor. In addition, the six-transistor cell provides higher internal noise margins in the circuit for the same reason. Most manufacturers of static RAMs use the four-transistor cell because it allows static RAMs of higher density to be fabricated with smaller die sizes.

2 Structure of dual-port memory

A dual-port RAM cell can be derived from the structure of a single-port cell. As illustrated in Figure A.3, the basic cell is created by the standard cross-coupled inverter pairs. There are two pairs of bit-lines associated with a cell, each pair acting as a read/ write port into the dual-port RAM. Two pass transistors (Q2) controlled by a row select line connect the cell outputs to a pair of bit-lines. Ignoring all the lines of port_2, the operation of the cell using port_1 is exactly as described for a single-port cell. Similarly, port_2 can access the cell using Q3 pass transistors.

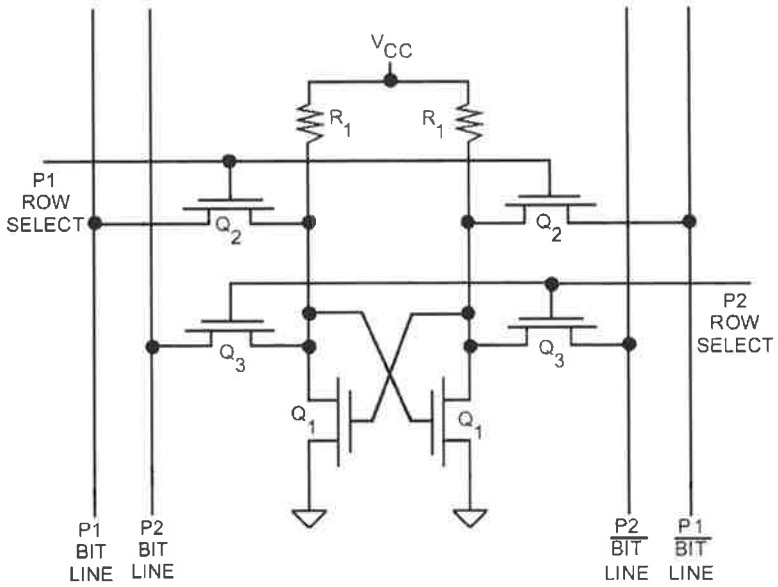

Figure A.3 Dual-port SRAM cell

Each cell can be accessed by two different ports using a separate row select and a pair of bit-lines. The operation of the cell from each port is similar to the operation of a single-port cell.

A schematic diagram of a 16x1 bits dual-port RAM is shown in Figure A.4. As each port has a separate address line, any memory cell can be selected by one port independent of the other port. Each pair of bit-lines in each column is connected to a sense amplifier and a write buffer via a data multiplexer so that each port can read from or write to its selected cell via separate data lines.

The ports can access the cells independently as long as both do not select the same cell; otherwise, data corruption may occur. Simultaneous reading of the same cell by two ports is not a problem; however, if two ports write to the same cell at the same time, one or both values may be lost. Likewise, if one port writes to a cell at the same time that the other port is reading the cell, the read may be corrupted even though the write is completed correctly.

2.1 Control logic for dual-port memory

Most dual-port memories include control logic to deal with three common application issues: conflict arising from simultaneous addressing of a memory location by both

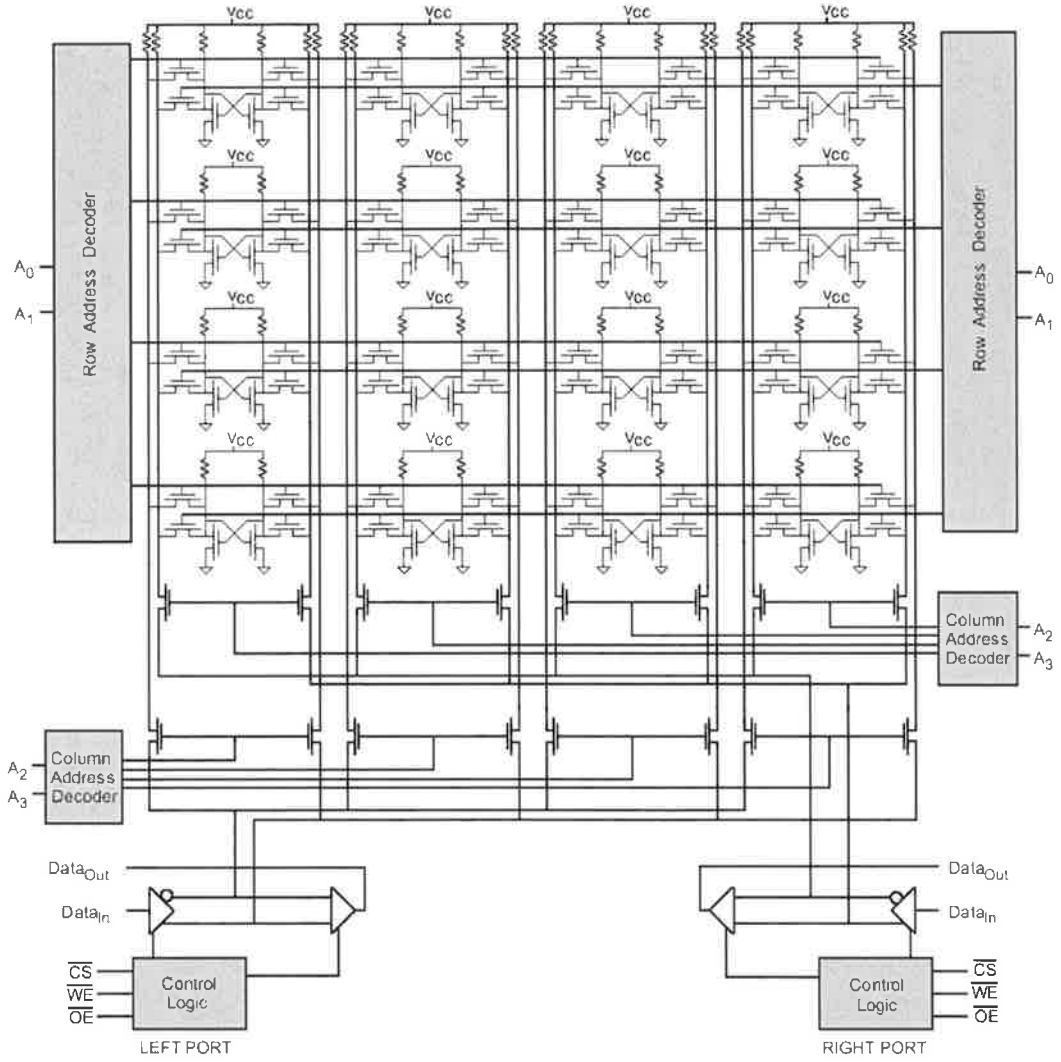

Figure A.4 Structure of 16-bit dual-port memory

Each port can select a different cell using separate address lines. The selected cells can be read or written independently using data lines that are isolated from the other port.

ports, hardware support for temporary assigning a block of memory to one side only (semaphore), and signalling between processors. Each case will be discussed briefly.

2.1.1 Busy logic for simultaneous access conflicts

A conflict can occur with dual-port memories when both ports attempt to access the same address at the same time. There are two significant cases: when one port is trying to read from the same location that the other port is writing to, and when both ports

attempt to write to the same location at the same time. If one port is reading while the other port is writing, the data on the read side will be changing during the read and corrupted data may be obtained. If both ports attempt to write at the same time, the memory cell will be driven by both sides and the result can be a random combination of both data words rather than the data word from one side or the other. Busy logic solves this problem by detecting when both sides are addressing the same location simultaneously.

Busy logic is a hardware address arbitration circuit that decides which side will receive a busy signal if the addresses are equal. It consists of a common address detection circuit and a cross-coupled arbitration latch. A typical circuit for busy logic is shown in Figure A.5. This circuit provides a $\overline{\text{BUSY}}$ signal to the port that has sent the address slightly later, inhibits the write request from that port, and makes a decision in favor of one side or the other when both addresses arrive exactly at the same time. The logic consists of a pair of address comparators, a pair of delay buffers, a cross-coupled arbitration latch, and a set of $\overline{\text{BUSY}}$ output drivers. The address comparator output is set true when both addresses at its inputs are equal.

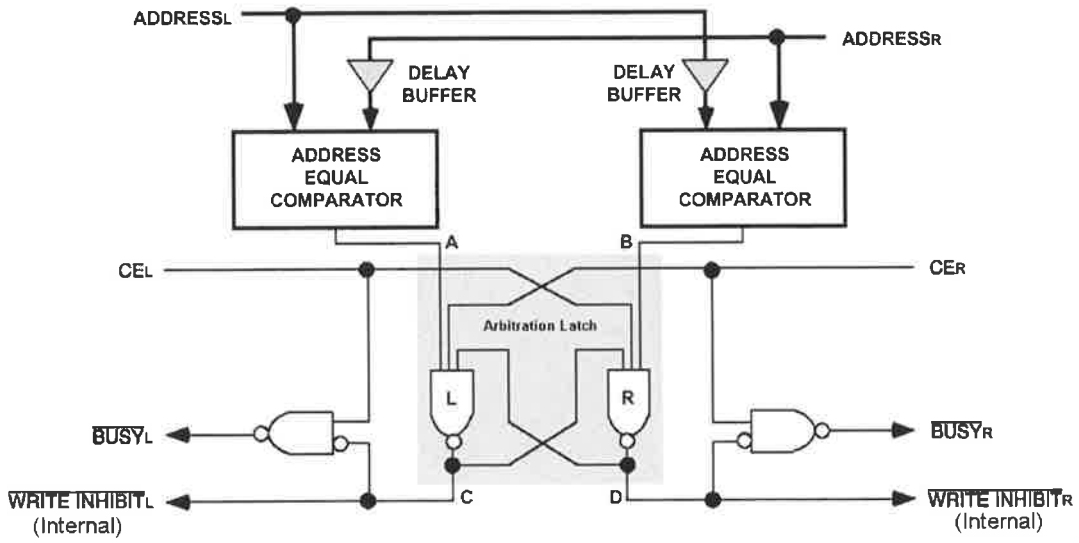

Figure A.5 Busy logic for dual-port memory

When accessing the same location from both ports, busy logic internally inhibits the write request that has arrived slightly later and sends a busy signal to the corresponding port. If both of the requests arrive exactly at the same time, the circuit will decide in favor of one side.

In the logic shown in Figure A.5, the ability to detect which address arrived last is provided by the time delay buffers between the address lines and the comparators. Assume that the L (left) address is stable and the R (right) address changes to match the L address. The R address comparator will go true immediately while the L address comparator will become active some time later as determined by the time delay gates.

The arbitration latch formed by the L and R gates settles with the timing of the address comparator outputs A and B. This latch has three stable states: both latch outputs C and D high, C low - D high, and C high - D low. Initially, both C and D are high because A and B are low. For the L address stable and the R address arriving later, B becomes active before A and forces D to go low. C will remain high because A will go high sometime later, but the input of L gate connected to D will go low before this occurs. Hence, D will be low and will inhibit the write request from the right port, which its address has arrived later. In addition, the right port will receive a busy signal.

The extreme case of decision making in busy logic occurs when both addresses arrive at exactly the same time. In this case, the outputs of both address comparators go high simultaneously activating both sides of the arbitration latch. The latch will settle into one of two states with either C or D latch outputs being active. The latch design must ensure that one side will be given priority in this case to avoid metastability.

A common way of using the busy line is to stretch the cycle for the operation performed by the losing processor until the other processor finishes its access. Note that for the read-read case, no arbitration is required, but this circuit generates a busy signal for one side. The arbitration latch can be modified to remove this case. As illustrated in Figure A.6, if both of the ports are performing read operations, the arbitration logic will be disabled and will have no effect on read operations.

2.1.2 Semaphore logic

Data integrity is a major issue in dual-port memories. Sometimes there is a requirement to assign a block of memory to one port temporarily. For example, if one processor needs to update a data table as a whole, the other processor should not interfere until the update is complete. Moreover, block allocation can be used to avoid the address

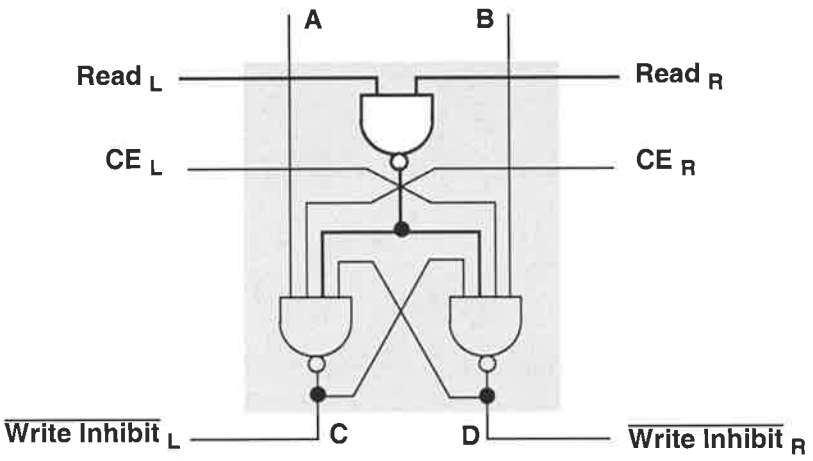

Figure A.6 Modified arbitration latch

If both of the ports perform read action, the arbitration latch will be disabled and no busy signal will be generated.

arbitration problem since it is a way of ensuring that both sides do not use the same address at the same time.

Semaphore logic is a hardware support mechanism to allocate a block of memory to one side. Most dual-port memories have up to eight semaphore latches. This method was explained in detail in "Section 2.1, Hardware semaphore" in Chapter 5.

2.1.3 Interrupt logic for signalling

A common problem in dual-processor systems is signalling between processors. For example, when processor A needs to request processor B to perform a task, it sends a signal to processor B. The task might be defined by a data word written in the common memory. When processor B completes the task, it needs to signal processor A that the task is done. Note that signalling must occur in both directions. A common form of signalling is to send an interrupt from one processor to the other one. This allows the receiving processor to be informed of a communication without having to check for it constantly.

Hardware support for this type of signalling is provided by the interrupt logic, which is available on most dual-port SRAMs. As shown in Figure A.7, in these devices, the two top memory locations serve as interrupt generators for the ports. If the left CPU writes

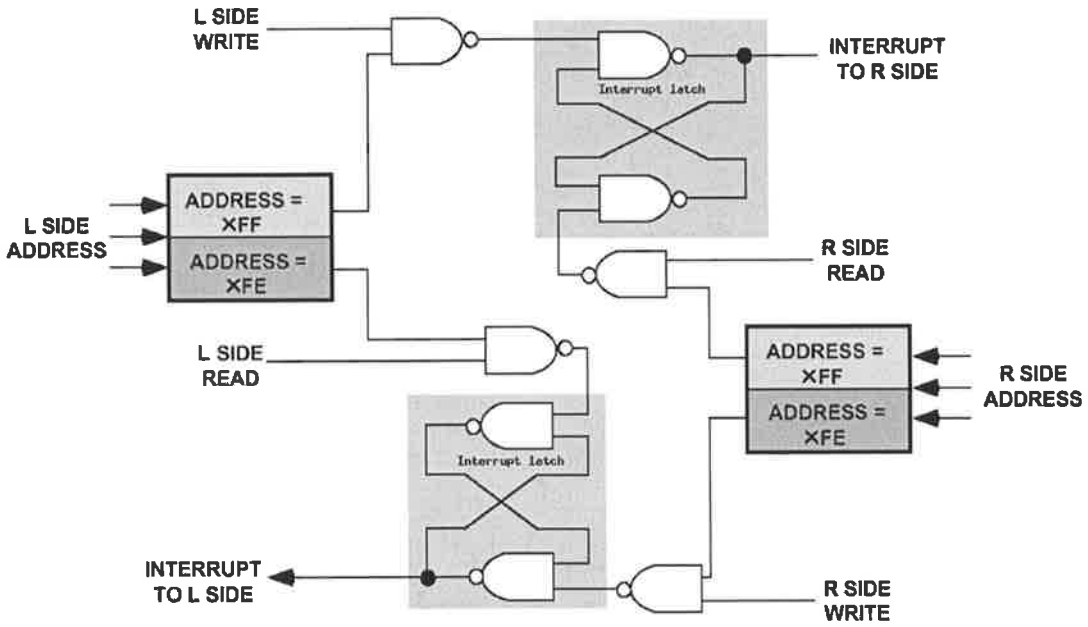

Figure A.7 Interrupt logic for signalling

Each processor can signal the other processor by sending an interrupt to it. If the processor on the left side writes a byte in the last memory location, an interrupt will be generated to the right side. Reading the same location by the right processor will clear the interrupt. The right processor can perform a similar task by writing in the penultimate location.

into the location XFF (X depends on the memory size), an interrupt latch is set and the interrupt line connected to the right CPU is activated. This interrupt latch is cleared when the right CPU reads from the same location. Similar logic is also provided to allow the right CPU to send an interrupt to the left CPU using the location XFE. The availability of this logic on memory chips simplifies the system design, because no extra logic is required for interrupt control. The interrupt logic is an additional feature on the memory chips and using it has no impact on the normal operation of the other memory locations.

3 Structure of multiport memory

The dual-port memory cell can be further expanded to form a multiport memory cell. Figure A.8 shows the schematic diagram for a 2-bit N-port RAM. The two inverters

making up the basic memory cell are similar to a single-port memory cell except that two P-channel pull-up transistors are used instead of pull-up resistors. This cell has similar benefits of a six-transistor cell as explained before. The inverters are connected in the normal cross-coupled fashion to create a single memory cell. N individual memory ports are generated by using N pairs of pass transistors for connection to N pairs of bit-lines. For each port, an individual row select line can activate the corresponding pair of pass transistor connected between the RAM cell outputs and the bit-line pairs. N sense amplifiers and write buffers are used to provide individual read/ write paths from each port to all the cells in the RAM. There are some practical limits on N such as the maximum number of sense amplifiers that a memory cell can feed.

As there are more ports in this memory, it is more likely that two or more ports will try to access the same cell simultaneously. Furthermore, the simultaneous access can happen for more than one cell by different groups of ports. Hence, data corruption problems in simultaneous access of the cells are more complicated in a multiport memory. This issue and several other issues in designing multiport memories are discussed in the next section.

4 Discussion

In the design of multiport memories with a large capacity and a large number of ports, several issues such as limited capacity, large pinout, and control logic should be considered. These issues are discussed in the following sections.

4.1 Limited capacity

The capacity of a multiport memory chip suffers from the bigger cell size and larger connection matrices. As illustrated in Figure A.8, the size of a multiport cell is big and for an N-port cell, 2N+4 transistors are needed. In addition, increasing the number of ports increases the amount of wiring inside the chip, and large connection matrices are required for routing vertical and horizontal lines. Consequently, a large section of chip area has to be devoted to the connection matrices and the chip area available to the cells (that are already big) is reduced. Therefore, the capacity of the multiport memory is

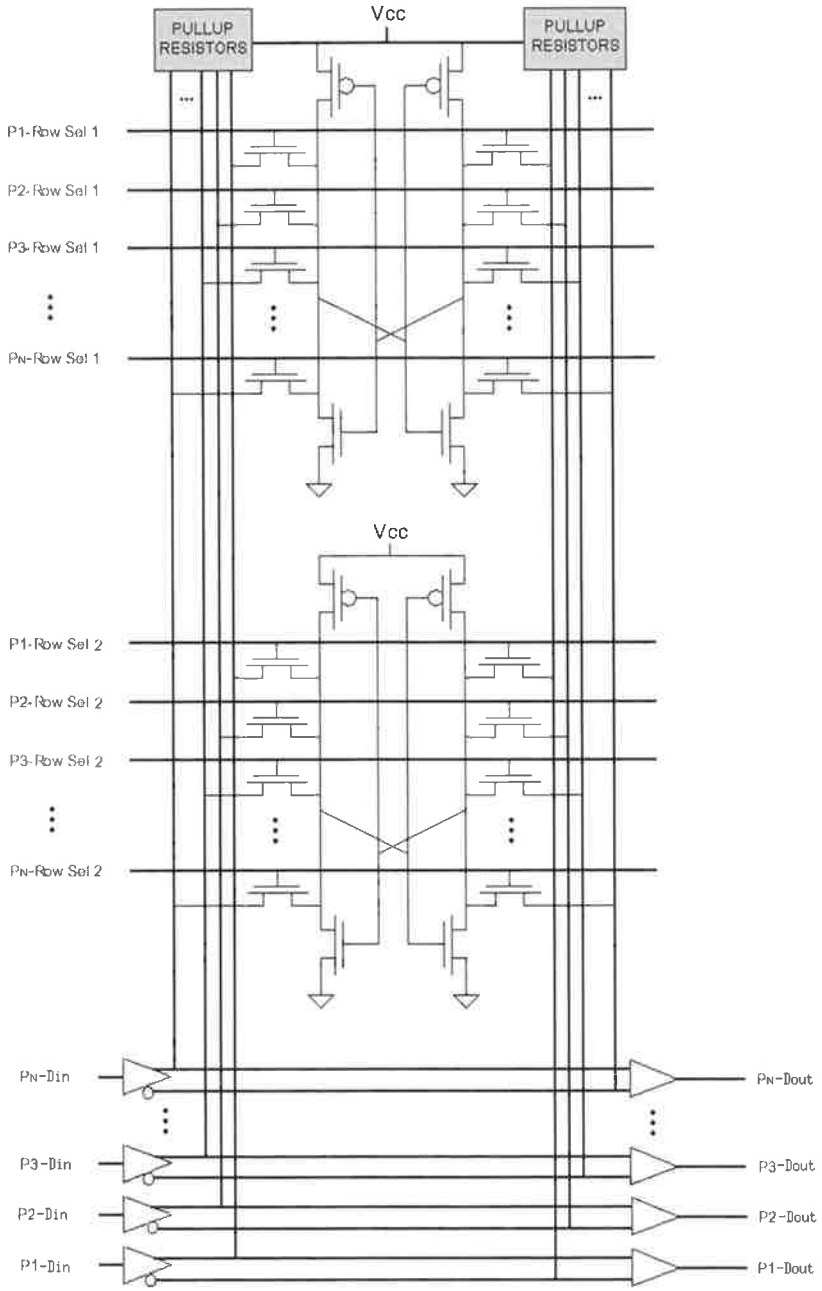

Figure A.8 Architecture of multiport memory

For each port, separate data and address lines are available. Within some constraints, each cell can be accessed independently for reading or writing by the ports.

limited. Nevertheless, the multiport memory is a very useful device and even a small capacity is valuable. Moreover, by using advanced packing technologies such as the

Vertically Expandable Memory (VEM) used by [DensePac], up to eight layers of memory can be packed into a single package. Hence, the limited capacity is not a major issue, especially if the structure using it, such as the communication scheme presented in this research does not require large memories.

4.2 Large number of pinout

As the number of ports of a memory increases, more pins are required to interface to the external devices. However, for the application discussed in this document, the required pinout is under control. Using similar calculations shown in Chapter 3, a 16K, 64-bit 8-port memory would require around 750 pins, which can be easily packed in a single chip using advanced packaging technologies such as the ones used by [Xilinx].

4.3 Control logic for multiport memory

Although the multiport cell was derived from the dual-port cell, extending the dual-port control logic to multiport memories is not trivial. This section discusses some of the issues in designing control logic for multiport memories.

4.3.1 Busy logic

Extending the busy logic to multiport memory is possible but requires considerable hardware. For example, for only three ports, the size of the hardware is tripled. In this case, each pair of address lines requires two comparators and an arbitration latch as illustrated in Figure A.5. As the address lines can be paired in three different ways, three separate circuits are required. For each port, the two busy outputs coming from different circuits should be combined together to form only one busy line for that port. Even with this circuit, if all the ports apply exactly at the same time, all of them may receive a busy signal, as arbitration latches may decide in favor of different ports. Although the probability of all of the ports applying exactly at the same time is very low, this case must be removed by careful latch design, otherwise it can keep all the ports in an endless waiting state. In general, for N ports, N(N-1) arbitration circuits are required and their outputs should be combined properly. In this case, the maximum number of cells that might be accessed by more than one port simultaneously is N/2.

4.3.2 Semaphore logic

Among the range of available control schemes, semaphore logic is the most effective but the hardest one to extend to multiport memories. The present concept works only on two ports and it requires major modifications if it is to be used on multiport memories. In the semaphore logic used for multiport memories, the semaphore latch should be modified to include all the ports and it should take into account the order in which the ports apply for semaphore. There should be also a priority scheme among the ports if more than one port apply at the same time.

Adding the order of applying for semaphore can increase the complexity the semaphore logic. A simplified version can be designed in which a fixed priority is used for the ports that request the semaphore. Figure A.9 illustrates a new proposal for semaphore logic. In this circuit, the shaded priority logic exerts a priority scheme in which Port_1 has the highest and Port_n the lowest priority. Each port requests the semaphore by writing "1" in its input latch. If the semaphore latches are all cleared, the request with the highest priority will be the winner and it write "1" in its semaphore latch. Once one of the latches is set, a NOR gate combined with AND gates at the input of the priority logic disables the other requests from entering the semaphore latches and keeps them in a pending state. The winning node receives "1" in the output and all the losers receive "0". When the winning node writes "0" in the input latch, its semaphore latch is cleared and the highest pending request wins the semaphore.

In this circuit, semaphore latches are clocked by delayed data. When the data input of a latch changes from "0" to "1", a low to high transition is generated at the clock input using the short delay of a buffer. The delay is inserted so that the data in the input settles before the clock is applied. When the request is removed from the input latch, if the semaphore latch is already set, an AND gate connected to the asynchronous reset of the latch resets it to zero. The output latches are included to avoid the change of state while reading the semaphore.

This circuit was simulated and successfully tested with the Xilinx' Foundation Tools.

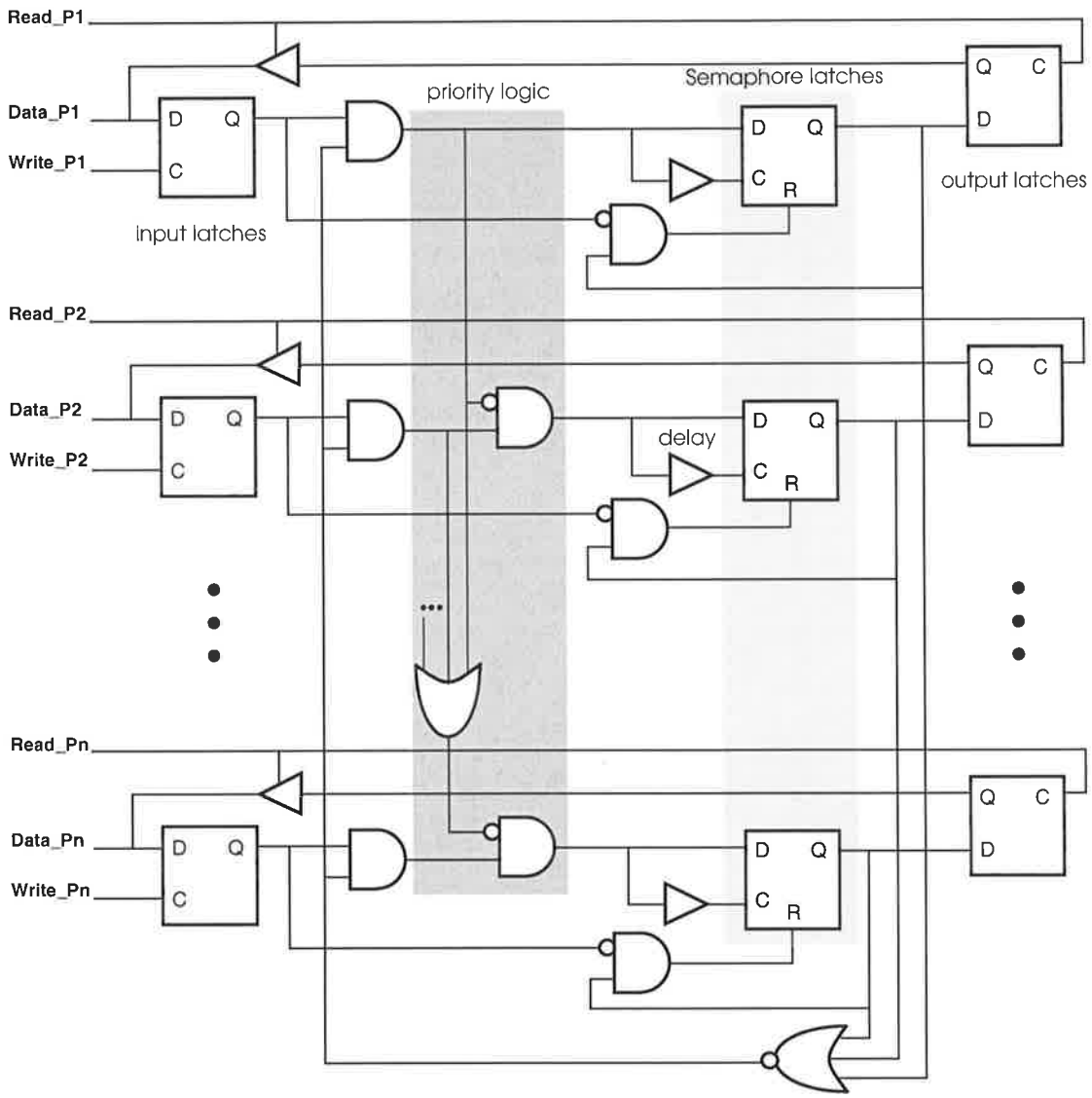

Figure A.9 Semaphore logic for multiport memory

There is a fixed priority among the nodes with Port_1 having the highest and Port_n the lowest priority. The ports register their request by writing "1" in the input latch. When the semaphore latches are free, the highest priority request wins the semaphore. For reading, the winning port will receive "1" at the output. Other pending requests will get "0" at the output as a denial.

4.3.3 Interrupt logic

In the multiport interrupt logic, each port should be able to generate interrupts for other ports, or clear the received interrupts from them. Although the individual interrupt circuits are isolated from each other, several issues should be considered in the design

of an N-port logic. For this logic, N(N-1) memory locations are required, and each processor should use a table to find the address of other processors. For large N, it is necessary to combine all of the interrupts generated for one processor into one line, as a processor may have few interrupt lines, with some allocated to other resources. In this approach, a mechanism is required to identify the source of generated interrupts. In addition, as the received interrupts can be cleared only one by one, a new signal may be included for clearing them all at once. Moreover, some other issues need to be resolved such as conflicts between clearing interrupts and receiving new interrupts at the same time, or receiving new interrupts while the interrupt line is active and other interrupts are being serviced. The latter case may result in missed new interrupts, or excessive delays before servicing them.

A new proposal for multiport interrupt logic is presented here. In order to reduce the size of the required logic and to simplify the operation of the circuit, a different approach from the one discussed for dual-port memory is used. Figure A.10 illustrates the proposed structure for four ports and it can be scaled easily. In this structure, there is no memory location associated with the interrupt generation logic. The number of available addresses that are accessible from each port is equal to the number of available ports on the memory. The chip-enable signal for this circuit can be different from the memory chip-enable. Each processor can send an interrupt to any other processor by writing to the address matching the port number of the destination processor. The write operation does not require any data as it only sets the corresponding interrupt latch. This will create an active high signal for the destination processor on its interrupt line. If a command needs to be sent along with interrupt, it can be passed in the mailbox, which is not in this circuit. For this purpose, each port of an N-port memory will require N-1 dedicated memory locations in the main multiport memory, each regarded as the mailbox of one of the ports. In total, N(N-1) such locations will be required.

All the interrupt latches related to one port share the same interrupt line for sending an interrupt to the processor connected to the port. In order to identify the source of interrupt, the target processor should perform a read from the address matching its port number. In order to avoid probable data change during read operation, the interrupt

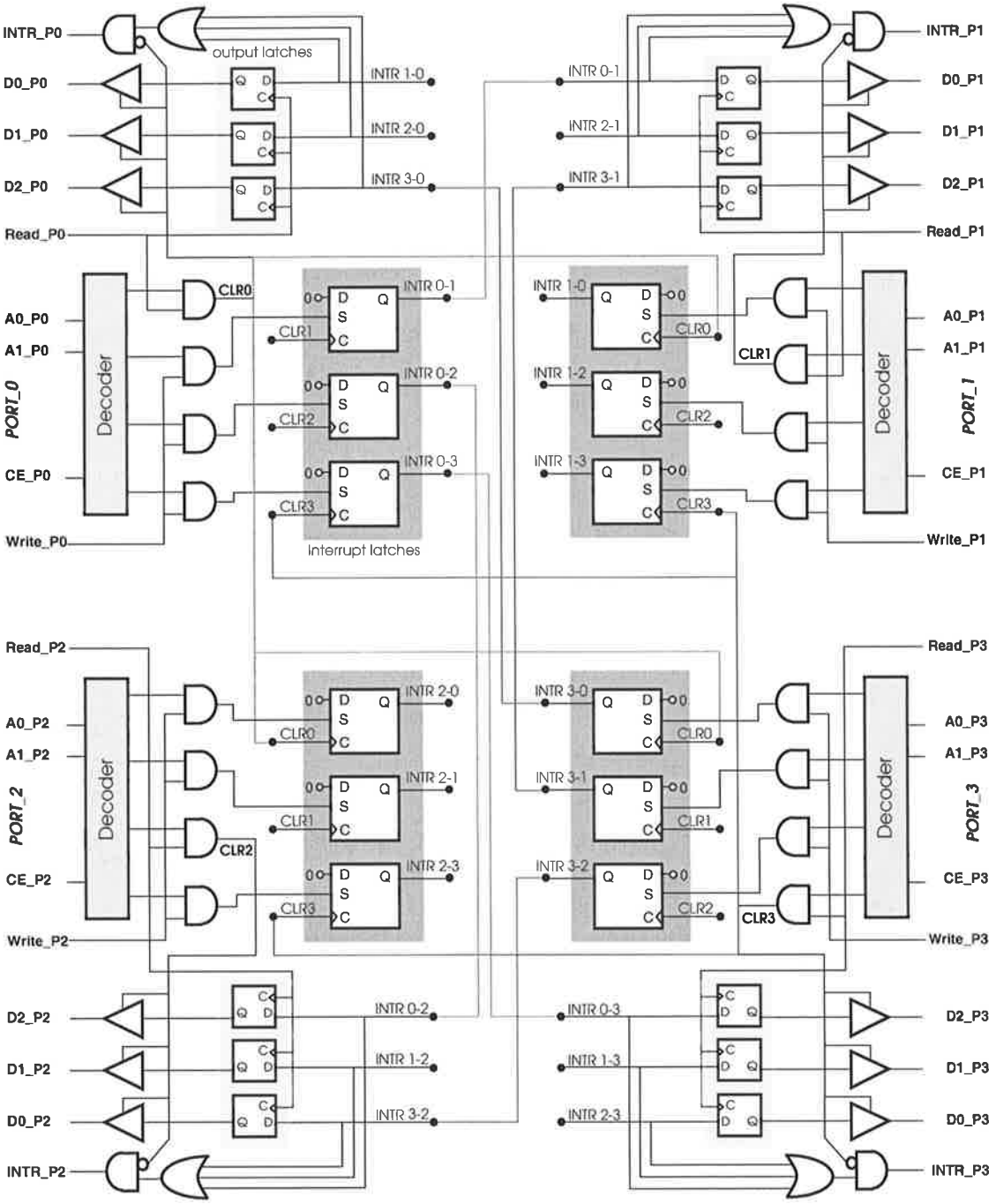

Figure A.10 Interrupt logic for multiport memory

Each port can generate an interrupt for another port by writing in the address that matches the destination port number. The interrupt is stored in an interrupt latch and the interrupt line of the destination port is activated. Reading from the address of each port clears all the generated interrupts for the port and returns the interrupt information on separate bit lines. Only the connecting wires for port_0 and port_3 are drawn to avoid confusion, and the identical wire names should be connected together.

latches of a port are stored in the output latches at the start of read. The interrupt information is available on separate bit lines during reading. An active bit indicates a pending interrupt from the corresponding port.

The start of a read operation also clears all the corresponding interrupt latches and frees the interrupt line. If another interrupt is generated for this port immediately after releasing the interrupt line, the interrupt system of the receiving processor may not be able to detect the short transient on this line. Hence, an AND gate forces the interrupt line to the inactive state during a read operation and provides sufficient time for the interrupt system of the target processor to settle before accepting another interrupt.

The design of the interrupt latch ensures that if an interrupt is generated during a read operation, the destination port will not miss it or receive it in duplicate. As shown in Figure A.11, this latch is cleared by clocking "0" using the rising edge of read operation, but it is set through the asynchronous SET input. By using an AND gate, the latch output is available when the set signal goes inactive, as if setting of the latch is performed by the trailing edge of write operation. A delay buffer is used in the SET input with the delay value matching the delay of the latch plus the AND gate. This structure ensures that in clearing all the interrupts, if the destination port resets a latch while the originating port is setting it, both of the operations will be performed without any conflict. In this case, the read operation will not reflect the newly generated

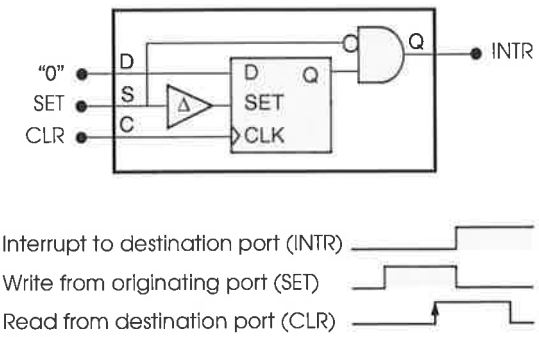

Figure A.11 Interrupt latch design

The destination port resets the latch by clocking "0" in the rising edge of read, whereas the originating port sets it by the asynchronous SET. The latch output is gated to be available when writing is finished. A read will not conflict with a write, even if performed at the same time.

interrupt. Instead, the port will receive another interrupt by re-activation of the interrupt line, and the subsequent read will show the interrupt information.

The interrupt circuit was simulated and successfully tested with the Xilinx' Foundation Tools.

4.4 Simultaneous read of a cell

A common problem in multiport memory is the simultaneous write of a cell by more than one port. In addition, simultaneous reading and writing of a cell can cause problems. Both of these issues should be resolved by the use of hardware or software tools, as explained earlier. Another issue in the design of multiport memory is the simultaneous read of a cell by many ports. Although there is no conflict in this case, the issue should be considered in the design of multiport memories.

The problem arises from the loading effect on the memory cell. Referring to Figure A.8, if many sense amplifiers attempt to read the same cell, the sink or source current of the cell will increase resulting in a poor logic state at the cell output. Hence, the output transistors of the cell should be strong enough to sink or source the required current. This may add to the complexity of the cell and a bigger area may be required. Furthermore, the capacitive loading of the sense amplifiers can also affect the logic state of the cell output and should be kept to a minimum.

5 Summary

In this Appendix, the structures of single-port and dual-port memories were presented and different hardware controls for dual-port memories were discussed. The structure of the multiport memory was shown to be an extension of the dual-port memory structure.

As currently there is no control circuitry for multiport memory, newly devised practical circuits have been proposed for easier control of multiport memories. The new semaphore logic and interrupt logic for multiport memories were tested by hardware design tools.

Another option for the control of multiport memory is the use of a central off-chip controller. In this method, as discussed in Chapter 8, different types of control circuitry are integrated into a single chip that can be connected to the main memory chip externally. Even the buffer allocation process can be integrated into this chip. The advantages are faster control of the memory and utilization of the entire memory chip for memory cells. Moreover, as stated earlier, in vertical or horizontal memory expansion performed by connecting several chips in series or in parallel, there will be no duplicate control circuitry, which would normally be left idle.

Bibliography

[Ahmad+ 93] Ahmad I and Chen C Y R, **"Datapath Synthesis Using Onchip Multiport Memories"**, *IEE Proceedings E: Computers and Digital Techniques,* vol. 140, no. 4, pages 227-32, July 1993.

[Almasi+ 89] Almasi G S and Gottlieb A, ***Highly Parallel Computing***, Benjamin Cummings Publishers, Menlo Park, USA, 1989.

[Asato+ 95] Asato C, Montoye R, Gmuender J, Wade Simmons E, Ike A, and Zasio J, **"A 14-port 3.8 ns 116-word 64b Read-Renaming Register File"**, in *Proceedings of International Solid-State Circuit Conference (ISSCC'95),* San Francisco, USA, pages 104-5, 345, and 440, February 1995.

[Asgari+ 01] Asgari N and Burgess N, **"Multiport Memory for High-Speed Interprocessor Communication in MultiCom"**, *the International Journal of Science and Technology, Scientia Iranica*, Sharif University of Technology, Tehran, Iran, vol. 8, no. 4, pages 322-31, October 2001.

[Asgari+ 98] Asgari N and Burgess N, **"The Structure and Design of MultiCom, a Small Multiprocessor with Multiport Memory Based Communication"**, in *Proceedings of the Third Australasian Computer Architecture Conference (ACAC'98),* Perth, Australia, pages 15-24, Springer-Verlag, February 1998.

[Asgari+ 99a] Asgari N and Burgess N, **"Memory Management and Dynamic Allocation of MultiCom"**, in *Proceedings of the Forth Annual International CSI Computer Conference (CSICC'98),* Tehran, Iran, pages 72-81, January 1999.

[Asgari+ 99b] Asgari N and Burgess N, **"Interprocessor Communication in Tree Structured Multiprocessors Using Multiport Memory"**, in *Proceedings of the Forth Australasian Computer Architecture Conference (ACAC'99),* Auckland, NZ, pages 161-72, Springer-Verlag, January 1999.

[Bakoglu+ 90] Bakoglu H B, Grohoski G F, and Montoye R K, **"The IBM RISC Sytem/6000 Processor: Hardware Overview"**, *IBM Journal of Research and Development,* vol. 34, no. 1, pages 12-22, January 1990.

[Baumann 96] Baumann M, **"The Most Commonly Asked Questions About Dual Ports"**, Application Note AN-91, Integrated Device Technology, Santa Clara, USA, 1996.

[Ben-Ari 82] Ben-Ari M, ***Principles of Concurrent Programming***, Prentice Hall, Englewood Cliffs, NJ, USA, 1982.

[Bhuyan+ 84] Bhuyan L N and Agrawal D P, **"Generalized Hypercube and Hyperbus Structures for a Computer Network"**, *IEEE Transaction on Computers*, vol. C-33, no. 4, pages 323-33, April 1984.

[Boianov+ 91] Boianov L K and Knowles A E, **"Higher Speed Transputer Communication Using Shared Memory"**, *Microprocessors and Microsystems,* vol. 15, no. 2, pages 67-72, March 1991.

[Campbell+ 96] Campbell S and Kumar M J, **"The Design and Development of the COMPS Architecture"**, in *Records of the Australian Computer Architecture Workshop (ACAW'96),* Melbourne, Australia, pages 177-87, January 1996.

[Castro+ 92] Castro Alves V, Lubaszewiski M S, Nicolaidis M, and Courtois B, **"Testing Embedded Single and Multiport RAMs Using BIST and Boundary Scan"**, in *Proceedings of the European Conference on Design Automation,* Brussels, Belgium, pages 159-63, March 1992.

[Chin+ 96] Chin-Chien C and Wooley B A, **"A 1.3-ns 32-word*32-bit Three-port BiCMOS Register File"**, *IEEE Journal of Solid-State Circuits*, vol. 31, no. 6, pages 758-66, June 1996.

[Culler+ 98] Culler D, Singh J P, and Gupta A, ***Parallel Computer Architecture: A Hardware/Software Approach***, Morgan Kaufmann Publishers, San Mateo, USA, 1998.

[Cypress 96] Cypress Semiconductor Corporation, **"Understanding Dual-port RAMs"**, Cypress Semiconductor Corporation, San Jose, USA, 1996.

[Dedrick 84] Dedrick J H, **"Multiport Register File Simplifies and Speeds up Digital Signal Processing"**, *Electronic Design*, vol. 32, no. 10, pages 213-22, May 1984.

[DeMara+ 91] DeMara R F and Moldovan D I, **"Design of a Clustered Multiprocessor for Real-time Natural Language Understanding"**, in *Proceedings of the Fifth International Parallel Processing Symposium,* Anaheim, USA, pages 270-7, April-May 1991.

[DeMara+ 93] DeMara R F and Moldovan D I, **"The SNAP-1 Parallel AI Prototype"**, *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, no. 8, pages 841-54, August 1993.

[DensePac] DPAC Technologies Corporation (formerly Dense-Pac Microsystems Incorporation), CA, USA, web address ***www.dense-pac.com***

[ElGuibaly+ 96] El Guibaly F, Sabaa A, and Shpak D, **"A New Shift-register Based ATM Switch"**, in *Proceedings of the First Annual Conference on Emerging Technologies and Applications in Communications*, Portland, USA, pages 24-7, May 1996.

[Elliot+ 89] Elliot D and Wenniger B, **"Dual-port RAMs; 'Unspecified Operation' "**, in *Wescon/89 Conference Records,* San Francisco, CA, pages 125-30, November 1989.

[Endo+ 91] Endo K, Matsumura T, and Yamada J Y, **"Pipelined, Time-Sharing Access Technique for an Integrated Multiport Memory"**, *IEEE Journal of Solid-State Circuits*, vol. 26, no. 4, pages 549-54, April 1991.

[Feng 81] Feng T, **"A Survey of Interconnection Networks"**, *IEEE Computer,* vol. 14, no. 12, pages 12-27, December 1981.

[Forsell 94] Forsell M J, **"Are Multiport Memories Physically Feasible?"**, *Computer Architecture News,* vol. 22, no. 4, pages 47-54, September 1994.

[Franch+ 97] Franch R L, Ji J, and Chen C L,**"A 640-ps, 0.25-μm CMOS, 16*64-b Three-port Register File"**, *IEEE Journal of Solid-State Circuits*, vol. 32, no. 8, pages 1288-92, August 1997.

[Gajski+ 83] Gajski D, Kuck D, Lawrie D, and Sameh A, **"Cedar- A Large Scale Multiprocessor"**, in *Proceedings of the International Conference On Parallel Processing,* Bellaire, MI, USA, pages 524-9, August 1983.

[Gaughan+ 93] Gaughan P T and Yalamanchili S, **"Adaptive Routing Protocols for Hypercube Interconnection Networks"**, *IEEE Computer,* vol. 26, no. 5, pages 12-23, May 1993.

[Hana+ 89] Hana H H and Hussain S J, **"High Speed Multi-port Static RAM Silicon Compiler"**, in *Proceedings of the IEEE 1989 Custom Integrated Circuit Conference*, San Diego, USA, pages 23.6.1-4, May 1989.

[Handy 90] Handy J, **"Specialty Static RAMs Fuel Development of Higher Performance Architectures"**, in *Electro International Conference Records,* New York, USA, pages 214-8, April 1991.

[Hayes+ 86] Hayes J P, Mudge T, Stout Q F, Colley S, and Palmer J, **"A Microprocessor-Based Hypercube Supercomputer"**, *IEEE Micro,* vol. 6, no. 5, pages 6-17, October 1986.

[Hennessy+ 94] Hennessy J and Patterson D, ***Computer Organization and Design, The Hardware/Software Interface***, Morgan Kaufmann Publishers, San Mateo, USA, 1994.

[Hirano+ 98] Hirano K, Ono T, Kurino H, and Koyanagi M, **"A New Multiport Memory for High Performance Parallel Processor System with Shared Memory"**, in *Proceedings of the 1998 Asian South Pacific Design Automation Conference (ASP-DAC'98)*, Yokohama Japan, pages 333-4, February 1998.

[ICmaster 99] *IC Master*, Hearst Business Communications Incorporation, Garden City, NY, USA, 1999.

[IDT] Integrated Device Technology, Santa Clara, USA, Multiport memory products website
www.idt.com/products/pages/Multi-Ports.html

[IDT 02] Integrated Device Technology, **"IDT Introduces Industry's Fastest and Lowest-Powered Synchronous FourPort Devices"**, Integrated Device Technology, Santa Clara, USA, Company Press Release, September 2002.
www.idt.com/news/Sep02/09_09_02_1.html

[IDT 95] Integrated Device Technology, **"High-Speed 2K x 8 FourPort Static RAM"**, Data Sheet for IDT7052S/L, Integrated Device Technology, Santa Clara, USA, August 1995.

[IDT 96] Integrated Device Technology, **"High-Speed 4K x 8 FourPort Static RAM"**, Preliminary Data Sheet for IDT7054S/L, Integrated Device Technology, Santa Clara, USA, 1996.

[IDT 99] Integrated Device Technology, **"IDT Introduces Industry's Fastest 36-bit Dual-Port"**, Integrated Device Technology, Santa Clara, USA, company press release, October 1999.
www.idt.com/news/Oct99/10_18_99_1.htm

[Inmos 88] INMOS Limited, ***Transputer Reference Manual***, Prentice Hall, New York, USA, 1988.

[Izumikawa+ 96] Izumikawa M and Yamashina M, **"A Current Direction Sense Technique for Multiport SRAM's"**, *IEEE Journal of Solid-State Circuits*, vol. 31, no. 4, pages 546-51, April 1996.

[Jagadish+ 89] Jagadish N, Kumar M J, and Patnaik L M, **"An Efficient Scheme for Interprocessor Communication Using Dual-ported RAMs"**, *IEEE Micro,* vol. 9, no. 5, pages 10-19, October 1989.

[Khan+ 93] Khan G N and Mahmud K, **"Scalable and Fault Tolerant Restricted Shared Memory"**, *Electronics Letters,* vol. 29, no. 9, pages 783-5, April 1993.

[Khan+ 94] Khan G N, Mahmud K, Iqbal M S, and Rashid H U, **"RSM - a Restricted Shared Memory Architecture for High Speed Interprocessor Communication"**, *Microprocessors and Microsystems,* vol. 18, no. 4, pages 193-203, May 1994.

[Kornaros+ 97] Kornaros G, Kozyrakis C, Vatsolaki P, and Katevenis M, **"Pipelined Multi-queue Management in a VLSI ATM Switch Chip with Credit-based Flow-control"**, in *Proceedings of the Seventh Conference on Advanced Research in VLSI*, Ann Arbor, USA, pages 127-44, September 1997.

[Kuskin+ 94] Kuskin J, Ofelt D, Heinrich M, Heinlein J, Simoni R, Gharachorloo K, Chapin J, Nakahira D, Baxter J, Horowitz M, Gupta A, Rosenblum M, Hennessy J, **"The Stanford FLASH Multiprocessor"**, *Computer Architecture News,* vol. 22, no. 2, pages 302-13, April 1994.

[Labrousse+ 90] Labrousse J and Slavenburg G A, **"A 50MHz Microprocessor with a Very Long Instruction Word Architecture"**, in *Digest of Technical Papers, 37th IEEE International Solid-State Conference (ISSCC 90),* San Francisco, USA, pages 44-5, February 1990.

[Lai+ 94] Lai F, Chung Y L, and Chen S J, **"A New Design Methodology for Multiport SRAM Cell"**, *IEEE Transactions on Circuit and Systems*, vol. 41, no. 11, pages 677-85, November 1994.

[Lamport 91] Lamport L, **"The Mutual Exclusion Problem Has Been Solved"**, *Communications of the ACM*, vol. 34, no. 1, pages 110-1, January 1991.

[Landsberg+ 93] Landsberg P, Tretz C, and Zukowski C, **"An Efficient Macromodel for Static Multi-port Memories"**, in *Proceedings of the 1993 IEEE Custom Integrated Circuit Design Conference*, San Diego, USA, pages 8.2.1-4, May 1993.

[Lee+ 95] Lee H D and Hwang S Y, **"A Scheduling Algorithm for Multiport Memory Minimization in Datapath Synthesis"**, in *Proceedings of the Asia and South Pacific Design Automation Conference / IFPI International Conference on Hardware Description Languages / IFPI Conference on VLSI,* Chiba, Japan, pages 93-100, August-September 1995.

[Lenoski+ 91] Lenoski D, Laudon J, Gharachorloo K, Weber W D, Gupta A, and Hennessy J, **"Overview and status of the Stanford DASH multiprocessor"**, in *Proceedings of the International Symposium on Shared Memory Multiprocessing*, Tokyo, Japan, pages 102-8, April 1991.

[Lin+ 96] Lin T, Lin J, and Chung Y, **"Using the IDT7050/7052 FourPort SRAM in DSP and Matrix Processing Applications"**, Application Note AN-42, Integrated Device Technology, Santa Clara, USA, 1996.

[Litazie+89] Litazie D, Elkhlifi F, Hammami O, Lalam M, Mzoughi A, Sainrat P, and Salinier J C, **"Serial Multiport Memory Multiprocessors"**, in *Proceedings of Parallel Architectures and Language Europe (PARLE'89)*, Eindhoven, Netherlands, vol. 1, pages 34-51, June 1989.

[Lum+ 92] Lum M, Brandt J, Vojta M, Wehner W, and Belt R, **"Aladdin: A High Performance, Distributed Memory Multiprocessor"**, in *Proceedings of the 25th Hawaii International Conference on System Sciences*, Kauai, USA, pages 181-9, January 1992.

[Maly+ 91] Maly W, Patyra M, Primatic A, Raghavan V, Storey T, and Wolfe A, **"Memory Chip for 24-Port Global Register File"**, in *Proceedings of the IEEE Custom Integrated Circuit Conference*, San Diego, USA, pages 15.5.1-4, May 1991.

[Mandal+ 96] Mandal C, Zimmer R M, Tewksbury S, and Chapman G, **"Use of Multi-port Memories in Programmable Structures for Architectural Synthesis"**, in *Proceedings of the Eighth Annual IEEE International conference on Innovative Systems in Silicon,* Austin, USA, pages 341-51, October 1996.

[Matsumura 95] Matsumura T, **"An Efficient Method for Embedded Multi-port RAM with BIST Circuitry"**, in *Records of the 1995 IEEE International Workshop on Memory Technology, Design and Testing,* San Jose, USA, pages 62-7, August 1995.

[McGeady 90] McGeady S, **"Inside Intel's i960CA Superscalar Processor"**, *Microprocessor and Microsystems*, vol. 14, no. 6, pages 385-96, July-August 1990.

[Mick 96] Mick J R, **"Introduction to IDT's FourPort RAM"**, Application Note AN-45, Integrated Device Technology, Santa Clara, USA, 1996.

[Mudge+ 87] Mudge T N, Hayes J P, and Winsdoe D C, **"Multiple Bus Architectures"**, *IEEE Computer,* vol. 20, no. 6, pages 42-8, June 1987.

[Mzoughi+ 93] Mzoughi A and Litazie D, **"The Processor Board of Multiprocessor M3S: Specific Aspects, Implementation Details"**, *International Journal of Mini and Microcomputers*, vol. 15, no. 2, pages 82-9, 1993.

[Nakamura+ 96] Nakamura K, Sakai K, and Ae T, **"A VILIW Processor for Real-Time Signal Processing"**, in *Proceedings of the Fourth International Symposium on Signal Processing and Its Applications (ISSPA 96)*, Gold Coast, Australia, vol. 2, pages 664-7, August 1996.

[Nanduri 91] Nanduri BVR, **"Specialty Memories Increase Performance and Reduce Cost in High Speed System Designs"**, in *Wescon Conference Records*, San Francisco, USA, pages 116-21, November 1991.

[Nii+ 95] Nii K, Maeno H, Osawa T, Twade S, Kayano S, and Shibata H, **"A Novel Memory Cell for Multiport RAM on 0.5 μm CMOS Sea-Of-Gates"**, *IEEE Journal of Solid-State Circuits*, vol. 30, no. 3, pages 316-20, March 1995.

[Patterson+ 98] Patterson D and Hennessy J, ***Computer Organization and Design, The Hardware/Software Interface***, Second Edition, Morgan Kaufmann Publishers, San Mateo, USA, 1998.

[Pryce 89] Pryce D, **"Dual-port Static RAMs. Specialized Memories Ease Communications"**, *EDN,* vol. 34, no. 8, pages 83-9, April 1989.

[Shibayama+ 87] Shibayama S, Sakai H, Monoi H, Morita Y, and Itoh H, **"Mu-X: An Experimental Knowledge Base Machine with Unification-Based Retrieval Capability"**, in *Proceedings of the Second Franco-Japanese Symposium*, Cannes, France, pages 347-64, November 1987.

[Shinohara+ 91] Shinohara H, Matsumoto N, Fujimori K, Tsujihashi Y, Nakao H, Kato S, Horiba Y, and Tada A, **"Flexible Multiport RAM Compiler for Data Path"**, *IEEE Journal of Solid-State Circuits*, vol. 26, no. 3, pages 343-8, March 1991.

[Silburt+ 93] Silburt A L, Phillips R S, Gibson G F R, Wood S W, Bluschke A G, Fujimoto J S, Kornachuck S P, Nadeau-Dostie B, Vemra R K, and Diedrich P M J, **"A 180-MHz 0.8 μm BiCMOS Modular Memory Family of DRAM and Multiport SRAM"**, *IEEE Journal of Solid-State Circuits*, vol. 28, no. 3, pages 222-32, March 1993.

[Stallings 98] Stallings W, ***Operating Systems, Internal Design and Principles,*** Third Edition, Prentice Hall, Englewood Cliffs, NJ, USA, 1998.

[Stodieck 96] Stodieck R, **"The IDT FourPort RAM Facilitates Multiprocessor Design"**, Application Note AN-43, Integrated Device Technology, Santa Clara, USA, 1996.

[Strohman+ 89] Strohman C R and Peck S B, **"Architecture and Performance of New CESR Control System"**, in *Proceedings of the 1989 IEEE Particle Accelerator Conference,* Chicago, USA, vol. 3, pages 1687-9, March 1989.

[Su+ 92] Su S-C and Biswas P, **"A Memory-mapped Interprocessor Communication Architecture Using FIFO RAMs"**, *Microprocessor and Microprogramming*, vol. 34, no. 1, pages 187-92, February 1992.

[Tabak 90] Tabak D, ***Multiprocessors***, Prentice Hall, Englewood Cliffs, NJ, USA, 1990.

[Tanenbaum 01] Tanenbaum A S, ***Modern Operating Systems***, Second Edition, Prentice Hall, Englewood Cliffs, NJ, USA, 2001.

[TI 96] Texas Instruments Incorporated, ***TMS320C5X DSP Starter Kit User's Guide,*** Texas Instruments Incorporated, USA, 1996.

[TI 97] Texas Instruments Incorporated, ***TMS320C5X User's Guide,*** Texas Instruments Incorporated, USA, 1997.

[Tuazon +85] Tuazon J, Peterson J, Prifet M, and Liberman D, **"Caltech/JPL Mark II Hypercube Concurrent Processor"**, in *Proceedings of the 1985 IEEE International Conference on Parallel Processing*, IEEE CS Press (microfiche), Los Alamitos, USA, Pages 666-78, August 1985.

[Uvieghara+ 90] Uvieghara G A, Nakagome Y, Jeong D K, and Hodges D A, **"An On-Chip Smart Memory for a Data-Flow CPU"**, *IEEE Journal of Solid-State Circuits,* vol. 25, no. 1, pages 84-94, February 1990.

[Varshneya+ 94] Varshneya A, Madan B B, and Balakrishnan M, **"Memory Coupled Scalable Multiprocessors"**, in *Proceedings of the First International Workshop on Parallel Processing (IWPP-94),* Bangalore, India, pages 424-9, December 1994.

[Wyland 88] Wyland D C, **"Dual-port RAMs Simplify Processor Communication"**, *Microprocessors and Microsystems,* vol. 12, no. 10, pages 585-94, December 1988.

[Xilinx] Xilinx, San Jose, USA, package information website ***www.xilinx.com/publications/products/packaging/pkg_info.htm***

[Yuejian+ 97] Yuejian W and Gupta S, **"Built-in Self-test for Multiport RAMs"**, in *Proceedings of the Sixth Asian Test Symposium (ATS'97)*, Akita, Japan, pages 398-403, November 1997.

[Zhang+ 95] Zhang J and Ma Y, **"Extending Method of Neurocomputer with a Shared Multi-port Memory"**, in *Proceedings of International Conference on Neural Information Processing (ICONIP'95)*, Beijing, China, vol. 2, pages 589-92, October-November 1995.

[Zhi+ 96] Zhi L, Smith E D, and Kwatra S C, "**A Compact Cell Design for a Multiport Register File"**, in *Proceedings of the 39th Midwest Symposium on Circuits and Systems,* Ames, USA, vol. 3, pages 1231-4, August 1996.