Scientia Iranica, Vol. 8, No. 4, pp 322–331

SCIENTIA
IRANICA

Research Note

Multiport Memory for High-Speed Interprocessor Communication in MultiCom

N. Asgari* and N. Burgess[1]

MultiCom is a small multiprocessor designed to check the efficiency of an interprocessor communication scheme based on multiport memories. This paper reviews the structure of MultiCom and discusses the shared memory management using both static and dynamic allocations. Two new software locks designed to control the ownership of the shared memory in dynamic allocation are presented and broadcasting is evaluated. The measured communication rate is 45.5 and 45.8 Mbytes/s for static and dynamic allocations, respectively, showing an 11 times improvement compared to a system using dual-port memories. The communication rate for the broadcasting is measured as 68.2 Mbytes/s and demonstrates even greater achievement in performance. In addition, the structure can be implemented even with a small shared memory and enjoys a four-fold reduction in the cost compared to a system using dual-port memories.

INTRODUCTION

Multiprocessors need a very high bandwidth for their interprocessor communications and widely use serial links for this purpose. In loosely coupled multiprocessors, each node is connected to the neighboring nodes using static links according to the topology of the system such as ring-mesh or hypercube [1]. Beside the slow nature of serial communication, i.e., data is sent bit by bit using a clock, this method may suffer from the complexity of the interconnection network. In addition, a message may need to pass many intermediate nodes to reach the destination. The links can also be created dynamically by setting active switching elements of the network. Common types of networks in this category use cross bar or multi-stage switches [1].

Using parallel links to send all bits of data at once has been studied as an alternative for a long time. A simple way of establishing a parallel link is to create a shared memory among the nodes by using a single-port memory and a bus. The structure suffers from the fact that only one node can use the memory at a time and the other nodes have to wait for their turn. Hence, it is limited to systems with small number of nodes. Connecting two microprocessors was an early attempt to use this structure [2]. In another attempt, a cluster of four transputers was constructed using a single-port memory [3].

The advent of Dual-Port Memory (DPM) introduced new techniques for interprocessor communication and many structures were proposed on this basis [4,5]. In a structure that uses dual-port memories, each node shares a DPM with its neighboring nodes. Data is written to a DPM by one node and is read by the other node. DPM has been used in the design of a few small systems [6-8].

One of the problems in this method is that the number of DPMs used is high. For example, for eight nodes arranged in a cube and a network controller (NC) of Figure 1, 20 modules of DPM are required. Even with this many DPMs, the communication between some nodes is not direct and requires the use of one or two other nodes as shown by the bold or dotted arrows. Moreover, the restriction of having only two ports for memory places an upper bound on this kind of structure and limits its application to small systems.

Multiport memory (MPM) is a better alternative for interprocessor communication. In this paper, the structure and design of MultiCom, a small multiprocessor with a communication scheme based on MPM, is reviewed and its memory management using both static and dynamic allocations is presented.

*. *Corresponding Author, School of Informatics and Engineering, Flinders University of South Australia, G.P.O Box 2100, Adelaide 5001, Australia.*

1. *Division of Electronics, School of Engineering, Cardiff University, Queen's Buildings, The Parade, P.O. Box 689, Cardiff CF2 3TF, Wales, UK.*

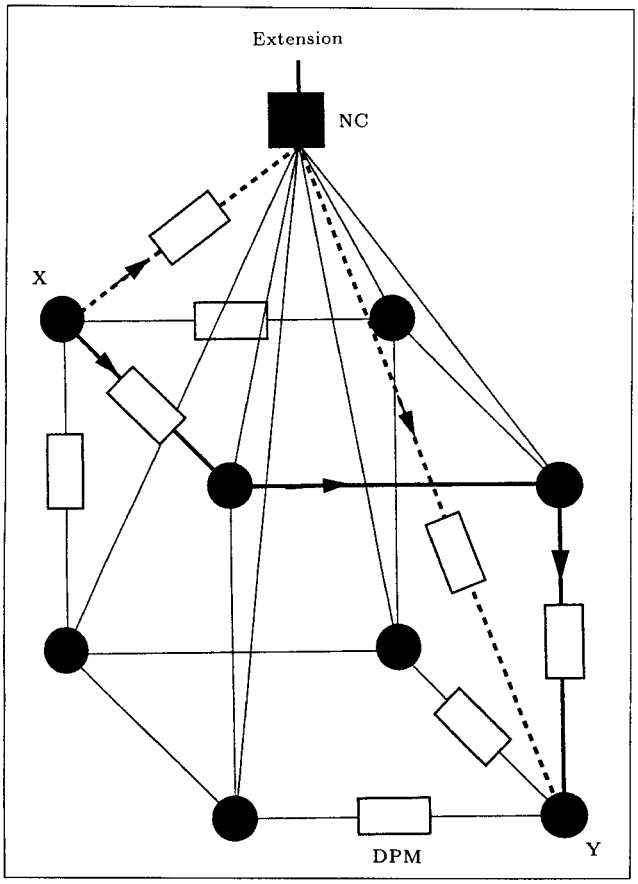

Figure 1. Communication with dual-port memory.

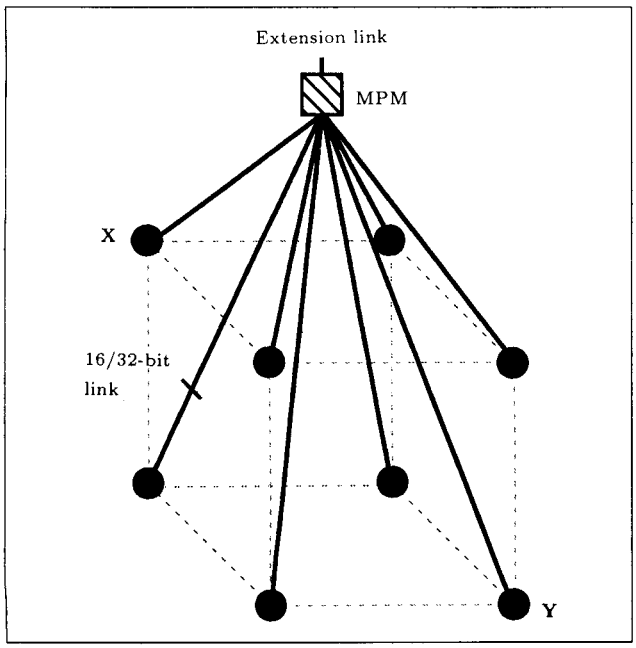

Figure 2. Communication with MPM.

MULTIPORT MEMORY FOR COMMUNICATION

Multiport memory can increase the performance of a multiprocessor system and reduce the cost. Figure 2 shows the full connection of 8 nodes through a 9-port memory. The extra port can be used as an extension link. Any node can send a message to others simply by writing the message into the shared memory. The others can receive the message by reading the memory. This structure offers the best solution in terms of cost and performance.

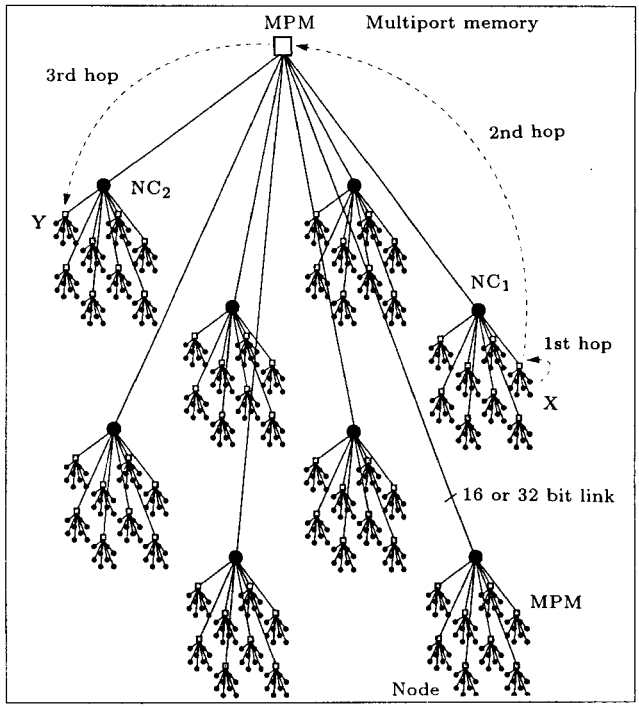

Figure 3. Scaling the structure to 512 nodes.

The feasibility of multiport memories has been studied in [9]; however, the structure has not been realized yet and multiport memory with large port numbers is not available on the market. Fortunately, the advent of 4-port memories, currently available in 4 Kbytes, shows that the availability of this kind of memory with larger port numbers is not far away. As the number of ports increases, the pinouts grow rapidly; however, a 2 Kbyte 9-port memory will require around 200 pins, which is easily achievable nowadays.

This structure can be extended to a large number of processors. Figure 3 shows a system of 512 nodes, in which 65 blocks of 9-port memory and eight Network Controllers (NC) are used. As illustrated, the communication of the furthest nodes such as X and Y can be performed in three hops: X writes to shared memory; NC_1 copies it to the next shared memory in the hierarchy; and finally, NC_2 transfers the message into the shared memory of the node Y, where it can be picked up. For a similar situation, a hypercube of order 9, using store and forward, would require 9 serial hops. Moreover, if the structure were to be implemented by dual-port memories, requiring 1460 blocks of dual-port memory and 73 NCs, it would require 6 hops.

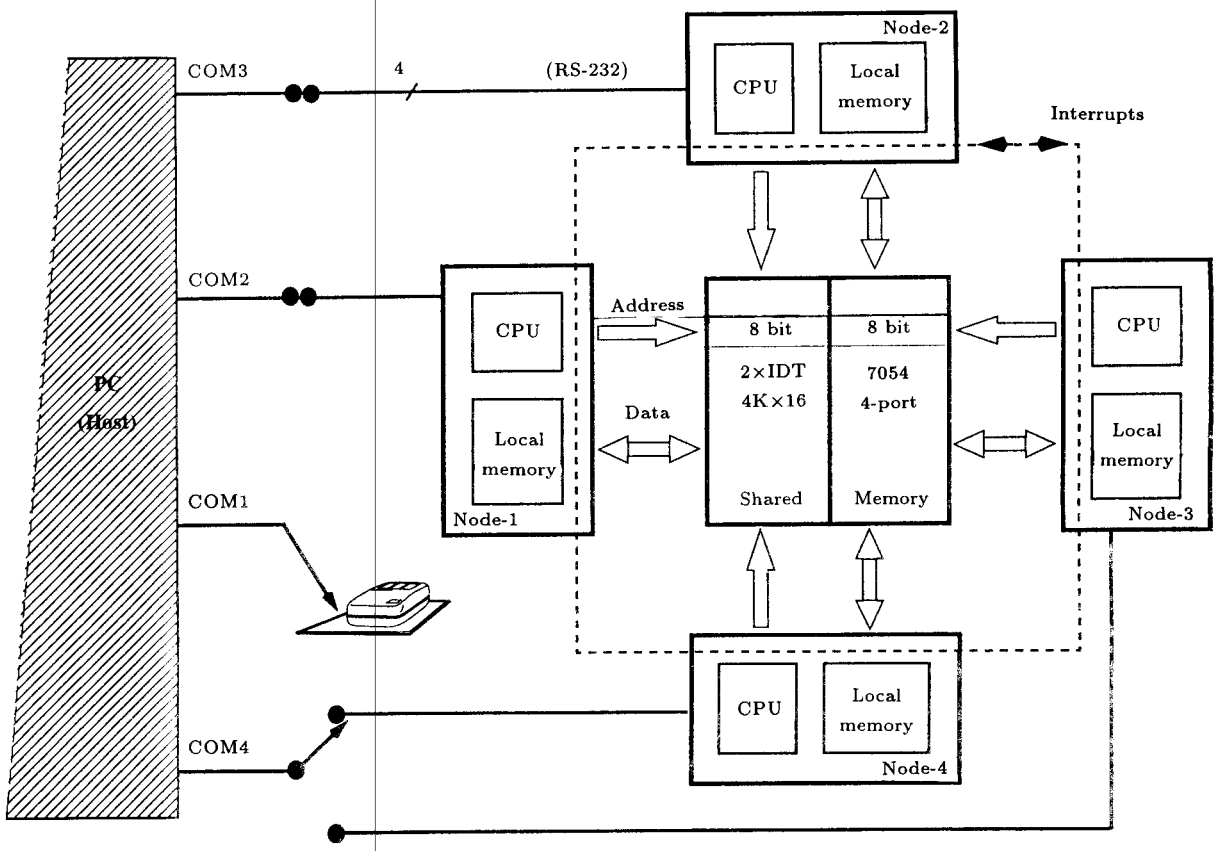

Figure 4. The block diagram of MultiCom.

The expected rise in performance and reduction in cost and the number of links would be very high. As the first stage in evaluating the proposed structure, a small multiprocessor called MultiCom was designed and built. In this model, four nodes were interconnected using 4-port memories as the communication medium and static allocation was tested on it [10]. More complicated shared memory management and broadcasting are presented in this paper. Using simulation techniques, a performance model was also constructed to evaluate the performance of the system in larger structures. The preliminary results are presented in [11] and more simulations are currently under development.

Hardware Design of MultiCom

The nodes of MultiCom were TMS320C50 DSP processors from Texas Instruments (TI) running with a cycle time of 50 ns (20 MHz). Each node had 10K 16-bit on-chip RAM, one 16-bit timer and four external interrupts. The nodes were connected to a PC acting as their host via RS232 ports. The TI assembler and debugger were used for programming and debugging.

The 4-port memory used in MultiCom was IDT7054 from Integrated Device Technology (IDT) with an access time of 35 ns. This memory is organized as a 4 Kbyte array and has four independent ports. Each port has its own control, address and data buses. The memory array can be accessed through different ports simultaneously, but there are two limitations:

- Simultaneous writes to the same location should be avoided;
- Simultaneous read from a location that is being written may return incorrect data.

For reliable memory operation, these limitations should be considered in the design of any system.

The block diagram of MultiCom is shown in Figure 4. In this diagram, four nodes are interconnected by two IDT7054 memories. Each node is connected to one of the serial ports of a PC acting as a host. The interface circuit consists of the required logic to connect the 4-port memory to each node and an interrupt bus. Each node generates three interrupts for the other three nodes and receives three interrupts from them. The interrupts are used as part of the handshaking for the communication through 4-port memory.

The on-chip timer was used to measure the time spent on communication of the nodes from beginning until all the nodes finished. This measurement was the basis of the achieved results. In order to get a correct time measurement, all the nodes should

be synchronized to start at the same time. The synchronization was performed by the aid of the shared memory.

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. In this program, each node sends 1792 words (700 hex) to the other three nodes through the shared memory. The message size of 700 hex was chosen by considering the capacity of the internal memory of the nodes.

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 (Tx) allocates a buffer for transmission and writes the proper message size in the buffer. Then, it transfers the data into the buffer and generates an interrupt for the receiving node. The proper message size is less than or equal to the maximum buffer length in the shared memory.

Upon receiving the interrupt, the receiver (Rx) identifies the transmitter and refers to the appropriate buffer in the shared memory. It reads and transfers the data to its local memory and appends it to the previous data received from the same transmitter. At the end, it activates a flag in the shared memory to inform the transmitter that the node is ready again to receive more data from that transmitter. This protocol shows that the handshaking between the Tx and Rx is performed with the aid of an interrupt from the Tx side and a flag in the shared memory from the Rx side.

After finishing all of the transmissions, each node announces 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 reported by the nodes is chosen as the time spent on the overall communication.

It is apparent from the algorithm that there are several overheads in communication among the nodes. Each node wastes some time for the management of the transmission and reception of the data. Checking the transmit buffers or the status of the recipient nodes, allocating a buffer for transmission and interrupt latency are a few examples of available overheads. In programming the system, every attempt was made to reduce the overhead to a minimum.

MANAGEMENT OF SHARED MEMORY

The allocation of the buffers in shared memory to different transmissions is one of the major issues in this structure. This can be done with static allocation where a buffer is already allocated for every possible Tx and Rx, or with dynamic allocation where the available buffers can be allocated to any Tx and Rx on request. Each method is discussed separately.

Static Allocation

In static allocation, as shown in Figure 5, the shared memory is divided among all the possible transmitters and receivers and there is a dedicated buffer for each Tx sending data to every Rx. 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 4K available memory, the maximum size of each buffer is 336 or 150 hex. The leftover words are reserved for administrative purposes such as the ready signal from Rx to Tx [10].

The advantage of the static allocation is that there is no overhead in assigning a buffer to a transmission, so the overall overhead is reduced. There is also no interference from other nodes in using a buffer for transmission. The drawback is that some part of the valuable shared memory is wasted because at any time, only some of the buffers are actively used in the transmission and the rest remain idle at that time. A better algorithm can use the available memory more efficiently.

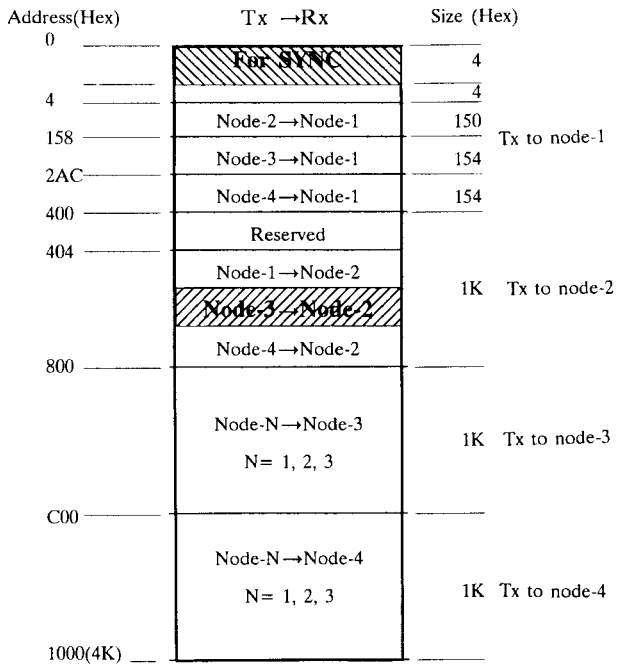

Figure 5. Memory layout for static allocation.

Dynamic Allocation

In dynamic allocation, as illustrated in Figure 6, the shared memory is divided into several buffers. All of them are general purpose and none is dedicated to a specific communication. The list of the buffers is kept in an allocation table. Any node requiring a buffer for transmission refers to this table and allocates a free buffer for its transmission.

At a given time, if there is more than one node performing buffer allocation, data loss may occur because of the probable contention. 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 should be a mechanism to control the allocation process so that it could be performed by one node exclusively. In order to achieve this goal, a lock mechanism implemented in hardware or software should be used and each node should exclusively own the lock before allocating a buffer. This prevents the others from performing a similar task until the lock is released.

It is worth mentioning that the lock is only used for critical activities such as buffer allocation, where there is a possibility of interference from the other nodes. Other tasks such as reading or writing to the previously allocated buffers can be performed in parallel with the ongoing activities on the shared memory. Therefore, the performance loss caused by the lock mechanism is minimal.

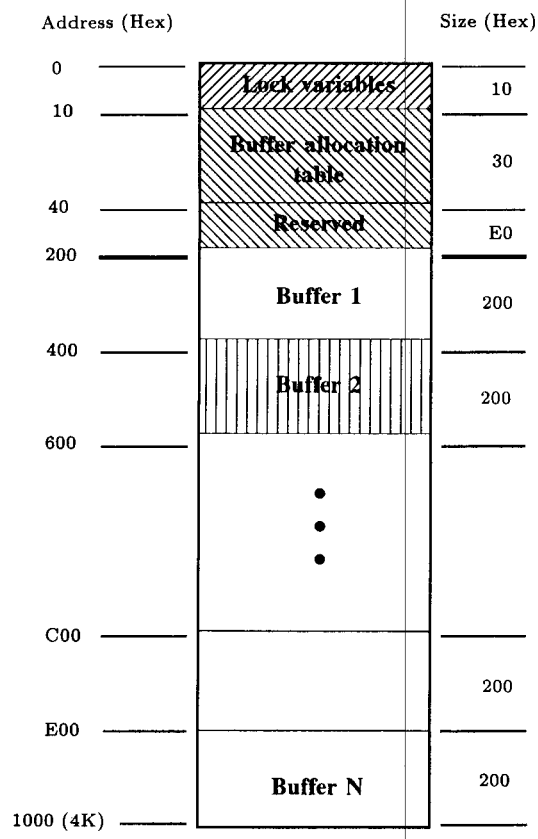

Figure 6. Memory layout for dynamic allocation.

Hardware Lock

In some dual-port memories, a hardware lock called "semaphore logic" is used to control the ownership of the memory [12]. The requesting node writes "0" in the lock followed by a read. If the lock is not owned by the other node, the read action will return "0" indicating a success, otherwise "1" will be returned indicating a denial. A success means that the node has the exclusive right to work in the locked area of the memory. Once the owner finishes with the memory, it releases the lock by writing "1" in it. On the other hand, a denial in acquiring the lock means that the other node possesses the lock and when it is released, the current request will automatically take effect. Hence, the node receiving the denial should either check the lock regularly until it is released or write "1" in the lock to quit and try another time. This concept is only effective for two ports.

As IDT7054 was a new product at the time of design, no semaphore logic had been implemented in it. The other possible reason for the lack of semaphore logic could be the complexity of logic for four ports. This is because there might be more than two nodes requesting the lock and other factors such as priority should be included in the mechanism. Consequently, the overall logic could be more complicated.

In the absence of a hardware lock, software locks should be exploited. TOKEN passing is the only method cited in the literature for this purpose [13]. This method and the two new methods devised to control the lock in software are discussed briefly.

TOKEN Passing

In this method, the lock belongs to a node with an ID that matches the TOKEN which resides in a dedicated location. When finished, the owner passes the lock to the next node in the cycle by writing the ID of the new owner in the token. Each node should check the token regularly to determine its turn in possessing the lock.

This scheme is easy to implement, but the problem is that the nodes that are busy with other tasks and are not involved in the communication at that time are also included in the token passing cycle. Therefore, regardless of its activity, each node should check the token regularly and pass it to the others. Failure to pass may result in long delays for the nodes waiting for the token.

The Lock with BUSY Signal

In this new method, only those nodes queued for the lock are considered in the TOKEN passing cycle. As shown in Figure 7, each requesting node should register in a dedicated location, provided that the BUSY flag is not active, otherwise the node should keep checking until the flag is cleared. The lock belongs to the node with the ID that matches the number in the OWNER. When the current owner finishes with the lock, it determines the next owner by checking all the registered nodes and passes the lock to the next node in a circular priority.

Before determining the new owner, the BUSY flag is set to prevent other nodes from registering. This is essential for blocking probable concurrent read/write operations.

In the process of determining the next owner, if there is no request pending, the current owner sets another flag called No-owner. This is to signal the next requesting node to take the responsibility for determining the owner. The details of the algorithm are as follows:

- Each node registers its request if BUSY is inactive, otherwise keeps trying;
- If there exists a current owner (No-owner flag is not set), the requesting node keeps checking the OWNER. When it matches the ID of the node, the lock belongs to the node;
- If there is no owner, the requesting node sets the BUSY flag to block other requests. Note that there might be more than one node attempting to register simultaneously at this situation and a short delay is required to make sure that other probable requests that might be still in action, have been registered successfully. Considering all the requests, each requesting node determines the owner in a circular priority where the previous owner has the lowest priority. The node with the ID that matches the determined owner, writes its ID in the OWNER, clears the BUSY and the No-owner flags and proceeds to use the locked area of the shared memory. The other nodes, if any, perform the regular checking of the OWNER to determine their turn in acquiring the lock;
- When the owner finishes with the lock, it sets the BUSY flag to disable the incoming requests and clears its own request. Then, it checks the other requests to find the next owner according to the priority cycle and writes the new owner's ID in the OWNER. If there is no pending request for the lock, the node sets the No-owner flag. At the end, it clears the BUSY flag.

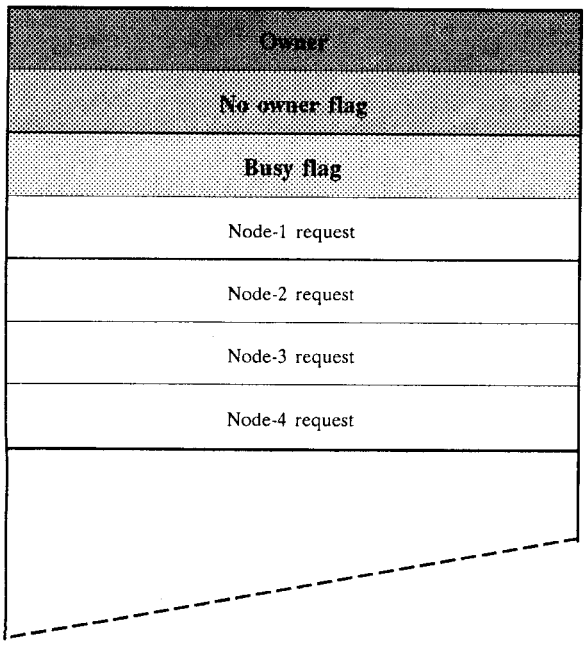

Figure 7. Lock with BUSY signal.

This algorithm was quite successful in practice and a fair distribution of the ownership was observed. Although the overhead of the algorithm is slightly high, the good point is that the registered nodes are free to do other tasks while their turn to acquire the lock is determined by the other nodes. The only remaining contention is the probable setting of the BUSY flag by more than one node at the same time. However, as they all write the same information in the BUSY flag, the contention has no drawbacks. If a lower overhead is desired, the algorithm can be changed to the one explained in the next section.

Fast Lock

The structure of this lock is the same as the previous one except that each node is responsible for acquiring the lock individually. As shown in Figure 8, the No-owner flag is no longer needed. The algorithm works as follows:

- If not BUSY, each requesting node raises the BUSY flag to block the other nodes from registering and enters its request in an appropriate location. In heavy demand for the lock, usually more than one request could be registered;
- After a short delay to make sure that all probable requests are entered, each registered node determines the potential owner using the circular priority. The winning node proceeds to use the shared memory, 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 removes its request and writes its ID in the Previous-

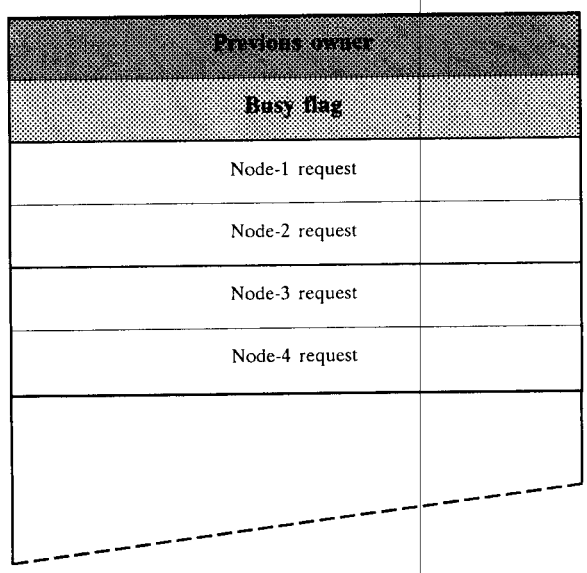

Figure 8. Fast lock.

Owner. If no other request is pending, it also clears the BUSY flag;

- By changing the Previous-Owner and removing the relevant request by the current owner, the other registered requests, if any, determine the new owner and proceed accordingly.

The algorithm is efficient and very simple. The associated overhead is also low. The only drawback is that the registered nodes should continuously determine the next owner to win the lock. In addition, the requesting nodes may face a raised BUSY flag more frequently.

It is worth mentioning that another algorithm was also tested on the system, which was based on queuing for the lock in a first-in-first-out list. However, this method was not completely successful because of the probable contention that it could cause when registering or moving in the queu.

Broadcast and Multicast

One advantage of using dynamic allocation is that if the same data is to be sent to more than one node, instead of sending it to each of them individually, multicast or broadcast can be used. The idea is that once the data is transferred into a buffer, it can be sent to more than one node. Each involved node can receive the data from the same buffer in parallel with the other nodes. Broadcasting and multicasting eliminate the overhead of getting multiple locks and multiple transfers of data into the buffers and reduce the overhead to the overhead of one attempt. Hence, performance is expected to rise.

Broadcast and multicast can be implemented by applying minor changes to the protocol. Each entry in the allocation table can also hold the broadcast information by assigning a bit to each node. If the bit is set, the corresponding node will receive the broadcast data.

The transmitter checks the status of all the receivers and if ready, allocates a buffer, writes the broadcast information in the relevant 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 generates an interrupt for each of them.

The receiver gets the number of the buffer in use from its mailbox, reads the data from the buffer and clears its flag bit in the corresponding entry in the allocation table. If the entry shows that there are no more nodes to receive the data, the last receiver deallocates the buffer.

RESULTS AND DISCUSSION

Multiport memory was interfaced as an external memory for TMS320C50 nodes. In the node processors, a 2-byte read from the external memory takes one cycle and a write takes two cycles. Therefore, in total, a 2-byte communication takes three cycles or 150 ns for a cycle time of 50 ns. Hence, the communication rate for each node is 13.3 Mbytes/s. If all of the four nodes communicate at this rate without any overhead, the aggregate communication rate for the entire system would be 53.4 Mbytes/s. Note that the ideal rate is stated for comparison purpose only and is not attainable in practice. This is because several overheads in the system such as checking the readiness of the receivers, queuing and allocating buffers, interrupt latency, and also the finite length of the shared memory buffers degrade the upper bound. The experiments illustrate that the length of the buffers in the shared memory is the main factor in rate measurements. Other important factors are the overhead of getting the lock and allocating a buffer in dynamic allocation.

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 sections.

Static Allocation

Static allocation was successfully tested on the system. Apart from other outcomes, it confirmed that the interprocessor communication with multiport memories was feasible and could offer a great increase in the system performance. As shown in Figure 9, by increasing the buffer length, the communication rate increases and approaches the ideal bandwidth. The maximum rate obtained was 45.5 Mbytes/s, showing 15% overhead. This rate was obtained for the buffer length of 336 (150 hex), which was the maximum buffer size for the

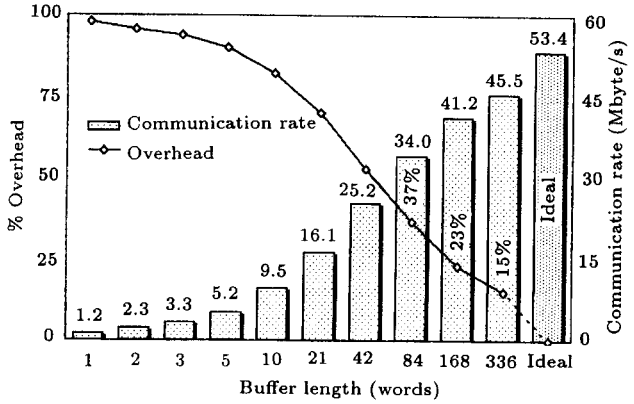

Figure 9. Results of static allocation.

shared memory size of 4K. Considering unavoidable overheads in the system, 15% is a reasonable value and demonstrates the effectiveness of the protocol under heavy traffic. Further performance rise can be expected for larger shared memory, however, the upper bound cannot be exceeded. For the memory size and, hence, the buffer size reduced to half, the rate dropped to 41.2 Mbytes/s and the overhead rose to 23%.

The results also demonstrate that a good performance with reasonable overhead is achievable even with small buffer sizes (100 or more). Therefore, a large shared memory is not a requirement for this structure. Increasing the buffer size beyond 336 will only improve the performance slightly and may not satisfy the cost/performance criteria. The latency for the static allocation was 2.4 μs for a 4-byte message and 12 μs for a 128-byte message. Higher processor speed and wider data path can lower the latency. The limiting factor is the speed of the memory in use.

Dynamic Allocation

Considering the reasonably close performance to the maximum rate achieved for the static allocation, there is not much to gain in applying dynamic allocation. However, there are two main reasons for using this kind of allocation:

- The ultimate goal of this structure is to use at least 8 or 9 ports for the shared memory. Under static allocation, a memory with N-port 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 probable 8-port memory will have a small capacity and dividing it into 56 buffers will yield even a smaller buffer size. This in turn will reduce the performance. Considering the fact that not all of the buffers are used simultaneously, a better management of the memory, such as dynamic allocation, is required to prevent the performance loss due to small buffer size;
- As explained later, broadcast and multicast, which increase the system performance considerably, are only attainable in dynamic allocation.

Figure 10 depicts the communication rate for different buffer lengths and buffer counts. The number of allocations made for each buffer was also recorded in the experiment. An interesting result was achieved from these statistical figures; they showed that for four nodes and under heavy traffic, no more than six buffers were required (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 was evaluated by the performance model in [11].

The highest communication rate was 45.8 Mbytes/s, which was achieved for 6 buffers of 672 words. As expected, the increase in performance was not significant compared to the static allocation, but it is anticipated that this method can perform much better if a large number of ports and small memories are to be exploited. Furthermore, its advantage in broadcasting will be shown in the next section.

For 5 buffers of 806 words, although the buffer length was increased, the performance dropped because some nodes had to wait for a buffer to be freed. Increasing the number of buffers to 7 declined the performance, as the extra buffer was not used, but the buffer size was reduced. The rate obtained for a buffer size of 336 was 40.8 Mbytes/s compared to 45.5 Mbytes/s for static allocation. This reveals the extra overhead endured by the system because of the lock mechanism and the allocation process.

Dynamic allocation enjoys the benefits of a better memory management, but suffers from the overhead imposed by the serial nature of the lock and its implementation in software. A better performance can be expected if a hardware lock, such as semaphore

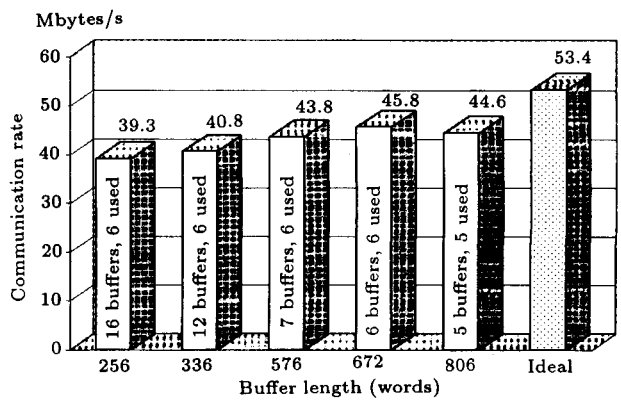

Figure 10. Communication rate in dynamic allocation.

logic, is exploited and/or the allocation procedure is improved. Compared to static allocation, latency is slightly higher due to the lock mechanism.

Broadcasting

Broadcasting is one of the major advantages of the dynamic allocation. Data is written into the shared memory buffer once, but it can be received by more than one node. Hence, a considerable saving in time can be achieved by avoiding duplicate writes and a higher communication rate is expected.

The best communication rate for broadcasting was 68.2 Mbytes/s for the buffer length of 806 words. This value is about 50% more than a normal communication. Even the upper bound communication bandwidth was exceeded by 30%. The total number of buffers involved in communication did not exceed four. These results confirm that dynamic allocation performs better than static allocation.

Comparison of Results

Different results obtained from MultiCom are compared in Figure 11. The results demonstrate that both static and dynamic allocations can achieve a reasonably good performance that is close to the unattainable ideal case. Furthermore, dynamic allocation illustrates its advantage by the use of broadcasting, which outperforms even the ideal case.

The result of a previous work in this area is included for comparison; in this system, Intel 486DX2/66 processors were used as nodes and IDT7006 dual-port memories (16 Kbyte, 45 ns) as shared memory. A cluster of five nodes was reported to be near completion in [8]. Under light traffic condition and for a buffer length of 1000 words, the measured rate for sending a packet of data to the dual-port memory was quoted around 2 Mbytes/s. The same rate could be applied for receiving data from the dual-port memory. Therefore, each node had a communication rate of 1 Mbytes/s. The overall communication rate was not stated; however, for comparison, an upper bound can be estimated for a four-node system. In order to have a full connection between all the nodes, this system would require six blocks of dual-port memory between the nodes and each node should be connected to three different memory blocks. Under these conditions, the upper bound communication rate would be 4 Mbytes/s.

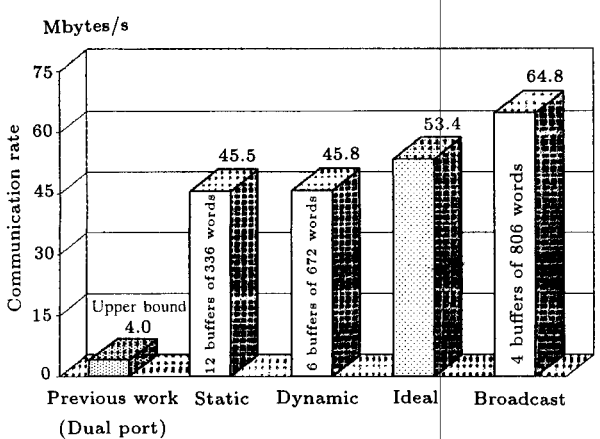

Figure 11. Comparison of results.

The results obtained from MultiCom show a great deal of improvement over the previous work 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 four times reduction in cost). The increase in performance could be even higher if an exact value of the communication rate rather than an upper bound was available. Moreover, broadcasting demonstrates an increase of at least 17 times in performance. It is worth mentioning that broadcasting is not possible with a structure that uses dual-port memories.

CONCLUSION

In this paper, the design of MultiCom, a small multiprocessor that uses multiport memory for its interprocessor communications, is discussed. The shared-memory management was achieved using both static and dynamic allocations. Dynamic allocation is superior to static allocation because of a better memory management and the ability to perform multicasting/broadcasting. Two new algorithms to implement software locks for obtaining the ownership of the shared memory were also presented. The achieved communication rate for the overall system was 45.5 Mbytes/s for static allocation, 45.8 Mbytes/s for dynamic allocation and 68.2 Mbytes/s for broadcasting. The system illustrated a great increase in performance as well as a good reduction in cost, with no requirement for the use of large shared memory. It also enjoyed a much simpler design and a great deal of reduction in the connection buses compared to a system using dual-port memories.

REFERENCES

1. Feng, T. "A survey of interconnection networks", *IEEE Computer*, **14**(12), pp 12-27 (1981).
2. Hoffner, Y. and Smith, M.F. "Communication between two microprocessors through common memory", *Microprocessors and Microsystems*, **6**(6), pp 303-308 (1982).
3. Boianov, L.K. and Knowles, A.E. "Higher speed transputer communication using shared memory", *Microprocessors and Microsystems*, **15**(2), pp 67-72 (1991).

4. Wyland, D.C. "Dual-port RAMs simplify processor communication", *Microprocessors and Microsystems*, **12**(10), pp 585-94 (1988).

5. Pryce, D. "Dual-port static RAMs. Specialized memories ease communications", *EDN*, **34**(8), pp 83-9 (1989).

6. Jagadish, N., Kumar, M.J. and Patnaik, L.M. "An efficient scheme for interprocessor communication using dual-ported RAMs", *IEEE Micro*, **9**(5), pp 10-19 (1989).

7. 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*, **18**(4), pp 93-203 (1994).

8. Campbell, S. and Kumar, M.J. "The design and development of the COMPS architecture", *Australian Computer Architecture Workshop (ACAW'96)*, Melbourne, Australia, pp 177-187 (1996).

9. Forsell, M.J. "Are multiport memories physically feasible?", *Computer Architecture News*, **22**(4), pp 47-54 (1994).

10. Asgari, N. and Burgess, N. "The structure and design of MultiCom, a small multiprocessor with multiport memory based communication", *3rd Australasian Computer Architecture Conference*, Perth, Australia, pp 15-24, Springer-Verlag (1998).

11. Asgari, N. and Burgess, N. "Interprocessor communication in tree structured multiprocessors using multiport memory", in *4th Australasian Computer Architecture Conference (ACAC'99)*, Auckland, NZ, pp 161-172, Springer-Verlag (1999).

12. Cypress Semiconductor Corporation, *Understanding Dual-port RAMs*, San Jose, CA (1996).

13. Mick, J.R. "Introduction to IDT's four port RAM", *Application Note AN-45*, Integrated Device Technology, Santa Clara, CA (1996).