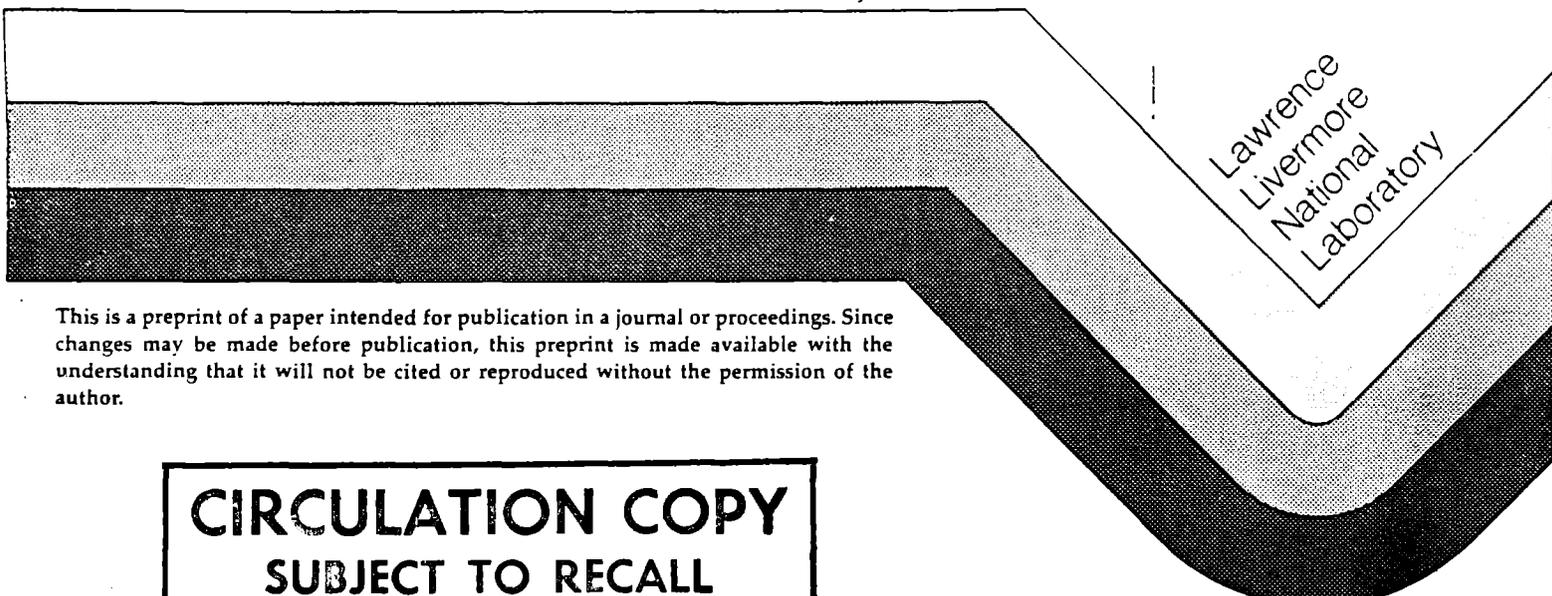


RING INTERCONNECTION NETWORK FOR THE
S-1 AAP MULTIPROCESSOR

Viki Y. Moldenahuer
P. Michael Farmwald
Owen T. Anderson
Jay C. Pattin
Jeffrey M. Broughton

This paper was prepared for submittal to The 15th
Annual International Symposium on Computer Architecture
Honolulu, Hawaii
May 30 - June 2, 1988

November 12, 1987



Lawrence
Livermore
National
Laboratory

This is a preprint of a paper intended for publication in a journal or proceedings. Since changes may be made before publication, this preprint is made available with the understanding that it will not be cited or reproduced without the permission of the author.

**CIRCULATION COPY
SUBJECT TO RECALL
IN TWO WEEKS**

DISCLAIMER

This document was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor the University of California nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial products, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement recommendation, or favoring of the United States Government or the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or the University of California, and shall not be used for advertising or product endorsement purposes.

Ring Interconnection Network for the S-1 AAP Multiprocessor

ABSTRACT

This paper presents an interconnection network for the S-1 AAP, a large-scale multiprocessor being constructed at Lawrence Livermore National Laboratory. The packet-switched interconnect consists of dual counter-rotating rings, with nodes modelled as non-uniform crossbars. The advantages of this interconnect are its conceptual simplicity, its suitability for high-speed implementation, and its extensibility. The ability to queue messages in the interconnect provides reliable transmission without acknowledgements or flow control. The motivations behind the design are discussed, and the ring organization is described. Both difficult design issues and benefits of the design are discussed.

INTRODUCTION

One of the classic problems in designing multiprocessors is that of connecting the processing elements and other system resources together. Ideally an interconnection network should provide high-speed, reliable delivery of messages under all conditions of system performance. The challenge of meeting these requirements has led to the proposal and construction of many solutions such as the global linear bus, direct-connection net, crossbar, hypercube, star net, and ring. Each of these has its respective merits, which are well documented in the literature. A good survey of many interconnection networks can be found in the IEEE Tutorial on Interconnection Networks,^[1] and rings, in particular, are discussed by Liu and Rouse.^[2]

This paper presents the design of an interconnection network for a large-scale multiprocessor, the S-1 Advanced Architecture Processor (AAP), being constructed at Lawrence Livermore National Laboratory.^[3,4] The interconnect design consists of nodes connected by dual counter-rotating slotted rings, with nodes constructed as fixed-size, non-uniform crossbar switches connecting the rings and processing elements.

The two major design objectives of the AAP interconnection network are high performance and adaptability for large-scale systems. The systems that will use this network are envisioned to be moderately large, on the order of two hundred processors. The global interconnect must be extensible and avoid fixed limits on system size such as those encountered by crossbars. The physical limitations of many interconnect schemes eliminate them from consideration in large systems. The limitations of extensibility, interconnection cable count, and adjacent-node latency are particularly important.

A ring has only two connections per node, regardless of the number of nodes. The size of a ring can be changed without affecting existing nodes, allowing a system to grow incrementally as new processing elements are added. Also, with a small, fixed number of connections per node, wide data paths are not prohibitively expensive. This allows data to be sent in parallel, preserving the per-processor bandwidth of the interconnect as the system size grows.

In large-scale implementations of some interconnect schemes, cable counts can reach such high values that transferring data bit-serially is the only reasonable solution. This forces a large latency even for communication between adjacent nodes. Because many problems intended for multiprocessors partition in such a way as to display nearest-neighbor locality, a short adjacent-node latency is desirable. Thus, the parallel data paths minimize latency for all transfers.

High performance in the AAP interconnect is achieved, in part, by a 15 ns cycle time. The AAP adopts a RISC-like approach to interconnection as well as processor design. Despite its low connectivity, the physical simplicity of a ring allows it to be readily optimized for speed to provide the desired performance. If the cycle time can be made fast enough, then traversing even the longer distances of a ring will still be faster than performing the more complicated routing control required by an interconnect of higher connectivity.

The cycle time goal requires that routing and control issues be solved quickly. Speed-of-light limits imply that there is not enough time for global flow control: routing decisions must

be made locally. The decisions required by a ring or a small, fixed-size crossbar are simple and can be made relatively quickly.

A third consequence of the high performance objective of the entire AAP system is that there are no acknowledgements for receipt of messages. The processors themselves are heavily pipelined to achieve high performance. With their fast, short pipelines, processors would inevitably wait for an end-to-end acknowledgement across the ring. Lack of an acknowledgement mechanism requires the ring to deliver all messages entrusted to it.

The rest of this paper first defines some terms and looks at the organization of the interconnect. Then the major design issues of deadlock avoidance, command time-ordering, latency minimization, and traffic management are discussed. Added hardware within a node and special properties of message types solve the deadlock avoidance problem. A generation count scheme provides a means of time ordering commands on the ring, while routing rules preserve the ordering within a node. The choice of a ring as the global interconnect provides high bandwidth and short adjacent-node distances in large systems, while the extremely high transfer rate around the ring minimizes the latency for long-distance transfers. Traffic management implemented as routing rules within the nodes helps maintain the system-wide performance. A peak transfer rate of 1.1 gigabyte per second per node is achieved by this design.

RING ORGANIZATION

The organization of the ring interconnect is shown in Figure 1. The *ports* are the entities to be connected together, e.g. the processors, memories, I/O processors, etc. The *nodes* provide the queuing and control logic that connect the ports to the rings. Under current implementation technology limitations, up to three ports may be attached to a node, and a system may contain up to 128 nodes.

Two counter-rotating rings, arbitrarily designated the *clockwise* and *counterclockwise*

rings, carry data. They can be classified as bit-parallel slotted rings, similar to Pierce rings, but with relaxed rules on slot access. Slots are not assigned to particular nodes and there is no token.

Messages are the objects which travel around the interconnect. A message contains 64 data bits, 40 bits of address, plus control and parity bits for a total of 142 bits. Messages are originated and consumed by ports. There are two types of messages: *commands* and *responses*. A command is a query or a request to perform some operation, e.g. read a doubleword from memory. A response is a message in answer to a query, such as a doubleword of data. The significant difference between the two types is that commands can generate further traffic whereas responses cannot.

Figure 2 shows a block diagram of the data path within a single node. The data flows from left to right. Messages coming in from the ports travel to other ports or out onto the rings. Messages coming in on the rings travel to ports on this node or continue on the rings to the next node. Messages cannot travel from one ring to the other.

Messages on a ring continue to circulate until they are removed by their destination node. The rings never stop.

DESIGN ISSUES

Deadlock Avoidance

The most complicated issue in the ring design is deadlock avoidance. The term *deadlock* is used to mean a situation in which *no* port can make forward progress because the message delivery system has become saturated, and delivery paths for all messages are blocked. The following example will illustrate how deadlock can occur for a simplified crossbar interconnection of two processors (see Figure 3).

The four boxes in the center are output and input registers for each of two entities, here shown as processors. They form a crossbar. For each processor, two sources generate messages

which are sent to the crossbar through the processor's output register. The instruction stream executed by each processor may generate any number of messages. The cache coherency logic may create new messages for each command that it receives.^[3] If both processors generate enough messages from both instruction stream and cache coherency logic, the messages will fill all four registers. The processors will then be deadlocked, unable to continue processing because there is no place for either source to put outgoing messages. This can be generalized to the situation of rings and ports as connected elements on a crossbar. In the AAP system a node acts as a crossbar switch.

The deadlock problem is caused by unconstrained message generation by processors and insufficient queuing in the destination ports.

Some attempts are made to curb the generation of messages by processors: a processor can have only two cache misses and one cache coherency operation outstanding, but there is no limit on the number of other commands. One cache flush can generate up to 2K doubleword writes. Control in the processors was kept as simple as possible in order to maintain high performance. Consequently, processors have little local flow control. Lack of global flow control compounds this situation as a single memory port may be the destination of messages from multiple sources. Thus no safe upper bound can be placed on the number of commands on the interconnect at any time.

Additional queuing would not help: there is sufficient queuing in the ports that near-deadlock situations are expected to be rare, but it is not possible to have enough queuing to ensure that deadlock never occurs.

However, even if the interconnection network is saturated with messages, and all paths everywhere are blocked, if one entity anywhere can make forward progress at any time, the situation can be eventually resolved. Therefore, a mechanism is provided in each node that can be used to resolve local saturation conditions one message at a time. This mechanism requires very carefully chosen constraints on messages and their routing.

There are three sequences of messages that can occur: a command is sent and consumed by the destination node; a command is sent to a destination, which then generates another command as a direct result; and a command is sent to a destination, which generates from one to four responses.

The first case is trivial: for example, a processor writes one doubleword of data to a memory. The memory does not generate any message in reply. This case cannot cause deadlock.

The second case is a difficult one. A processor may send another processor a cache coherency command.^[3] The second processor does not know in advance that the command is coming, and cannot have reserved a space for it. Unexpected commands such as these may back up, blocking data paths throughout the interconnect. In particular, a port's own output path may become blocked, preventing that port from making forward progress because it cannot send messages out. The solution used by the interconnect is an extra register in the node which allows the node and port to perform a swap of commands: the node takes a command from the port first, then as the port makes forward progress, the port clears its input register so that it may accept a command from the node.

This extra register is called the *emergency register* and is part of the node's port-in logic (see Figure 4). This register is only used in an *emergency*, which is defined as a situation in which the node tries to send a command to a port which the port cannot accept, and the port simultaneously tries to send a message to the node which the node cannot accept. Nodes or ports refuse messages when they have no place to queue them.

Figure 5 shows the identification and resolution of an emergency.

The third sequence of messages causes problems because of the multiplication of messages. A processor executing a cache miss will send a single read command for four doublewords of data, and the memory will return four responses, one for each doubleword of data.

This multiplication of messages is desirable for performance. The processors could send

four separate doubleword reads to satisfy a cache miss. This would allow use of the swapping scheme to handle deadlock cases, but this would also increase message traffic significantly. Because they are expected to occur frequently, it is worthwhile to optimize cache misses by reducing the number of required commands.

Multiple responses can be supported by requiring that a port be able to store, *i.e.* immediately accept, all responses that it requests. Because responses are explicitly requested by a port, and not spontaneously generated, this can be done. In the current implementation, a processor is limited to two outstanding cache misses because there are only eight spaces for responses.

With multiple responses returning from one command, the swapping procedure is insufficient to avoid deadlock and another means of preventing deadlock must be provided. All responses have a destination willing to accept them if they can get there. The node must always be able to route a response to its destination port, even if the path to that port is cluttered up with unaccepted commands. Thus responses must be able to bypass backed-up commands in order to avoid deadlock.

Within a node, structures called *quelets* provide queuing and allow responses to pass commands. A quelet is shown in Figure 6. The associated control must ensure that only one command may be in a quelet at a time. One or two responses, or one response and one command may also be in a quelet at a time. Simultaneously allowing two responses in a quelet optimizes the cache miss case.

At the system level, each ring must always have at least one slot reserved for responses. This ensures that responses do not become blocked by a ring full of commands. One spare slot for responses would be sufficient to avoid deadlock, but designating more slots for responses improves performance. Slots are marked according to type of message they can contain. Either a command or a response may use a slot marked "command," while only a response can use a slot marked "response."

Naturally, emergencies and larger scale near-deadlock situations are not expected to happen often, but the design must handle them. The process of emergency resolution shown in the simplified example extends to multiple rings and ports at the expense of additional complexity in the control logic in the node.

Maintaining Time Ordering

The AAP processor's cache coherency scheme demands that commands be delivered in time order.^[3] Specifically it requires that from any destination's viewpoint, commands from any single source must be delivered in the order in which they were generated by the source. Thus the interconnect must ensure both that commands on the ring are accepted into a destination node in the order in which they were first seen by that node, and that within a node a command does not pass another earlier command. Because responses are not subject to time ordering, a response can pass a command.

To support time ordering, a node must accept commands in the order in which they are generated. Although the cache coherency scheme requires only that commands from a single writer be time-ordered, in practice the node cannot keep track of the writers of any commands refused. If a node is unable to accept a command, that command will circulate around the ring and can be received on a subsequent pass. This introduces the possibility that commands could be accepted out of order. Therefore, that node cannot accept any other commands until it can accept the first refused command.

The time ordering of commands is maintained across both rings with a *generation count* scheme. A node maintains two special counters called the *Accepting (A)* and *Setting (S)* generation counts. The value of the Setting counter is recorded in messages when they are first refused. The Accepting counter follows the Setting counter and is incremented when refused messages are finally accepted. These are actually special linear feedback shift registers^[5] whose values never become zero; zero always indicates a new message.

Normally, a node clocks in messages coming around the ring. The node's A and S counters are equal, the node is accepting commands, and there are no commands refused by this node on the ring.

When a node refuses to accept an incoming command, the node sends the command around the ring after changing the command's generation count to the node's S count. The node then increments its S count for the next command. At some future time, the node can again accept commands, but because it has once refused a command (i.e. A is not equal to S) the node must not accept new commands and commands where the generation count is not equal to A. When the command with a generation count equal to A returns, the node clocks in the command, increments the A counter, and looks for the next command.

Figure 7 demonstrates this.

An interesting complication arises when generalizing to two rings. If two commands, one on each ring, simultaneously arrive at a node, they both must receive the same value of S. If both can be accepted when they return, A is incremented. However, if only one can be accepted when they return, A cannot be incremented, or the unaccepted command would remain in the ring until A cycled through its full count again. This case is detected and handled properly.

Latency Minimization

One major problem with a ring interconnect is the large latency for distant transfers. This problem can be minimized by making the interconnect run as fast as possible. This is achieved in three ways: transferring bits in parallel, making node-to-node transfers very fast, and generating control signals a cycle in advance of their use.

The nodes operate at twice the speed of the processors, and ring transfers occur at twice the speed of the node by means of a distributed register scheme. Consequently, a message moves through four nodes in the time a processor can execute a single instruction.

A slot on one of the rings corresponds to a physical message. The message is held in the ring-out latches, shown in Figure 2 after the ring-out block. Latches in adjacent nodes form a distributed register. Adjacent nodes' clocks run 180 degrees out of phase, permitting messages to exist in two nodes at any one clock cycle. Messages leapfrog around the ring, advancing two nodes during a single node clock cycle. As a result, the ring must have an even number of nodes.

The node cycle time only allows two levels of discrete 100K ECL logic between registers; signal propagation delay consumes a significant portion of the available cycle time. The complexity of the routing calculations requires some control signals to be generated one cycle ahead of their use.

The node-to-port interface includes a bidirectional data bus, arbitration for that bus, and handshaking for notification of message acceptance. To allow a short cycle time, arbitration is performed for the next cycle rather than for the present one. Accept-command and accept-response signals are also generated for the next cycle.

The node-to-node interface pipelines its control functions across two adjacent nodes. Because of the tight timing constraints, a node N shows its previous node on each ring which types of messages N is willing to accept. Assume P precedes N on the clockwise (CW) ring. P examines its incoming messages on the CW ring and determines if they are destined for N and if N can accept them. P directs N to accept or recirculate the messages accordingly. The node preceding N on the counterclockwise ring performs the same function for N on that ring. The control signals for the generation count scheme work in a similar fashion. A node shows its A and S values to its two previous nodes, which factor them in to the generation of the clock-message-in and pass-message-on signals.

Traffic Management

Local emergency situations are expected to occur temporarily, especially around highly-

utilized system resources; therefore, the network is designed to handle them. The network will also handle system-wide near-deadlock situations: the collective effect of each node's resolution of its local emergencies is to prevent system-wide deadlock. System-wide near-deadlock is expected to occur rarely. If the system were to approach near-deadlock frequently, performance would drop drastically, making the system useless. The only expected need for deadlock avoidance is to prevent an unfortunate timing coincidence from halting the system.

Although there is nothing the interconnect can do about the generation rate of messages, there is one further routing decision that can significantly affect performance: trying to keep the ring as empty as possible. Thus, within a node, messages coming from a ring and going to a port have routing preference over messages originated by and destined for a port at this node. If the port-in messages were given priority, the node's ring-in queue could fill up. If this happens, the node will refuse to accept commands, and refused commands to this node could fill every ring command slot. If all command slots were filled, no node could place commands onto the ring. In general, it is better to make a local port wait a cycle or two than to allow even one command to back up onto the ring. However, as in the emergency example, it is sometimes essential that a node route a port-to-port message before a ring-to-port message.

CONCLUDING REMARKS

Performance

The transfer rate of the interconnect past a node can be calculated as follows:

$$\frac{8 \text{ bytes}}{\text{message}} \times \frac{1 \text{ message}}{15 \text{ ns}} \times \frac{2 \text{ rings}}{\text{node}} = 1.1 \text{ gigabytes/second}$$

Adding nodes increases the bandwidth, although at the expense of increased latency. The total interconnect bandwidth is 1.1 gigabytes per second times the number of nodes. Port-to-port transfers within a node can occur in parallel with ring-to-ring transfers, with a bandwidth of 0.5 gigabytes per second.

Potential Problems

Potential problems of rings include issues such as the fate of lost messages and the effect of a broken link between nodes. The AAP interconnect does not have a supervisory node to identify and remove lost messages. Although local area networks can contain supervisory nodes, such centralized control is inconsistent with the goals of a multiprocessor interconnect.

The network does not detect faults such as broken cables or dead nodes, nor does it attempt to route around them dynamically. Without message-level acknowledgements, it is very difficult to detect such faults. The system can, however, be reconfigured off-line by a front-end processor to avoid dead nodes and allow continued operation with the malfunctioning hardware still in place.

Status and Future Work

A prototype is expected to be operational by May, 1988. The logic for a single node has been successfully simulated in the SCALD logic simulator.^[6] A design consisting of a node, a processor, and a memory controller has also been simulated. A full multiprocessor with several nodes, processors, and memories will be simulated in the near future.

With improved implementation technologies it will be possible to speed up the ring slightly, but more leverage would be gained by increasing the amount of queuing in the ports, increasing the size of the local crossbars, or enhancing the global connectivity of the rings.

The generalization of this interconnection scheme to other processors is also an area for future work.

Summary

Designing the interconnect of a high-performance multiprocessing system is hard — *really* hard! The AAP multiprocessor system has taken a RISC-like approach to solving this problem. The choice of a ring as the global interconnect provides high bandwidth and short adjacent-

node distances at large scales, while the extremely high transfer rate around the ring minimizes the latency for long-distance transfers. A peak transfer rate of 1.1 gigabyte per second per node is achieved by this design.

ACKNOWLEDGMENT

Work performed under the auspices of the U. S. Department of Energy by the Lawrence Livermore National Laboratory under contract number W-7405-ENG-48 with support from the Office of Naval Technology.

REFERENCES

1. *Tutorial: Interconnection Networks for Parallel and Distributed Processing*, Chuan-lin Wu and Tse-yun Feng, eds. IEEE Computer Society Press, Silver Spring, Md, 1984.
2. M. T. Liu and D. M. Rouse, "A Study of Ring Networks," *Ring Technology Local Area Networks*, I. N. Dallas and E. B. Spratt, eds., Elsevier Science Publishers B. V. (North Holland) IFIP, 1984.
3. J. D. Bruner, et. al., *Cache Coherency on the S-1 AAP*, Lawrence Livermore National Laboratory, UCRL-97646.
4. E. H. Jensen, et. al.,
5. D. E. Knuth, *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms*, Addison Wesley Publishing Co., Reading, MA, pp. 29-30.
6. T. M. McWilliams and L. C. Widdoes, Jr., *SCALD: Structured Computer-Aided Logic Design*, Lawrence Livermore National Laboratory, UCRL-80950.

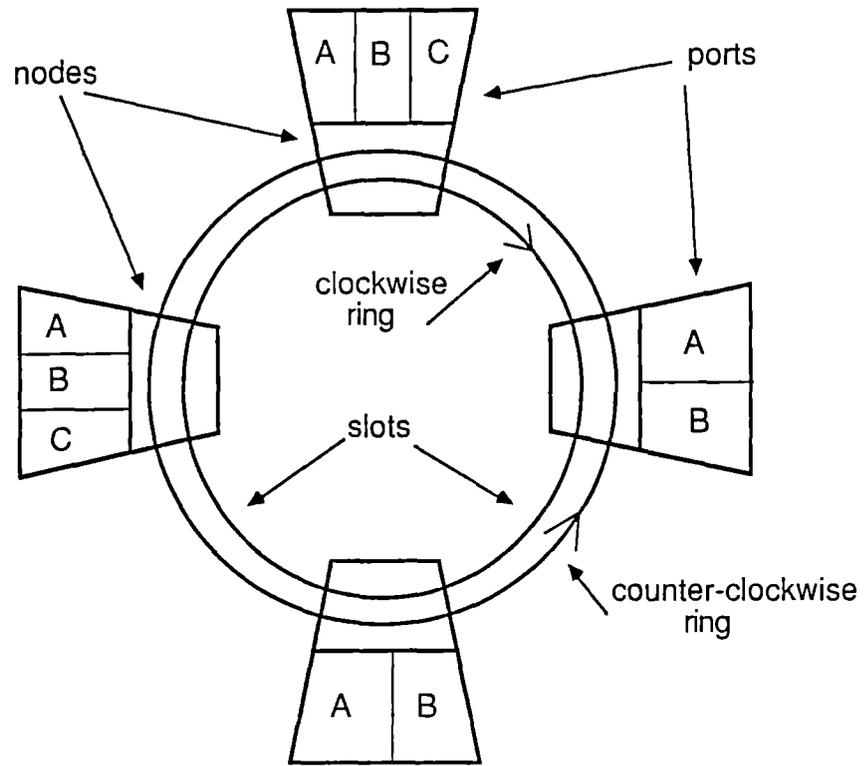


Figure 1: Organization of the Ring Interconnect

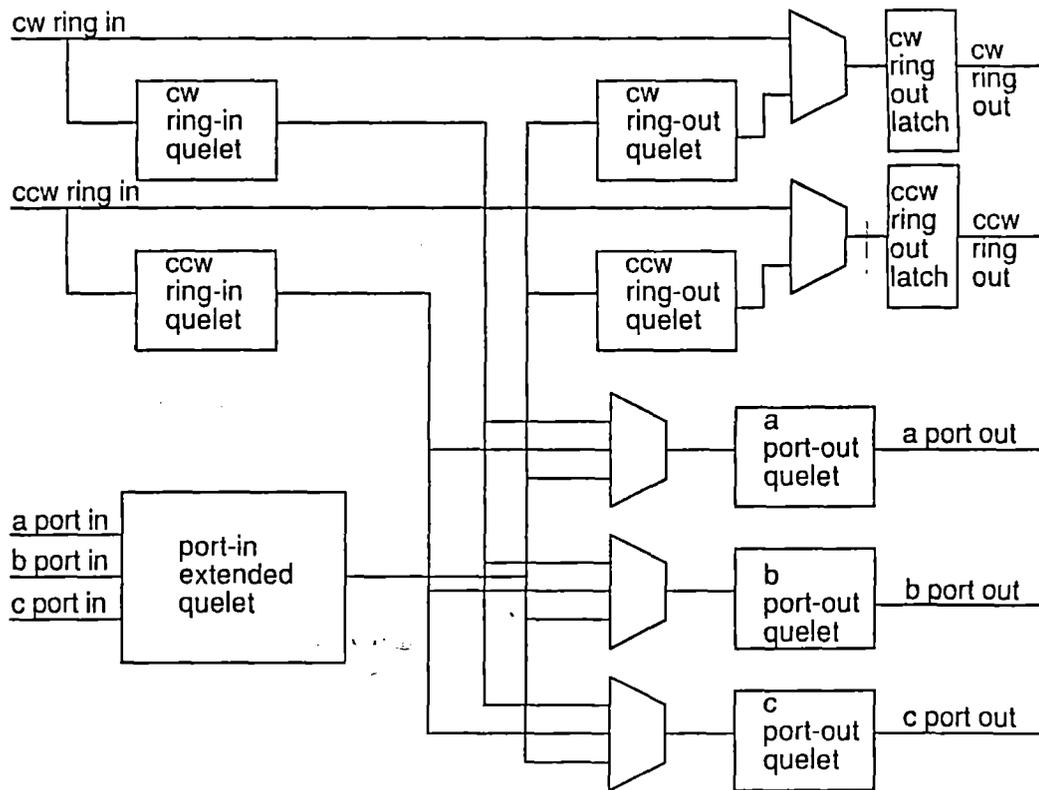


Figure 2: Block Diagram of the Data Path within a Node

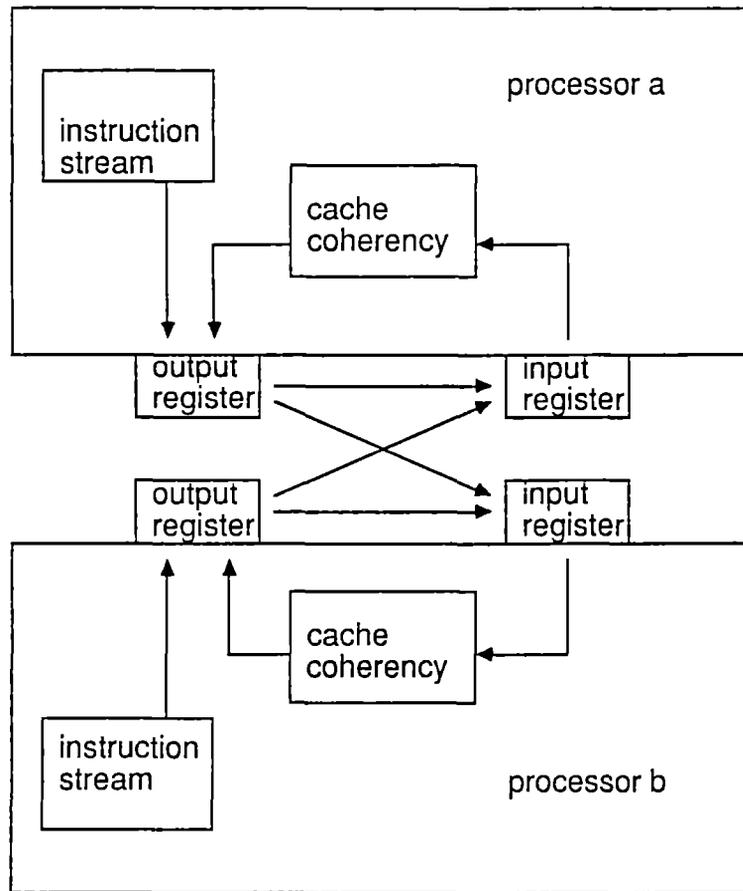


Figure 3: Simplified Crossbar Example

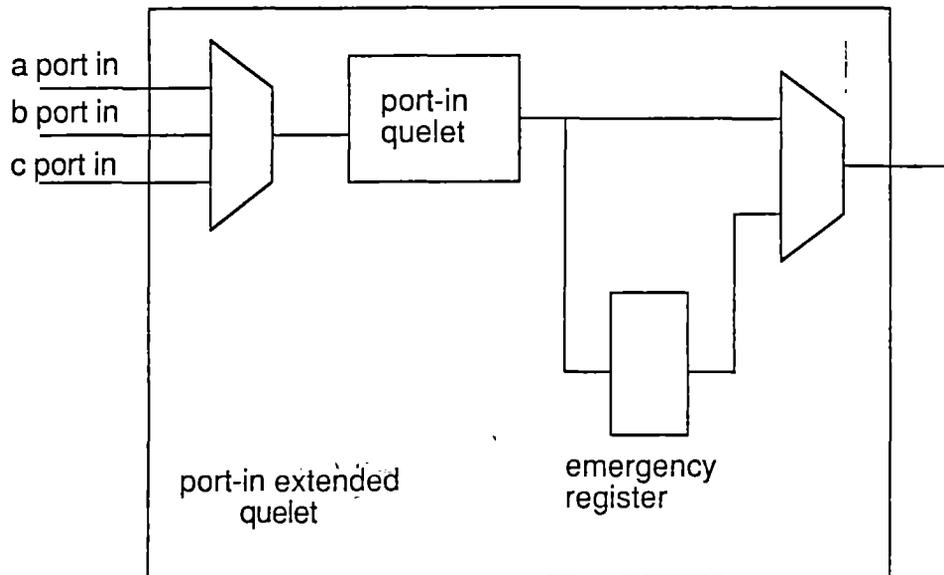
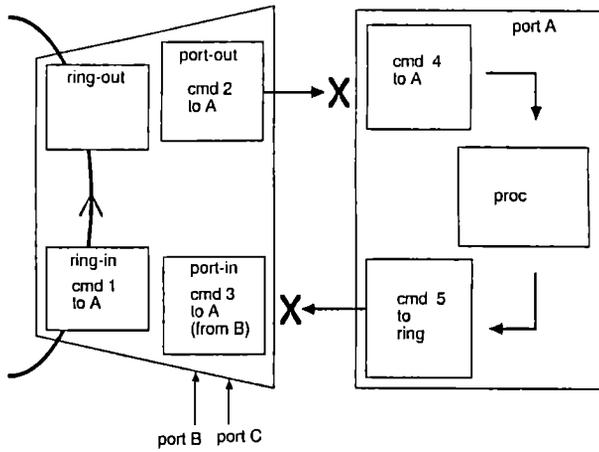
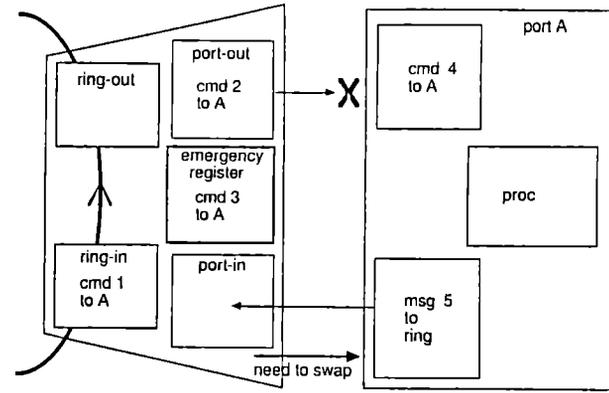


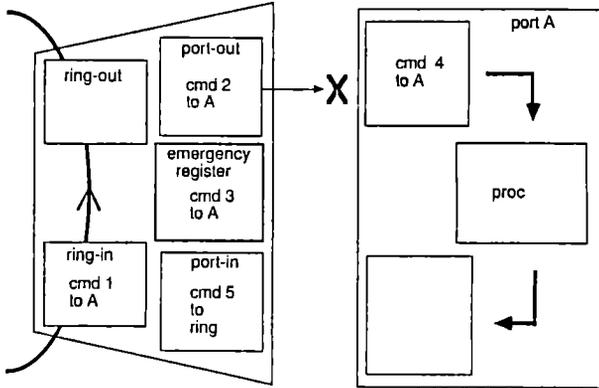
Figure 4: Port-In Extended Quelet containing the Emergency Register



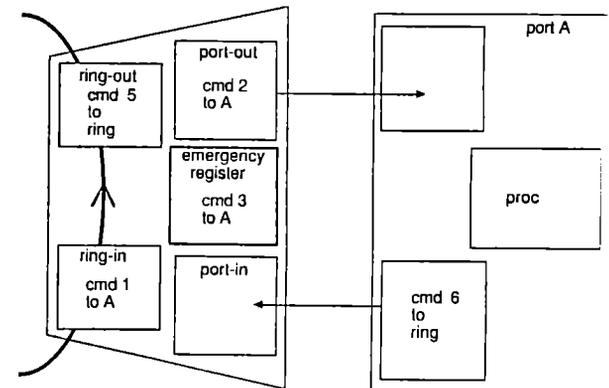
5a: An emergency implies that both node and port are full or nearly full. The X's show that both the node and the port are trying to send a command to each other and being refused. A swap of *cmd 2 to A* and *cmd 5 to ring* is required.



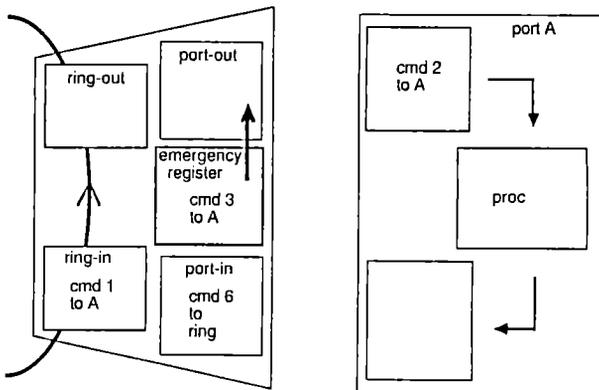
5b: The emergency register in the port allows the node to accept the port's command first. The node tells the port that a swap must occur.



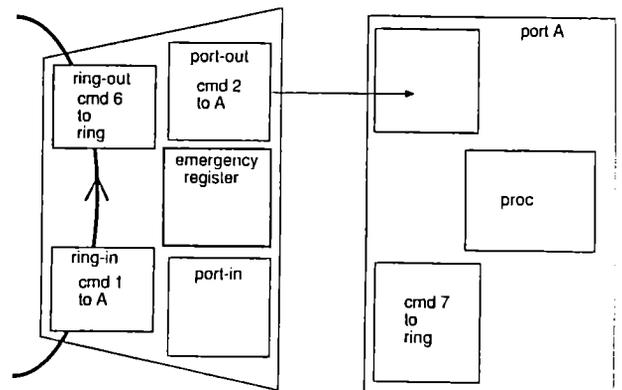
5c: The port's newly-emptied output register creates a hole that must be propagated back to the port's input register so the port can accept *cmd 2 to A*. The node warned the port of the swap, so the port must internally choose a command such that the command it places in its output register will free up its input register. When nodes and ports are both saturated, this is the one hole available to resolve the routing.



5d: In consuming *cmd 4 to A*, *cmd 6 to ring* was generated. Now the port can accept *cmd 2 to A* from the node. Here, because the port's *cmd 5 to ring* was destined for the ring, a second hole appears in the node's port-in register. In the worst case, the destination of this command would be another port, and the command would wait in the node's port-in register waiting for the emergency register to empty.



5e: Now the hole is in the node's port-out register. Here the node must choose a command that allows the emergency register to be emptied in the near future. This is usually the command in the emergency register, unless that command is destined for a ring. Then the node must select the command in that ring's ring-in register, allowing a swap of commands with the ring, and allowing the emergency register to send its command to the ring-out register. The node can perform a swap with a ring without requiring an extra register. An incoming command can be clocked into the ring-in logic at the



5f: The emergency is over when the emergency register is emptied. In the worst case, port A's commands would have been destined for another port, so both node and port would again be full of commands. Another swap would be needed, and it could be performed because the emergency register is empty again. The port is able to make forward progress. If all entities can be serviced, albeit slowly, then full deadlock can be avoided.

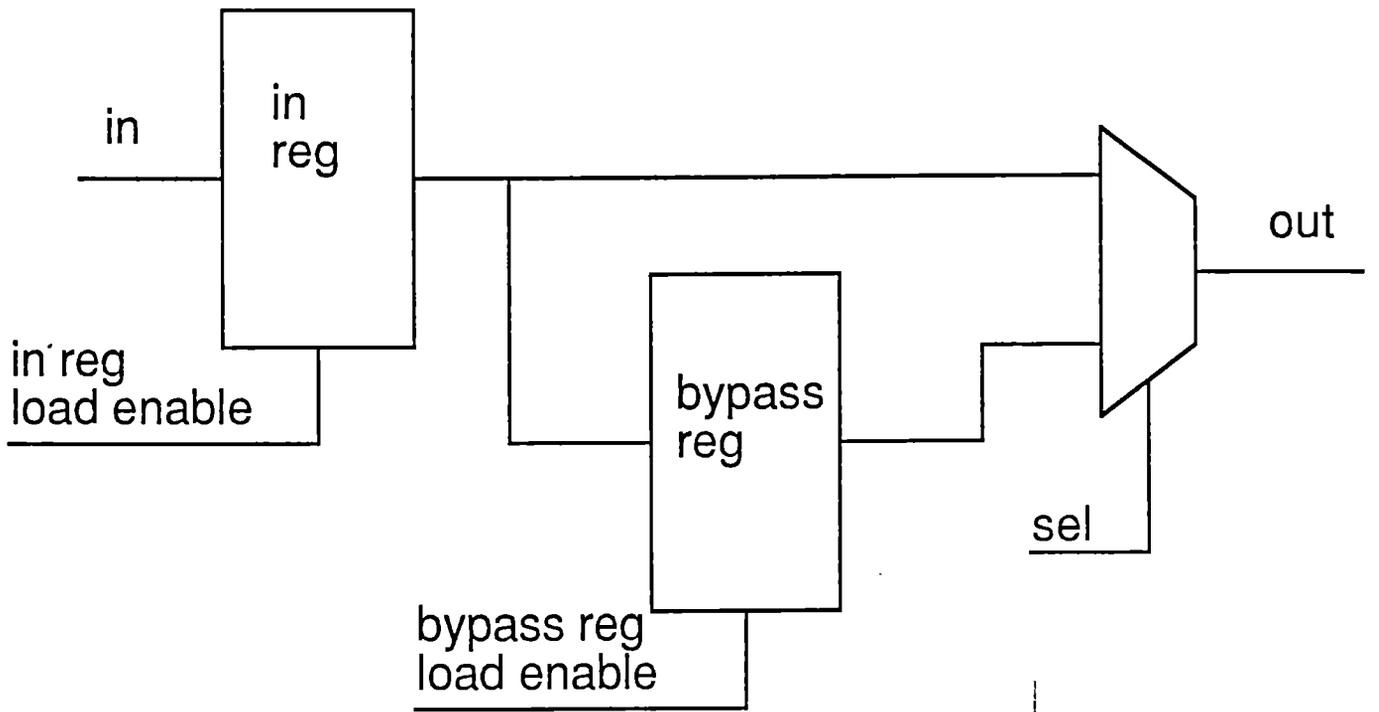
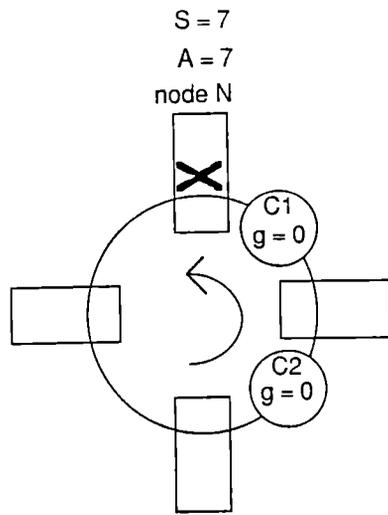
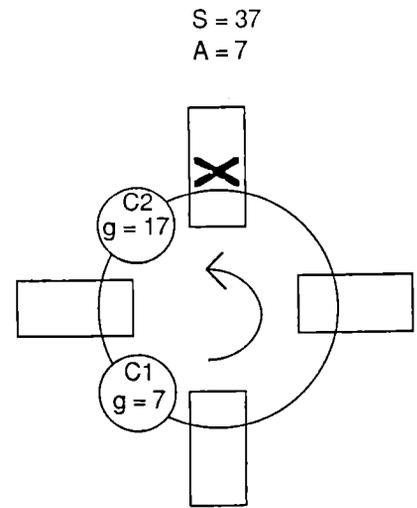


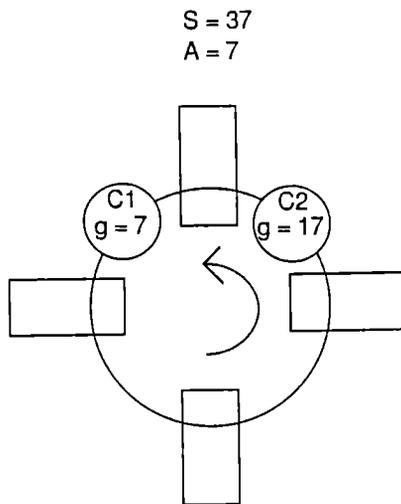
Figure 6: A Queue



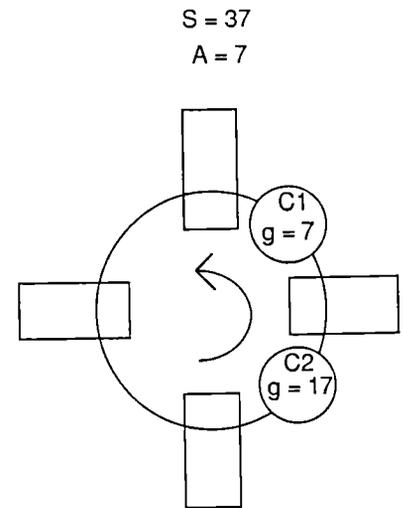
7a: Here is a simplified single ring, with the node of interest labeled N. Command C1 is a new command, originated in some other node. N's S and A counters are equal, but N must now refuse commands. The command remains on the ring with its generation count changed to the value of N's S counter. N then increments its S counter. The same will happen to command C2.



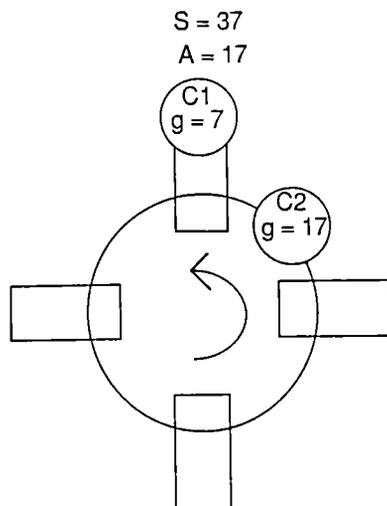
7b: The commands have been sent on, and N's S counter has been incremented.



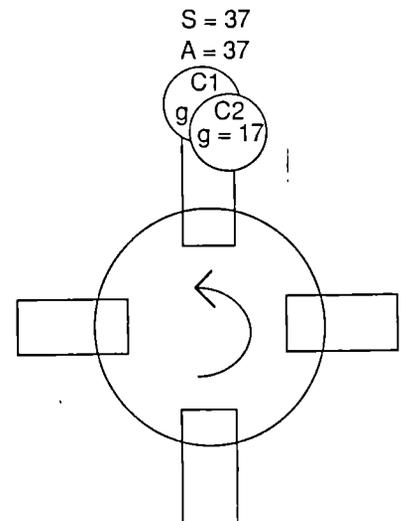
7c: N can accept commands, but because its S and A counters are not equal, it cannot accept commands with generation counts not equal to A. C2 continues around the ring, but its generation count is not changed this time.



7d: N's S and A counts are not equal and N is accepting commands, but now a command with a generation count equal to A is seen, so N accepts the command and increments the A counter.



7e: N has accepted C1. N is still accepting commands and the generation count of C2 is equal to A, so N accepts C2 and increments the A counter.



7f: Now S and A counts are equal, so there are no refused commands destined for N on the ring. N will accept any new commands.