Proceedings of the
IEEE International Computer Conference — COMPCON 94,
San Francisco, California, pp. 292-297, February 1994.

Starvation Prevention for Arbiters of Crossbars with Multi-Queue Input Buffers

Hsin-Chou Chi and Yuval Tamir
Computer Science Department
University of California
Los Angeles, California 90024

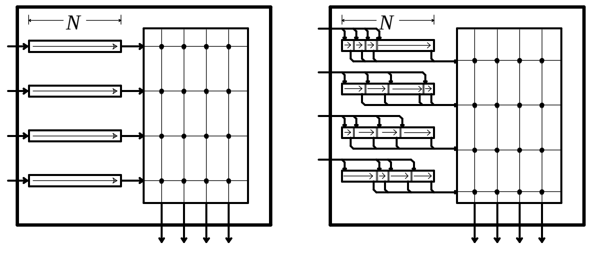
Abstract

Crossbars are key components of communication switches used to construct multiprocessor interconnection networks. Multi-queue input buffers have been shown to lead to high performance in such networks by allowing packets at an input port to be processed in non-FIFO order. *Symmetric crossbar arbiters* efficiently resolve conflicting requests in switches with multi-queue input buffers. While these arbiters lead to excellent performance in terms of throughput and average latency, they do not guarantee *fairness*. Hence, it is possible for an ‘‘unlucky’’ packet to be left in a switch buffer for a long time, potentially forever, while other packets are forwarded quickly through the switch. This paper introduces and evaluates a technique for preventing such *starvation* situations. The viability of the technique is demonstrated by implementing it in VLSI. Simulations show that the starvation-free arbiters may outperform arbiters that lack a starvation prevention mechanism for certain nonuniform traffic patterns at a cost of minor performance degradation for uniform traffic.

I. Introduction

Multiprocessors and multicomputers achieve high performance using interconnection networks that provide high-bandwidth low-latency interprocessor communication [1, 3, 4]. Small $n \times n$ crossbars are key components of the communication switches which are the building-blocks of such interconnection networks. Multi-queue input buffers can maximize the performance of the communication switches by allowing packets at an input port to be forwarded by the switch in non-FIFO order [7, 6, 2, 9]. Some of the packets arriving at a communication switch may be delayed due to conflicting demands for resources, such as buffer space or output ports. Crossbar arbiters which efficiently resolve these conflicts are critical to realizing the potential for high performance of non-FIFO processing of packets [10].

Figure 1-a shows the organization of a crossbar switch with conventional FIFO buffers. The strict FIFO order of handling packets at each input port unnecessarily reduces the throughput of the switch [9]. When the packet at the head of the queue is blocked, all other packets in the same buffer are also blocked, even if they are destined to idle output ports. Multi-queue buffers avoid this shortcoming of FIFO buffers by partitioning each input buffer into several FIFO queues, one for each output port. Even if one of the queues of the buffer is blocked, it may be possible to transmit a packet from the head of another queue, which is destined to a different output port.



(*a*) *FIFO Switch* (*b*) *DAMQ Switch*

Figure 1: Switches with FIFO buffers and DAMQ buffers

A particularly efficient organization of a multi-queue buffer is to *dynamically* partition the buffer space between the queues so that the storage is available where needed. Such buffers are called *dynamically-allocated multi-queue* (DAMQ) buffers [9]. Figure 1-b shows the organization of a crossbar switch with DAMQ buffers.

A key goal of the crossbar arbiter is to maximize throughput and minimize average latency. This can be accomplished by maximizing the number of packets that are transmitted simultaneously, taking into account the resources required by the packets ready for transmission. We have previously introduced *symmetric crossbar arbiters*, which meet this goal for switches with multi-queue input buffers [10]. However, these symmetric crossbar arbiters do not guarantee fairness. Specifically, it is possible for a packet to remain in a switch input buffer indefinitely. This paper deals with the design of symmetric crossbar arbiters which prevent this *starvation* condition. Symmetric crossbar arbiters are described in Section II. Section III discusses possible causes of the starvation problem and presents a technique to overcome it. Section IV presents an evaluation of the performance impact of this technique under uniform traffic and a particular nonuniform traffic pattern. The VLSI implementation of a starvation-free arbiter is described in Section V.

II. Symmetric Crossbar Arbiters

Figure 2 shows an example of arbitration for an $n \times n$ switch with two alternative input buffer organizations: FIFO and DAMQ. As discussed earlier, it is possible to transmit more packets simultaneously (connect more crosspoints) with multi-queue buffers than with FIFO buffers. With multi-queue buffers, there are up to n^2 requests (one from each queue) but only one request can be granted in each row and only one in each column. Each input port contends for *multiple* output

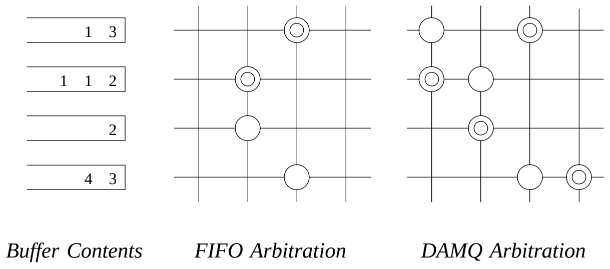


Figure 2: Example arbitrations for switches with FIFO and DAMQ buffers. Double circles indicate granted requests. Single circles indicate denied requests. The numbers in the buffers indicate the destinations of the packets.

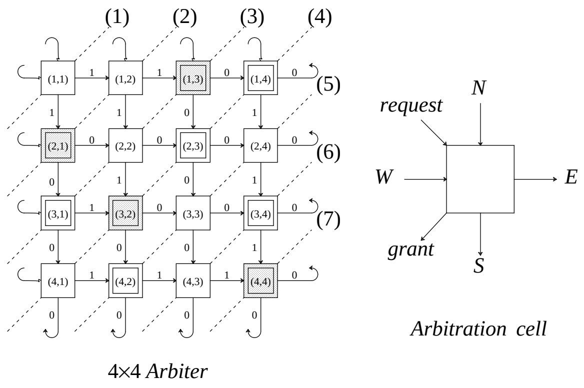


Figure 3: A wave-front symmetric crossbar arbiter. Cell (1,1) has the top priority. The numbered diagonals indicate the progression of the arbitration wave front. Double squares indicate that the corresponding crosspoint has been requested. Shaded squares indicate that the corresponding crosspoint has been granted.

ports but needs only one for full utilization. Similarly, each output port contends for multiple input ports and needs one for full utilization. The arbitration task is thus *symmetrical* with respect to inputs and outputs.

A 4×4 symmetric crossbar arbiter, called the *wave-front* (WF) arbiter [10], is shown in Figure 3. The arbiter consists of n^2 arbitration cells, and each cell corresponds to a crosspoint. For each crosspoint, there is a *request* (R) input and a *grant* (G) output. Each cell also has two inputs, *north* (N) and *west* (W), and two outputs, *south* (S) and *east* (E). The N signal indicates that there are no granted requests for the crosspoints above in the same column. The W signal indicates that there are no granted requests for the crosspoints to the left in the same row. The G output is asserted if, and only if, the crosspoint is requested and both the N and the W inputs are asserted, All the N and W inputs for the highest priority row and column, respectively, are set to 1.

The entire arbitration array is implemented as a combinational circuit, and the arbitration is completed in one clock cycle. Arbitration starts with one top priority cell. The arbitration cells reach their final configuration in a ‘‘wave front’’ that moves diagonally from the top left corner to the bottom right corner of the arbiter. The top priority is given to a different cell every cycle such that fairness is improved. Assuming the propagation delay for a cell is T time units, the whole arbitration completes after $(2n-1)T$ time units.

III. Starvation Prevention

Without a starvation prevention mechanism, communication switches with multi-queue input buffers cannot guarantee that a packet arriving at the switch will be forwarded to the appropriate output within a finite period of time. This is obvious if the top priority is fixed at a queue (arbitration cell) which is not the one holding the packet in question. Even if the top priority is rotated to a different cell every cycle, eventual transmission of the packet is *not* guaranteed. Specifically, although the queue containing the packet periodically gets the top priority, it is possible that whenever that happens the output port that corresponds to this queue is blocked (e.g., due to a full buffer in the next switch). Hence, low priority queues may be allowed to transmit their packets while the top priority queue remains blocked and priority is then shifted to another queue. Such *starvation* conditions may delay packets in an ‘‘unlucky’’ queue indefinitely.

Starvation prevention in *synchronous* networks is relatively simple [10]. In a synchronous network, packets are transmitted and received by the communication switches in lock-step. The steps are called *stage cycles* [11]. Assuming no contention, an entire packet is transmitted from one switch to its neighbor in a single stage cycle. No packet is ‘‘in transit’’ across a stage cycle boundary so *all* the resources assigned by the crossbar arbiter — input ports and output ports — are freed at the end of each stage cycle. In such networks the WF arbiter can guarantee prevention of starvation if a nonempty top priority queue maintains its top priority until it succeeds in sending one packet [10]. We will call a WF arbiter that uses this scheme to rotate the top priority, a *round-robin* (RR) arbiter.

Starvation prevention in asynchronous networks is more difficult than in synchronous networks. In asynchronous networks it takes multiple clock cycles to transmit an entire packet from one switch to its neighbor. However, packets may arrive at a switch during any clock cycle and may be of variable length. Thus, different packets utilize input and output ports for transmission for different durations. Once an entire packet is transmitted to a neighbor switch, the crossbar row and column it has been using become available to another packet. Hence, the resources assigned by the crossbar arbiter are not all freed simultaneously so that they can be assigned at will to the top priority queue. It is possible for the top priority queue to be starved indefinitely since the row and column that it needs are never free at the same time.

Figure 4 demonstrates a possible scenario of starvation in an asynchronous networks with multi-queue buffers. Queue (3,2) is starved despite the fact that it maintains the top priority since row 3 and column 2 are never available at the same cycle. When row 3 is released by some other queue in the same row,

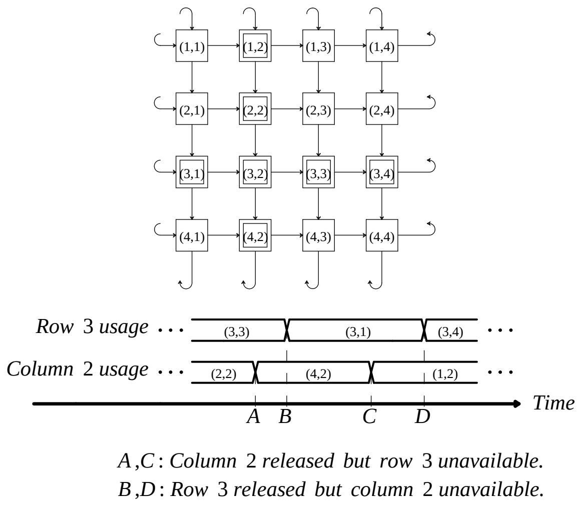


Figure 4: An example of starvation for symmetric crossbar arbitration. The requesting queue (3,2) keeps being rejected because row 3 and column 2 are occupied by other queues and are not both available at the same cycle for queue (3,2).

column 2 is still being used by another queue in the same column. Hence, the arbiter allocates row 3 to another requesting queue in the same row before column 2 is released. Similarly, when column 2 is released by some other queue in the same column, row 3 is still being used by another queue in the same row. Hence, column 2 is allocated to another requesting queue in the same column before row 3 is released.

For an asynchronous network, a starvation-free arbiter can be implemented by modifying the RR arbiter discussed above so that it may *reserve* a resource required by the top priority queue even if the other resource required by this queue is not available. Specifically, in addition to stopping the priority rotation, the top priority queue prevents the assignment of the row and column it needs to other queues until its request is granted. If one resource (either the row or the column) becomes available, the arbiter does not allocate this resource to any other requesting queues. Since the other resource is eventually freed, the request from the top priority queue can be granted then. This starvation-free arbitration scheme is called the *symmetric greedy reservation* (SGR) arbiter. Note that while a row is unavailable only if another queue is using it, a column is unavailable if either another queue is using it or the corresponding output port is currently blocked.

Since some resources are kept idle while reserved, the reservation mechanism for an SGR arbiter may cause performance degradation. To alleviate this problem, a *starvation counter* can be incorporated in the arbiter. The purpose of the starvation counter is to check if the requesting top priority queue is ‘‘near starvation.’’ If the top priority request has been rejected for a certain number of clock cycles (*starvation count threshold*), the arbiter starts to reserve the two requested resources. Otherwise, the arbiter operates as an RR arbiter. An SGR arbiter with a starvation count threshold equal to k is denoted as an SGR-k arbiter.

Although both the row and the column are needed for the top priority queue to be granted, reserving one resource is sufficient for resolving the starvation problem. A *row-greedy reservation* (RGR) arbiter is similar to an SGR arbiter except that only the row can be reserved. A column is never reserved and whenever it is freed it is assigned to a queue that can use it immediately. However, a row can be reserved by a top priority queue. After the row has been reserved, the top priority queue obtains and keeps the row the moment it becomes available. Eventually the column required by the top priority queue becomes available, and then the requesting top priority queue can obtain both the row and the column. Likewise, we can reserve the column only. This arbiter is called a *column-greedy reservation* (CGR) arbiter. Initial simulation studies indicate that the SGR arbiter performs slightly better than the RGR arbiter and the CGR arbiter. Hence, the rest of this paper focuses on SGR arbiters.

IV. Performance Evaluation

We have evaluated the performance of the starvation-free arbiters under uniform traffic and nonuniform traffic for single switches using event-driven simulation [8]. Links and buffers are assumed to be byte wide. Each buffer has independent read and write ports and can store 128 bytes. A byte can be read from the buffer, written into the buffer, or transferred through a link in one cycle. Virtual cut-through switching is used and the minimum delay for a packet going through the communication switch is five cycles [5, 9]. The simulations use packet sizes which are uniformly distributed between 8 and 32 bytes. *Senders* generate packets and send them to the switch. The interval between packet creation follows a geometric distribution. However, a packet is sent into the switch only if the corresponding input buffer has room for the entire packet. Otherwise, the sender blocks.

The latency of a packet is the number of cycles that elapse from when the first byte of a packet is generated in the sender to when it leaves the switch, including the wait for the blocked input port. The normalized throughput is the average number of bytes received by each output per clock cycle.

The starvation-free arbiters are compared to two arbiters that do not provide guaranteed packet delivery. One of these two is the round-robin (RR) arbiter, discussed earlier. The other is the *oblivious round-robin* (ORR) arbiter, which is a WF arbiter that shifts the priority every cycle.

Under *uniform traffic* load, packet destinations are uniformly distributed over all the switch outputs. Figure 5 shows how different arbiters impact the performance of a 4×4 switch under uniform traffic. The results indicate that the reservation mechanism used by the SGR arbiters results in a small performance degradation. A high starvation count threshold minimizes this performance degradation.

As shown in Figure 6, under uniform traffic, the performance degradation caused by the starvation prevention mechanism is larger for the 2×2 switch than for the 4×4 switch. The reason for this is that when a row or a column is reserved for the top priority queue, the number of resources the arbiter can allocate to the other queues is reduced. As the switch size

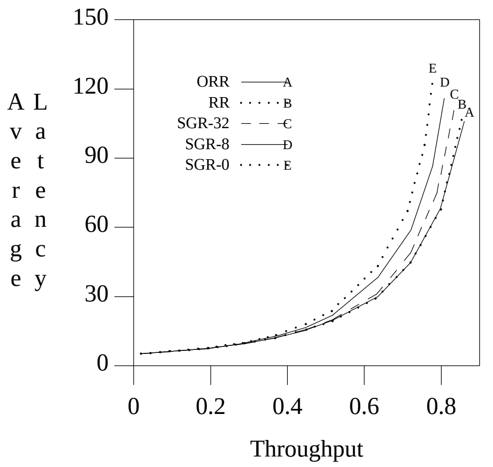


Figure 5: Average latency vs. normalized throughput of a 4×4 switch for the different arbiters under uniform traffic.

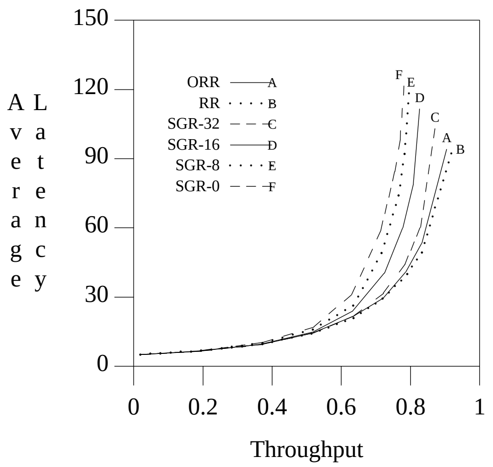


Figure 6: Average latency vs. normalized throughput of a 2×2 switch for the different arbiters under uniform traffic.

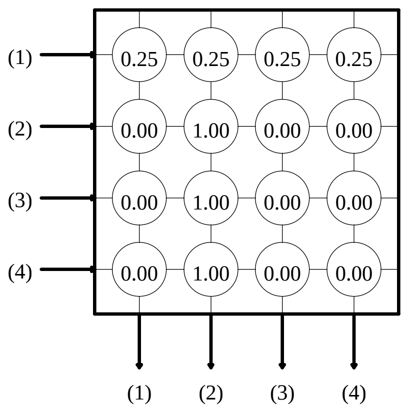


Figure 7: A nonuniform traffic pattern for a 4×4 switch. The numbers at each crosspoint are the probability distribution of the destinations for packets arriving at the input port.

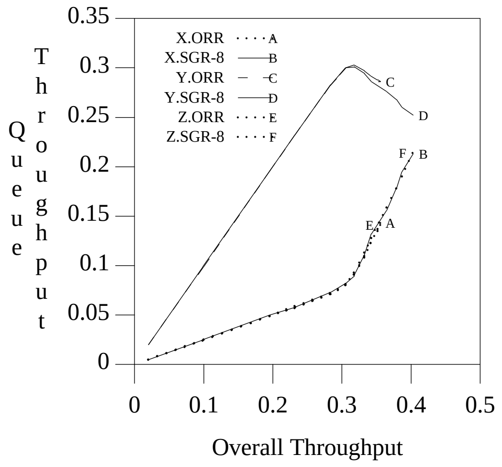


Figure 8: The impact of starvation prevention on the performance of different queue groups — an SGR-8 arbiter is compared to an ORR arbiter under the traffic pattern in Figure 7. Queue throughput vs. overall throughput. X includes queue (1,2) only; Y queues (2,2), (3,2), and (4,2); Z queues (1,1), (1,3), and (1,4).

increases, the percentage of the reserved resources over the total resources decreases.

One of the goals of our simulation studies was to evaluate the performance of the starvation prevention mechanism under nonuniform traffic patterns that could cause some packets to be ‘‘stuck’’ in the switch for an inordinately long time. To this end we have used the nonuniform traffic pattern shown in Figure 7. With this traffic pattern, queue (1,2) is ‘‘unfavored’’ because it is the only queue that receives packets and has to compete for *two* resources in order to send these packets. The queues associated with the other three inputs only compete for output port (column) 2. The other three queues of input 1 only compete for row 1.

To evaluate the performance of different arbiters under nonuniform traffic, we use *overall throughput*, *queue throughput* and *queue latency* as performance measures. Overall throughput is the average normalized throughput over all the outputs (different outputs can have different normalized throughputs in the nonuniform traffic case). Queue throughput is the average number of bytes transmitted from a defined group of queues. Queue latency is the average latency of the packets which are transmitted from a defined group of queues.

Using the traffic pattern from Figure 7, Figure 8 shows the performance comparison between the ORR arbiter and the SGR-8 arbiter in terms of queue throughput vs. overall throughput for three groups of queues. Figure 9 shows the performance comparison in terms of queue latency vs. overall throughput. Group X includes queue (1,2) only. Group Y includes queues (2,2), (3,2), and (4,2). Group Z includes queues (1,1), (1,3), and (1,4). The results indicate that the reservation mechanism for a starvation-free arbiter has little impact on groups Y and Z. However, under high throughput, it significantly reduces the queue latency of queue (1,2).

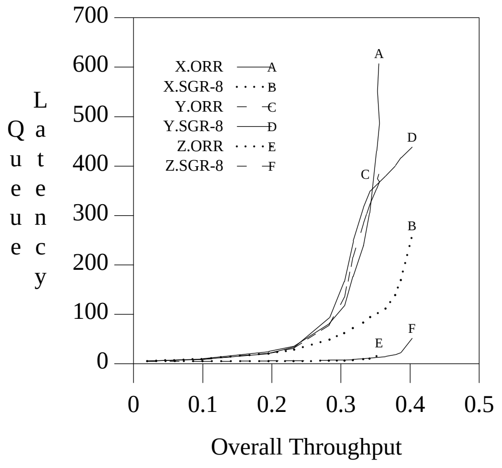


Figure 9: The impact of starvation prevention on the performance of different queue groups — an SGR-8 arbiter is compared to an ORR arbiter under the traffic pattern in Figure 7. Queue latency vs. overall throughput. X includes queue (1,2) only; Y queues (2,2), (3,2), and (4,2); Z queues (1,1), (1,3), and (1,4).

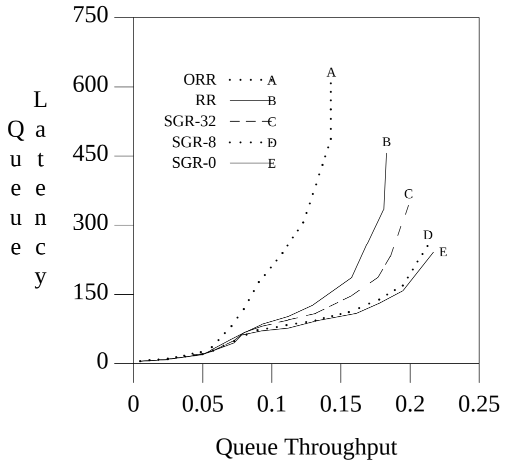


Figure 10: The performance impact of different arbiters on queue (1,2) under the traffic pattern in Figure 7. Queue latency vs. queue throughput.

Figures 10, 11, and 12 show queue latency vs. queue throughput — the latency and throughput shown are only for the specific queue group in question. However, in all three figures the *applied load* from all the senders is varied uniformly to obtain these results.

Figure 10 shows the performance impact of different arbiters on queue (1,2) under the traffic pattern in Figure 7. A significant reduction in latency is achieved by changing the priority shifting scheme from ORR to RR. However, the reservation mechanism of the SGR arbiters leads to additional reductions in latency. As the starvation count threshold is decreased, the queue latency decreases since the reservation mechanism is triggered earlier.

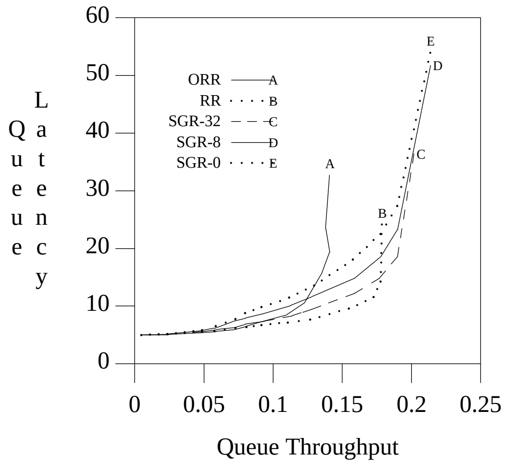


Figure 11: The performance impact of different arbiters on queues (1,1), (1,3), and (1,4) under the traffic pattern in Figure 7. Queue latency vs. queue throughput.

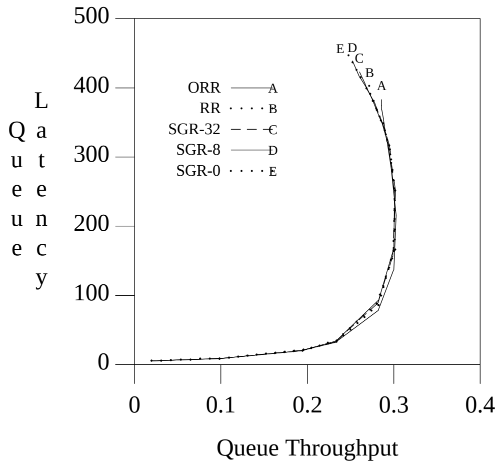


Figure 12: The performance impact of different arbiters on queues (2,2), (3,2), and (4,2) under the traffic pattern in Figure 7. Queue latency vs. queue throughput.

Figure 11 shows the performance impact of the different arbiters on queues (1,1), (1,3), and (1,4) under the traffic pattern in Figure 7. The ORR arbiter achieves the lowest maximum queue throughput. The reason for this is that with ORR queue (1,2) is more likely to be starved, causing the row 1 input buffer to fill up with packets destined for output 2. In this case, no sender 1 packets destined to the other outputs can enter the switch. The SGR arbiters can achieve a slightly higher queue throughput than the RR arbiter since they are more likely to prevent input buffer 1 from filling up with queue (1,2) packets. This effect increases with decreasing starvation count threshold since the reservation mechanism is triggered earlier. On the other hand, for a given queue throughput, decreasing the starvation count threshold increases the latency since the reservation mechanism is more likely to be triggered, allowing transmission from queue (1,2) and delaying packets in the other queues of input 1.

Figure 12 shows the performance impact of the different arbiters on queues (2,2), (3,2), and (4,2) under the traffic pattern in Figure 7. It might be expected that queues (2,2), (3,2), and (4,2) would exhibit poorer performance for starvation-free arbiters due to the reservation of column 2 for queue (1,2). Figure 12 shows that the maximum achievable queue throughput is the same for all the arbiters. However, as shown in Figure 8, the maximum queue throughput for this group (group Y) is not achieved when the overall switch throughput is maximized. Instead, the maximum queue throughput for group Y is achieved at a point where the reservation mechanism has no impact on the queue throughput for queue (1,2). As the applied load is increased beyond this point, the group Y throughput actually decreases. This decrease is most pronounced with the SGR-0 arbiter precisely because this arbiter reserves column 2 for queue (1,2) most often.

V. Implementation

In order to demonstrate the viability of the design of the starvation-free arbiters, we have laid out a 4×4 SGR-32 arbiter in custom VLSI. In the SGR arbiters, the capability of starvation prevention is efficiently implemented owing to the regular structure of the symmetric crossbar arbiters. To reserve the row and the column, the top priority arbitration cell simply ‘‘disables’’ the wave front output signals in horizontal and vertical directions.

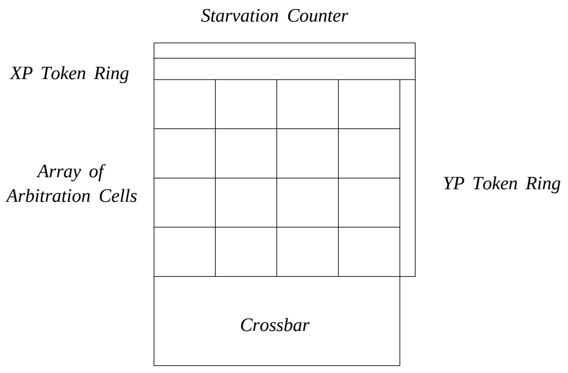


Figure 13: The floorplan of a 4×4 crossbar of eight-bit wide buses with the SGR-32 arbiter. The modules are drawn to scale.

Figure 13 shows the floorplan of the SGR-32 arbiter. The XP and YP token rings are used to point to the top priority cell in the cell array. The layout of the crossbar is 990 λ wide by 380 λ tall, while the layout of the arbiter is 1050 λ wide by 930 λ tall. SPICE circuit simulations using 2 μ CMOS technology indicate that the worst case delay for the arbiter is 23 ns.

A 4×4 RR arbiter has also been laid out in order to measure the cost of adding the starvation prevention mechanism. The chip area of this RR arbiter is 1010 λ wide by 870 λ tall. The worst case delay for the RR arbiter is 20 ns. Hence, if the arbitration delay is part of the critical path of the switch, adding the starvation prevention mechanism does have a small performance penalty which was not taken into account in the previous section.

VI. Summary and Conclusions

While high bandwidth and low latency are critical requirements from interconnection networks, fairness and a guarantee of timely delivery of *every* packet must also be provided. In an asynchronous network where the communication switches use multi-queue input buffers, eventual packet delivery cannot be guaranteed unless special starvation prevention mechanisms are employed. We have presented a simple starvation prevention mechanism for the wave-front arbiter used in such communication switches. We have laid out in custom VLSI wave-front arbiters with and without the starvation prevention mechanism. The overhead of the mechanism was found to be 11% in terms of layout area and, in the worst case, 15% in terms of arbiter circuit performance. The mechanism presented does guarantee eventual packet delivery. Furthermore, simulation studies have shown that, under certain nonuniform traffic patterns, the mechanism can improve performance for packets in what would otherwise be a particularly slow queue at a cost of a relatively small decrease in performance for packets in some other queues.

References

1. W. J. Dally and C. L. Seitz, ‘‘The Torus Routing Chip,’’ *Distributed Computing* **1**(4), pp. 187-196 (October 1986).
2. W. J. Dally, ‘‘Virtual-Channel Flow Control,’’ *17th Annual International Symposium on Computer Architecture*, Seattle, WA, pp. 60-68 (May 1990).
3. A. Gottlieb, R. Grishman, C. P. Kruskal, K. P. McAuliffe, L. Rudolph, and M. Snir, ‘‘The NYU Ultracomputer - Designing an MIMD Shared Memory Parallel Computer,’’ *IEEE Transactions on Computers* **C-32**(2), pp. 175-189 (February 1983).
4. W. D. Hillis and L. W. Tucker, ‘‘The CM-5 Connection Machine: A Scalable Supercomputer,’’ *Communications of the ACM* **36**(11), pp. 30-40 (November 1993).
5. P. Kermani and L. Kleinrock, ‘‘Virtual Cut Through: A New Computer Communication Switching Technique,’’ *Computer Networks* **3**(4), pp. 267-286 (September 1979).
6. M. Kumar and J. R. Jump, ‘‘Performance Enhancement in Buffered Delta Networks Using Crossbar Switches and Multiple Links,’’ *Journal of Parallel and Distributed Computing* **1**(1), pp. 81-103 (1984).
7. R. J. McMillen and H. J. Siegel, ‘‘The Hybrid Cube Network,’’ *Distributed Data Acquisition, Computing, and Control Symposium*, pp. 11-22 (December 1980).
8. S. M. Swope and R. M. Fujimoto, ‘‘Simon II Kernel Reference Manual,’’ Technical Report UUCS 86-001, University of Utah, Salt Lake City, UT (March 1986).
9. Y. Tamir and G. L. Frazier, ‘‘Dynamically-Allocated Multi-Queue Buffers for VLSI Communication Switches,’’ *IEEE Transactions on Computers* **41**(6), pp. 725-737 (June 1992).
10. Y. Tamir and H.-C. Chi, ‘‘Symmetric Crossbar Arbiters for VLSI Communication Switches,’’ *IEEE Transactions on Parallel and Distributed Systems* **4**(1), pp. 13-27 (January 1993).
11. H. Yoon, K. Y. Lee, and M. T. Liu, ‘‘Performance Analysis of Multibuffered Packet-Switching Networks in Multiprocessor Systems,’’ *IEEE Transactions on Computers* **39**(3), pp. 319-327 (March 1990).