Proceedings of the
International Conference on Computer Design
Cambridge, Massachusetts, pp. 466-471 (October 1989).

THE DESIGN AND IMPLEMENTATION OF A MULTI-QUEUE BUFFER FOR VLSI COMMUNICATION SWITCHES †

Gregory L. Frazier and Yuval Tamir

Computer Science Department
University of California
Los Angeles, CA 90024

Abstract

Small $n \times n$ switches are key components of multistage interconnection networks and communication coprocessors used in multiprocessors and multi-computers. Communication latency and throughput are critically dependent on the structure of the internal buffers in these switches. We have previously introduced the architecture of a new type of buffer, called a *dynamically-allocated multi-queue* (DAMQ) buffer, that provides non-FIFO message handling and can support higher throughput at lower latency than the commonly used FIFO buffer. In this paper, we present a micro-architecture and VLSI implementation of a DAMQ buffer. We discuss design tradeoffs for the DAMQ buffer's datapath and present a floorplan and the timing of the major functional units. This paper shows that in VLSI switches, with buffers that can store multiple packets, additional chip area is better used for the control of DAMQ buffers than for increased buffer space in simpler FIFO buffers.

I. Introduction

Interconnection networks composed of small $n \times n$ switches are used for interprocessor communication in both multiprocessors with multi-stage networks [2, 4] and multicomputers with point-to-point links [6]. The architecture and implementation of the $n \times n$ switches, particularly their internal buffers, are critical for achieving, at low cost, the performance goals of high-throughput low-latency communication.

One of the key issues in the design of $n \times n$ switches is the handling of messages at different input ports which are simultaneously ready to be sent out through the same output port (*output port contention*). This situation can be handled by allowing internal buffering of packets in the switch. In many communication switches [3] buffering is done in first-in first-out (FIFO) queues located at the input ports. The problem with FIFO buffers is that only the packet at the head of the queue can be read and transmitted. If the packet at the head of the queue is blocked, all other packets in the buffer, even those destined to different output ports, are also blocked. To overcome this limitation of FIFO buffers, we have designed a new type of buffer, called a *dynamically-allocated multi-queue* (DAMQ) buffer, in which several queues of packets, one for each output port, are maintained at each input port [7, 8]. As shown in Figure 1, the idle time of buffers and ports due to output port contention can be reduced using non-FIFO buffers, thus leading to significant increases in the throughput of the switch.

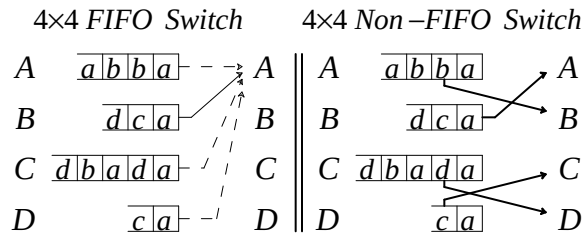

Figure 1: Reducing the performance penalty of output port contention. In this example, the non-FIFO buffers quadruple the throughput relative to the FIFO buffers.

In order to evaluate the performance of the DAMQ buffer, we have simulated communication in a 64×64 Omega network constructed from three stages of 4×4 switches [7, 8]. As shown in Figure 2, due to the buffer's ability to utilize the output ports despite output port contention, the network with DAMQ buffers was able to achieve significantly higher throughput and lower latency than the network of equal size FIFO buffers.

The research described in this paper is part of the UCLA ComCoBB (**Com**munication **Co**processor **B**uilding-**B**lock) project, whose focus is the design and implementation of a single-chip high-performance communication coprocessor for VLSI multicomputers. While the DAMQ buffer was designed for use within the ComCoBB chip, this paper describes a ''stand-alone'' chip, the *DAMQ Chip*, that contains only the DAMQ buffer. A complete communication switch will require one or more DAMQ Chips together with several other key building blocks, such as a router and arbiter.

† This research is supported by Rockwell International and the State of California MICRO program.

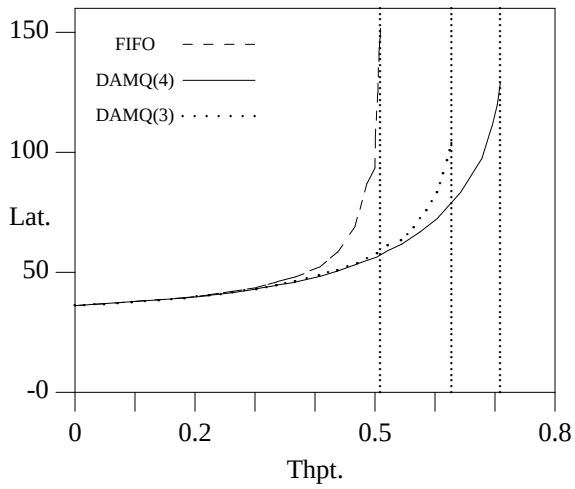

Figure 2: Latency vs. Throughput. 64×64 Omega network. 4×4 switches. Four packet slots per buffer FIFO. Four and three packet slots per buffer DAMQ. Uniform traffic.

The next section describes the requirements of the DAMQ Chip, its basic organization, and its microarchitecture. Layout and circuit performance are discussed in Section III.

II. The Architecture and Microarchitecture of the DAMQ Chip

The DAMQ Chip is part of a communication switch that forwards packets through the network and is also the interface between the network and one computation node. Incoming packets are stored in a memory array (the buffer at the input port) while their *header byte*, which contains routing information, is sent to the router. The router returns the switch output port to which the packet should be forwarded. With some routing schemes, such as virtual circuits [1], the router also returns a new header byte to be prepended to the packet on its next ''hop'' through the network.

In order to forward a packet, a connection must be established between the DAMQ Chip and the appropriate output port. The switch arbiter determines which connections to establish based on the state of all the input buffers and the switch output ports. Upon receiving notification from the arbiter that a connection has been established, the DAMQ Chip commences transmission of the first packet destined to the output port specified by the arbiter. The packet is transmitted at a rate of one byte per clock cycle, with the first byte being the new header supplied by the router.

The performance goals for a communication switch are to minimize latency (delay) through the switch and maximize total throughput (bandwidth). In the rest of this section we describe the design and implementation of several features in the DAMQ buffer which help achieve these goals. As discussed in Section I, non-FIFO access to packets in the input-port buffers can reduce unnecessary delays and low utilization of the ports due to output-port contention. Dual port buffers, which allow simultaneous reception and transmission of packets, are critical for maximizing port utilization, thus increasing switch throughput. When the network is lightly loaded, latency can be dramatically reduced by switches that support *virtual cut-through* [5] — packet transmission out of the switch can begin soon after the first byte of a packet arrives, instead of after the entire packet is received. Efficient handling of variable length packets improves the utilization of the internal buffers, thus increasing the throughput of the switch.

A. The Basic Organization of the DAMQ Chip

The DAMQ buffer is a single storage array in which multiple FIFO queues are maintained, one for each output port. Such dynamic partitioning of the buffer between the queues is significantly more efficient than static partitioning [7]. While the packets in each queue are maintained in FIFO order, the order in which queues are serviced is ''random'' (determined by the arbiter). High performance requires the ability to rapidly locate, access, and transmit the first packet in any one of the queues. In order to handle variable-length packets efficiently, the DAMQ buffer must be able to quickly allocate memory for incoming packets without wasting buffer space due to internal or external fragmentation.

The memory array of the DAMQ buffer is partitioned into fixed-size *buffer blocks* [7]. Each queue is a linked list of these blocks. *Head registers* and *tail registers*, which point to the first and last block of each queue, are located outside the array of buffer blocks, thus allowing access to any one of the queues. Each packet is stored in one or more buffer blocks. For each queue, the linked list of buffer blocks defines the FIFO order between packets as well as the order of blocks that make up each packet. To reduce internal fragmentation due to variable size packets, the buffer blocks (eight bytes) are significantly smaller than the maximum packet size (thirty-two bytes in this implementation). With fixed-size blocks, there is no external fragmentation and rapid memory allocation is facilitated. A linked list of ''free buffer blocks'' (the *free list*) is used to keep track of storage available for incoming packets. For each buffer block there is a *pointer register*, stored outside the packet memory array, which contains the address of the next buffer block or a *NULL* value signifying the end of the last packet in the queue.

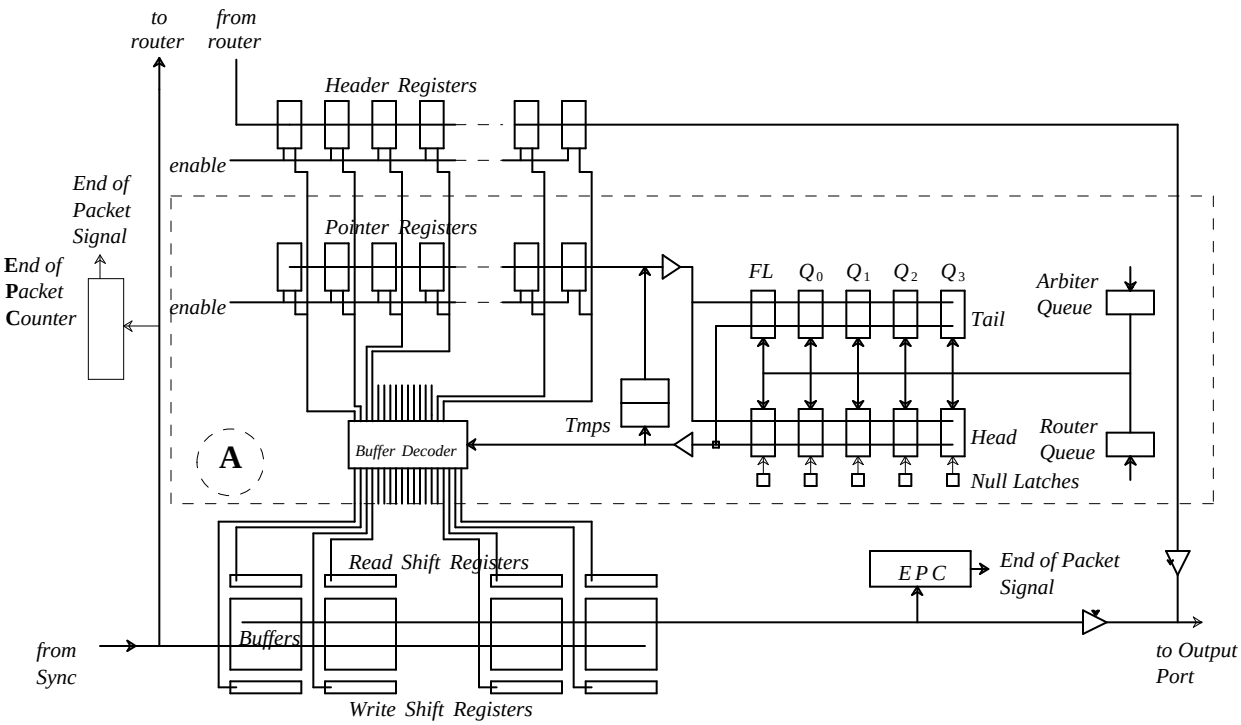

Figure 3: The control logic data path of the DAMQ buffer. The area marked 'A' contains the logic which is specific to the DAMQ buffer. The rest of the logic would be necessary for a FIFO buffer design.

B. The DAMQ Buffer Control

The control logic of the DAMQ buffer is quite complex. The process of receiving a packet involves: (1) locating the block at the head of the free list and directing incoming data to that block, and (2) altering the linked list pointers, logically moving the block containing the beginning of the new packet to the tail of the appropriate queue. When a packet is transmitted, the block which is at the head of the current queue is moved to the tail of the free list while its data is being read.

The control of the DAMQ buffer must be able to manipulate the linked list pointers, moving buffer blocks from one linked list to another, in as few clock cycles as possible. The control must also satisfy three additional important requirements: (1) the silicon area required for the buffer control hardware should be minimized, (2) the hardware must allow the simultaneous reception and transmission of packets, and (3) the control logic should not be on the DAMQ buffer's critical path. The last requirement is key to allowing the DAMQ buffer to realize its performance advantage [7, 8] over FIFO buffers. With a FIFO buffer, due to its simple control, other factors (e.g. reading from the memory array or the raw bandwidth of the links) will determine the chip clock rate. Due to the complexity of the DAMQ buffer control, there was the potential for it to force a significantly slower clock rate, thus eliminating the performance gained by non-FIFO access to the packets.

The datapath and key control circuitry of the DAMQ buffer are shown in Figure 3. Logically moving a block from one queue to another requires three clock cycles:

cycle 1: The value in the head register of the queue the block is being removed from is copied to one of the two temporary registers. This value is also sent to the decoder of the pointer-register array, where it is used to fetch the pointer register containing the address (block number) of the next block in the queue. This address is copied to the head register, thus removing the block from the queue.

cycle 2: The value in the temporary register, which is the address of the block being ''moved,'' is copied to the pointer register whose address is in the tail register of the destination queue. The value in the temporary register is also copied to the tail register of the destination queue.

cycle 3: The *NULL* value is stored in the pointer register whose address is in the tail register of the destination queue.

When a packet is received, a block is removed from the free list queue and placed at the tail of the queue specified by the router. When a packet is transmitted, the arbiter specifies the queue from which the blocks are removed, and they are appended to the free list. Since the buffer must be able to simultaneously receive and transmit packets, the buses and registers are multiplexed.

The transmission logic and reception logic control the data path on alternate clock cycles. Hence, a minimum of six clock cycles are needed for the linked list manipulations involved in receiving or transmitting a packet. However, when a packet destined for an empty queue arrives, there is no need to wait for these operations to complete before beginning to forward it (Section II.C).

The data path and buffer memory are controlled by three finite state machines (FSMs). FSM_{sto} stores incoming packets into buffer blocks. FSM_{rtr} removes blocks from the free list and appends them to the queue specified by the router. It uses the data path on the same cycles that FSM_{sto} does. FSM_{fwd} forwards the packets. This division of control allows concurrent packet reception, routing and transmission.

C. Support for Virtual Cut Through

With virtual cut through [5] the switch begins to forward a packet before it has completely arrived. A packet can only be cut through a switch if the buffer at which it is arriving is not already transmitting a packet, the output port for which the packet is destined is not connected to another buffer, and there are no packets already in the buffer waiting to be transmitted through that same output port. Optimally, when these conditions exist, the packet will be forwarded as soon as it is routed and the buffer is connected to the appropriate output port.

Thus, there are two keys to virtual cut through. First, the routing of the packet and arbitration of the switch must be performed as quickly as possible in parallel with packet reception. Second, the packet must be available for transmission as soon as the arbitration is complete, no matter how much of the packet has been received at that point. The first condition is satisfied by keeping the routing information at the head of each packet, and by keeping the router and arbiter units independent of the buffer itself. These two measures allow the routing to commence the cycle the packet begins to arrive, and the arbitration to commence as soon as the routing is complete. The second condition is partially met by having the three separate FSMs, but some special-purpose hardware is necessary to ensure that the arriving packet is available for transmission when the arbitration is complete.

As was previously mentioned, the internal organization of the DAMQ buffer is based upon linked lists. Each queue is referenced by two registers; the head register points to the first block in the queue, and the tail register points to the last. When the queue is empty, however, the tail register holds the NULL value, and the head register holds garbage. Using the same queue manipulating sequence described in Section II.B, the tail register will be set to point to the arriving packet when the packet is enqueued. In the case of an empty queue, however, the head register must also be set to point to the arriving packet. Furthermore, this must happen quickly enough to not delay the packet from being cut through (FSM_{fwd} acquires the address of the block to be transmitted from the head register).

The head register need not have the correct address until the arbiter has connected the buffer to the output port. The switch arbiter is notified of the output port request on the same clock cycle that the DAMQ buffer receives the queue number and new packet header from the router. On this clock cycle or the following, the block number is stored in the tail register of that queue. Every time a value is written to a NULL tail register, it is also written to the corresponding head register. Thus, the head register will also be set to point to the first block of the incoming packet. Even if the arbiter connects the buffer to the output port in a single clock cycle, the correct block number is in the head register in time for the new header to be read.

Little hardware is required to support virtual cut through. A register latches the packet's header as soon as it passes through the synchronizer, and presents it to the router along with an enable signal, allowing the routing to take place immediately. The same separate control and addressing hardware which allows the DAMQ buffer to simultaneously receive and transmit separate packets also allows it to cut a single packet through. Finally, five *NULL latches* cause dual tail/head register writes to empty queues. With an on-chip router based upon simple algorithmic routing or small a routing table, and an on-chip arbiter which operates in a single clock cycle, the minimum latency across a single DAMQ switch is four clock cycles. Each additional cycle required to route a packet or arbitrate the switch adds a cycle to this latency.

III. Floorplan and Circuit Performance

In Figure 2, it was shown that, given equal clock rates, a network made up of DAMQ buffers outperforms a corresponding FIFO network. The microarchitectures of the DAMQ and FIFO buffers must be compared, however, to determine whether and to what degree the additional complexity of the DAMQ buffer affects its performance. This complexity could cause the DAMQ buffer to operate at a slower clock rate than the FIFO buffer. In addition, the control logic could occupy silicon area which the FIFO buffer would use for additional storage. Having more storage for incoming data increases the FIFO buffer's performance relative to that of the DAMQ.

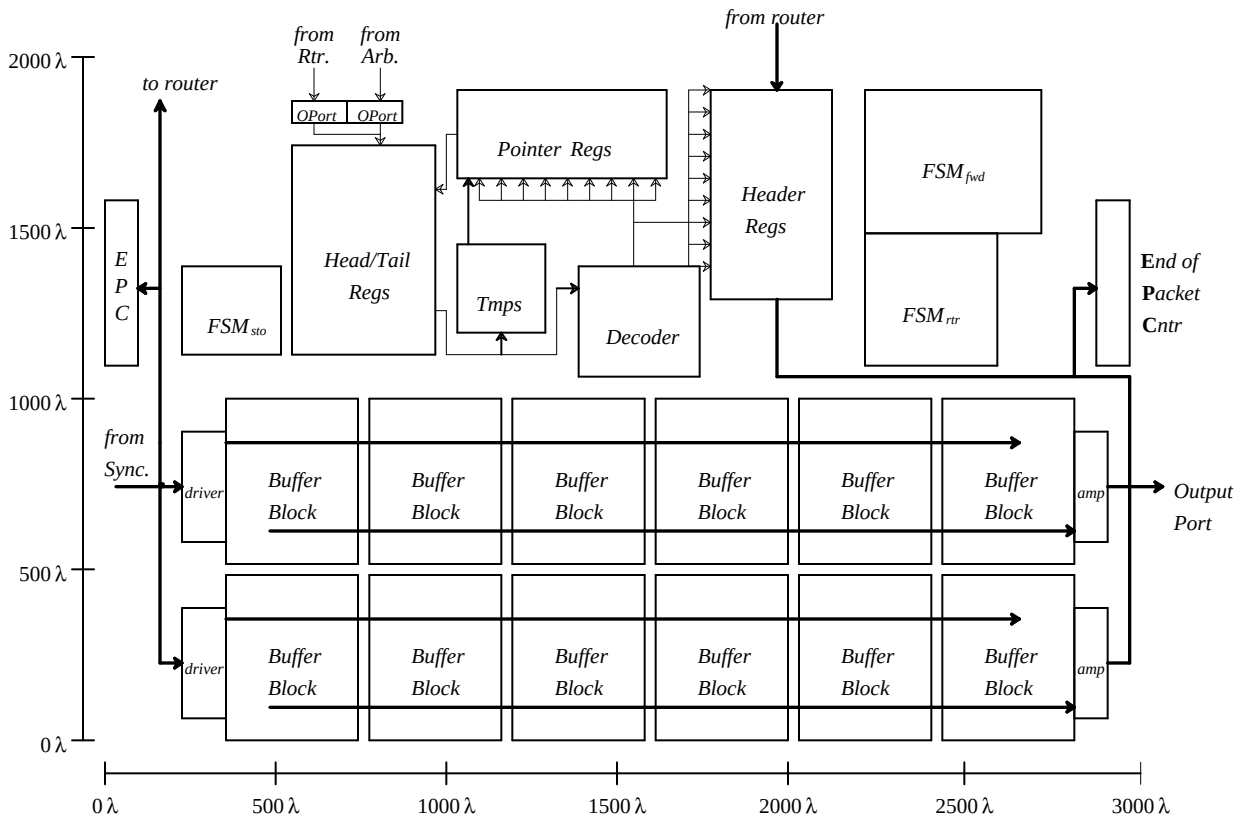

Figure 4: The floorplan for the DAMQ buffer.

In order to compare the performance of the DAMQ buffer to that of the FIFO, both buffers have been designed and layed out using MOSIS scalable (2 μ) CMOS design rules. The DAMQ and FIFO buffers have been designed with ''similar'' features to restrict the causes of any difference in performance to the microarchitecture. Thus, both use three clock phases (Figure 5), both are currently designed with ninety-six bytes of buffer memory divided into two forty-eight byte arrays (Figure 4), and both are controlled by multiple FSMs. In addition, both follow the same basic operating sequence, as follows. On the rising edge of the first clock phase (ϕ_1), the FSMs drive their outputs. During ϕ_1, the bus lines across the memory arrays are precharged. During ϕ_2, the memory array is written/read, and any control signals are fed back to the FSMs to be latched for the following cycle. ϕ_3 rises during ϕ_2 and causes the FSMs to latch their inputs and calculate their outputs, to be driven on the next rising edge of ϕ_1. The forty-eight byte memory arrays require 5 ns to be precharged and 10 ns to be read.

The DAMQ buffer must perform its queuing functions in addition to the basic sequence described above. Specifically, in a single clock cycle, it must be able to read a head or tail register, decode the value read, read the specified pointer register and write to a head or

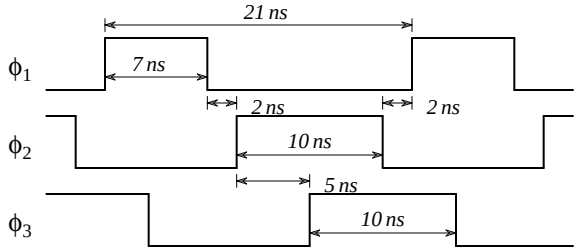

Figure 5: The clocking for the DAMQ buffer. The clocking is the same for the FIFO buffer, except that ϕ_1 is only 5 ns (total cycle 19 ns).

tail register. The head/tail register read occurs in the first phase, and requires 7 ns. On the falling edge of ϕ_1, the value is decoded to address the pointer registers. This happens in less than the 2 ns which separate ϕ_1 from ϕ_2. When ϕ_2 rises, the selected pointer register is read. The pointer register bus carries dual values and is equiped with a sense amplifier which allows the pointer register value to be cranked into the selected head/tail register in 10 ns (matches the memory array read). Finally, 2 ns separate the falling edge of ϕ_2 and the rising edge of ϕ_1. Instead of reading from a pointer register, one of the two temporary registers is sometimes read, and its value is written to both a pointer register and a head/tail register. This is slightly faster than the sequence given above.

The buffer storage is accessed by *shift register addressing* [9] where shift registers are used to access successive bytes, thus eliminating the delays of conventional address decoding which would be on the chip's critical path. The FSMs which control the buffer also have the potential of being on the chip's critical path. The largest of these, FSM_{fwd}, uses a NOR-NOR programmed logic array and requires 5 ns to generate its outputs. FSM evaluation is triggered by the rising edge of ϕ_3, which occurs 7 ns before the rising edge of ϕ_1. Thus, the latency of the FSMs is not on the DAMQ buffer's critical path, either.

Based on the above discussion this implementation of the DAMQ buffer can operate at 48 MHz. The memory array itself requires 5 ns and 10 ns for phases ϕ_1 and ϕ_2, respectively. With a FIFO buffer, its simple control is faster than the memory array, resulting in a maximum potential clock rate of 53 MHz.

Given the above speeds of the FIFO and DAMQ buffers, it is likely that the buffers themselves will not determine the clock rate at which the chip operates. Both the DAMQ and the FIFO buffers operate at the same rate that data is transmitted across the communication links. As the FIFO and DAMQ buffer implementations are scaled to smaller feature sizes, it becomes increasingly likely that the off-chip transmission times will determine the rate at which they operate. If, however, faster links are available, the FIFO buffers can operate at a higher clock rate and the performance advantage of the DAMQ buffers is reduced.

The remaining issue, then, is a comparison of the DAMQ and FIFO layouts to determine how much more buffer space is available to the FIFO buffer, given fixed chip area. The floorplan for the DAMQ buffer is shown in Figure 4. The 96 bytes of storage (slots for three thirty-two byte packets) are broken into two rows with separate read and write buses in order to reduce the latency of reads and writes and to produce a rectangular layout suitable for use as a building block in larger chips. This layout occupies approximately the same area as a FIFO buffer with 128 bytes of storage (slots for four packets). Our simulations indicate that a network of DAMQ buffers with three packet slots outperforms one with FIFO buffers of four packet slots [7] (Figure 2), achieving throughputs 23% higher than the maximum throughput of the FIFO network. Furthermore, even in comparison with the maximum throughput of a network with eight-slot FIFO buffers, a network with three-slot DAMQ buffers saturates at 10% higher throughput and a network with four-slot DAMQ buffers saturates at 24% higher throughput.

IV. Summary and Conclusions

The DAMQ Chip is a VLSI implementation of a multi-queue buffer for use in multiprocessor and multicomputer communication networks. As an interface chip in the node of a multicomputer, it asynchronously receives packets from and transmits packets to other DAMQ Chips. As an example implementation of the DAMQ buffer, it demonstrates that a non-FIFO buffer can be efficiently implemented in VLSI. In this paper, we have presented the design of the DAMQ Chip. Details of the implementation are given, including the floorplan and critical path timing.

We have demonstrated that the DAMQ buffer can operate at high clock rates, despite its complex control. Hence, other factors, such as the inter-chip links, are more likely to limit the raw bandwidth. For a DAMQ buffer with four queues and a packet size of 32 bytes, it was shown that the DAMQ buffer will have one less packet slot available to it than a FIFO buffer occupying approximately the same chip area. Previous work has indicated that, with as few as two packet slots (64 bytes), a network with DAMQ buffers can outperform a FIFO buffer network which has an additional packet slot [7]. With three packet slots, the DAMQ buffer network will saturate at a throughput approximately 23% higher than a FIFO buffer network with four packet slots. The DAMQ buffer is thus shown to be a highly effective building block for packet switches, thus demonstrating that for VLSI switches, increased control complexity may lead to higher performance.

References

1. D. Bertsekas and R. Gallager, *Data Networks*, Prentice Hall (1987).
2. W. Crowther, J. Goodhue, R. Gurwitz, R. Rettberg, and R. Thomas, ''The Butterfly Parallel Processor,'' *IEEE Computer Architecture Newsletter*, pp. 18-45 (September/December 1985).
3. W. J. Dally and C. L. Seitz, ''The Torus Routing Chip,'' *Distributed Computing* **1**(4), pp. 187-196 (October 1986).
4. 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).
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. C. L. Seitz, ''The Cosmic Cube,'' *Communications of the ACM* **28**(1), pp. 22-33 (January 1985).
7. Y. Tamir and G. L. Frazier, ''High-Performance Multi-Queue Buffers for VLSI Communication Switches,'' *15th Annual International Symposium on Computer Architecture*, Honolulu, Hawaii, pp. 343-354 (May 1988).
8. Y. Tamir and G. L. Frazier, ''Support for High-Priority Traffic in VLSI Communication Switches,'' *9th Real-Time Systems Symposium*, Huntsville, AL, pp. 191-200 (December 1988).
9. Y. Tamir and J. C. Cho, ''Design and Implementation of High-Speed Asynchronous Communication Ports for VLSI Multicomputer Nodes,'' *International Symposium on Circuits and Systems*, Espoo, Finland, pp. 805-809 (June 1988).