Pipelined Memory Shared Buffer for VLSI Switches

Manolis Katevenis, Panagiota Vatsolaki, and Aristides Efthymiou

Institute of Computer Science (ICS)
Foundation for Research and Technology − Hellas (FORTH)
Science and Technology Park of Crete, Vassilika Vouton, P.O.Box 1385, Heraklion, Crete, GR 711 10 Greece
E-mail: katevenis@ics.forth.gr Tel.: +30 (81) 391664 Fax: +30 (81) 391661

ABSTRACT: Switch chips are building blocks for computer and communication systems. Switches need internal buffering, because of output contention; shared buffering is known to perform better than multiple input queues or buffers, and the VLSI implementation of the former is *not* more expensive than the latter. We present a new organization for a shared buffer with its associated switching and cut-through functions. It is simpler and smaller than wide or interleaved organizations, and it is particularly suitable for VLSI technologies. It is based on multiple memory banks, addressed in a pipelined fashion. The first word of a packet is transferred to/from the first bank, followed by a ‘‘wave’’ of similar operations for the remaining words in the remaining banks. An FPGA-based prototype is operational, while standard-cell and full-custom chips are being submitted for fabrication. Simulation of the full-custom version indicates that, even in a conservative 1-micron CMOS technology, a 64 Kbit central buffer for an 8×8 switch operates at 1 Gbps/link (worst case) and fits in 45 mm^2 including crossbar and cut-through.

KEYWORDS: *crossbar switch, shared buffering, input queueing, multi-port buffer, pipelined memory, gigabit VLSI switch buffer.*

1. Introduction

Switches (or routers) are basic building blocks for present and future computer and communication systems. They are used to build interconnection networks for large-scale parallel computers [AtSe88], gigabit local area networks for high performance distributed computing [Kung92], and wide area communication networks. Switches are connected, through their links, to other switches or terminal devices. Their function is to make routing decisions and to forward packets that arrive through the incoming links to the proper outgoing link(s). Since the incoming traffic is usually not known or scheduled in advance, multiple packets arriving (almost) simultaneously through different incoming links may have to be forwarded along the same outgoing link.

To deal with such *output contention,* some systems use deflection routing: packets failing to get selected for the desired link are sent along different links, in the hope that when they later return they will not collide again with another packet. Other systems use *buffering:* packets failing to get selected for the desired link are buffered inside the switch and go out at a later time. When very large scale integration (VLSI) technology is used, buffering is more efficient than deflection routing, because VLSI memory is less expensive than off-chip communication (pins and chip-to-chip wires) − in effect, deflection routing uses the latency of network links as an (expensive!) form of buffer memory. This paper deals with VLSI switches that use buffering to resolve contention.

Switch buffers can be organized in various ways − input, output, cross-point, shared − and can be managed as FIFO queues or as non-FIFO buffers. Performance-wise, *shared* (centralized) buffering is the best architecture. These points are reviewed in section 2. Implementation-wise, even though several designers prefer input buffers because they consider shared buffering expensive, the latter occupies in fact less silicon area for comparable performance, as concluded in section 5. This paper presents, in section 3, a new shared buffer organization, which we call *pipelined memory,* that reduces appreciably the cost and complexity of shared buffering. Relative to other shared buffer organizations, the pipelined memory reduces significantly the I/O circuitry needed around the buffer, and eliminates the need for a cut-through crossbar. The pipelined memory organization was used in three versions of the *Telegraphos* switch, using discrete, semi-custom ASIC, and full-custom VLSI technology, respectively; section 4 reports on these implementations.

M. Katevenis and A. Efthymiou are also with the University of Crete, Department of Computer Science.

2. Switch Buffer Architectures

This section reviews the main organizations for the buffer memory in a switch, their performance, and their implementation tradeoffs. We deal with switches that have a low enough number of links so that they fit in a single chip and so that their switching fabric is a crossbar. Such switches can be used by themselves, or they can be the building blocks for larger, multi-stage switches and networks; our discussion applies equally well to both uses.

2.1 Architectures using Low-Throughput Buffers

Figure 1 shows the switch architectures where each buffer memory needs a throughput proportional to the link bandwidth, but not to the number of links. We use 3 by 3 switches for illustration. *Input queueing* is the simplest but worst performing architecture. At each occasion, only the front (head-of-line) packet of each input queue is considered for possible routing to its destined output; if it collides with the front packet from another queue, then it has to wait, and so do all the packets behind it in its queue, even if the latter are destined to currently idle outputs. This is known as *head-of-line blocking.* Because of it, a switch with equal input and output throughput, with fixed (small) packet size, and with independent, randomly destined packet traffic, saturates at about 60% of the link capacity [KaHM87]. When the traffic is bursty and the bursts are larger than the buffers − for example with multi-flit packets in wormhole routing − saturation occurs sooner: in [Dally90 (fig. 8, 1 lane)], with 20-flit messages and 16-flit buffers, simulation showed saturation at about 25% of link capacity.

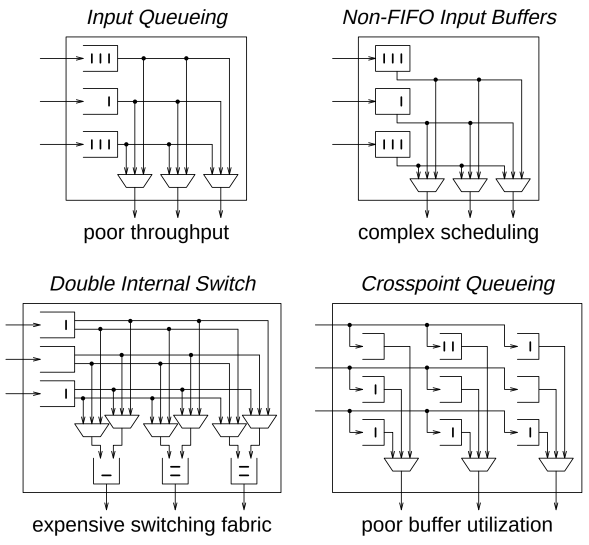


Figure 1: Architectures needing low-throughput buffers

To improve the performance of input queueing, one may drop the requirement that the buffers behave like FIFO queues, thus eliminating head-of-line blocking. In this *non-FIFO input buffering* architecture, the buffers still have a single read port. This organization provides quite better performance than input queueing [AOST93] [TaCh93] [LaSe95], although output throughput may be less than optimal with practical schedulers, and latency is worse than optimal whenever some output remains idle because all buffers that contain packets for it are busy with forwarding other packets to other outputs. Buffer implementation is more complicated than with input queueing [TaFr88]. Also, a more complicated scheduler is needed, because now the scheduling of each output depends on the scheduling of the other outputs: only one output port is allowed to use each buffer at any given time. Schedulers for this architecture were studied in [AOST93] [TaCh93] [LaSe95].

Another method to improve the performance of input queueing is to provide an internal *switching fabric of higher throughput* than that of the incoming links [PaBr93]; figure 1 shows an example with a double internal switch. This is equivalent to input queueing operating at a reduced input load. Output queues are also needed here. The cost of this architecture is in the additional queues, in the requirement for the queues to be three-ported (rather than two-ported as above), and in the increased throughput of the switching fabric.

To reach maximum throughput, one can go to *crosspoint queueing:* there is a single queue for each input-output link pair. Every outgoing link can now be kept busy, by selecting and forwarding a packet to it, independent of what the other links do. This architecture achieves optimal link utilization, but has the disadvantage of needing many buffers with a total memory capacity considerably higher than the architectures of the next sub-section, for a given level of performance.

2.2 Architectures with High-Throughput Buffers

Figure 2 shows the architectures where buffer throughput is proportional to the number of links. In *output queueing,* each buffer is associated with one outgoing link, and must be able to accept, in the worst case, packets arriving simultaneously from all inputs. Each output queue plays the role of the union of the queues in one column in the crosspoint architecture. Merging multiple queues into one improves buffer memory utilization, while optimal link utilization is maintained.

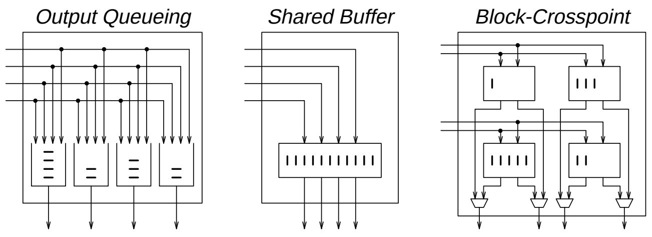


Figure 2: Architectures needing high-throughput buffers

Shared (centralized) buffering uses only one buffer memory for the entire switch; this must have a throughput at least as much as the aggregate throughput of all incoming and all outgoing links. This shared buffer plays the role of the union of all output queues in the previous organization. The shared buffer architecture yields the same optimal link utilization as crosspoint and output queueing, and, in addition, it achieves the best buffer memory utilization. For example, according to [HlKa88], a 16×16 switch with incoming link load of 0.8 (uniformly distributed destinations), needs the following buffer sizes in order to achieve packet loss probability of 0.001: *(i)* 86 packets under shared buffering (5.4 per output); *(ii)* 178 packets under output queueing (11.1 per output); and *(iii)* 1300 packets under ''input smoothing'' (80 per input) − input smoothing has considerable similarities to non-FIFO input buffering. Concerning latency, the simulations in [AOST93, fig. 3] showed output queueing (or equivalently shared buffering) to be about twice faster than input buffering, under the particular scheduling algorithm that that paper uses, for link loads between 0.6 and 0.9.

A mixture of crosspoint and shared buffering, *block-crosspoint buffering,* is also possible. It consists of a number of shared buffers, each dedicated to a certain subset of incoming and outgoing links. It features lower throughput-per-buffer requirements than a single shared buffer, and better buffer space utilization than crosspoint queueing.

2.3 Discussion

While the shared buffer architecture is performance-wise the best, several designers consider it expensive because of the high throughput requirements on the buffer − see e.g. [TaFr88, p. 344] and the motivation for the development of advanced schedulers for input buffering [AOST93] [LaSe95]. Because of this impression, a number of real switches use input queueing, e.g. [Arno89], [HoLa93], [C104], [Souz94].

Reality is different, however. An $n \times n$ switch, with link throughput 1, needs n input buffers of throughput 2 each, or it needs one shared buffer of throughput $2n$. The aggregate throughput of the n input buffers is $2n$; by suitably arranging these n memories, one buffer of throughput $2n$ can be constructed, as needed for shared buffering. Section 3.1 reviews the existing techniques for implementing such high-throughput buffers, and switches that use them. The *pipelined memory* − this paper's contribution − is a simpler and more compact implementation of such buffers; it is presented in sections 3.2 through 4.4. Section 5 compares the pipelined shared buffer to other shared and input buffering designs, and demonstrates the superiority of the former when cost-performance is considered.

When packet sizes are not integer multiples of the memory width (see section 3.5), shared buffer designs encounter difficulties. These difficulties may be one reason why designers turn to input queueing/buffering. We believe that this is either unjustified, because the above memory width quantum can often be made quite small, or that this will not be an issue with the evolution of modern networks. In fact, we believe that high-speed networks will converge to using fixed-size packets, cells, or flits, just like processors converged to using fixed-size instructions (RISC versus CISC architectures). ATM, with 53-byte fixed-size cells, is a big step in that direction.

3. Multi-Port Buffers: The Pipelined Memory

This section reviews existing techniques for implementing multi-port buffers and switches that use them, and then presents our new organization for them − the pipelined memory.

3.1 Existing Techniques for Multi-Port Memories

True multi-port memory is very expensive, because each storage bit must have multiple word lines and bit-lines, thus raising significantly the memory cost per bit. As a result, for almost every application other than the smallest memories (e.g. processor register files), multiple *interleaved banks* of single-ported memory are used in place of multi-port memory. The buffers in the ''Knockout Switch'' [YeHA87] use this technique, in an output queueing architecture. The ''Prelude'' switch [DeCS88] shared buffer memory operates as an interleaved memory when cells are extracted from it (in that case, neighboring bank addresses differ by 1 in succeeding clock cycles). The shared buffers of [Turn93] and of the PRIZMA switch architecture [DeEI95] also consist of interleaved banks, where each packet is stored entirely within one bank rather than being spread among multiple banks as is customary in other interleaved memories; see section 5.3 for further discussion.

When an interleaved memory is used as buffer for multi-word packets, most of the accesses to it are sequential. This eliminates most or all of the bank conflicts, and also allows one to use a single address register for all memory banks, effectively merging the multiple memory banks into a single *wide memory.* This works best when the size of the items accessed − the packet size in switches − is a multiple of the wide memory word size, so that all accesses are aligned on wide-word boundaries. A wide memory is particularly feasible and attractive inside a VLSI chip, because storage arrays inside chips are naturally wide: widths of 128 to 2048 bits are common. The IBM ''Vulcan'' switch [Stun94] uses a wide memory shared buffer. The ''Prelude'' switch [DeCS88] buffer memory operates as a wide memory when cells are stored into it. The VLSI ATM switches [Koza91] and [Shob91] also use wide memory shared buffers.

We have used the wide memory organization in an earlier switch chip design [KaSC91], in a shared buffer architecture. Figure 3 shows that buffer, with its associated input and output latches, drivers, and cut-through circuitry, for a simple case of a 2×2 switch. Packets travel on the links as sequences of w-bit words, at the rate of one word per clock cycle. The memory cycle time is assumed equal to the link cycle time. In the simple example of figure 3, the packet size is 4 words, or a multiple thereof. In the wide-memory organization each operation has to be performed on an entire packet, and thus it can only be performed after the entire packet has been assembled. Double buffering is needed on the input side, because it is *not* possible to guarantee that the wide memory will be available for storing the packet into it at precisely the desired time, since packet arrivals are not synchronized. Double buffering was also used in the outgoing link circuits in [KaSC91] as a feature rather than requirement.

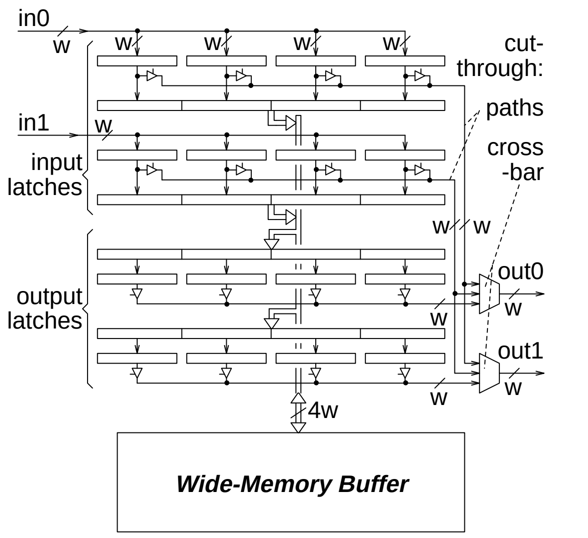


Figure 3: Input, output, and cut-through overhead in a 2×2 shared-buffer wide-memory switch

Cut-through is the capability of a switch to start forwarding a packet along an outgoing link before the entire packet has arrived. Cut-through is essential for modern low-latency switches; even if the store-and-forward delay of an ATM cell at Gbps rates (a few hundred ns) is negligible for telecom applications, it is *not* so for distributed and parallel computer applications, especially when the rest of the communication is optimized to eliminate software overheads (e.g. as in Telegraphos [Kate94]). Since a packet cannot be stored into the wide memory before all of it has arrived, and since cut-through must start before that time, it follows that cut-through must be performed through paths *other than* the wide memory or its data bus. Additional tristate drivers, bus wires, and an output crossbar are needed (figure 3). Notice that the paths provided in this design do not suffice in order to be able to initiate cut-through between the time a packet is transferred into the second row of input latches and the time when it is transferred onto the wide memory bus.

3.2 The Pipelined Memory Organization

The wide memory is a simplification of the interleaved memory, applicable when most accesses are sequential. The *pipelined memory* organization proposed in this paper is a different and better simplification, applicable in the same cases. The pipelined memory is like an interleaved memory where the address used by a bank is the same as that used by the previous bank in the previous cycle. Using the same address for subsequent banks in subsequent cycles *(i)* simplifies control relative to the interleaved case, *(ii)* makes the VLSI implementation faster than wide memory (see section 4.3), and *(iii)* significantly reduces the size of the peripheral circuitry relative to the wide memory.

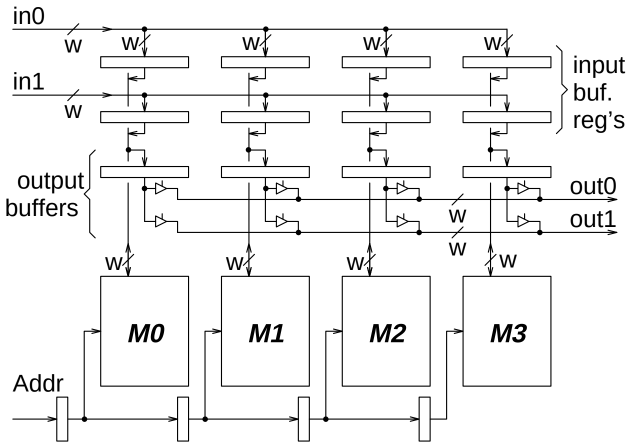


Figure 4: The 2×2 shared-buffer switch of figure 3, using pipelined memory

Figure 4 shows the same 2×2 shared-buffer switch example as figure 3, this time using pipelined memory. Again, the memory cycle time is equal to the link cycle time, the link throughput is one word (*w* bits) per cycle, the number of pipeline stages is equal to the number of incoming plus outgoing links (four), and the packet size is equal to or a multiple of that same number. As a packet arrives, it is written into the corresponding input buffer registers, one word at a time. Imagine this as a *wave* of new packet words entering into the input buffer registers (overwriting the old data in them). During *some* cycle after the arrival of the first word, *W0*, of this packet, and before the arrival of the fifth word (or first word of the next packet), *W0* is written into the first memory stage, *M0*. In the three subsequent cycles, the next three words of the packet will be written into the next three memory stages, *M1*, *M2*, and *M3*, at the rate of one word per cycle. Think of this as another wave of words being written into the buffer memory from left to right. All words of the same packet are written into the same address in each of the memory stages. The tail of the packet may be written into the last memory stage after the head of a new packet has entered into the first input buffer register. This does *not* lead to any loss of information, since the wave of storing the old packet into the buffer memory was initiated before the new packet wave started overwriting the input registers, and since both waves proceed at the same rate from left to right. Thus, *no double buffering* (as with wide memory) is needed here. The precise cycle when *W0* is written into *M0* depends on the resolution of contention for access to the buffer memory by the incoming and the outgoing links. Normally, higher priority is given to the outgoing links, because any delay to supply data to an outgoing link leads to idle time on that link, while delays to store incoming packets into the buffer memory have no direct consequence on the switch performance.

On the output side, again, pipelined memory needs one level of buffering less than wide memory. Figure 4 uses only one row of output buffer registers shared among all outgoing links, with the restriction that no two outgoing links can start sending out packets in the same cycle. Output operation is as follows. The first word of the outgoing packet is read out of *M0* and placed in the leftmost output buffer register. In the next cycle, this register drives the desired outgoing link. The wave of reading from the memory stages and transmitting successive words proceeds left to right, at the rate of one word per cycle.

3.3 Pipelined Control and Automatic Cut-Through

Control of the pipelined memory and associated I/O circuits is simple and straightforward. Every cycle, a new operation wave can start from *M0*: we can start reading another outgoing packet and send it on another outgoing link, or we can start writing a new incoming packet into the buffer memory. Each pipeline stage performs exactly the same operation as the previous stage in the previous cycle, and thus we only need to generate the control signals for the *first* memory stage; the control signals for subsequent stages are delayed versions of the former. Figure 5 shows the control signals for the switch of figure 4; control for stages *M1*, *M2*, and *M3* is identical to stage *M0* − delayed by the appropriate number of clock cycles. The load enable signals of the input latches are activated when new packets enter through the corresponding links. In parallel, in each cycle, arbitration circuitry decides whether to initiate a new read wave or a new write wave; as mentioned above, normally, priority is given to read's (output links). For each read or write initiation, an outgoing or incoming link is specified (*linkID*), and a buffer address is given. The circuits that provide these − in particular the buffer (address) management circuits − are independent of the pipelined memory, and their choice is orthogonal to the shared buffer organization examined in this paper; see [Kate94] or [KVES95] for the way Telegraphos I does that. Once an operation wave is initiated in *M0*, the same operation will automatically follow in the subsequent stages.

In the pipelined memory organization, *cut-through* is *automatic* − no additional datapath or control is required. This represents a major savings relative to wide memory, which needs an extra set of buses on the input side, an extra crossbar, and special control in order to implement cut-through. Any cycle after a packet's first word arrives and is written into the leftmost input buffer register, this register can drive the data bus for storing the word into *M0*, as normal. In the same or in any subsequent cycle, this word can also be loaded (or read out of *M0*) into the leftmost output buffer register, and in the very next cycle it can be sent out on an outgoing link. All this may well happen before the

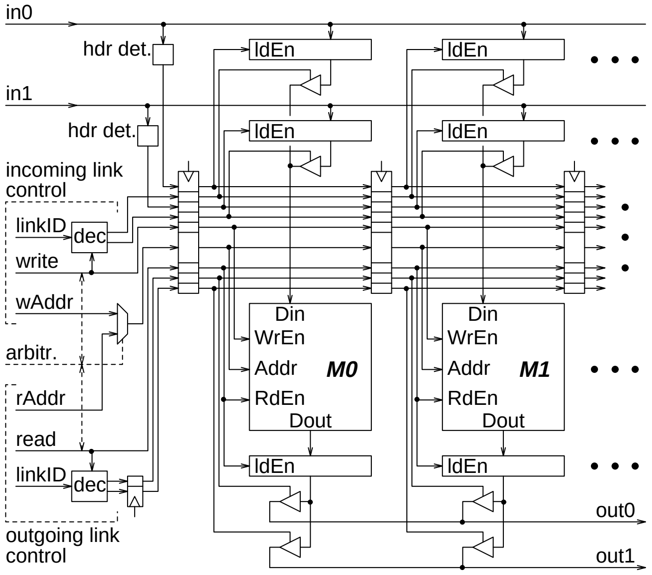


Figure 5: Control signals for the datapath of figure 4

tail of the packet has arrived into the switch, and thus it constitutes cut-through. Since the wave of packet arrival started before the wave of packet departure, and since they both proceed left to right at the same rate, transmission of the packet's tail will only be attempted after that tail has arrived into the switch and has been written into the rightmost input buffer register.

3.4 Latency due to Staggered Initiation

The pipelined memory shared buffer of figure 4 places the restriction that no two packet transmissions or cut-through operations can be initiated in the same clock cycle. This increases slightly the packet latency, but, as we will show here, this increase is *negligible.* Under heavy traffic load, the effect under discussion is not important, because every packet has to wait for other packets to go out in front of it, anyway. Since the previous packets had staggered departure times, the same will be true for the new packets, and simultaneous initiations will not be attempted. Under light load, the above effect will increase the cut-through latency of all but one of the packets that happen to enter into the switch in the same clock cycle. Bellow we show that, probabilistically, the expected value of this delay is approximately 0.25 clock cycles times the link load. For 40% load, this amounts to one tenth of a clock cycle, i.e. negligible.

The expected cut-through latency increase due to multiple packet headers arriving at the same time is $(p/4) \cdot (n-1)/n$, where p is the link load and n is the fan-in of the switch. The precise analysis is as follows. Since the packet size is $2n$, the probability of a packet head appearing on a given link in a given cycle is $p/2n$. Consider a particular "tagged" packet, whose head appears on incoming link l in cycle c. The tagged packet will suffer a cut-through latency increase (in clock cycles) equal to one half the number of packet heads appearing on the other $n-1$

input links, in cycle c (one half because for each pair of conflicting packets one wins and one loses). The $n-1$ other links carry independent traffic, with head probability in cycle c equal to $p/2n$ each. Thus the expected number of heads above is $(n-1)\cdot(p/2n)$, and hence the expected cut-through latency increase for the tagged packet is $(1/2)\cdot(n-1)\cdot(p/2n)$.

3.5 Packet Size Quantum

The pipelined memory operation requires that the size of each packet (cell) be an integer multiple of a basic quantum. This basic quantum is the total width of the shared buffer, or half of that width if the technique described below is used. How large is this basic quantum? Does it pose serious problems? The answer is *no*. To see why, consider a quantum as small as 32 to 64 bytes (ATM cells are 53-byte wide). This corresponds to buffer widths of 256 to 1024 bits. With an (on-chip) memory cycle time of 5 ns, which is reasonable in CMOS today, the aggregate throughput of such a buffer is 50 to 200 Gbits/s (12 to 25 GBytes/s) − enough for 16 incoming and 16 outgoing links near the Giga-*Byte* per second range, each. Since the above quantum is proportional to both link throughput and number of links, some designers consider this as a non-scalable architecture. However, as the above numbers show, chip I/O throughput rather than memory cycle time is the bottleneck; with time, the former increases and the later decreases, so their relationship is likely to persist. Alternatively, if more links or more throughput is desired, one can always go to block-crosspoint buffering, still using pipelined memory to construct each of the buffers.

Although in the straightforward pipelined memory organization the packet size must be an integer multiple of the total width of all memory stages, packets of half that size can also be handled. This is achieved as follows. Consider a shared-buffer $n \times n$ switch with $2n$ pipelined memory stages. Consider the operation when the packets are of size n words each. The peak throughput of each link is one packet every n cycles. Since there are n incoming and n outgoing links, the shared buffer must be able to initiate one write operation of one incoming packet *and* one read operation of one outgoing packet in each and every cycle. The shared buffer will consist of *two* pipelined memories, with n stages each. Each packet is stored into one or the other of these two memories. In each and every cycle, one read operation of one outgoing packet is initiated from one of the two memories − whichever the desired packet happens to be in. In the same cycle, one write operation of one incoming packet must also be initiated; this will be initiated into the *other* one of the two memories.

4. Example: the Telegraphos Switch Buffers

The work presented in this paper is being carried out in the context of the *Telegraphos* project [Kate94], in which we build switches and processor-network interfaces that enable parallel processing on workstations clustered through gigabit LAN's. We are working on three prototypes of our Telegraphos system, using different technologies for each − FPGA, semi-custom ASIC, full-custom VLSI. The next sub-sections describe the pipelined memory shared buffer that forms the core of the switches in these prototypes, with particular emphasis on the optimizations in VLSI switch buffering that the pipelined memory makes possible.

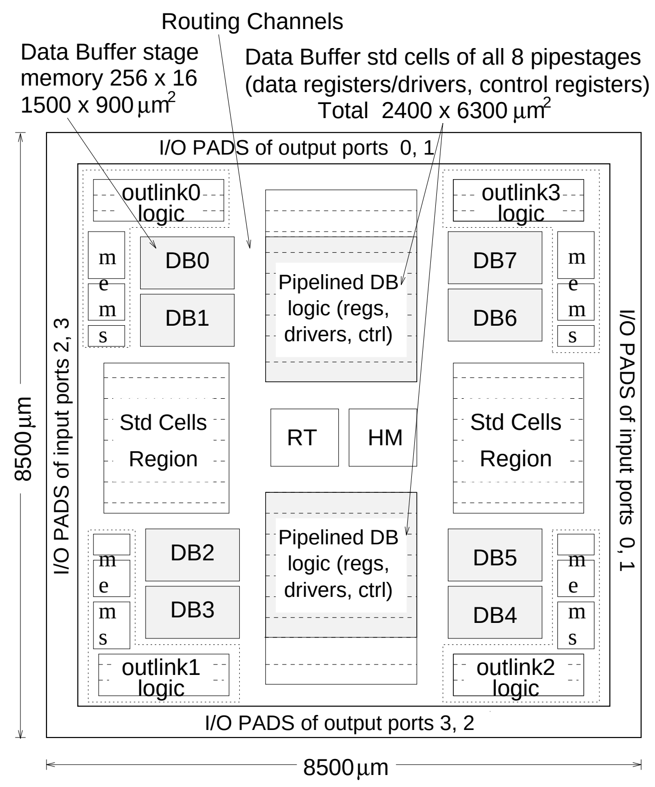
4.1 Telegraphos I: FPGA-based Prototype

The Telegraphos I switch is a 4×4 crossbar implemented with field-programmable gate arrays (FPGA) and SRAM chips. It has been built and successfully tested in our lab. It operates correctly at 13.3 MHz; each link carries 8 bits per clock cycle, i.e. 107 Mbps/link. The packet size is 8 bytes, and the shared packet buffer consists of 8 pipelined stages; it is built out of 8 SRAM chips. The logic for access arbitration among the incoming and the outgoing links, as well as for control signal generation for the first pipeline stage, is contained in one Xilinx 3130PC84 FPGA, and it is approximately equivalent to 500 gates. The peripheral circuitry of the shared buffer (input/output registers/drivers, control signal pipeline registers) can be considered as an 8-bit wide datapath; this datapath was sliced into four 2-bit-wide slices, and was implemented in four Xilinx 3164PC84 FPGA's, each of them containing the equivalent of 1500 gates. The PCB wiring is quite dense in the area of the shared buffer, requiring 4 signal layers with 0.2mm wide traces to route all the interconnections.

4.2 Telegraphos II: Standard-Cell ASIC

The Telegraphos II switch is a single-chip version of the Telegraphos I switch, operating at higher speed. This ASIC uses the 0.7 μm CMOS standard-cell process of European Silicon Structures Inc. (ES2). At the time of this writing, the placed and routed ASIC is undergoing final simulation in order to be submitted for fabrication. Figure 6 shows its floorplan; the chip size is 8.5×8.5 mm^2. Like Telegraphos I, this is also a 4×4 crossbar, but each link operates at 400 Mbps − 16 bits / 40 ns on-chip, 8 bits / 20 ns off-chip − as verified by loaded worst-case simulation. The packet size is 16 bytes. The shared packet buffer has eight pipeline stages; the shaded areas of figure 6 identify the pipelined memory and its peripheral circuits. Each memory stage, DB0 to DB7, is a 256×16 compiled SRAM of size $1.5 \times 0.9\ mm^2$. All eight SRAM megacells occupy 11 mm^2. The peripheral circuits of the shared data buffer including input and output data registers, tristate drivers, and control registers, are implemented with standard cells placed in two regions in the middle of the chip, occupying 15 mm^2. To this, we must add the routing area used by the memory buses: 5.5 mm^2. Thus, the total shared buffer area amounts to 32 mm^2.

The other blocks on the floorplan in figure 6 are: the outgoing link logic and memories, including the credit-based flow control, the list of ready to depart packets, and the output registers and drivers, are in the L-shaped areas marked by dotted line at the four corners of the chip. At



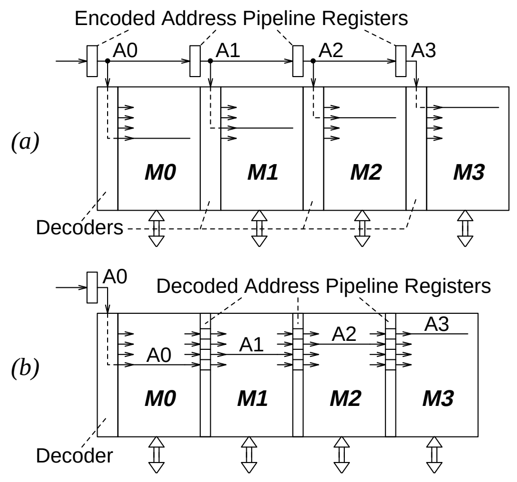
Routing Channels
Data Buffer stage memory 256 x 16 1500 x 900 µm²
Data Buffer std cells of all 8 pipestages (data registers/drivers, control registers) Total 2400 x 6300 µm²
I/O PADS of output ports 0, 1
outlink0 logic
outlink3 logic
mems
DB0
DB1
DB7
DB6
Pipelined DB logic (regs, drivers, ctrl)
Std Cells Region
RT
HM
DB2
DB3
DB5
DB4
outlink1 logic
outlink2 logic
I/O PADS of input ports 2, 3
I/O PADS of input ports 0, 1
I/O PADS of output ports 3, 2
8500µm
8500µm

Figure 6: Floorplan of the Telegraphos II switch chip

the center of the chip, the RT block is the translation routing memory, and the HM is the untranslated packet header memory. The two standard cell regions at the left and right sides of the chip contain the input port logic, arbitration for the translation memory, and the control and management unit of the switch.

4.3. Pipelined Memory Optimizations in VLSI

Figure 7(a) shows how traditional RAM arrays can be arranged in VLSI, when implementing the pipelined memory of figure 4. Relative to the wide memory (figure 3), the pipelined memory has more address decoders, but shorter word lines. This is an advantage, since it reduces the "RC" delay [WeEs93] of activating the addressed word. Actually, in current and future VLSI technologies, the RC delay of wide memory word lines is so large that they are split into multiple narrower memory blocks anyway, each with its own address decoder, thus arriving at a floorplan and area similar to figure 7(a). In many cases, a further optimization of the pipelined RAM implementation is possible, as shown in figure 7(b), which is *not* possible for the wide RAM. Since the arrays touch each other, the decoded address can be transferred from one stage to the next *directly,* without being re-decoded. Effectively, the word lines of all stages are connected through pipeline flip-flops into long word lines, which are activated in a wave-like fashion. Oftentimes, these flip-flops are smaller and/or faster than the decoder that they replace.



Encoded Address Pipeline Registers
A0
A1
A2
A3
(a)
M0
M1
M2
M3
Decoders
A0
Decoded Address Pipeline Registers
(b)
A0
A1
A2
A3
M0
M1
M2
M3
Decoder

Figure 7: Pipelined RAM in VLSI:
(a) traditional, (b) novel organization and floor-plan

Three further optimizations are possible in future very-high-speed IC technologies, where RC delays of long wires will be comparable to clock cycle time. First, the long lines carrying the input and output link data (see figure 4) can be split in two or more pipeline stages each. At each splitting point, an additional pipeline stage is also inserted into the word lines, in the pipelined memory. The net effect is that *all* packet data are delayed by an equal number of cycles on their way from an input to an output link, and thus the logic of the switch operation remains unaffected. Second, the outgoing link circuitry can be placed at the bottom of the pipelined memory, as in figures 5 and 8 − rather than the top, as in figure 4. In this way, all information flows left-to-right and top-to-bottom. By distributing the clock signal in the same direction, the *relative skew* between clock and data is minimized, which will be crucial in the high-speed technologies of the future. Lastly, this uniform direction of flow of all data can be exploited in one more way: bit lines can be split into multiple pipeline stages, thus speeding up memory cycle time; the address decoder also has to be split in the same manner.

4.4 Telegraphos III: Full-Custom Pipelined Buffer

To show the advantages of full-custom over standard-cells for a regular VLSI structure such as the pipelined memory buffer with its datapath, we are also implementing a pipelined memory buffer in 1.0 µm full-custom CMOS by ES2. At the time of this writing, this design is being submitted for fabrication inside a test chip. By going from standard-cells to full-custom, the datapath of the shared buffer gains approximately a *factor of 22* in speed, capacity, and area: full-custom has twice the number of links, the clock is 2.5 times faster, and the peripheral circuit area is 4.5 times smaller. Figure 8 shows the floorplan of this Telegraphos III pipelined memory shared buffer. A decoded address pipeline register is 2.3 times smaller than the normal address decoder for the SRAM. The largest gains relative to the standard-cell version come from the use of dynamic latches for the incoming links, the use of precharging for all long wires that have multiple drivers (e.g. outgoing link buses), and the overlap of the peripheral circuitry with the link wire routing. The peripheral circuitry area is just about 9 mm^2, compared to 41 mm^2 that the standard-cell design would occupy in this 1.0 μm technology for the half-sized (4×4) Telegraphos II switch. Given that the peripheral circuit area grows with the square of the number of links, an 8×8 standard-cell design would be about *18 times* larger than this same configuration in full-custom.

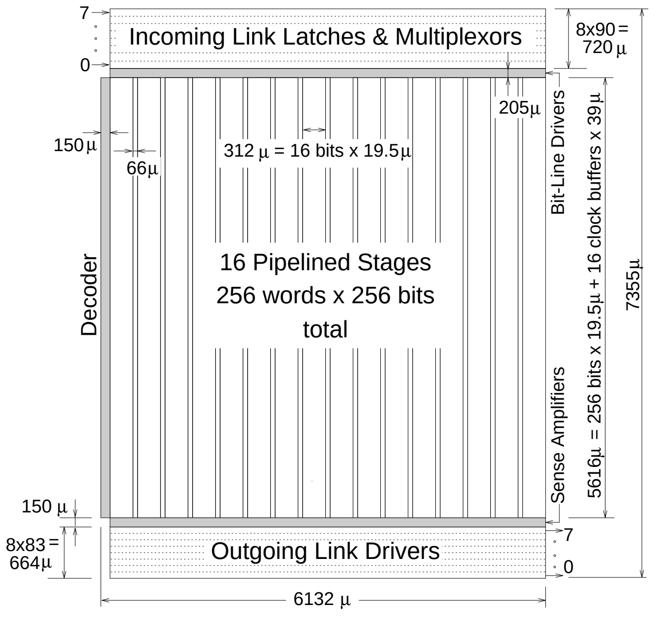


Figure 8: Floorplan of 16 Gbps, 64 Kbit pipelined buffer in 1 μm full-custom CMOS

The CMOS technology used has 1.0 μm minimum drawn feature size, 5 Volts VDD, one polysilicon layer, and two metal layers. We have simulated extensively (using HSPICE) the final layout of the buffer and its peripheral datapath. The worst case clock cycle time is 16 ns (worst case is 4.5 V supply, 125 °C junction temperature, slow transistors, high parasitics); typical clock cycle time is 10 ns. The buffer consists of 16 pipelined stages, and it provides storage for up to 256 packets of 256 bits each. There are 8 incoming and 8 outgoing links, with worst-case throughput of 1 Gbps/link (1.6 Gbps/link typical). On-chip, each link consists of 16 wires, carrying 16 bits every clock cycle. In the area of the input and output link datapath, all active circuits are laid out *under* the horizontal link wires. Thus, the area of this block approaches the minimum possible area of a crossbar, since every crossbar has to have at least the data wires. This area also includes the control-signal pipeline registers (figure 5).

5. Comparisons

In this section we compare the silicon area cost of the pipelined memory shared buffer switch architecture, implemented in full-custom CMOS VLSI technology, against input buffering, wide memory shared buffering, and the PRIZMA interleaved shared buffering. We assume an $n \times n$ switch, with clock period T, memory cycle time T, and link throughput w bits/T.

5.1 Shared versus Input Buffering

Figure 9 presents a floor-plan comparison of input and shared buffering. Under input buffering, each buffer memory needs a write throughput of w bits/T, plus a read throughput of w bits/T. Consequently, it must either be dual-ported with a width of w bits, or single-ported with $2w$ width; in either case, its horizontal size is approximately $2w$, in units of single-ported bit cells. Figure 9 assumes that the n input buffers are organized as tall and skinny memory arrays that are placed next to each other, so that their outputs come out of their bottom. Under this arrangement, the total buffer memory width is $2nw$. The shared buffer has *the same width,* since its throughput must equal the aggregate throughput of all switch links (which is equal to the aggregate throughput of all input buffers). The total input buffer width includes n address decoders, while the shared buffer width includes one address decoder and $2n-1$ pipeline registers; to a first order of approximation, these are about the same. Let H_i be the height of the input buffers, and H_s the height of the shared buffer. If H_i and H_s are measured in units of bit cells, and if the input buffers are single-ported (the more economical option), then the input buffer size is $2wH_i$ bits per input or $2nwH_i$ total, while the shared buffer size is $2nwH_s$ bits. To achieve a similar level of switch performance, fewer total buffer bits suffice under shared buffering (section 2.2), so we can let H_s be (significantly) smaller than H_i. This represents a net advantage of the shared buffer.

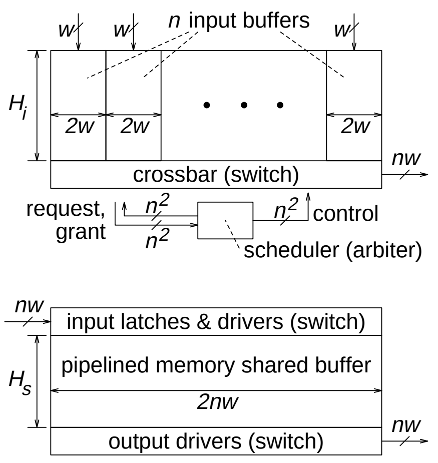


Figure 9: Input versus shared buffering VLSI cost

Under input buffering, a w-bit-wide $n \times n$ crossbar fabric is needed. Practically, we believe that it is best to *pitch-match* this crossbar to the input buffers; then, its size is roughly $2nw \times nw$, where the length $2nw$ is in units of storage cells, and the height nw is in units of wires (n w-bit crossbar outputs). For the shared buffer case, as we saw in section 4.4, the area occupied by the input and output datapath is dominated by the incoming/outgoing link wires. Thus, each one of these two blocks is again roughly $2nw \times nw$, measured in the same units as above. The input buffering scheme needs only one crossbar, while the shared buffer needs two such blocks. On the other hand, non-FIFO input buffering needs a quite complex scheduler (arbiter), with many control wires connecting it to the other blocks. Additionally, the management of the n queues in *each* input buffer is quite more costly than the management of n (total) queues in the single shared buffer.

In conclusion, to a first-order approximation, the single crossbar and the scheduler of the input buffers occupy comparable area with the two crossbars of the shared buffer, while $H_s < H_i$ for similar performance. Thus *shared buffering has better cost-performance* ratio than (non-FIFO) input buffering.

5.2 Pipelined versus Wide Memory Shared Buffer

As discussed in section 3.2, one of the advantages of the pipelined relative to the wide memory organization is the reduction in peripheral circuitry area. To estimate this, we calculated what that area would be in [KaSC91] (our previous wide-memory design), if its parameters (technology, number of links, link width) were modified to match those of Telegraphos III (section 4.4). This adjusted wide-memory peripheral circuitry area would be 13 mm^2, versus 9 mm^2 of Telegraphos III, i.e. pipelined memory has about 30 % smaller peripheral area than wide memory.

5.3 Pipelined versus Interleaved Shared Buffer

Pipelined memory, as a special and simplified case of interleaving, costs less than the latter organization. Here, we will make a comparison to the particular form of interleaving used in [Turn93] and in the PRIZMA architecture [DeEI95]. There, each packet (cell) is stored in one bank of the interleaved shared buffer, and each bank can only contain one packet. According to [DeEI95], this choice was made in order to ensure scalability of the shared buffer throughput with increasing link throughput and link counts for constant cell size − which is not true for the wide or the pipelined organization. As discussed in section 3.5, while, strictly speaking, the above argument is true, on the other hand, practical considerations (chip I/O throughput, in particular) make it unlikely that one will ever need that much shared buffer throughput; even if one does, block-crosspoint buffering is probably be a better alternative.

The silicon area cost of the PRIZMA architecture is much higher than the pipelined memory cost. First consider the storage area for the packet bodies. Because the buffer consists of many, small, independent memories − one per packet − it suffers from a considerable overhead for address decoders. Implementing the banks as shift-registers would not solve this problem, because one (dynamic) shift-register bit is 4 times larger than one (3-transistor dynamic) RAM bit. Shift-registers would also preclude cut-through.

Second, consider the ‘‘router’’ and ‘‘selector’’ circuits of PRIZMA. Each of them is a (w-bit wide) $n \times M$ crossbar, capable of connecting each of n links to any of M memory banks. The function of the PRIZMA router is comparable to our incoming link latches and multiplexors (figure 8), and the selector is comparable to our outgoing link drivers. However, the PRIZMA crossbars have a complexity proportional to $n \times M$ each, while our crossbars have a complexity proportional to $n \times 2n$ each, where M is the number of PRIZMA memory banks i.e. the shared buffer capacity measured in packets (cells). Since usually the packet capacity of the buffer is much larger than the total number of links, the PRIZMA circuits cost much more than the pipelined memory circuits. For example, in Telegraphos III, $2n = 16$, while $M = 256$; thus, the shared-buffer crossbars would cost *16 times more* in the PRIZMA architecture relative to the Telegraphos III architecture. The PRIZMA crossbar cost could be reduced by placing more than one packets per bank, but that would complicate control and scheduling and may hurt performance.

Conclusion

In VLSI switches, input buffering (FIFO or non-FIFO) and shared buffering cost about the same per bit of storage, to a first-order approximation. However, by its nature, shared buffering performs better than input buffering for a given total buffer capacity. Thus, contrary to an existing impression, shared buffering should be the architecture of choice. We described a new ‘‘pipelined memory’’ organization, which is particularly suitable for VLSI shared buffers, and we showed a few prototype implementations of it. Relative to interleaved or wide memories, the pipelined memory simplifies control, reduces input and output datapath, and eliminates cut-through circuitry.

Acknowledgements

Telegraphos was developed in the ESPRIT/HPCN project ‘‘SHIPS’’, and will be used by the ACTS project ‘‘ASICCOM’’, funded by the European Union (DG III and DG XIII). We deeply appreciate this financial support, without which Telegraphos would have not existed.

Telegraphos is a collective effort, with many contributors. The authors wish to acknowledge in particular E. Markatos, S. Kapidakis, P. Constanta-Fanouraki, and Ch. Nikolaou for valuable contributions to the Telegraphos design; G. Kalokerinos, M. Stratakis, Ch. Xanthaki, A. Dollas, and G. Papadourakis for implementing the rest of the Telegraphos I system prototype; G. Kornaros and N. Houstis for porting the Telegraphos I switch design to the Telegraphos II ASIC technology; and Ch. Kozirakis and M. Grammatikakis for contributing to the switch simulation. Also, the comments of D. Serpanos, M. Grammatikakis, and the anonymous reviewers were valuable. We thank all of them.

References

[AOST93] T. Anderson, S. Owicki, J. Saxe, C. Thacker: ‘‘High-Speed Switch Scheduling for Local-Area Networks’’, *ACM Trans. on Computer Systems,* vol. 11, no. 4, November 1993, pp. 319-352.

[Arno89] E. Arnould, F. Bitz, E. Cooper, H.T. Kung, R. Sansom, P. Steenkiste: ‘‘The Design of Nectar: A Network Backplane for Heterogeneous Multicomputers’’, *Proceedings of ASPLOS-III Conference,* ACM SIGARCH vol. 17, no. 2, April 1989, pp. 205-216.

[AtSe88] W. Athas, C. Seitz: ‘‘Multicomputers: Message-Passing Concurrent Computers’’, *IEEE Computer Magazine,* vol. 21, no. 8, August 1988, pp. 9-24.

[C104] ‘‘IMS C104 Packet Routing Switch − Preliminary Datasheet’’, *INMOS Ltd,* 1000 Aztec West, Almondsbury, Bristol, BS12 4SQ. UK, June 1993.

[Dally90] W. Dally: ‘‘Virtual-Channel Flow Control’’, *Proc. of the 17th Int. Symp. on Computer Architecture,* May 1990, pp. 60-68.

[DeCS88] M. Devault, J-Y. Cochennec, M. Servel: ‘‘The "Prelude" ATD Experiment: Assessments and Future Prospects’’, *IEEE Journal on Sel. Areas in Communications,* vol. 6, no. 9, December 1988, pp. 1528-1537.

[DeEI95] W. Denzel, A. Engbersen, I. Iliadis: ‘‘A Flexible Shared-Buffer Switch for ATM at Gb/s Rates’’, *Computer Networks & ISDN Systems* 27 (1995), pp. 611-624, Elsevier Science B.V.

[HlKa88] M. Hluchyj, M. Karol: ‘‘Queueing in High-Performance Packet Switching’’, *IEEE Journal on Sel. Areas in Communications,* vol. 6, no. 9, December 1988, pp. 1587-1597.

[HoLa93] M. Homewood, M. McLaren: ‘‘Meiko CS-2 Interconnect Elan - Elite Design’’, *Meiko Ltd.,* 650 Aztec West, Bristol, UK, May 1993.

[KaHM87] M. Karol, M. Hluchyj, S. Morgan: ‘‘Input versus Output Queueing on a Space-Division Packet Switch’’, *IEEE Trans. on Communications,* vol. COM-35, no. 12, December 1987, pp. 1347-1356.

[KaSC91] M. Katevenis, S. Sidiropoulos, C. Courcoubetis: ‘‘Weighted Round-Robin Cell Multiplexing in a General-Purpose ATM Switch Chip’’, *IEEE Journal on Sel. Areas in Communications,* vol. 9, no. 8, October 1991, pp. 1265-1279.

[Kate94] M. Katevenis: ‘‘Telegraphos: High-Speed Communication Architecture for Parallel and Distributed Computer Systems’’, Technical Report FORTH-ICS/TR-123, *ICS, FORTH, Heraklio, Crete, GR,* May 1994, 38 pages; on-line: ftp.ics.forth.gr: tech-reports/1994/94.TR123.Telegraphos.ps.Z

[KVES95] M. Katevenis, P. Vatsolaki, A. Efthymiou, S. Stratakis: ‘‘VC-level Flow Control and Shared Buffering in the Telegraphos Switch’’, *Proceedings of the Hot Interconnects III Symposium,* Stanford Univ., CA, USA, August 1995.

[Koza91] T. Kozaki, N. Endo, Y. Sakurai, O. Matsubara, M. Mizukami, K. Asano: ‘‘32 $\times$ 32 Shared Buffer Type ATM Switch VLSI’s for B-ISDN’s’’, *IEEE Journal on Sel. Areas in Communications,* vol. 9, no. 8, October 1991, pp. 1239-1247.

[Kung92] H. T. Kung: ‘‘Gigabit Local Area Networks: A Systems Perspective’’, *IEEE Communications Magazine,* vol. 30, no. 4, April 1992, pp. 79-89.

[LaSe95] R. Lamaire, D. Serpanos: ‘‘Two-Dimensional Round-Robin Schedulers for Packet Switches with Multiple Input Queues’’, to appear in *IEEE/ACM Trans. on Networking,* 1995.

[PaBr93] A. Pattavina, G. Bruzzi: ‘‘Analysis of Input and Output Queueing for Nonblocking ATM Switches’’, *IEEE/ACM Trans. on Networking,* vol. 1, no. 3, June 1993, pp. 314-328.

[Shob91] Y. Shobakate, M. Motoyama, E. Shobakate, T. Kamitake, S. Shimizu, M. Noda, K. Sakaue: ‘‘A One-Chip Scalable 8 * 8 ATM Switch LSI Employing Shared Buffer Architecture’’, *IEEE Journal on Sel. Areas in Communications,* vol. 9, no. 8, October 1991, pp. 1248-1254.

[Souz94] R. Souza, P. Krishnakumar, C. Ozveren, R. Simcoe, B. Spinney, R. Thomas, R. Walsh: ‘‘GIGAswitch System: A High-Performance Packet-Switching Platform’’, DEC, *Digital Technical Journal,* vol. 6, no. 1, Winter 1994, pp. 9-22.

[Stun94] C. Stunkel, D. Shea, B. Abali, M. Denneau, P. Hochschild, D. Joseph, B. Nathanson, M. Tsao, R. Varker: ‘‘Architecture and Implementation of Vulcan’’, *Int. Parallel Processing Symposium,* April 1994.

[TaFr88] Y. Tamir, G. Frazier: ‘‘High-Performance Multi-Queue Buffers for VLSI Communication Switches’’, *Proc. of the 15th Int. Symp. on Computer Architecture,* ACM SIGARCH vol. 16, no. 2, May 1988, pp. 343-354.

[TaCh93] Y. Tamir, H-C. Chi: ‘‘Symmetric Crossbar Arbiters for VLSI Communication Switches’’, *IEEE Trans. on Parallel & Distributed Systems,* vol. 4, no. 1, January 1993, pp. 13-27.

[Turn93] J. Turner: ‘‘An Optimal Nonblocking Multicast Virtual Circuit Switch’’, Technical Report WUCS-93-30. *Washington Univ., St. Louis, MO, USA,* June 1993, 22 pages.

[WeEs93] N. Weste, K. Eshraghian: *‘‘Principles of CMOS VLSI Design − a Systems Perspective’’,* second edition, Addison-Wesley, ISBN 0-201-53376-6, 1993.

[YeHA87] Y. Yeh, M. Hluchyj, A. Acampora: ‘‘The Knockout Switch: A Simple, Modular Architecture for High-Performance Packet Switching’’, *IEEE Journal on Sel. Areas in Communications,* vol. SAC-5, no. 8, October 1987, pp. 1274-1283.