Weighted Round-Robin Cell Multiplexing in a General-Purpose ATM Switch Chip

Manolis Katevenis, *Member, IEEE*, Stefanos Sidiropoulos,
and Costas Courcoubetis, *Member, IEEE*

revision of 15 May 1991
(first version was submitted on 15 February 1991)

To be published in the IEEE Journal on Selected Areas in Communications,
special issue on Large Scale ATM Switching Systems for B-ISDN, Fall 1991.
This paper has *not* been presented at any conference yet.

The authors are with the Institute of Computer Science, Foundation of Research and Technology – Hellas (FORTH), and with the Department of Computer Science, University of Crete, Heraklio, Crete, Greece.

Correspondence to them can be addressed at: Computer Science Inst., FORTH, P.O.Box 1385, Heraklio, Crete, GR-711 10 Greece. Electronic mail: "katevenis@csi.forth.gr". Telephone: +30 (81) 239713, 221171. Facsimile: +30 (81) 229342.

Abstract – We present the architecture of a general-purpose B-ISDN switch chip and in particular its novel feature: the cell (packet) multiplexing algorithm and its implementation in hardware. The chip is intended to be connected, via its bit-parallel links, to other similar switch chips or to link-interface chips, providing maximum flexibility in building small or large networks. Our chip implements buffering, routing, flow control, cell scheduling, and cut-through all in hardware. We present a detailed design of how our chip implements its scheduling function consisting of a weighted round-robin multiplexing scheme, using a counter, frequency weights stored in reverse bit order, and a content-addressable memory. This algorithm allocates the available bandwidth to the virtual circuits that can use it, in proportion to the prescribed weights. Other features are chip-to-chip flow control with a built-in window mechanism, a pool of shared buffers, and a set of dedicated buffers, thus offering powerful buffer management capabilities. The above features are key for our chip to successfully switch traffic with different service requirements such as real-time traffic and non-interactive data traffic; the mechanisms built into the chip can be used by the network manager to offer guaranteed service performance to the real-time traffic, and to fully utilize the spare capacity of the links by serving the lower priority traffic without allowing congestion (fairly allocating buffers and bandwidth). We are currently designing this chip for a full-custom CMOS technology; its crucial parts have been laid out and simulated, thus proving its feasibility.

1. INTRODUCTION

In Broad-band Integrated-Services Digital Networks (B-ISDN) the rate at which the cells (packets) of traffic must be processed is so high (on the order of 1 µs or less per cell) that there is almost no room for software to operate – all the processing must be done in hardware [Toba90]. We present the architecture of a switch chip designed to do exactly that: handle buffering, routing, flow-control, cut-through, and multiplexing of cells all in hardware. The chip operates according to the Asynchronous Transfer Mode (ATM): cells, consisting of 53 bytes each, flow through Virtual Circuits (VCs) or Virtual Paths (VPs) and the switching nodes forward them to the proper outgoing link as soon as contention for that link allows. Our switch chip handles a number of issues in unconventional and novel ways: it uses chip-to-chip flow-control and dedicated buffers in order to be able to operate at full capacity even when some particular traffic stream (or even a traffic class, e.g. data) is congested, and it multiplexes (schedules) the flow of cells through each outgoing link using a weighted round-robin algorithm implemented in hardware.

Figure 1.1 presents a block diagram of the switch chip. Our current design is for a 4×4 switch, with a throughput of 400 Mbits/s on each of the 4 links, but the architecture of the chip is scalable, so that the chip capacity can be easily extended, with more advanced technology and packaging, to a dozen or two of multi-Gb/s links. Central to the chip is the buffer RAM, which is shared among the links. The bytes that enter through the incoming links are first assembled into ATM cells in special input buffers, and then each entire cell is written into a row of the buffer RAM (all bits being written in parallel). A routing table (RAM) associated with each incoming link specifies the outgoing link and new VC identifier (ID) for all incoming VCs. A ‘‘scanning memory’’ is associated with each outgoing link, and keeps track of all the cells waiting to be forwarded on that link; it contains special circuits which implement our cell multiplexing algorithm. Under the control of these circuits, cells are read out of the buffer RAM, and placed into the appropriate output buffer, from where they get forwarded, one byte at a time, to the corresponding outgoing link. A special cross-bar switch is used to implement cut-through, whenever an incoming cell can be forwarded directly out of the switch, without having to be buffered first.

The chip is designed so that it can be used as a general-purpose building block for B-ISDNs. Multiple such chips can be connected together to form larger switches of arbitrary topology. The number of chips and the size and topology of the composite switch are arbitrary. When the chips to be connected together are next to each other, the connection can be made directly between their pins – a bit-parallel, clock-synchronous link is used. For longer distances, one can use, among others, coaxial cable, optical fiber and microwave links, together with special interface chips, as explained in section 3. Thus, the user of these chips has considerable flexibility in configuring a network with large, centralized switches, or with smaller, distributed ones.

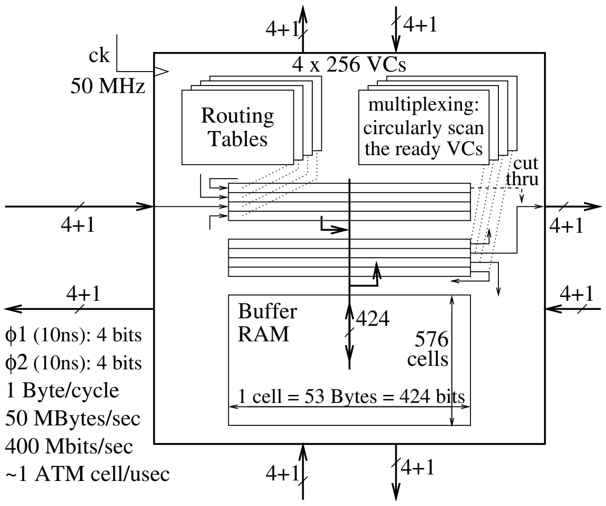

Figure 1.1: Block diagram and capacity of the current switch chip design

The other aspect of the general-purpose character of the chip concerns the ability to support multiple types of traffic with different service requirements, by providing the network manager with a number of powerful mechanisms to control the flow of traffic through the switch. In the rest of the paper we assume that each traffic flow occurs through some virtual circuit or some virtual path already set through the network; for simplicity, since our chip can not distinguish among the two, we use the term VC to refer to either of these routing entities. The first mechanism is the support of priorities. There are four priority classes for VCs, and the cells of a certain class have absolute priority over the cells of the lower classes. The second mechanism is the allocation of bandwidth within each priority class. In every priority class, cell contention for a link is resolved in a *weighted round-robin fashion,* where each VC has its own weight factor and hence shares the link throughput in proportion to this weight. The third mechanism is a chip-to-chip window flow control scheme, which operates on a VC base. An upstream chip does not send a cell on a particular VC to the next chip down the line, unless it receives a token from the downstream chip. This mechanism can be used to guarantee that no cell is transmitted unless there is a buffer for it. As we will mention later, this "backpressure" mechanism can also be used to keep the downstream chip as busy as possible. The fourth mechanism is the support of different buffering mechanisms: a subset of the VCs have buffers dedicated to each of them, while the remaining VCs share a common buffer pool. The advantage of the shared buffer pool is that it provides better buffer utilization than having dedicated buffers, due to the statistical nature of the traffic. On the other hand, when not having dedicated (reserved) buffers, a few VCs can occupy all buffers in the pool, and hence congest the system. An important point we should stress is that the chip has been designed so that the amount of fast memory on the chip is enough to implement *all* the buffering needed for the operating conditions we anticipate. Interestingly enough, a key reason for this is the large number of simultaneous VCs supported by the architecture of the chip. The multiplexing of this large number of relatively small traffic flows makes large fluctuations of the aggregate traffic to be highly unprobable, and hence we anticipate the queues to be relatively small.

From the above discussion it follows that the network manager can choose any combination of priority class, weight factor, and buffer allocation policy for every VC, thus having a very wide spectrum of alternatives for serving the different quality-of-service requirements of traffic such as management traffic, isochronous traffic (voice, video), interactive data traffic, bulk data transfers, etc., as discussed in section 5. Indeed, our chip has been designed to meet the quality-of-service requirements of B-ISDN. Similar approaches have been considered in [LPTZ85], [LaTG88].

Section 2 of the paper presents the flow control and buffer management strategies that allow the chip to operate at top performance under congestion, and it explains why round-robin scheduling should be used under those circumstances. Section 3 discusses the chip architecture, and how the key choices were made. The weighted round-robin algorithm and its hardware implementation are presented in section 4. Section 5 analyzes the statistical performance of the switch. The critical parts of the chip have been laid-out and simulated, thus proving the feasibility of our architecture; the results are presented in section 6. We are currently designing the full chip.

2. FLOW CONTROL AND CONGESTED OPERATION

In this switch chip, we use the same general flow-control and buffer-allocation philosophy as in our previous study [Kate87], which we will briefly review here. We start from the general view that link throughput is a more precious network resource than buffer space in modern technology. This follows from a rough relationship that the two have, *"(buffer space) = (throughput)* × *(round-trip delay)",* which, when converted to numbers, amounts to about 1 Kilobyte per kilometer and Giga-bit/second. Even when this refers to buffer space per virtual circuit or virtual path, the cost of the total buffer space implemented using a few DRAM chips is still very much less than the cost of the long optical fiber that they correspond to. Two conclusions follow.

The first conclusion is that it is better to buffer packets than to drop them and later retransmit them. (This refers to traffic that absolutely has to reach its destination – e.g. data; for isochronous traffic – e.g. voice – where a delayed packet is useless, it is still better to drop packets when they cannot be transmitted in time). For this reason, we implement hop-by-hop (node-to-node) flow control, based on a sliding-window mechanism and on "permits" (tokens) transmitted backwards. A token is sent to the upstream neighbor every time when room is made in a local buffer for more packets (cells). This guarantees that buffers will never overflow and packets will never have to be dropped; instead, the system works with a kind of "back-pressure", which stops the flow of data when enough buffer space does not exist. In order for the flow to be able to proceed at a given peak rate when there is nothing to stop it, the window size (buffer space) must be roughly at least as large as the desired peak throughput times the "round-trip delay"; the latter is the total time for a flow-control token to travel back, for the upstream neighbor to respond to it, and for the data packets to travel forward from that neighbor to the local node.

The second conclusion is that it is better to fully utilize the available link throughput than to economize on buffer space. In traditional networks, minimization of the required buffer space is achieved *(i)* by sharing that space among all VCs, and *(ii)* by guaranteeing that the traffic load offered to each link is below the link capacity by a certain safety margin. If the network fails to satisfy property *(ii)* on some link, then the packets of the VCs which encounter heavy traffic on that link get concentrated and stay in the (shared) buffers of the nodes upstream from this link, taking up buffer space which thus becomes unavailable for other VCs. These other VCs then get slowed down, even though their own path may not pass through heavy-traffic areas. This is the phenomenon of high traffic load (*congestion*) in a part of the network causing the dramatic decrease of the network capacity in a much wider area. We adopt a different approach to buffer allocation: for those classes of traffic and those VCs for which it is not feasible or desirable to guarantee property *(ii)* above, we provide *dedicated* buffer space, individually to each VC.

This buffer preallocation policy completely solves the problem of such undesirable interactions between VCs (as well as the store-and-forward deadlock problem), since it makes them independent of each other [Tane81, p. 216]. In this way, *congestion has no negative effects* and it even becomes *desirable,* since it is needed for achieving full utilization of the network throughput. Here, and in the rest of this paper, we use the word "congestion" to mean that the load offered to the network is higher than the network capacity, rather than the situation where an increasing load causes a decrease of useful work performed. Clearly, in order for the links to never stay idle, the buffers in the switches should never be empty, which implies that the load is higher than the capacity. We believe that this is a desirable situation, since in our view the scarce network resource is throughput. A good method to achieve this full link utilization is to allow the *low-priority* traffic to be congested; massive data transfers that are not time-critical are a good candidate.

The amount of buffer space needed in order to implement this preallocation policy is considerable, but not huge. Each of the potentially-congested VCs needs approximately the buffer size that was computed four paragraphs above. For example, one thousand VCs, *each* of which is allowed to reach a peak instantaneous rate of 1 Gb/s, would need a total of 1 MByte of buffer storage for each kilometer of link length; this is well within the limits of technology, and of minor cost compared to the cost of the link. An important method that can be used to reduce these buffer space requirements, or to concentrate the buffers at the end-points of long links, is to temporarily merge multiple VCs into one virtual path, over a *common* portion of their path. We assume that such aggregation of VCs does happen in the networks in which our switch chip will be used, so that the sizes of the buffer memory, routing memory, and scanning memory can be kept low (section 3.3).

In the absence of congestion, the network does not have to make any bandwidth (throughput) allocation decision; since the offered traffic is less than the network's capacity, all the packets that are sent will eventually get-through. When congestion exists, *the network* has to make the bandwidth allocation decisions, i.e. it has to arbitrate among all the VCs that try to use more throughput than there is of it. In that case, the cell multiplexing method (which cell to forward next along an outgoing link) is the primary means of making these decisions. Priority classes obviously control this allocation, but in a very crude manner. If within a congested priority class all the cells of all the VCs are placed on a single queue and serviced in a FCFS (FIFO) order, then an "unfair" allocation of the available throughput is made: those VCs that "merge into" a path at later stages get more service than the ones which merged earlier. If each VC has a queue of its own, instead, and these queues are served in a *round-robin* fashion, then a "fair" allocation is made. Round-robin scheduling means circularly scanning all the VCs (of a given priority) and transmitting one cell from each of them that is found to be "ready"; a VC is ready iff its buffer on the local node contains at least one packet and its buffer on the destination node is known to have empty space for at least one packet. The round-robin scheduling of cells *(i)* equally distributes all the available link throughput to all the virtual circuits that can use it, and *(ii)* if some virtual circuits do not need to or cannot use all of their share of throughput, then they get allocated as much as they need, and the remaining portion is equally distributed to all the other VCs that can use it. Round-robin scheduling was used in "TYMNET", whose VCs are low-speed (one end of them is a terminal) [Tyme81, p. 395], was also proposed in [Nagl84], [Nagl87], [Morg89], and was also studied in [DeKS89]. For practical purposes, exact equality among the VCs may not be the desired proportionality factor in the allocation of throughput. For that reason, as described in section 4, our chip will implement a *weighted* round-robin scheduling, which allocates the available bandwidth to the virtual circuits that can use it, in proportion to arbitrary prescribed weights. Round-robin multiplexing also has other nice properties, which we analyze in section 5.

3. ARCHITECTURE OF THE SWITCH CHIP

This section presents the hardware architecture of the chip, discusses the important alternatives that were considered, and explains how the crucial decisions were made. We discuss first the interface between the chip and the external world, then the hardware organization of the buffer RAM and its scalability to higher switch capacities, then the routing and scanning memories, and finally the buffer management hardware.

3.1 System Architecture and Pin Interface

Figure 3.1 illustrates the overall network architecture that we envision and for which this switch chip is designed. The network can be arbitrarily large. It consists of parts that are separated by considerable physical distance from each other and which are linked with each other via links, usually of bit-serial nature, like coaxial cables, optical fibers, microwave links, etc. One part of the network, that resides at one physical location, may be as small as a few chips on a small printed-circuit board or as large as thousands of chips on many boards in several boxes – all located physically close to each other. An entire such subsystem, or appropriate parts of it (e.g. a few boards in a box), will be concentrated enough for its core to operate synchronously under a single clock and for its chips to communicate via bit-parallel links. In our architecture, these subsystems will be built using very few different types of chips: the switching fabric will consist of chips of a single type – the switch chip that this paper describes – while the "external" links will be interfaced to this fabric using *link interface* chips, of one particular type for every class of external links. In figure 3.1, within the lower dashed box, a 10×10 switching fabric is built by connecting two 6×6 switch chips. Obviously, a form of "internal blocking' exists in this particular topology – the middle link will be a bottleneck if traffic is not localized on the left and on the right. In every case, the network designer can choose a topology that has a tolerable amount of internal blocking for the expected pattern of traffic, or even one that has no blocking at all.

One of the first choices that had to be made was the pin-level interface between switch chips that are connected together to synthesize larger switching fabrics. In electronic (i.e. non-

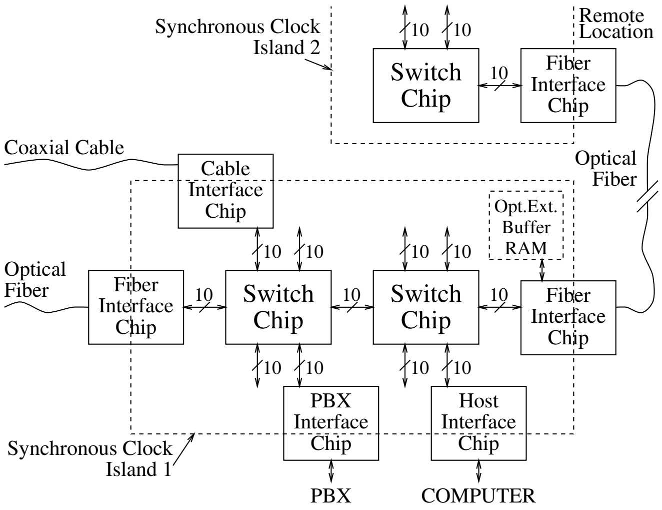

Figure 3.1: Envisioned network architecture

photonic) digital systems, a basic method for achieving high throughput is to use wide datapaths operating on bit-parallel data. Furthermore, digital operation is greatly simplified by adopting a synchronous clocking discipline: no synchronizers, clock regenerators, or elastic buffers are needed, and transfer delay is minimized. Thus, our switch chip operates under a single, external clock, and communicates with its neighbors via synchronous, bit-parallel links. Operating several chips synchronously from one *high frequency* clock is not easy, but it is not impossible either. If the chips are kept in close proximity, if they only communicate with close neighbors, and if the clock is distributed via transmission lines with carefully equalized delays, then one can achieve such high-speed synchronous operation. When synchronous operation becomes impractical, one can adopt "messochronous" operation among neighboring switches, using a small elastic FIFO and a timing recovery circuit to compensate for the difference in clock phase [Mess90].

Whenever two of these synchronous "islands" (e.g. island-1 and island-2 in figure 3.1) are far from each other – whether these are two boxes in the same room or two boxes many kilometers apart – their communication has to be performed in a different way. Such sub-systems have to have different clocks, and, usually, their communication has to be bit-serial. Rather than encumbering all switch chips with bit-serial, non-synchronous interface circuits, it is preferable to provide separate building-block chips for this purpose. *Link-interface* chips can be used to connect remote switching fabrics. Analogous interface chips can be used to connect to host computers, PBX's, video equipment, etc. The network designer has maximum flexibility in choosing the size and the topology of the local switching fabric, and the ways to connect it to the rest of the network. There is a large difference in round-trip delay – as defined in section 2 – between the connections from one switch chip to another and the connections from one synchronous island to another. The former is very short, and thus we only need small buffer space in the switch chips. The latter is longer to very much longer, thus entailing the need for more buffer space in the link-interface chips. This, however, is quite acceptable, since the link-interface chips do not need to provide the routing and switching functions. In the case of very long links, it is acceptable to connect external RAM chips to the corresponding link-interface chip, whose number (and cost) is proportional to the length (and a small portion of the cost) of the link, as discussed in the previous section. This paper does not deal with the design of the interface chips.

Each one of the bidirectional links in and out of the switch consists of two sets of unidirectional wires, as in figure illustrated 1.1. We prefer unidirectional pins, rather than tri-state ones, to avoid arbitration overhead and turn-around delays. In the current design, each unidirectional path consists of 5 wires; more wires per path can be used if the designer wishes to increase the link throughput. Thus, the chips of the present configuration need 40 pins for the links, and can be housed in 48-pin packages. The 5 wires in each link direction carry 10 bits of information per clock cycle − 5 bits during the positive half-cycle of the clock and 5 more bits during the negative half; in this way, the signaling rate on the clock pin does not have to be twice the desired signaling rate on the data pins. Assuming that each output pin cycles a 20 pF capacitance through a 5 Volt swing 50 million times per second (50 MHz clock, 100 Mb/s signaling rate), the maximum power consumption of each pin driver is 12.5 mW. Thus, driving the 4 links in our current design takes 0.25 Watts of power, which is comfortably low; an 8×8 switch chip with 9 wires per link direction (800 Mb/s data rate per link) would need 1 Watt to drive all of its links.

The ten bits of information that are transferred in each link direction during each clock cycle contain one 8-bit data byte, one signaling bit, and one flow-control bit. The current chip design operates under a 50 MHz clock, giving a link throughput of 50 MBytes/s (400 Mbits/s) of data in each link direction. The signaling bit differentiates between a normal data byte in an ATM cell and the special cell-delimiter control byte. Thus, a new 53-byte ATM cell can be transmitted through a link every 54 clock cycles − 53 for data and 1 for the delimiter. When a link is idle, consecutive cell-delimiters are transmitted. The flow-control bits are enough to carry 6 flow-control commands of 9 bits each (=54 bits) per ATM-cell time; we will see in a while how these are used. A more compact encoding of the signaling and flow-control bits could allow about 100 bits of flow-control information per cell time, at the expense of a more complex cell-synchronization (framing) sequence; these would be reduced to 50 bits in a wider, 9-wire link carrying two data bytes per cycle.

3.2 Buffer RAM

Inside the switch chip, the central part of its datapath is the cell buffer memory, which is *shared* among all links (figure 1.1, bottom). A shared buffer memory is certainly preferable over separate link-dedicated memories, if one can manage the high throughput requirements that sharing places upon it. The advantages of a shared buffer memory include better utilization of buffer space, simpler routing of the buses to and from the links (the shared memory bus replaces the cross-bar switch), and less overhead owing to having less memories to manage [Toba90]. The on-chip throughput of a modern RAM can be made very high by providing parallel access to all the bits in a row. A RAM of a few hundred Kbits is internally organized as several hundred rows of several hundred bits each; every single access to this array internally reads and/or writes (can write) an entire row. We can take advantage of this RAM organization and of the fact that our accesses only need to be at the granularity of entire ATM cells, to get very high throughput out of a memory in which each row contains one ATM cell, that is 53 Bytes = 424 bits. For example, the memory of our current chip version, with its cycle time of 50 ns (section 6), has a throughput of 20 Mega-accesses/s × 424 bits = 85 Gbits/s. A shared buffer RAM for an $L \times L$ switch needs a throughput of $2L$ times the link throughput, to be able to receive L cells and transmit L other cells per link cell time. Table 3.1 lists the cycle time that a 424-bit wide RAM should have in order to be able to support the above throughput requirement, for various numbers L of links and various link speeds.

Table 3.1: Shared buffer RAM requirements
(RAM row size = 424 bits = 1 ATM cell)

number of links L	link throughput *Gbits/s*	required RAM cycle time *ns*
4	0.4	132
4	1.0	53
4	5.3	10
8	1.0	26
8	2.6	10
16	1.0	13
16	4.0	3.3

The first line of this table reflects our current design; a cycle time of 130ns can be trivially achieved in the current CMOS technology (our cycle time is 50 ns). Cycle times of a few tens of nanoseconds are also common today, while the shorter cycle times, in the range of a few nanoseconds, are achievable in BiCMOS or ECL technology. For example, a 4 Mbit SRAM chip with a 7 ns access time has already been demonstrated [OSKY91]; if the technology and the capacity of this chip were available to us, the memories in our chip could be three to four times larger as well as significantly faster, thus yielding switch chips with 12 to 16 links of 1 or more Gbits/s each, and with the same number of VCs *per link.* It follows that the shared buffer architecture will not run out of steam for some more years to come.

In order to access the shared buffer RAM at the ATM cell granularity, the data from the incoming links are first "shifted into" special *input buffers,* until an entire cell has been assembled, and then all the bits of the latter are written *in parallel* into the buffer RAM. Similarly, to feed the outgoing links, entire cells are read from the buffer RAM into special *output buffers* and then shifted out to the links. There are 4 double input buffers and 4 double output buffers in a 4×4 switch chip. Double-buffering is required, on the input side, to ensure that no incoming data are lost from the time a cell gets assembled to the time the (shared) buffer RAM is available to accept that cell; on the output side, double-buffering ensures that cells can be sent out "back-to-back". An important part of the input buffers are the *cut-through* buses, which are required in order to immediately forward a cell if its outgoing link becomes available while the cell gets assembled in its input buffer [KeKl79]. Figure 6.2 (section 6) shows details of these circuits.

3.3 Routing and Scanning Memories

Each time the head of a new cell enters into the switch chip, its VC ID is looked up into the *routing table* of the corresponding incoming link to find the outgoing link and the new VC ID on that link. The routing tables normally would also contain the service class (priority) of the VCs, their service frequency weights, and their flow-control states (stopped or not), but we maintain all this information in the output-multiplexing memory to be described below. Each routing table is implemented as a RAM memory, addressed by the VC ID. We provide a separate routing table for each incoming link, rather than a single shared table equal to their total size, *(i)* in order to avoid access conflicts, so that the tables are available for consultation as soon as the header of a cell enters the chip, and *(ii)* so that, with a given number of bits for the VC ID, more VCs can pass through the switch, since VCs on different links can use the same ID.

In the current design we allow a maximum of 256 VCs simultaneously open on each link. We imposed this limit in order to keep down the sizes of the chip memories – primarily the output-multiplexing, and secondarily the buffer RAM and the routing tables. Thus, the VC ID's have 8 bits, and each routing table has a size of 256 words by 10 bits – 2 bits to identify an outgoing link and 8 bits to identify a VC on it. Allowing only 256 VCs per link would be a severe limitation for a large network if these were end-user VCs. However, these will in fact be "virtual paths" (VPs) formed by hierarchically grouping multiple user-VCs together over the common part of their path, as mentioned in section 2. Under these circumstances, the limit of 256 is quite comfortable. As technology progresses and more storage can be placed on a chip, both the number of routing tables and the number of VCs in each can grow, thus providing scalability to the switch architecture.

As soon as the outgoing link and the new VC ID of an incoming cell have been determined, the responsibility for keeping track of the cell is passed to the control circuit of its outgoing link. The main part of each such controller is a memory that records all administrative information related to the VCs and the cells destined to go on the corresponding outgoing link. A finite-state machine (FSM) continuously scans that memory, trying to decide which cell should be forwarded next – we refer to that memory as the *scanning-memory.* During the design of our chip we considered several alternatives for the organization of the buffer and the scanning memories, which we briefly review next.

3.4 Buffer Management Hardware

First we looked at how we might implement in hardware the management of traditional queues of cells (see e.g. [Fuji83], [TaFr88]), in a buffer memory that is shared among the VCs, as a measure against which to compare our own eventual scheme. Since we would want to implement round-robin scheduling, rather than FCFS, we would need a separate queue of cells for each VC. Figure 3.2 shows two alternatives, the second of which seems better. In **(a)**, a special queue-memory, with the same number of words as the buffer memory, is used to hold pairs of VCIDs and buffer-addresses for the cells that are currently buffered, *in the order* in which these cells have arrived; effectively, this memory maintains all the queues for all the VCs. To support the dequeueing of the oldest cell of a given VC – say VC#5 – this memory has to be content-addressable (CAM) and the corresponding priority encoder has to select the top-most matching entry. This memory also has to be organized as partial-shift-up registers, so that an entry is dequeued by shifting all entries below it up by one position. These two requirements, CAM and shift, make it an expensive memory. In addition to this queue memory we would still need the scanning memory, to hold the various administrative information for each open VC; thus, solution (a) is not very attractive.

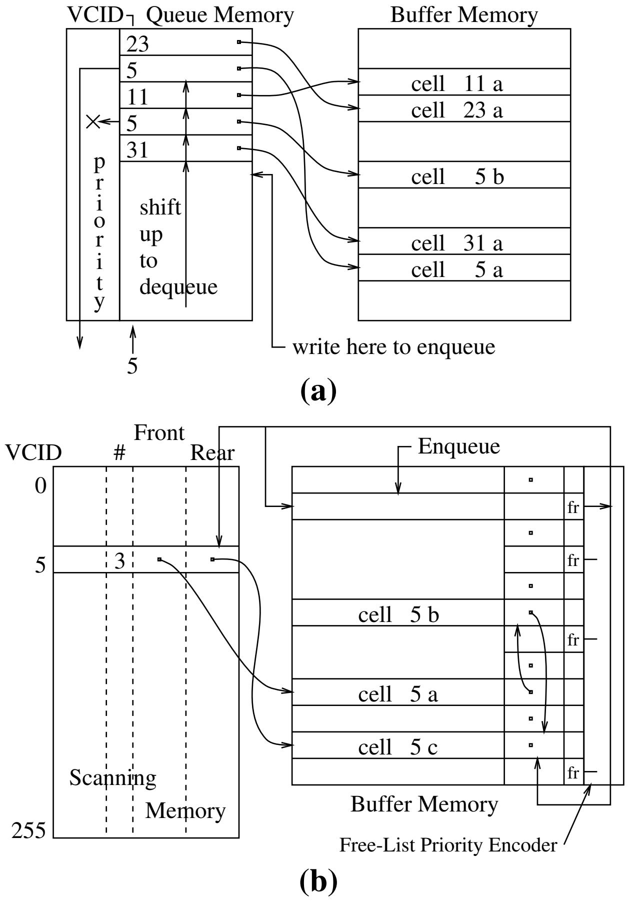

Figure 3.2: Traditional cell queues in a shared buffer organization

The organization shown in part **(b)** of figure 3.2 is better. It augments the buffer memory with a pointer field in every word (and with an additional address-decoder for that field, as we'll see); this pointer is used to link together all the cells in the queue of a particular VC. The scanning-memory, which contains one entry for every VC, is also augmented with two pointer fields per entry, used as front and rear queue pointers. No third memory, analogous to the queue memory in (a), is needed in this scheme. Dequeueing a cell is relatively easy: using the front pointer, we read a single word from the buffer RAM; this provides us both with the cell and with the new value for the front pointer. To be able to enqueue a cell in a single buffer-RAM cycle time, we need a pointer to a free word and two address decoders. Using the free word pointer and the first address decoder, the cell is written into the data part of the buffer RAM. Using the rear pointer and the second address decoder, the value of the (ex-)free pointer is written into the pointer part of the buffer RAM. The free-word list of the buffer memory can be implemented as another linked list, using the pointer field of that RAM. During cell dequeueing, this necessitates a read-modify-write access to the pointer field of the buffer RAM, so that the dequeued word is simultaneously enqueued into the free list; during cell enqueueing, it necessitates a pointer-read from the (ex-)free word, simultaneously with the pointer-write into the (ex-)rear word. Alternatively, the free list can be implemented using a flag bit in each buffer-RAM word and a priority encoder that selects one of the true bits (this implementation of the free list would also have to be used in solution (a)). The scanning memory in (b) also contains a count of the number of buffered cells per VC, in order for flow-control decisions to be made.

As discussed in section 2, a shared buffer memory has the advantage that less total buffer space is required, but also the disadvantage that overload conditions yield loss of cells and/or underutilization of the throughput. Our switch chip manages part of its buffer memory as shared, and part of it as dedicated, while it also radically simplifies queue management. We will present and briefly justify this organization here, while a deeper analysis of it appears in section 5. Virtual circuits 0 through 127 on each outgoing link are intended for traffic that is allowed to – or even expected to – try to overload the network, and thus they each have a buffer dedicated to itself. Virtual circuits 128 through 255, on the other hand, can only be used by traffic for which the management of the network can guarantee that it will never exceed a certain portion of the network capacity, and thus all these VCs of *all* outgoing links share a common buffer pool of 64 cells.

An important decision, that significantly simplifies the hardware of our switch chip, is to allow a *maximum of one buffered cell per VC.* For the VCs that have dedicated buffer space, this significantly reduces the required buffer RAM size. For the VCs that share the 64-cell buffer pool, this greatly simplifies (actually eliminates) queue management. Also, for all VCs, this decision simplifies flow-control. One reason why this decision has a negligible negative effect on the chip performance is analyzed in section 5. Another reason is the following: as discussed below, the round-trip delay of a flow-control command to the next upstream switch chip and of its effect on the downstream traffic is about *two cell times,* on the average. Thus, limiting the buffer space per VC to one cell, limits the peak instantaneous throughput *per VC* to one half the link throughput – not a very important restriction.

Figure 3.3 illustrates our buffer and scanning memory organization. The entire buffer RAM has a capacity of 576 = 4×128 + 64 cells. Each of the 4 outgoing links has 128 of these cells dedicated, one-to-one, to its 128 first VCs. Thus, the first 128 entries of each scanning-memory do not need a buffer pointer. The other 128 entries of the scanning memories need a 6-bit pointer to indicate the position of their buffered cell, if it exists, in the 64-cell shared buffer pool. The 64-word shared part of the buffer RAM has one additional bit per word, used as an empty/full flag, and a priority encoder, to implement the "free list". Relative to the scheme of figure 3.2(b), this solution has 35 % less bits in the scanning memory, and a few less bits per word and one less decoder in the buffer memory. The scanning memory has the following fields:

Valid (1 bit), indicating whether this VC is currently open;

Source link and VC ID (10 bits), needed in order to know where to send flow-control commands to;

Priority class of this VC (2 bits), needed in choosing the next VC to serve;

Weight of service frequency (12 bits) – see section 4;

Still-unvisited flag (1 bit) – see section 4;

Stopped flag (1 bit), indicating whether the flow of this VC is currently stopped because the next downstream switch chip has a full buffer;

Full flag (1 bit), indicating whether this VC currently has a buffered cell.

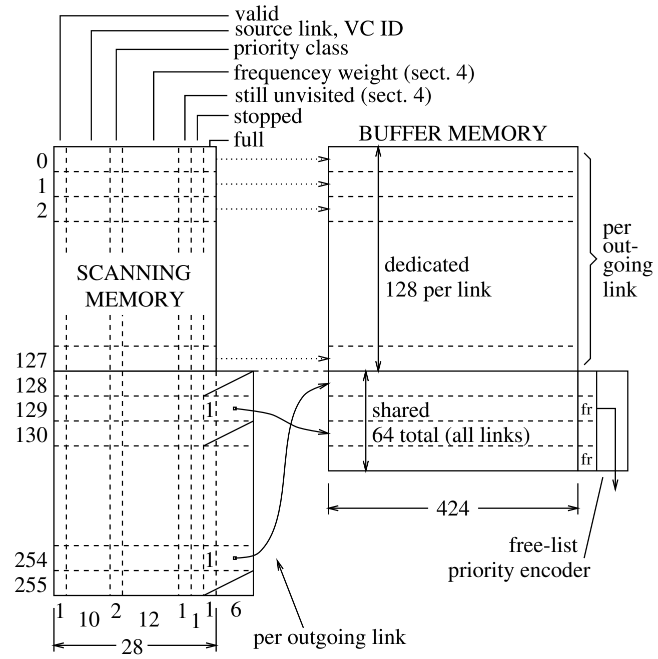

Figure 3.3: Buffer memory and scanning memory organization

Given the limitation of having at most one buffered cell per VC, the *flow control* algorithm becomes trivial: whenever a cell is forwarded downstream a link, a corresponding notification (*token*) is sent upstream, saying that one more cell of this VC can be received. This token only needs to have 8 bits, identifying a particular VC; a 9th validity bit is also needed, unless one particular VC ID is reserved to signify an invalid token. Six tokens can be transmitted through a link per cell time; in our switch, this safely exceeds the peak requirement for token transmission over a link, which is three tokens per cell time, since three cells from VCs of this link may happen to get forwarded through the three other links, all in one cell time. The round-trip delay of tokens and cells in our chip design varies between a minimum of one and a maximum of three cell-times, mostly according to the contention among tokens for access to the upstream link, according to the contention among link-circuitry for access to the buffer RAM, and according to the relative time-alignment of cell-boundaries on the various links (given that cell scheduling is non-preemptive). Thus, the average round-trip delay between two switch chips connected directly to each other is about two cell-times.

4. WEIGHTED ROUND-ROBIN MULTIPLEXING IN HARDWARE

The idea of round-robin scheduling, in general, is that a server process circularly and repeatedly "visits" a number of clients and performs one job for each of them that has such a need at the time of the visit. In the case of multiplexing ATM cells for an outgoing link of the switch chip, visiting the VCs in a round-robin fashion and forwarding one cell from each ready VC upon each visit is in principle fairer than FIFO (FCFS) multiplexing, as it was discussed in section 2. However, to be really fair, this mechanism should not treat all VCs as exactly equal, but rather as equal within a given weight factor: when the throughput requirements of two VCs cannot be met, the available throughput should be distributed to them in proportion to their weight factors W_1 and W_2. This can be achieved with the *weighted round-robin* algorithm illustrated below.

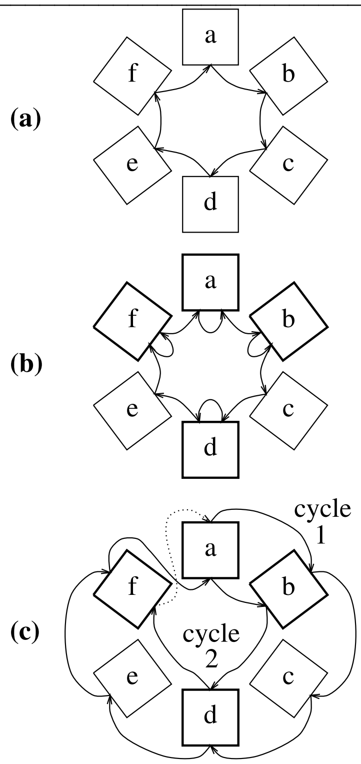

Figure 4.1: Round-robin **(a)** *and weighted round-robin* **(b)**, **(c)** *visits*

Figure 4.1(a) illustrates the classic round-robin algorithm: during every cycle of the server process, all clients are visited exactly once each. Part (b) of this figure is a first approach to the weighted round-robin scheduling. Clients a, b, d, and f receive twice more frequent service than clients c and e, since the former are visited twice while the latter only once in every 10 visits. In part (c) of the figure, the frequencies of visits are maintained the same, but the visits to the "frequent" clients are spread more evenly in time. For example, two successive visits to a are separated by either zero or eight visits to other clients in schedule (b), while in schedule (c) they are separated by either three or five other visits. We will adopt the scheduling style of figure 4.1(c): the successive visiting cycles will be labeled $1, 2, \cdots, N-1, N, 1, 2, \cdots$ and each VC will have a certain frequency weight that entitles it to receiving one unit of service during every cycle whose label belongs to a specified subset of the numbers $1, 2, \cdots, N$.

Figure 4.2 shows the main idea of how we implement the weighted round-robin visits in hardware. All the relevant weight and state information of all the VCs is kept in a memory, the *scanning memory,* which is organized similarly to a content-addressable memory (CAM). The "ready" bit of each word represents the flow-control and cell availability information: it is true whenever the buffer RAM contains at least one cell of the corresponding VC, and that VC is not stopped due to back-pressure (flow-control) from the destination switch. The "still unvisited" bits are all set to 1 before each visiting (scanning) cycle begins; during the cycle, after a certain

VC is visited, its *still-unvisited* bit is cleared. In the situation illustrated in figure 4.2, we are in the process of performing a scan cycle labeled "2"; only the VCs a, b, d, and f are entitled to a visit during this cycle, as illustrated by the asterisk that appears in their weight entry. Out of these four VCs, only a, d, and f remain candidate for a visit after the *ready* bit is taken into consideration (see the three asterisks in that column). Further on, VC a was just visited during the last access to the scanning memory, thus leaving only d and f as the eventual candidates for the rest of this scanning cycle. The priority encoder selects d as the VC to be served now: an ATM cell of d will be transmitted, and the *still-unvisited* bit of d will be cleared. During the next query to the scanning memory, the *still-unvisited* flags will only let VC f get through to the priority encoder, and thus it will be served in its turn. The query after that will yield no result, thus signaling that a new scanning cycle must be initiated by setting all *still-unvisited* flags and advancing the cycle label to "1".

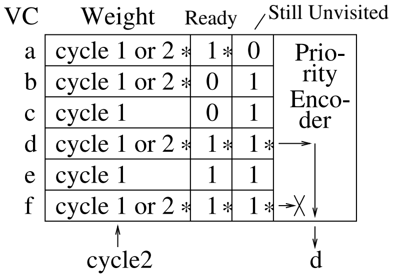

Figure 4.2: The VC scanning memory

The next issue to resolve is how the weight entry of each VC should specify the exact cycle labels during which that VC is entitled to a visit. If the scanning cycles are labeled 1 through N, a VC that is visited during w of these N cycles receives a throughput (service) share of relative size w/N. The minimum service rate is obviously $1/N$ and the maximum is N/N. Which ones of the cycles 1, 2, $\cdots$, N should a w/N-rate VC get visited in? One solution might be to visit it during the cycles 1, 2, $\cdots$, w, but that would result in an uneven spreading of the visits on the time axis. A much better spreading would result by serving this VC during the cycles N/w, $2N/w$, $\cdots$, $wN/w = N$ (or anyway some integer approximations to these values). However, computing these cycle labels in hardware or storing their precomputed values would be unrealisticly expensive. We use a different choice of w labels out of N, that is very easy to generate in hardware, and whose spreading in the time axis does not differ much from the above even spreading. More precisely, in the above even spreading, two successive visits to the VC occur approximately N/w cycles apart. In our method, they occur a maximum of $(N+1)/w_1$ and a minimum of $(N+1)/2w_1$ cycles apart, where w_1 is the largest power of 2 that is less than or equal to w (N must be a power of 2 minus one) (when w is a power of 2, the maximum is $(N+1)/w$ and the minimum is $((N+1)/w)-1$).

Figure 4.3 will help us explain our method of choosing w labels "relatively evenly" among the N cycle labels, where N is a power of 2 minus one ($N = 2^n - 1$). Our method is based on writting the cycle labels, 1 through N, as binary numbers and identifying the rightmost (least significant) "1" digit in this notation. There are $(N+1)/2$ labels in which that rightmost-1 is in the

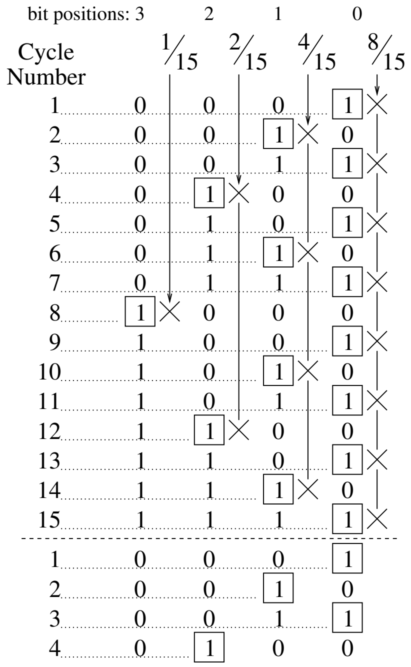

Figure 4.3: Cycle identification and binary frequency components

least significant bit-position (position 0), $(N+1)/4$ labels where it is in position 1, $(N+1)/8$ labels where it is in position 2, etc. These $\log_2(N+1)$ sets of labels contain decreasing powers of 2 of labels each, they are *distinct* (their intersection is null), and they are spread “quite evenly” in the time axis (vertically in figure 4.3). We choose w labels out of these N as follows: write w as a binary number, that is as a sum of powers of two; choose all the labels from all the above sets whose cardinalities are those same powers of two. For example, in figure 4.3, if $w = 5$, i.e. if we wish to serve a particular VC at a relative rate of $5/15 = (4/15) + (1/15)$, we will visit that VC once during every scanning cycle whose label has its rightmost-1 at bit positions 1 or 3.

Figure 4.4 shows the details of the cycle-matching part of the scanning memory (left part of figure 4.2). A counter contains the current cycle label. A priority enforcer drives a bus through the weight part of the scanning memory with a single “1” at the position of the rightmost “1” of the cycle counter. The scanning memory contains the weight of every VC, in binary notation, stored in *reverse* bit order. This reversal occurs because the leftmost positions are the ones where the priority enforcer sends a “1” less frequently, and thus they are the least-significant weight positions. Those VCs whose weight has a “1” at the position where the priority enforcer sends its “1” are enabled as candidates to be visited during the current scanning cycle.

There is one more practical problem to be solved in the above hardware set-up; it has to do with how fast we can search for the next VC to serve (next VC to forward a cell from). In order not to waste any link throughput, the search for the next VC must be performed during the time when the current cell is being transmitted over the outgoing link. In our current design, this time is 53 Bytes × 20 ns = $1.06\ \mu s$, i.e. it is enough for about two dozens of accesses to the scanning memory. This is a comfortable limit if lots of VCs are ready for transmission, but things are different if only the VCs with the smallest weights are ready. Consider, in the example of figure 4.4,

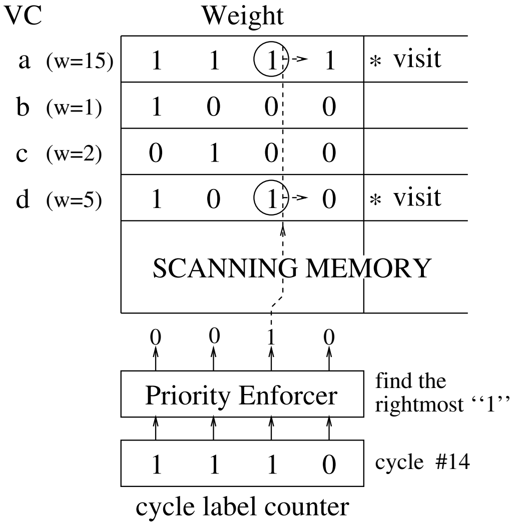

Figure 4.4: Cycle matching – visit those VCs whose weights, stored in *reverse* bit order, have a "1" at the position of the rightmost "1" of the cycle counter.

the case where only VC *b* is ready. Doing one access to the scanning-memory we discover that there is no VC to be served during the current cycle #14, since VCs *a* and *d* are not ready. Consequently, the cycle counter is advanced to cycle #15 and a new access is performed to the scanning memory, which however is again "sterile", because only VC *a* is entitled to a visit during this cycle and that VC is not ready. It will take seven more incrementations of the cycle counter, and seven more sterile accesses to the scanning-memory, before the counter reaches cycle #8 when VC *b* is entitled to a visit and thus the next cell to be transmitted is determined. This is clearly impractical when the cycle counter has many bits (N is large). We can see that during the above multiple sterile accesses, similar searches are uselessly repeated. During all of cycles 15, 1, 3, 5, and 7, the priority enforcer of figure 4.4 asserts its rightmost output wire. Since the search of cycle #15 yields no ready VC, it is useless to perform the exact same search for cycles 1, 3, 5, and 7. Similarly, the searches of cycles 2 and 6 are useless, given that the similar search of cycle #14 found no ready VC with a binary component of 4/15 in its weight (see figure 4.3). What is needed then, is a cycle counter that can *"jump"* over those counts whose rightmost-1 is at bit-positions already determined to be sterile.

Figure 4.5 shows the idea of how to design such a counter. In this figure, the boxes at the top that are crossed out represent bit-positions known to be sterile; positions 0, 2, 3, 5, 7, and 8 were known to be sterile from beforehand, while in the current cycle, whose rightmost-1 is at position 1, we just determined that this position 1 is also sterile. Jumping over the sterile counts (and just them) means to add to the current count the *smallest* number that will generate a next count with a rightmost-1 in a non-sterile bit-position. Of key importance is the *LS sterile string* – the rightmost string of *consecutive* sterile positions – in this case positions 0 through 3. Adding to the current count anything less than or equal to 1001 – the 1's-complement of the current-count bits that fall inside the LS sterile string – is obviously not enough, since any such addend would leave a rightmost-1 in the LS sterile area after the addition; thus, we have to add at least 1 more than that 1's-complement. Now, we distinguish two cases, according to the position of the

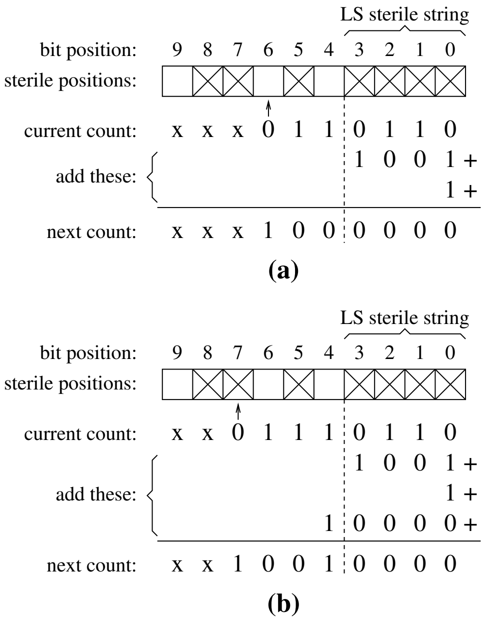

Figure 4.5: Incrementing the cycle counter so as to jump over sterile bit positions

rightmost zero of the current count *beyond* the LS sterile string. In case **(a)**, that zero is at a non-sterile bit-position – position 6 in our example. In this case, the two addends above are enough, since they will produce a next count with a rightmost-1 at that non-sterile position 6. Expressed in equivalent terms, in case (a) the next count can be generated by clearing all the bits of the counter that correspond to the LS sterile string and adding one to the first bit-position of the counter beyond that string. In case **(b)**, the rightmost zero of the current count beyond the the LS sterile string is at a sterile bit-position – position 7 in our example. In this case, the two previous addends are *not* enough, since they would produce a next count with a rightmost-1 at that sterile position 7. Thus, we have to add more; anything less than 10000 would *not* be enough, since it would leave a rightmost-1 in the sterile string, while 10000 *is* enough, giving a rightmost-1 at the first non-sterile position. Expressed in equivalent terms, in case (b) the next count can be generated by clearing all the bits of the counter that correspond to the LS sterile string and adding *two* at the first bit-position of the counter beyond that string. Notice that this latter case (b) also covers the "wrapping around" of the counter from N (all 1's) directly to 1, rather than going through 0 (see the transition from 15 to 1 in figure 4.3). For this, it is enough to consider that the counter value has an imaginary "0" bit beyond its most-significant bit, and that that imaginary bit-position is always sterile.

5. PERFORMANCE ANALYSIS

In this section we will provide an analysis of the performance of the switch. It is our purpose to keep this analysis as simple and realistic as possible, since exact models of the input traffic do not currently exist, and the exact effects of the flow control mechanisms are extremely hard to analyze. Our analysis shows that, in the range of the operating parameters of our system, the round-robin service discipline together with the use of priorities exhibits an extremely satisfactory behavior in terms of buffer occupancy for the real-time traffic, and regularity in serving the lower priority traffic. The reason for this nice behavior is the fact that the large capacity of the switch is shared among many VCs, each carrying a small amount of traffic by comparison to the total capacity of the switch. In the rest of this section, we will provide an intuitive justification of the above statement. We should note that there are a number of papers dealing with the exact analysis or the asymptotics of similar systems with different service policies; see [Weis86], [Mitr88]. To our knowledge, there is no existing analysis of the head-of-the-line service discipline which corresponds to the round-robin non-preemptive policy of our switch. The closest results are in [GrWh91] and concern the approximate analysis of head-of-the-line processor sharing. Some simulation results can be found in [DeKS89]. A general introduction to the analysis of polling systems is presented in [LeSi90].

We assume that there are two types of traffic going through the switch, the real-time traffic and the data traffic. Furthermore, we assume that the switch operates in a mode according to which the VCs of the real-time traffic are not congested and receive high-priority, whereas the VCs carrying data are congested and receive lower priority. In a typical setting a proportion of the data VCs will not be congested; as we will show, we can extend the analysis for this case as well. Consider first the behavior of the switch for the real-time traffic by looking at one among its outgoing links. Since this traffic has higher priority, it uses the server of the link at its maximum capacity. As the number of the VCs carrying real-time traffic grows, under the assumptions that the traffic of each VC remains individually small, the behavior of the total input traffic becomes more deterministic in the sense that its fluctuations scaled to the size of its mean tend to zero, and resembles to a fluid. A consequence of this is that if we consider a sequence of systems in which the number of the VCs grows while the utilization of the switch (traffic intensity) remains constant, the event that at a given time the input traffic rate exceeds the switch capacity is a large deviation (see [Buck90]), and hence its probability decreases exponentially fast in the number of VCs. This implies that a queue (more than one cells) will be rarely formed, and when formed it will be small. Although this is an asymptotic result, it becomes accurate by the large number of VCs that our system supports. The other observation concerns the VCs carrying data traffic. The intervals of time during which these lower priority VCs are being served as well as the time between these intervals have a mean and variance which is small and independent on the number of the data VCs. By the round-robin nature of the service policy, the fact that these VCs are congested and that there is a large number of them, it follows that the time between successive visit the same VC will be almost deterministic. This is extremely important for effectively implementing our flow-control strategies for the data traffic. In what follows we provide some analysis to further justify these estimates. We will keep the analysis simple and approximate with the goal to investigate the order of magnitude of the crucial design parameters such as the buffer occupancy for the real-time VCs and the variance between successive services for the data VCs. A more accurate analysis will be provided in [CRW].

We first consider the case of the real-time traffic going through some output link. Since it receives priority over the data traffic, we can analyze its performance by assuming that there is no data traffic in the system. Of course by doing this approximation we neglect the effect of the data cells in service when a higher priority cell arrives (the service discipline is not preemptive). One can easily show that this effect is small and does not significantly change our results. Let N denote the number of VCs (real-time traffic), and assume that each VC has the same input rate λ_{vc} cells/second, and that the input processes are Poisson and independent. We also let C denote the capacity of the server, $\lambda = N\lambda_{vc}$ be the total input rate of the system, and $\rho = \lambda/C$ denote its utilization. Then by standard results of the M/M/1 queue we have that the expected number of real-time traffic cells will be $\rho/1-\rho$, and since usually the switch will operate in light traffic with respect to real-time traffic (λ will not be close to C because of the need for spare bandwidth for the data traffic), it follows that this number will be small. For example, if $\rho = .5$, it follows that half of the time there will be no real-time traffic cells in the system, that this number will be on the average 1 and have variance equal to 2. Hence there will rarely be more than 6 such cells in the system, and these cells will be randomly distributed in the N VCs, which makes the probability of a single queue having length bigger than 1 to be extremely small (remember that the shared buffer pool of the current chip design has 64 buffers). Of course, this heavily depends on the assumption that the input processes are Poisson. In the case of bursty inputs, the above results must be multiplied by the expected burst size. We believe that the bursts in our system will be very small. The reason is that if we look at an individual VC, its traffic rate will be much smaller than the switch capacity. By the round-robin nature of the service discipline, any initial traffic burst entering the network in some VC will be "broken down" by the first link server and mixed with the other cell streams leaving that switch.

It is tempting to compute the rate at which a real-time VC gets served. Intuitively, since most of the time most of the other VCs will have no packet to send, it follows that the average capacity offered to a VC when it is busy must be almost C, with small variance. This is extremely important since it will induce almost deterministic delays to the real-time cells while they travel through the network. We can approximate the above average capacity C_{vc} as follows. We assume that all VCs carry traffic with similar characteristics, and we approximate each VC by an M/M/1 queue with input rate λ_{vc} and service rate C_{vc}. If we further assume that the above queues are independent it follows that the probability for a queue to be empty is $(1 - \rho_{vc}) = (1 - \lambda_{vc}/C_{vc})$. Now, we consider one among the N VCs at an instant at which its queue is non-empty. If there are n VCs with empty queues, this VC will be served at a rate of $C/(N-n)$, and the probability of n out of the $N-1$ remaining queues to be empty is equal to $\binom{N-1}{n}(1-\rho_{vc})^n \, \rho_{vc}^{N-n-1}$. From this it follows that

$$C_{vc} = \sum_{i=0}^{i=N-1} \frac{C}{N-n} \binom{N-1}{n} (1-\rho_{vc})^n \, \rho_{vc}^{N-n-1},$$

and since $C_{vc}/\lambda_{vc} = 1/\rho_{vc}$ and $C/\lambda_{vc} = N/\rho$, by solving the above fixed-point equation for ρ_{vc} we get $\rho_{vc} = 1-(1-\rho)^{1/N}$, which remains small even for modest values of N. For $N = 100$, and $\rho = .5$ we get that the corresponding C_{vc} is almost equal to C.

We continue now our simple analysis by investigating the performance of the link with respect to the data traffic. Let N denote this time the number of the VCs carrying the data traffic. We assume that all of them run in a congested mode which implies that their queues can be considered to be always non-empty. We will analyze the service process of these VCs, i.e., we will investigate how the statistics of the time between successive visits of the server to the same VC depends on N. We will argue that the mean value of this time increases linearly with N (as expected), whereas its standard deviation increases as $N^{1/2}$. This makes the service process of a data VC to become almost deterministic for large Ns, which will be the case in our design due to the large number of possible VCs. From the discussion in the previous sections it follows that this enables us to accurately select the size of the buffer for the data VCs. The following analysis is kept at an intuitive level for reasons of simplicity. It can be formalized if needed.

Consider the server of an output link. Since data traffic has lower priority, the server will visit the data VCs at its spare time, when no real-time traffic cells are in the system, and it will be serving these VCs until the first such cell arrives. Let T_{RT} denote a random time interval during which the server serves the real-time VCs, and T_D denote a random interval during which the data VCs are continuously served. Then, as time progresses, the service process of the link is described by an infinite interleaved sequence of intervals of the above types, and we can assume that the random variables corresponding to these intervals are mutually independent and i.i.d.. Now if we assume Poisson arrivals for the real-time traffic, then the average number of cells of the data traffic served during T_D is equal to $1/(1 - e^{-\lambda/C})$, which is a relatively small number (for $\lambda/C = .5$, this equals 2.5). Since its standard deviation is of the same order of magnitude, it follows that for large enough N, the same VC can not be served twice during the same T_D interval, with probability almost one (because the server must serve $N - 1$ other cells before returning to the same VC). Consider now a particular VC, and assume that it is being served at time 0. Let $D(t)$ be the number of data cells served from all data VCs between time 0 and time t. For t sufficiently large, since $D(t)$ is the sum of a large number of independent random variables having finite mean and variance (each represents the number of the data cells served during a $T_D + T_{RT}$ cycle), we will have that most probably $D(t) = (C - \lambda)t + \alpha(t)$, where $\alpha(t) = O(t^{1/2})$. Note that $(C - \lambda)t$ is the mean number of data cells served during t for large t. One can easily see that this implies that the time τ at which $D(t) = N$, for large N, will be $\tau = N/(C - \lambda) + \beta(N)$, where $\beta(N) = O(N^{1/2})$. This implies that the standard deviation of τ grows as $N^{1/2}$, while its mean is almost $N/(C - \lambda)$ for large N.

One can extend this analysis in the case of the data VCs not all operating in a congested mode. We expect the same results to hold with N being the average number of data VCs with non-empty queues in the steady-state. In this case N will be less than the total number of data VCs and will depend on the proportion of the congested versus the non-congested ones. A rigorous analysis of this case is beyond the scope of this paper.

6. CHIP FLOOR-PLAN AND CIRCUIT PERFORMANCE

We have laid out and simulated the parts of the chip which are critical in determining its size and speed, in order to prove the feasibility of our architecture; we are currently laying out the rest of the chip. Our layout was done for a full-custom 1 µm CMOS technology with double metal and single polysilicon layers, using the MOSIS Scalable-CMOS design rules (λ= 0.5µm); we used the *Magic* layout editor. The size-critical parts of the chip are the five blocks shown in figure 1.1; buffer RAM, I/O buffers, routing tables, and scanning memories. Table 6.1 lists the sizes of these blocks, in µm per bit-cell, total number of bits, and total area.

Table 6.1: Sizes of the critical blocks

Block	Bit Size (in µm) w	h	Total Bits	Tot. Area (in mm^2)
Buffer RAM	10	10.7	244,224	27.0
Input Buffer & Cut-Thru	10	138	1,696	2.3
Output Buffers	10	90	1,696	1.5
Routing Tables	21	17	10,240	3.7
Scanning Mem.	21	23	31,744	17.5

Using these sizes, plus the estimated sizes of the decoders, priority encoders, wiring, etc., we arranged the floor-plan of figure 6.1. As expected, the buffer memory and the scanning memories occupy the majority of the chip – 55 % of its ‘‘real estate’’. The routing tables and the I/O buffers occupy another 10 percent. A remaining large area (almost 10 mm^2) is reserved for the scanning finite-state machines and for control PLAs of the switch.

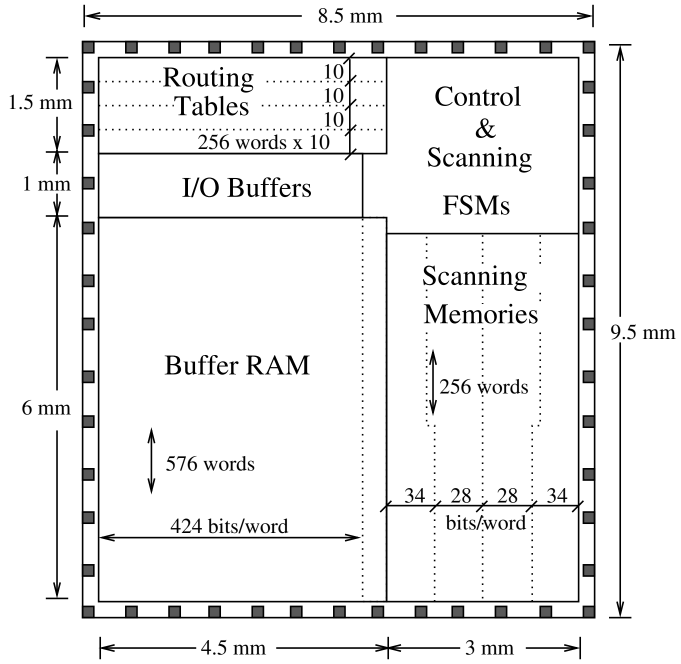

Figure 6.1: Floor-plan of the switch chip (1 µm CMOS technology)

To verify the correctness of our logic design we performed switch-level, gate-level, and behavioral simulations with *Esim* and *Verilog*. Among others, these were used to verify the cycle counter of figure 4.5, and the algorithm according to which the scanning memory is searched by its FSM. To estimate the speed of the switch, we performed circuit-level simulations with *Spice-2g6* †.

The buffer memory is a dynamic RAM, made using the 3-transistor cell. We did not use the 6-transistor static cell or the 4-transistor dynamic cell because they are significantly larger; we did not use the 1-transistor dynamic cell due to unavailability of the appropriate DRAM fabrication technology and expertise. We refresh the dynamic RAM by reserving two slots in every cell time for that purpose. The simulation of the buffer RAM indicated that a cycle-time below 50 ns can be easily achieved – 10 ns for evaluating the precharged address-decoder in parallel with precharging the bit lines, and 27 ns for discharging the 2.5 pF bit-line till a sensing inverter switches.

Figure 6.2: Details of an input-buffer and associated cut-through

Figure 6.2 shows an input buffer in more detail. The 4-bit incoming link is converted to an internal 8-bit bus that operates at half the link rate, that is it carries one byte per clock cycle (20 ns). During the 53 clock cycles of one ATM cell time, the contents of this bus are loaded into 53 latches; the load-enable signals come from a 53-bit shift register. (This implementation is less expensive than a 424-bit shift-by-8 register). On the 54th clock cycle, the contents of all these latches are copied into the lower 53 latches (double-buffering). Sometime during the subsequent 53 cycles, the contents of these lower latches must be written into the buffer RAM. The right-most part of Figure 6.2 illustrates the cut-through data-path, which works as follows. Normally, each outgoing link is fed from its corresponding output buffer. If the outgoing link for the current incoming cell becomes available while the bytes of this cell are being loaded into the upper latches, then the cut-through bus is used to start forwarding these bytes to the outgoing link before

---

† Some of the parameters used in our simulations are: VT=0.8 (NMOS) and -0.8 (PMOS), TOX=200×10^{-10}, LD=0.07×10^{-6} (N) and 0.1×10^{-6} (P), UO=650 (N) and 220 (P), GGSO=CGDO=1.2×10^{-10} (N) and 3.44×10^{-10} (P), CJ=1.89×10^{-4} (N) and 2.44×10^{-4} (P), CJSW=0.6×10^{-11} (N) and 2.31×10^{-10} (P)

the entire packet is received. The driving of the cut-through bus is controlled by another 53-bit shift register. The enable-bit in the cut-through shift register follows behind the enable-bit in the load shift register. The forwarding of the tail of the current cell via cut-through will continue when the head of the next cell has started entering into the left latches, but obviously no information will be lost. Notice that the buffer memory bus (i.e. the RAM bit-lines) can be used for direct cell transfers from input buffers to output buffers, without going through the buffer RAM itself. The simulation of the input buffer proved that it can operate with a 20 ns cycle time. For precharging the 2 pF input and cut-through bus lines, 7 ns are enough. A logic-zero value can discharge the input line and get latched within 8 ns. A logic-one value in the input latch can drive the cut-through bus low in 9 ns.

During the time that it takes for one cell to be transmitted through an outgoing link – approximately 1 μs – the scanning memory has to be searched, by a finite-state machine that repeatedly accesses it, for determining the next VC to be served. There are two distinct types of accesses to that memory: *(i)* a *match* operation is performed, indicating whether one or more VCs match, and *(ii)* the first one of the matching VCs is selected by the priority encoder, and its address is output. All but the last accesses by the FSM are of type *(i)*, and are thus quite fast. A single type-*(ii)* access is performed at the end, as late as possible, when the central controller notifies this FSM that a decision must be made right away because this link's read operation out of the buffer RAM is imminent. Any higher-priority cell that may happen to have just come into the switch before the type-*(ii)* access will be selected during this last step, and will cut-through.

Figure 6.3: The scanning memory priotrity enforcer

The simulation showed that type-*(i)* accesses can be repeated at a 20 ns cycle time. Type-*(ii)* accesses, involving the priority encoder, are more challenging. The heart of the priority encoder is the chain that disables ("kills") the next *match* flags beyond the first one (i.e. the priority enforcer). If this circuit were constructed using a 256-bit ripple structure, it would be too slow; instead, we use a look-ahead circuit. Figure 6.3 shows the main idea of the two-level Manchester chain that we use. It operates during the clock phase after the *Match* lines of the scanning memory have settled. Each pass transistor of the first-level chain corresponds to a match line of the scanning memory, while each pass transistor of the second-level chain corresponds to a group of sixteen match lines. Memory words that did not match turn-on their pass-transistor, thus allowing the *kill* signal to be cleared; however, the topmost line that matched disables any further discharging of the chain, thus leaving an asserted *kill* signal for all the words below it. The 16-bit chains are broken every 4 bits by inverters which accelerate the discharging process. The worst-case delay is when only the bottom-most word matched, thus requiring 15 level-two bits to discharge, followed by 15 level-one bits discharging until the *kill* signal is cleared. The simulation indicated that the worst case delay of this priority enforcer is below 24 ns, which is a comfortably short delay.

7. CONCLUSIONS

We presented and discussed the architecture of an ATM switch chip that uses unconventional and novel buffer management and cell scheduling techniques, and we demostrated that a number of sophisticated scheduling ideas can indeed be efficiently implemented with available resources in a single-chip switch. The chip is designed as a general-purpose building block for B-ISDNs: it can be connected to form larger switches of arbitrary topologies, and it provides the network manager with powerful and versatile tuning mechanisms for serving the varying needs of differented classes of traffic. We showed how to efficiently organize the buffer memory into a shared buffer pool and a set of dedicated buffers, how to use the dedicated buffers and the chip-to-chip flow-control in order to fully utilize the link throughput and to guard against misbehaving VCs, and how to implement in hardware a weighted round-robin scheduling algorithm for the ATM cells, which guarantees fair allocation of the entire spare link capacity to the low-priority traffic, even under congestion. The realization of such switch chips is feasible with the current CMOS technology, for chip sizes of four to ten links, with link throughput of half to one Giga-bit per second, and for about one thousand open VCs per switch; as technology progresses, the scalability of the architecture will enable the integration of larger and faster switches on single chips. Our work is currently continuing with the layout of the rest of the chip, so that it can then be fabricated, and with organizing simulation experiments in order to validate our initial performance analysis. Further work is needed in several areas, especially in designing the interface chips, and in studying and defining how the network management should take advantage of the numerous mechanisms that this chip provides.

ACKNOWLEDGEMENTS

We would like to thank M. Reiman, A. Weiss, and J. Walrand for many helpful discussions concerning the analysis of the performance.

REFERENCES

[Buck90] J. Bucklew: ‘‘Large Deviation Techniques in Decision, Simulation, and Estimation’’, Wiley, 1990.

[CRW] C. Courcoubetis, M. Reiman, A. Weiss: ‘‘An Asymptotic Analysis of Round-Robin Head-of-the-Line Disciplines with Priorities’’, in preparation.

[DeKS89] A. Demers, S. Keshav, S. Shenker: ‘‘Analysis and Simulation of a Fair Queueing Algorithm’’, Proceedings of the ACM SIGCOMM ’89 Symposium, ACM Comp. Communic. Review, Vol. 19-4, Sept. 1989, pp. 1-12.

[Fuji83] R. Fujimoto: ‘‘VLSI Communication Components for Multicomputer Networks’’, Doctoral Dissertation, Dept. of EECS, University of California, Berkeley, CA 94720, August 1983.

[GrWh91] A. Greenberg, W. Whitt: Private communication, 1991.

[Kate87] M. Katevenis: ‘‘Fast Switching and Fair Control of Congested Flow in Broad-Band Networks’’, IEEE Journal on Selected Areas in Communications, Vol. 5-8, pp. 1315-1326, Oct. 1987.

[KeKl79] P. Kermani, L. Kleinrock: ‘‘Virtual Cut Through: A New Computer Communication Switching Technique’’, Computer Networks, vol. 3(4), Sept. 1979, pp. 267-286.

[LaTG88] A. Lazar, A. Temle, R. Giton: ‘‘MAGNET II: A Metropolitan Area Network Based on Asynchronous Time Sharing’’, Proceedings of the IEEE International Conference on Communications, Boston, MA, June 1988.

[LeSi90] H. Levy, M. Sidi: ‘‘Polling Systems: Applications, Modeling, and Optimization’’, IEEE Transactions on Communications, vol. 38, no. 10, October 1990, pp. 1750-1760.

[LPTZ85] A. Lazar, A. Patir, Takahashi, E. Zarki: ‘‘MAGNET: Columbia’s Integrated Network Testbed’’, IEEE Journal on Selected Areas in Communications, Vol. 3-6, 1985, pp. 859-871.

[Mitr88] D. Mitra: ‘‘Stochastic Theory of a Fluid Model of Producers and Consumers Coupled by a Buffer’’, *Adv. Appl. Prob.,* Vol. 20 (1988), pp. 646-676.

[Mess90] D. Messerschmitt: ‘‘Synchronization in Digital System Design’’, IEEE Journal on Selected Areas in Communications, vol. 8, no. 8, October 1990, pp. 1404-1419.

[Morg89] S. Morgan: ‘‘Queueing Disciplines and Passive Congestion Control in Byte Stream Networks’’, IEEE INFOCOM’89 Proceedings, 1989, pp. 711-720.

[Nagl84] J. Nagle: ‘‘Congestion Control in IP/TCP Internetworks’’, RFC 896, January 1984.

[Nagl87] J. Nagle: ‘‘On Packet Switches with Infinite Storage’’, IEEE Tr. on Communications, Vol. 35-4, Apr. 1987, pp. 435-438.

[OSKY91] Y. Okajima, Y. Sato, K. Kurosaki, S. Yamada: ‘‘A 7ns 4Mb BiCMOS SRAM with a Parallel Testing Circuit’’, Proceedings of the IEEE International Solid-State Circuits Conference (38th ISSCC), February 1991.

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

[Tane81] A. Tanenbaum: "Computer Networks", Prentice-Hall Inc., 1981.

[Toba90] F. Tobagi: "Fast Packet Switch Architectures for Broadband Integrated Services Digital Networks", Proceedings of the IEEE, Vol. 78-1, Jan. 1990, pp. 133-167.

[Tyme81] L. Tymes: "Routing and Flow Control in TYMNET", IEEE Transactions on Communications, Vol. COM-29, No. 4, April 1981, pp. 392-398.

[Weis86] A. Weiss: "A New Technique for Analyzing Large Traffic Systems", *Adv. Appl. Prob.,* Vol. 18 (1986), pp. 506-532.

BIOGRAPHIES

MANOLIS KATEVENIS was born in Athens, Greece, in 1955. He received the Diploma degree in Electrical Engineering from the National Technical University of Athens, Greece, in 1978, and the M.Sc. and Ph.D. degrees in Electrical Engineering and Computer Science from the University of California, Berkeley, in 1980 and 1983, respectively. From January 1984 to March 1985 he was Assistant Professor of Computer Science at Stanford University, Stanford, California. Since September 1985 he has been Assistant Professor of Computer Science at the University of Crete, and a researcher at the Computer Science Institute of FORTH, in Heraklio, Crete, Greece. His interests are in Computer Systems Architecture, VLSI, Digital Communications, and CAD. During his doctoral studies (1980-83), he was the chief ‘‘microarchitect’’ and implementor of the RISC II single-chip microprocessor at U.C.Berkeley (precursor of the SUN SPARC architecture), and for this work he received the 1984 ACM doctoral dissertation award. He is a member of IEEE, ACM, the Greek Computer Scientists’ Association (EPY), and the Technical Chamber of Greece.

STEFANOS SIDIROPOULOS was born in Ioannina, Greece, in 1966. He received his B.Sc. degree in Computer Science from the University of Crete, Heraklio, Crete, Greece, in 1989. He is currently pursuing graduate studies in Computer Science, at the same University. His interests are in VLSI, Digital Communications, Computer Systems Architecture, and Operating Systems.

COSTAS COURCOUBETIS was born in Athens, Greece. He received his Diploma (1977) from the National Technical University of Athens, Greece, in Electrical & Mechanical Engineering, his M.S.(1980) and Ph.D (1982) from the University of California, Berkeley, in Electrical Engineering & Computer Science. He was a Research Assistant in ERL (Electronic Research Lab) in U.C. Berkeley, an AMTS (Associate Member of Technical Staff), Summer 1981, Bell Laboratories, an Adjunct Assistant Professor in the EE department, Columbia University, Visiting Lecturer at the EECS department, U. C. Berkeley, 1986, from 1982 until 1990 Member of the Technical Staff (MTS) in the Mathematical Sciences Research Center, Bell Laboratories, Murray Hill, NJ, and since 1989 Assistant Professor at the Computer Science department, University of Crete, Heraklion, Greece. His interests are Distributed Algorithms and Computation, Queueing Theory, Performance Analysis of Computer Systems and High-Speed Networks, and Design and Verification of Computer Communication Protocols.