Chapter 1

Introduction

Nowadays, even though the necessary bandwidth for the creation of Broadband Intergrated Services Network (BISDN) is provided by advanced fiber optic technology, this creation still remains a challenge. The major difficulty lies in the construction of switching circuitry that can cope with the bandwidth provided. The concept of Asynchronous Transfer Mode (ATM) – the switching technique selected for the realization of BISDN – is presented in chapter 1, followed by an overview of existing approaches for building ATM switches, and some notes on the organization of this thesis.

1.1 Broadband ISDN and Packet Switching

The rapid evolution of high speed telecommunication technologies has prompted the introduction of BISDN – an all-purpose digital network. It will support a wide variety of applications in an intergrated and unified fashion. There is an enormous difference in the requirements for bandwidth and quality of service between many of the applications supported by BISDN and those supported by today's digital networks. For example, High Definition Television (HDTV) will require over 100 Mbps bandwidth, with acceptable error probability up to five orders of magnitude smaller than that of today's digital telephony. In addition, a key requirement of BISDN is flexibility – the same network will be used for all services, including services whose characteristics have not yet been defined.

A key point for the realization of BISDN is the transfer mode (switching technique) that will be used. Early thoughts of researchers and study groups considered two approaches: circuit switching or Synchronous Transfer Mode (STM), and packet switching or Asynchronous Transfer Mode (ATM). Traditionally, circuit switching has been used for connection oriented applications such as telephony, and packet switching has been used for data transfers. Today it seems very difficult to achieve a multiservice (i.e. multibit rate) network by using circuit switching [Minz89]. This is because the bandwidth required by the various services depends both on the coding algorithm and the state of the art in technology. These parameters are evolving so fast that whatever the choice of the channel bit rate, it may prove inefficient in the near future. It was this inflexibility of STM, that led to the selection of ATM as the transfer mode of BISDN. In the following paragraph we will briefly describe the most important notions of the current ATM standard.

The concept of ATM, as defined by CCITT Study Group XVIII in [CCIT90], is simple and almost identical to that of the traditional packet switching. The usable bandwidth is divided into fixed-size information-bearing time slots, called cells. Each 53-byte cell consists of a 5-byte header (for routing and multiplexing purposes) and a 48-byte information field – the resemblance to the packets of traditional packet switched networks is obvious. The cells are multiplexed and routed by using Virtual Circuits (VCs). This implies that all services at the network layer are connection oriented. The switching and routing functions are performed by using the Virtual Circuit identifier (VC ID) field of the cell header. In addition, the current draft ATM standard defines the notion of Virtual Path (VP), which is an aggregation of many VCs that are being routed together in a segment of their itinerary in the network. Another important topic, addressed by [CCIT90], is the classification of various applications according to their quality of service requirements (end to end delay, error and loss rate etc.). Four classes of services have been defined for: circuit emulation, variable bitrate video, connection oriented data transfers, and emulation of connectionless services. This definition implies that a priority-based multiplexing mechanism must be implemented in the network layer of BISDN: it must differentiate among cells in different VCs according to information provided during connection establishment, and give preferential treatment to higher priority cells.

Some of the above functional requirements have more or less been implemented in traditional packet switching networks, so one may argue that the implementation of BISDN presents no difficulty with the available technology. However the main difference between traditional networks and BISDN is that in the latter the offered bandwidth is at least two orders of magnitude larger. Thus, it is evident that there is almost no chance of implementing the network layer functions in software, as done in traditional packet switching networks [Toba90] – instead all of them must be implemented in hardware. So, the main challenge is to built fast packet switches being able to handle rates in the order of 100,000 to 1,000,000 packets† per second per input line, and implementing the described functionality in hardware. Additionally, these switches must be able to cope with the peculiarities present in the traffic characteristics of each service. The next section

† In this work the terms packet and cell will be used indistinguishably

describes the functionality that an ATM switch must have and briefly reviews the existing architectures that have been considered for implementing ATM switches.

1.2 Overview of Fast Packet Switch Architectures

An $N \times N$ packet switch is a box with N inputs and N outputs which routes packets arriving on its inputs on the appropriate outputs. Usually all input and output lines have the same transmission capacity and all packets are of the same size. There is no constraint on the destination port of the packets arriving at a given time on the inputs of the switch, so more than one packets may be required to be forwarded to the same output port simultaneously. Such an event is referred as an *output conflict*. Due to output conflicts, buffering of packets within the switch must be provided.

Besides the basic routing and buffering functions performed by a switch some other functions may be required. The first is *multicast* operation, required by some applications such as video-conferencing and commercial TV-channel distribution. A switch capable of multicasting must be able to replicate a packet at several of its output ports on demand. Second there is the *priority* function. As explained in the previous section, a switch implementing a priority function must be able to differentiate high priority packets and prefer to serve them (i.e. forward them to their destination port) before other packets of lower priority. Another function that may substantially improve the overall network performance is *virtual cut-through*. The term has been originally introduced in [KeKl79]. It represents the ability of the switch to start the transmission of a packet over an output port, as soon as its destination port has been determined, without having to wait for the complete packet to arrive into the switch. The last function is *flow control*. The need for flow control is imposed by the fact that the buffer space within a switch is limited. As a result, the bursty nature of traffic may lead to unacceptable packet loss rates, when a switch in the network transmits packets without having any information on the buffer occupancy of the switch that receives them.

Several architectures for implementing fast packet switches have emerged in recent years. In [Toba90] they are classified into three main categories; namely: *shared memory*, *shared medium,* and *space division.* The rest of this section is devoted to a brief description of these three types.

1.2.1 Shared Memory Switches

An abstract model of the shared memory switch architecture is depicted in figure 1.1. Its basic component is a ‘‘virtually’’ dual-ported memory shared by all input and output lines. Packets arriving on input lines are stored to the common memory. Internally to the memory, the cells are organized into separate queues, one for each output line. Packets are simultaneously retrieved from the output queues and transmitted over the output lines. The two main constraints of the

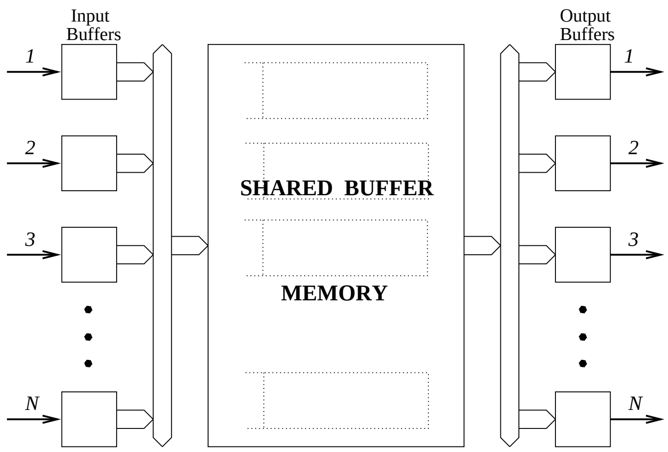

Figure 1.1: Basic structure of a shared memory switch architecture

shared memory architecture as identified in [Toba90] are that: *(i)* the time consumed by the routing decision must be sufficiently small to keep up with the flow of incoming cells, and *(ii)* the memory bandwidth should be sufficiently large to simultaneously accommodate all input and output traffic. If N is the number of ports and B the port bandwidth, the shared memory bandwidth must $2 \times N \times B$. However, as can be seen in [DeCS88] and [KEOK89] these two constraints do not impose any insurmountable design obstacles. Additionally, shared memory switches present a great conceptual resemblance to traditional packet switches. Thus, the expertise gained by the construction of these networks may be transferred to BISD networks, if elegant ways can be found to implement old algorithms, while conforming with the new high speed requirements.

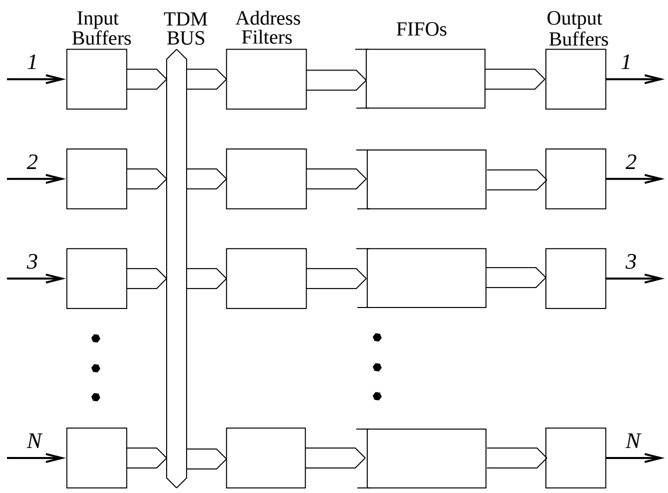

Figure 1.2: Basic structure of a shared medium switch architecture

1.2.2 Shared Medium Switches

The shared medium approach presents great similarity to Time Division Multiplexing (TDM) switches : All packets arriving on the input ports are synchronously multiplexed onto a common bus (figure 1.2). Each output line is connected to the bus via an address filter and an output FIFO. The address filter of a particular output port, based on the VC ID of a packet, determines whether or not the packet on the bus has to be written into the port's FIFO. The main constraint of the shared bus architecture is that the common bus must operate at $N \times B$ speed and each output FIFO at $(N+1) \times B$ speed. Some shared bus architectures that can cope with this constraint have been presented [SNSTI90]. Nevertheless, an inherent disadvantage of the shared medium architecture is the partitioning of the available storage to N distinct portions. Thus, in cases of ''hot spot'' traffic [PfNo85] much of the available storage will be wasted.

1.2.3 Space Division Switches

In space division switches, multiple concurrent paths are established from the inputs to the outputs. Thus there is no need for any buffering component in the switch, to have bandwidth more than $2 \times B$. The major problem of space division based fabrics is *internal blocking*, i.e. the state in which it is impossible for the internal fabric to set up concurrently all the required paths. The space division switches are classified into three main categories: *(i) crossbar fabrics*, *(ii) banyan based fabrics*, and *(iii) fabrics with N^2 disjoint paths.*

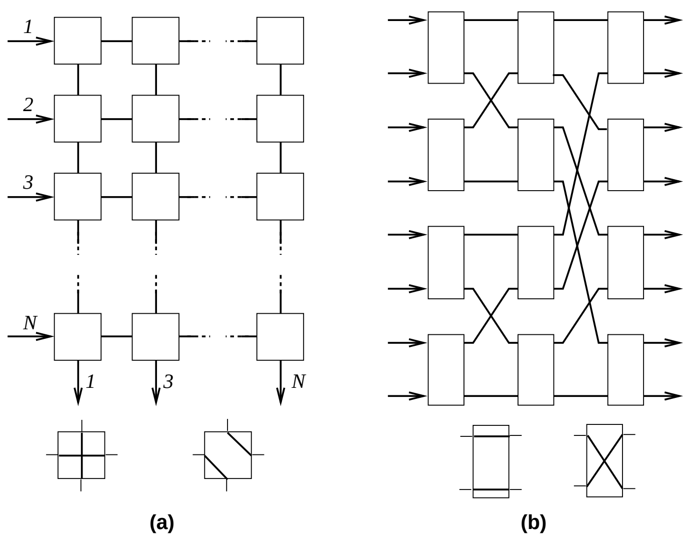

Figure 1.3: Crossbar **(a)**, *and Banyan* **(b)** *switches and switching elements*

The *crossbar fabric* consists of a square array of N^2 crosspoint switches each one corresponding to one input output pair – see figure 1.3(a). If there is no output conflict all incoming packets can reach their destinations. In case of an output conflict only one packet can be routed to the desired output; those remaining either have to be dropped or buffered somewhere. Several designs utilizing the crossbar architecture have been presented. Some of them have their buffers placed in the inputs [FrTa88], and others in the crosspoints of the switch. The main drawback of crossbar fabrics remains their limited scalability, due to the N^2 cost of the crossbar circuit.

The *banyan switching fabrics* are based on a class of multistage interconnection networks that consists of $\log_2 N$ stages, each one comprising $N/2$ binary switching elements. The interconnection lines between the elements are placed in such a way as to allow the formation of a unique path from each input to each output – see figure 1.3(b). The establishment of such a path is accomplished by using a self routing procedure: the switching element reached by a packet at $stage_i$, is set to the appropriate cross or bar state according to the i_{th} bit of the packet's destination number. The major problem of banyan based switches is *internal blocking*. It occurs whenever two packets arrive simultaneously at a switching element and request the same output link. The existence of such conflicts – which may occur even in the absence of output conflict – introduces severe performance limitations. Several alternate architectures have been proposed to circumvent this performance degradation. Examples are buffered banyan switches [Turn88], sorting banyan switches (usually referred to as Batcher/banyan) [GiLS90], and multiple banyan switches (such as Tandem Banyan or parallel Banyan investigated in [KwTo91] and [ChAT91]).

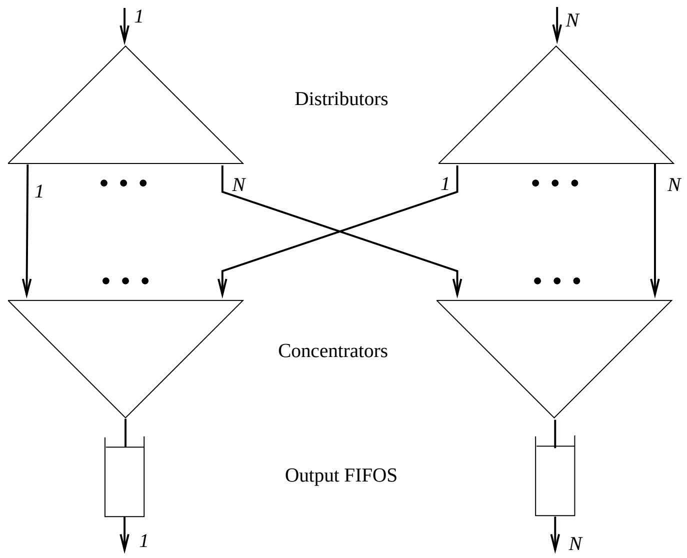

Figure 1.4: An abstract model of the N^2 disjoint paths switch architecture

The final class of space division switches is the *fabrics with N^2 disjoint paths.* This class of switches avoid blocking by providing as many concurrent paths as needed, to handle all possible packet arrival patterns. They distribute the packets arriving at each input port to N bins, one for each output, and the packets destined to each output are multiplexed into a single queue (see figure 1.4). The main drawback here is the limited scalability, as with crossbar switches. Several approaches have been presented to overcome this drawback. Some of them (e.g. [YeHA87]) are based on the observation that under some typical traffic patterns, the number of packets destined to a given output is small compared to N, and thus there is a value L such that if every output port receives up to L packets, the desired cell loss rate is achieved. Others (e.g. [WaTo91]) based on the same observation, try to limit the number of disjoint paths, by organizing the switch as a binary tree, in which each branch constitutes a group of paths that is shared by all the packets destined to a subset of output ports. Although these revised versions manage to limit the cost of this type of switching fabric, their complexity still remains high. Additionally they suffer from the inability to guarantee zero packet loss rate for a specific class of service, unless they incorporate additional input buffering and a flow control mechanism (which will further increase their complexity).

1.3 Overview of this work

As can be inferred by the presentation of the previous section, researchers throughout the world have studied extensively the issue of fast packet switching for ATM. However Coudrese, et *al.*, in [CoST88], note that there is a lack of efficient buffer and bandwidth management techniques in Broadband Packet Communications. In addition, they note that *''computationally efficient algorithms to decide how to combine traffic flows are needed to allow rapid route selection in an operational switch''*. Furthermore, the concept of quality of service does not explicitly appear in most of the existing architectures, with the exception of the ring local area network switches of Columbia's MAGNET (see [LaTG90a] and [LaTG90b]). The present work – an extension of [Kate87] and [KaSC91] – attempts to focus exactly on these issues. It defines a switch architecture, that provides the network manager with powerful mechanisms for buffer and bandwidth management. In addition, it is suitable for single chip VLSI implementation; the resulting chip can be used as a building block for constructing distributed or concentrated switches.

Chapter 2 of this thesis discusses our view of the overall Broadband Network architecture as the context in which the studied switch chip will operate. Chapter 3 deals with the architecture of the switch chip; it mainly discusses the major architectural decisions and presents some alternatives. Chapter 4 discusses the implementation issues and describes in detail the circuit design, layout, and simulations performed, in order to prove that the proposed architecture is feasible, in the contemporary CMOS VLSI technology. Finally, Chapter 5 provides some conclusions and briefly presents the future extensions of the present work

Chapter 2

An Approach to the Broadband Network Architecture

Given the flexibility provided by packet switches, we still need efficient ways to fully utilize the resources of a network. That is, the design of a switch has to focus not only on efficient routing and minimal packet loss due to output conflicts, but also on the achievement of maximum utilization of network resources. The methodology that leads to better bandwidth utilization is presented in § 2.1, followed by a preliminary investigation of possible allocation policies for various types of traffic (§ 2.2). The chapter concludes by presenting how a broadband network can be constructed by connecting together many copies of a small switch component that implements the proposed techniques (§ 2.3).

2.1 A Possible Use of Congestion in Broadband Networks

The term ‘‘congestion’’ in conventional networks is used to signify the situation where *‘‘performance degrades due to too much offered load’’* [Tane81, p. 216]. However in [Kate87], it was observed that offering to the network more traffic than its carrying capacity, is a prerequisite for achieving continuous and full utilization of all its bandwidth. That proposal has been the cornerstone of the present work, so we will briefly review it here.

It is obvious that in order to fully utilize a certain resource, the requests for using that resource must be equal to or exceed the capacity of the resource – otherwise, if there is a lack of sufficient requests, the resource would necessarily remain underutilized. But why would one choose to fully utilize the bandwidth of the links of a network rather than some other network resource? The main observation that justifies Katevenis’ approach is that buffer memory is no longer a scarce resource in Broadband Networks – instead bandwidth is a more precious resource. 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 Kilo-Byte 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 much less than the cost of e.g. the long optical fiber that they correspond to [Kate87]. Two conclusions follow.

The first conclusion is that it is better to buffer packets than to drop them and later retransmit them. For this reason a hop-by-hop flow control mechanism must be implemented throughout the net. The basis of this mechanism can be a sliding window protocol which utilizes backward transmitted ‘‘permits’’. Every switching or terminal node in the network sends a ‘‘permit’’ (token) to its upstream neighbor whenever room for more packets is available in a local buffer. This guarantees that buffers will never overflow, and packets will never have to be dropped, if they encounter a congested area somewhere in the network. On the other hand the flow of packets must be able to proceed at its maximum rate whenever there is nothing to stop it. For this reason 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, a ‘‘chain-reaction’’ phenomenon takes place. The packets of the VCs which encounter heavy traffic on the congested 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. In order to avoid such a situation a different approach to buffer allocation can be adopted: for those classes of traffic and those VCs for which it is not feasible or desirable to guarantee property *(ii)* above, *dedicated* buffer space to each VC can be provided.

This buffer preallocation policy solves the problem of undesirable interactions between VCs [Tane81] and the store and forward deadlock [MaZa90], since it makes VCs independent of each other. 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. 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, it is the network itself that 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 (hopefully, that will be the *lowest-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 if 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 of it as they need, and the remaining portion is equally distributed to all the other VCs that can use it. Round-robin scheduling was initially proposed in [BCHW72], used in TYMNET [Tyme81], and was also proposed in [Morg89] and studied in [DeKS89]. An extension that gives more power to the basic round robin cell multiplexing is to use *weighted round robin scheduling* – certain VCs are visited more than once during each scanning cycle, in proportion to a prescribed ‘‘frequency of service weight’’, and thus they receive a proportionately higher share of bandwidth.

The satisfactory function of the discussed scheme depends on whether the kind of traffic presented to the network requires minimal loss probability, and on whether it is statistically expected to try to overload the network. Not all kinds of traffic have the above two characteristics – e.g. digital telephony requires a constant bandwidth of 64 Kbps and may tolerate a reasonable rate of packet loss. Therefore, providing individual buffers to each VC in the network may not always be the most reasonable choice. The next section discusses the characteristics of various kinds of traffic, and presents appropriate buffer allocation schemes for each of them.

2.2 Quality of Service and Network Resource Allocation

How should network resources be allocated to a specific service call, to achieve better service performance? Answering this question requires knowledge of the diverse characteristics and requirements of various types of service. A classification of services according to some of their characteristics has been proposed in [CCIT90]. This classification – based on: isochronality, bandwidth stability, and connection mode (see table 2.1) – may be sufficient for determining various protocol functions, but is incomplete when one tries to answer the above question. The characteristics that must be used in such case are whether the service is: *(i)* loss sensitive – i.e. its performance decreases by packet loss, and *(ii)* delay sensitive – i.e. its performance is affected by end-to-end or cell-to-cell delay [OhON88]. In this section we will attempt to identify which resource allocation scheme can be used for the various services, when the multiplexing mechanism used is the weighted round robin method explained in previous section, in combination with a priority mechanism – i.e. the scheduling is done with respect to priority classes, and within a class the bandwidth is allocated in proportion to prescribed weights.

Table 2.1 : CCITT Service Classes

	Class 1	Class 2	Class 3	Class 4
Timing between source and destination	Related		Not Related	
Bitrate	Constant	Variable		
Connection mode	Connection Oriented			Connectionless

The definition of the first class above, relates to Quasi-STM services – using [OhON88] terminology. Digital telephony may be considered as a typical representative of this class. Although voice traffic may have a bursty nature when silence suppression is used, the most commonly used PCM or ADPCM coding results in a constant rate of 64 or 32 Kbits/s respectively. The principal requirement of voice services is bounded delay. Voice packets that are not delivered to their destination within some period have to be discarded. Although the exact length of the maximum tolerable period is subject to debate, it is generally accepted to be in the range of 100-600 ms. In addition a packet loss rate up to 5% is considered as acceptable by most quality standards. The above requirements call for high priority voice VCs in our scheme. This implies that in most switching nodes in the network, voice VCs will not be congested, because they will be serviced as soon as possible when they arrive at a node, and hence they will not need dedicated buffers or even flow control.

The second class of services has been created mainly for classifying Variable Bitrate Video (VBV). VBV requires bandwidth in the range of 2-150 Mbits/s, depending on the frame resolution. The end-to-end delay requirements for video services depend on the type of the service: for one way television applications such as transmission TV, delay is irrelevant compared to that of two way applications like video-conferencing. On the other hand video as an information carrier is complementary to sound, and must stay synchronized with it. The acceptable cell loss rate varies, depending on both the coding algorithm, and the amount of motion in the scene. Taking into account the above requirements, we may infer that a similar policy to telephony can be applied for VBV. The video traffic VCs must have high priority –in fact the same with that of their corresponding sound VCs. In addition, shared buffer space can be provided to them in most cases. The bursty nature of video traffic does not seem to impose any particular problem since, when many video sources are combined – e.g. in one virtual path – the total rate exhibits less burstiness [KaVe89]. Thus, in case the total bandwidth of the aggregated VCs is guaranteed to be smaller than the link's capacity shared buffers can be assigned to the video carrying virtual paths.

Typical examples of the third class of services are interactive communications and massive data transfers. Interactive data traffic is highly bursty and has moderate end-to-end delay requirements. In addition, although packet loss may be tolerable it is better to be avoided. These requirements show that interactive data VCs can have moderate priority and that it is better to use flow control; dedicated buffers must be assigned to them if the bandwidth they require is not guaranteed to be less than the link's capacity (e.g. graphics terminals). A higher priority can be used, in case the transaction the VCs correspond to requires bounded end-to-end delay (e.g. critical military or government transactions). On the other hand, massive data transfer traffic is the perfect candidate for becoming the low priority (congested) class in our system. It is loss sensitive – a lost packet must be retransmitted – and no time critical. Hence, the data carrying VCs must use dedicated buffers and flow control, and their frequency of service weight must be set to a value according to the agreed cost of call.

Class 4 corresponds to connectionless data services: the type of services similar to these of user datagrams in conventional packet switched networks. Since BISDN is by definition a connection oriented network, this type of service will be emulated by the means of a connectionless server. A user that requires such a service will be connected to the server, which subsequently will route the cells to their destination via a semi-permanent virtual path [CCIT90]. Connectionless services are expected to generate highly bursty traffic with no guaranteed maximum bandwidth. Therefore the most reasonable choice would be to treat the connectionless service VCs similar to massive data transfer VCs, i.e. assign them low priority, provide them dedicated buffers, and use flow control to increase their reliability.

Although no special class has been defined for supporting information transfer for network management and control, we cannot ignore the stringent requirements of this type of traffic. Network management traffic requires minimum delay – e.g. cells carrying flow control information must reach their destination as soon as possible in order to minimize round trip delay. In addition, it is loss sensitive – e.g. VC setup packet loss is not tolerable. These requirements clearly show that network management VCs must be assigned top priority in our system and use hop-by-hop flow control to ensure that their cells will not be lost.

The above discussion did not attempt to define a set of general rules, that solve all the problems that can occur in a real integrated network. An establishment of such an overall policy is not a simple matter and is beyond the scope of this study. Nevertheless, we can easily see that adding an option for shared buffering to the dedicated buffering scheme described in the previous section, leads to better buffer utilization while it still maintains maximum bandwidth utilization. Additionally, it can be seen that four levels of priority are enough for supporting: *(i)* network management traffic, *(ii)* critical interactive communications (e.g. police, ambulance, government or military communication channels), *(iii)* real time voice, video, and data, and *(iv)* massive data transfers. This priority-based mechanism along with the weighted round robin scheduling, gives the network manager a powerful tool for allocating the available bandwidth fairly, while in combination with some other policies (e.g. call admission, bandwidth quota etc.) contributes to satisfying the quality of service requirements of the provided services.

2.3 Hierarchical Organization of Broadband Networks

A large system can be constructed by replicating a small number of building block modules. This approach, which offers greater flexibility, modularity, and incremental expandability, can be adopted for the construction of the Broadband ISDN networks as well. For this purpose, it is sufficient to develop a number of building block components that plug together with each other implementing the appropriate functions.

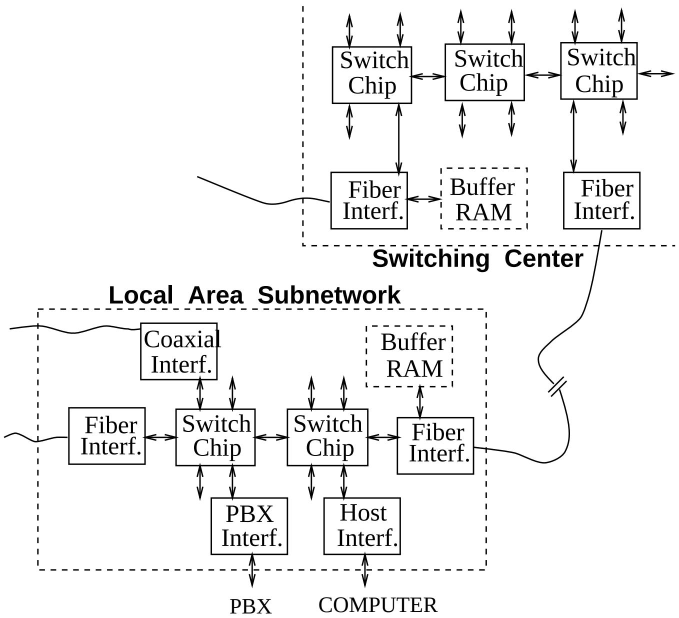

Figure 2.1: Envisioned network architecture

Figure 2.1 shows our view of the broadband network architecture. The network can have arbitrary size and topology, but is constructed using a small number of different VLSI components. It consists of parts that are separated by considerable physical distance from each other via links such as coaxial cables, optical fibers, microwave links, etc. One part of the network, that resides at one physical location, may be constructed with a few chips on a small printed circuit board (PCB), constituting a small scale subnetwork. Other parts of the network – e.g. large centralized switches – may incorporate thousands of chips on several boards. An entire such subsystem, or appropriate parts of it, will be concentrated enough for its core to operate synchronously and for its chips to communicate via bit parallel links (e.g. the lower left part of figure 2.1). Any such subsystem can be built using a few different types of chips. The switching fabric can be made of chips of a single type; various service equipment can be connected to it via special interface chips. The external links will be connected to the switching kernel by using special link interface chips for each particular type of external link. By using this approach the goals of network flexibility and incremental expandability are met in a straightforward way – new subsystems with particular configurations can be constructed easily by plugging together the appropriate building block chips on a board, and adding it to the existing system. Furthermore, a large switching systems may be made out by smaller ones, following any topology that the network designer will choose depending on the expected pattern of traffic and the tolerable amount of internal blocking.

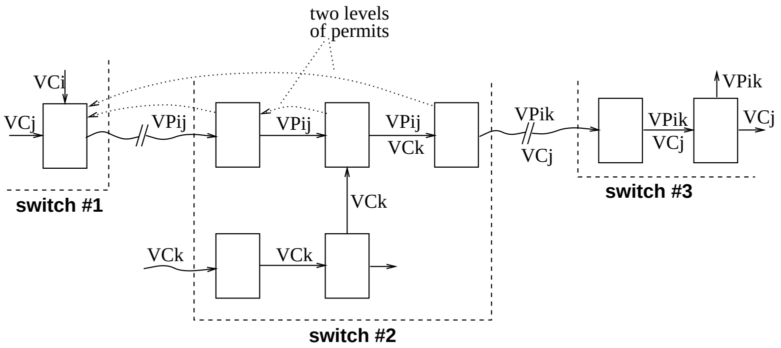

Figure 2.2: Routing and flow control of VCs and VP's

The network architecture just described has a hierarchical nature. The higher levels of this hierarchy – i.e. the large composite switches and local area subsystems at the end points of long links – will be able to serve thousands of end user VCs. As already described, these large systems will be constructed ''hierarchically'', by connecting together multiple, smaller switches, which will constitute the lower levels of the hierarchy. The traffic seen by these lower level switches will contain fewer but composite virtual circuits – ''virtual paths'' (VP's) using the terminology of CCITT – which will be derived by grouping together multiple end user VCs over a common portion of their path in the network. This is illustrated in figure 2.2 : VC_i and VC_j are merged at the fiber interface of the composite switch #1 and follow the same path through the component switches of the composite switch #2. Subsequently, at the exit point of switch #2 they are separated, and VC_i is merged with VC_k, while VC_j follows a different path through the component switches of switch #3.

The flow control policy described in § 2.1 can be applied in the case of composite VCs as well [Kate87]. Two levels of permits are needed in this case, as illustrated in figure 2.2. The lower level corresponds to the merged VP_{ij} and is transmitted from each switch chip lying on the path of the VP to its upstream neighbor. The higher level refers to the individual VCs and is transmitted from the interface chip in which the two VCs are separated, to the interface chip in which they are merged. We must mention however, that such an aggregation of VCs cannot always be applied using as only criterion whether or not two or more VCs follow the same path within a composite switch. For example, it is not practical to merge a VC carrying voice traffic with one carrying non interactive data, due to their different priority and the different buffer allocation policies that must be applied to each of them. Nevertheless, we anticipate that in a large composite switch a substantial number of end user VCs will be aggregated, thus leading to reduced buffer and routing table storage requirements in the switch chips.

In order for the bandwidth allocation to be fair, the multiplexing of multiple VCs into composite ones must be performed by using a round robin mechanism. This means that the link interface chips, besides the considerable amount of buffer storage, will also need to incorporate special hardware for implementing the round robin multiplexing. This may be a severe limitation for the single chip implementation of these building blocks, especially in cases in which they reside at the end point of a long high speed link (recall that each VC needs approximately 1 Kbyte of storage per kilometer and Gbit/sec). However, in such cases, it is acceptable to connect external RAM chips to the link interface chips without significantly increasing the overall cost of the system, since the cost of memory chips will be only a small fraction of the cost of the corresponding link. The cycle time of this external RAM modules will not impose any problem, since the data path for accessing them can be made wide enough to match the link bandwidth. The number of additional pins needed for this purpose will be reasonably small, e.g. 32 data pins will be needed for a 400 Mbits/s link interface connected to a RAM module with 40 ns cycle time.

Having defined the environment in which the switch chips will operate, it is easy to see that their required functionality allows a single chip implementation. This study discusses the architecture and implementation of such a single chip switching building block. Our current design is for a 4×4 switch with a throughput of 400 Mbits/s in each direction of its 4 links. The parallel link interface of the chip allows it to be directly connected as described above to other switch chips or to interface chips. Each link is capable of routing 256 composite VCs, 128 of which have a dedicated buffer space of one cell; the remaining 128 of all links (totally 512) share a common buffer pool of 64 cells. The stored cells are selected for transmission by using the priority class of their corresponding VC and according to a weighted round robin policy implemented in hardware. The flow control mechanism is straightforward, given the buffer space that can be occupied by each VC and the small round trip delay – every time a cell is selected for transmission a permit is send backwards indicating that there is room in the local buffer for receiving one more cell. Another feature that is essential for the performance of the composite switches composed by using this building block chip, is that the chip implements a ''virtual cut-through'' mechanism, thus dramatically reducing the latency of the non-congested VCs in a large composite switch. These features of the switch chip are described in the next chapter, together with the trade-offs that led us to choose them.

Chapter 3

The Switch Chip Architecture

In this chapter we present the hardware architecture of the building block switch chip, and discuss the principal alternatives that were considered. The presentation of the high level architecture is intermixed with micro-architectural discussions, since many of the decisions were affected by implementation issues.

We present first the overall block diagram and operation of the chip, then we discuss the organization of its storage parts (routing and buffer memories). Subsequently, we describe the weighted round robin multiplexing mechanism and the way we augmented it with priority classes. After a calculation of the round trip delay, the chapter concludes with a discussion about the initialization of the switch chips in a composite switch.

3.1 Block Diagram and Operation of the Chip

Figure 3.1 presents the overall block diagram of the current switch chip design. As mentioned earlier, this design is for a 4×4 switch chip, with a throughput of 400 Mbits/s on each one of its four links in each direction, but the architecture can be extended with a more advanced fabrication process and packaging technology up to a dozen of multi Gb/s links. The current chip design can route and multiplex 256 virtual paths† on each of its links. The format of the packets recognized by our architecture is simple: 53-byte long cells with 2-byte header (one byte for VC ID and one byte for control, as described in § 3.9). This format is not compatible with the

† from now on we will refer to the terms virtual circuit and virtual path indistinguishably, since our chip cannot signify the difference between the two of them.

standard cell format described in [CCIT90], but this incompatibility is not a problem: our system will perfectly cooperate with systems following the CCITT standard if a simple header translation is implemented in the (fiber and media) interface chips.

The main parts of the chip, illustrated in figure 3.1, are : four input buffers, four routing table memories, the centralized buffer memory, four outgoing link multiplex controllers, and four output buffers. The bytes belonging to a packet that arrives through an incoming link are successively stored into that link's input buffer. The routing table memory associated with that link is accessed when the header (i.e. the VC ID) of that cell has arrived into the switch. This access provides the destination outgoing link (VC routing), and the new VC ID for this cell (VC translation).

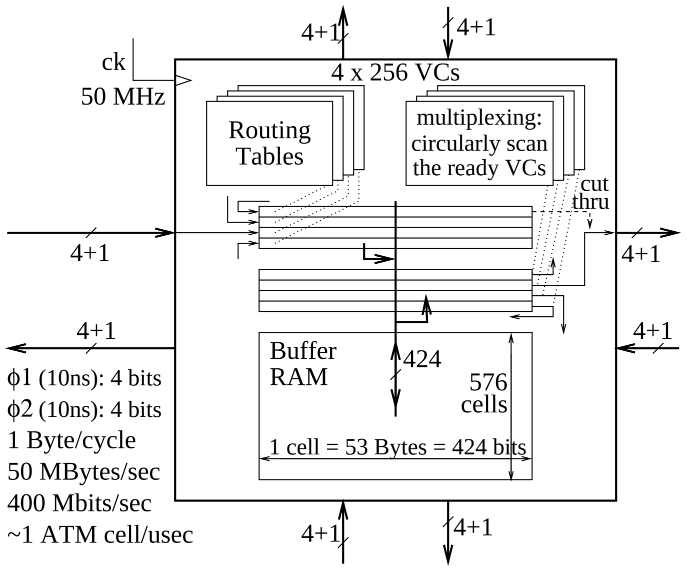

Figure 3.1: Block diagram and capacity of the switch chip

The packet will be stored into the buffer memory as soon as all of its bytes has been arrived and assembled into an entire ATM cell; all the bits of the cell are written in parallel into a row of the buffer memory. Under appropriate conditions (e.g. high priority VC, idle outgoing link), the cell will start being forwarded over its destination outgoing link, right away, without first being written into the buffer memory and then read into the output buffer. This is called ‘‘virtual cut-through’’ and is done through a special cross-bar switch circuit. The buffer memory is internally partitioned into five segments. Four of them have size 128 cells and each of their cells is dedicated to one of the 128 VCs of an outgoing link. The fifth segment is shared among the remaining 512 VCs of all the four links and has a size of 64 cells. A cell multiplexing circuit is associated with each outgoing link. It consists of a special ‘‘cycle counter’’, a finite state machine, and a scanning memory that keeps track of all the cells waiting to get forwarded over the particular outgoing link. Under the control of these circuits, a VC is selected for service (i.e. transmission), according to the weighted round robin multiplexing policy and its priority class; the cell belonging to this VC is read out of the buffer memory, placed into the appropriate output buffer, from where it gets forwarded out of the switch, one byte at a time, over the corresponding outgoing link. Whenever a cell belonging to a particular VC is selected for transmission, a permit is send to the upstream chip, indicating that there is room in the local switch buffers for receiving one more cell of this VC. This back-pressure flow control policy ensures that the cells belonging to the VCs that have assigned dedicated buffers will never be lost due to contention for buffer space.

Conceptually, the switch architecture described here resembles more to the shared memory model described in chapter 1. However, the input buffering and virtual cut-through mechanisms it implements resemble to cross-bar space division switches, while the static way that part of the memory is partioned could be an argument for classifying it to the shared medium architectures. Additionally, this architecture does not have a restriction that most of the switch architectures described in chapter 1 have: It does not require the arrival of cells in the input lines to be time-synchronized. Furthermore, its unconventional buffer management, flow control, and cell multiplexing mechanisms distinguish it from the other approaches – these features constitute a set of efficient primitives for supporting quality of service in hardware, while most of the other approaches rely on end to end software protocols for this purpose.

3.2 The Switch Chip Interface

The described architecture is independent of the pin level interface of its chip implementation. This implementation, however, is simplified by adopting a synchronous clocking discipline and a bit parallel interface between neighboring switches. In this way, no synchronizers, clock generators, or elastic buffers are needed, and the transfer delay is minimized. For the current version of the switch we have adopted this approach – a group of neighbor switch chips operate under a single external clock and communicate through bit-parallel links. We anticipate that such synchronous operation of several switches and interface chips on a board will not be impossible, provided that each chip will communicate only with close neighbors and the clock will be distributed through transmission lines with carefully equalized delays.

Each of the (bidirectional) links among switches consists of two sets of unidirectional wires (see figure 3.1). 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 a higher link throughput is desired. Thus the chips of the current configuration can be packaged in 48-pin packages (40 pins for the link interface, 4 power/ground pins, and 4 pins for clocking and miscellaneous functions like reset, test mode e.t.c). 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.

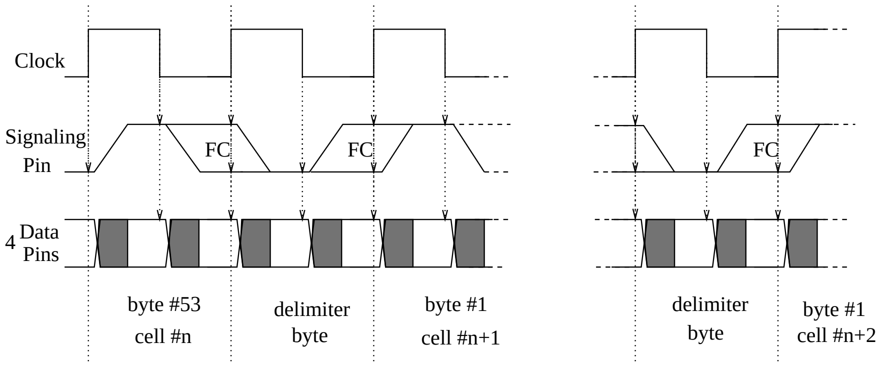

Figure 3.2: Synchronous operation and cell delineation on the chip interface

Chip to chip communication is obviously not error free; a noise spike in e.g. the clock signal may lead to a byte loss and consequently to a cell corruption. A single cell corruption is acceptable in our system; end to end retransmission protocols will be used to handle such cases. What is not acceptable however is the cell framing loss that may be caused by such a transient error. Therefore, a mechanism must be present in the switch to ensure that a single byte loss will not lead to overall synchronization loss. Our mechanism is illustrated in figure 3.2. 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. 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. By using this framing mechanism, an error during a cell transmission will not affect the framing of subsequent cells. Of course the cell that has been affected by the transient error, will arrive to its destination corrupted (or even to a wrong destination) – the current version of the chip does not implement any error recovery mechanism for this. However, we anticipate that such errors will rarely happen in a carefully designed switch PCB, and thus the overall error rate will be acceptably low.

It is evident that the synchronous operation of the chip interface is not part of the switch chip architecture; future extension of this architecture to higher link speeds will not be feasible to operate synchronously with their neighbors. In such cases one can adopt ‘‘messochronous’’ operation: neighboring switches operate under different clocks that are derived from a common system clock, and therefore have the same average frequency, while their clock phase difference is bounded. Such a messochronous operation is possible, regardless of how high the clock frequency is. A timing recovery circuit and an elastic FIFO can then be used to compensate for the difference in clock phases [Mess90].

3.3 Buffer and Routing Table Memories

Perhaps the most critical decision in the design of a switch is the buffering scheme that will be used. Buffering may be done by independent buffers at each incoming link, centralized buffer(s), or independent buffers at each outgoing link. As already explained in chapter 1, in the case of centralized buffers and output buffers switching is done by time-division multiplexing on the buffer bus, while in the case of independent input buffers switching is performed by an $N \times N$ space division switch. Our decision was to use buffer memory shared among all the links (figure 3.1, 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, and solve the buffer ‘‘hogging’’ problem [Fuji83]. The internal static partitioning of the shared buffer that we use solves the buffer ‘‘hogging’’ problem in a brute-force manner. The advantages of a shared buffer memory include better utilization of buffer space, simpler routing of the busses to and from the links (the shared memory bus replaces the space division switch), and less overhead owing to having less memories to manage [Toba90]. If we used input buffering, we would have to manage N distinct memory modules, and the implementation of the round robin multiplexing mechanism would be more complicated: the selection of a cell from an input buffer, would have to take into account the selections from the other input buffers. Output buffering does not suffer from the latter problem, but still has the overhead of managing N distinct memory modules, and does not relax the throughput requirements on each output buffer (see §§ 1.1.1, 1.1.2).

The problem of the shared buffer and output buffer architectures, that lead the majority of the designers to adopt input buffering scheme, is the high throughput requirements of the buffer RAM. However, 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, i.e. 53 Bytes = 424 bits. For example, the memory of our current chip version, with its cycle time of 40 ns (§ 4.2), has a throughput of 25 Mega-accesses/s × 424 bits = 106 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 40 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 that 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 becomes 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* busses, 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]. The input and output buffers and the implementation of the cut-through mechanism are further described in § 4.3.

As explained in § 3.1, the VC translation and routing in our switch are implemented by accessing the routing table memories (upper left of figure 3.1). Each routing table memory in the current version of the switch chip has a size of 256 words times 11 bits per word – 8 bits for storing the new VC ID, 2 bits for specifying the destination outgoing link and 1 bit to signify if the VC is currently open. Normally, the routing memories 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 in the next section.

Effectively our architecture supports 1024 VCs over all its links, so a straightforward implementation would be to provide a 1024-word routing table memory for all the input links. However, providing separate routing table memories for each incoming link has certain advantages. First, a shared routing table memory would introduce additional delay in the critical (for the round trip delay) path of VC lookup due to the access conflicts of the cells from the four links. A solution be to make the shared routing table quad-ported, but this would overly increase its size (and delay). The second advantage of partitioning the routing table is that less bits can be used for identifying a given number of VCs. This economizes about 16 % of total routing table area – the 13,312 bits of storage are reduced to 11264 and four 8-to-256 decoders are used instead of one 10-to-1024. Also, the size of the flow control token is decreased so that more tokens can be sent to an upstream chip during a cell transmission time (6 rather than 4.9 on the average). An argument in favor of the shared routing table memory might be that it allows more than 256 VCs to be routed in one (input or output link) and hence it supports better the ‘‘hot-spot’’ [PfNo85] patterns of traffic. However, in the envisioned network architecture, we anticipate that traffic will be uniformly distributed in most cases. Furthermore, if the network designer expects some kind of ‘‘hot-spot’’ traffic in a particular network, he can appropriately configure the topology of the composite switch for supporting the expected pattern of traffic. It follows that the partitioning of the routing table is a better choice for the purposes of this 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 review now.

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] and [FrTa88]), 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 3.3 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 VC ID's 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.

The organization shown in part **(b)** of figure 3.3 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. In addition, a two stage pipeline increases the maximum rate of cell enqueueing. During the first stage, the free list priority encoder operates, while during the second stage we equeue the cell; during cell dequeueing, the free flag bit is written concurrently with data read, thus eliminating the need for the read-modify-write access. Note that

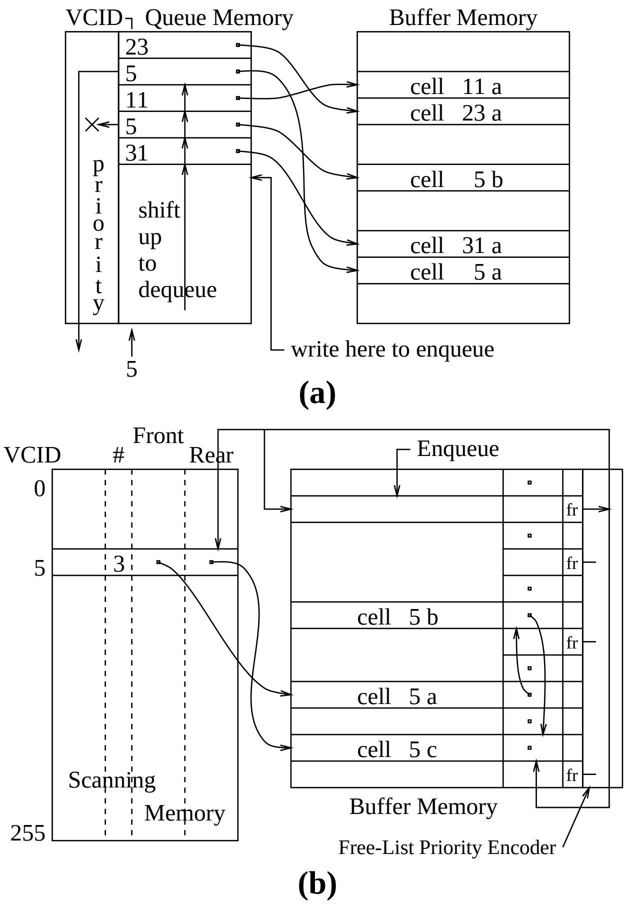

Figure 3.3: Traditional cell queues in a shared buffer organization

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.

A third alternative is to statically partition the buffer RAM space between VCs, allowing each cell belonging to a particular VC to occupy some predetermined buffer RAM rows. The number of rows dedicated to each VC must be equal to the round-trip delay of tokens and cells *counted in cell-times.* In our chip design, where this delay is 2 cell times on the average (§ 3.8), dedicating two rows of the buffer RAM to each VC would be a reasonable choice – i.e. store cells belonging to VC i either to row $2 \times i$ or to row $2 \times i + 1$. The advantages of this scheme are obvious. First, it completely eliminates the VC and free list management overheads. Second, it reduces the size of the scanning memory by eliminating the need of storing front and rear pointers for each VC queue. For example scheme (b) above would require 18×256 bits of memory for storing the *front* and *rear* pointers, in a chip with a total of 256 supported VCs and 512 cells of buffer memory. Instead, the new scheme requires only 3×256 bits of storage, since three bits per VC are enough for designating how many cells are stored in the two cells buffer space for the VC, and which of the two cells is the ‘‘head’’ of this VCs list.

The organization of the buffer memory described above, statically allocates all of the buffer memory space, which as discussed in chapter 2 is not the best buffer allocation scheme for all the kinds of expected traffic. On the other hand, 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. A deeper analysis of it appears in [KaSC91]. 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. The negative effect of this decision, which however is negligible, is the following: as discussed in § 3.8, 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, since more than one system VCs can be used if the network manager wishes to allow one end-user VC to momentarily use all of the link throughput.

Figure 3.4 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 3.3(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. Relative to the third alternative (two buffered cells per VC) this scheme quadruples the total number of VCs supported by the switch chip at the expense of a 10

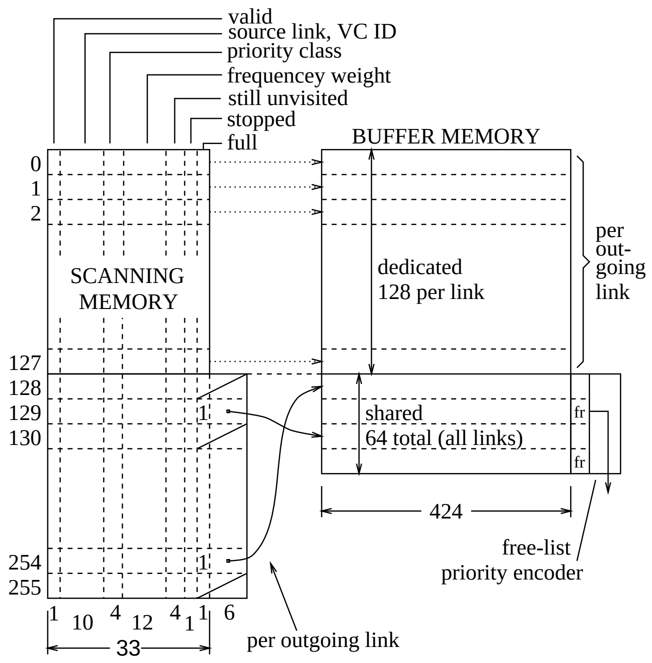

Figure 3.4: Buffer memory and scanning memory organization

% increase on the silicon area. 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 (4 bits, in decoded format), needed in choosing the next VC to serve – see § 3.6;

Weight of service frequency (12 bits) – see § 3.5;

Still-unvisited flag (4 bits) – see §§ 3.5, 3.6;

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.

A finite state machine uses the above fields of the scanning memory (the memory is organized as content addressable) to determine which VC to serve next, according to the prioritized weighted round-robin scheduling. The implementation of this scheduling described in detail in next section, while FSM that searches the scanning memory is described in § 3.6.

3.5 Weighted Round Robin Multiplexing in Hardware

This section† presents the weighted round-robin scheduling algorithm and discusses the hardware requirements of its implementation. 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 chapter 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.

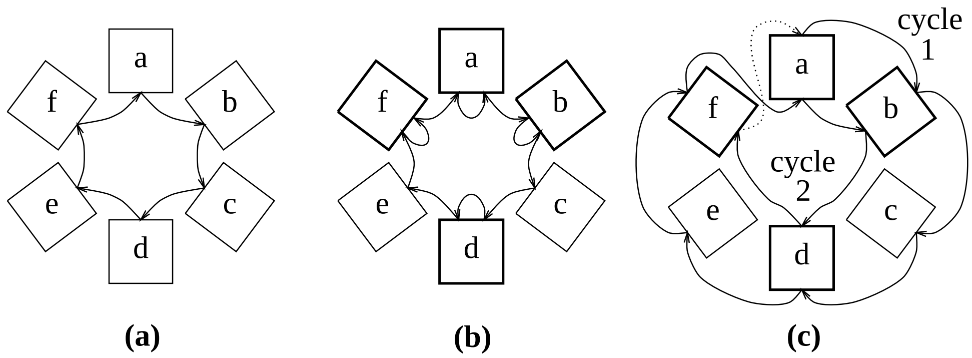

Figure 3.5: Round-robin **(a)** *and weighted round-robin* **(b)**, **(c)** *visits*

Figure 3.5(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 $1, 2, \cdots, N$.

† written by M. Katevenis, as § 4 of [KaSC91], and reproduced here.

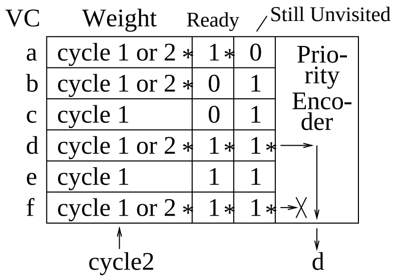

Figure 3.6: The VC scanning memory

Figure 3.6 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’’.

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 $((N+1)/w)-1$).

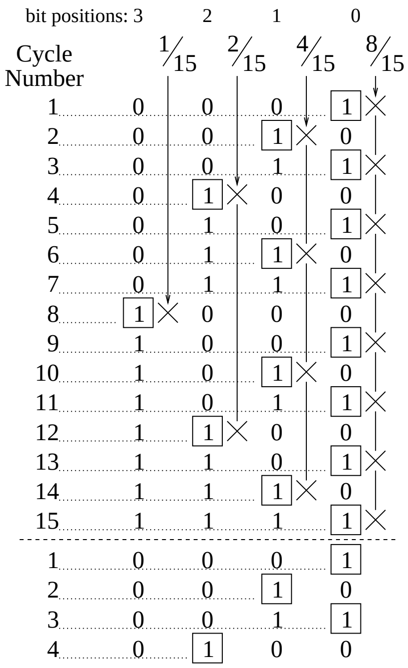

Figure 3.7: Cycle identification and binary frequency components

Figure 3.7 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 writing 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 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 3.7). 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 3.7, 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.

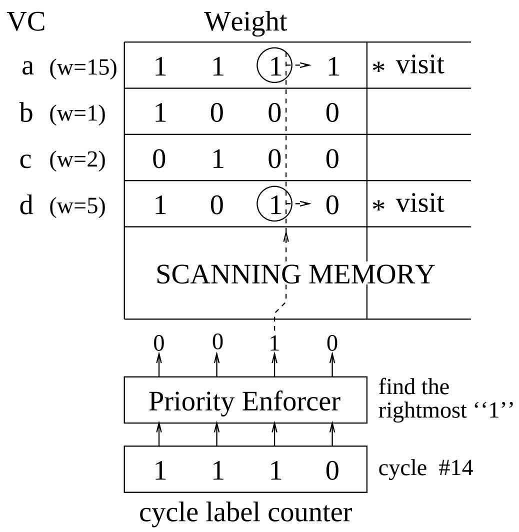

Figure 3.8: Cycle matching – visit those VCs whose weights, stored in *reverse* bit order, have a ''1'' at the position of the rightmost ''1'' of the cycle counter.

Figure 3.8 shows the details of the cycle-matching part of the scanning memory (left part of figure 3.6). 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 $\times$ 20 ns = 1.06 μs, 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 3.8, 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 3.8 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 3.7). 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.

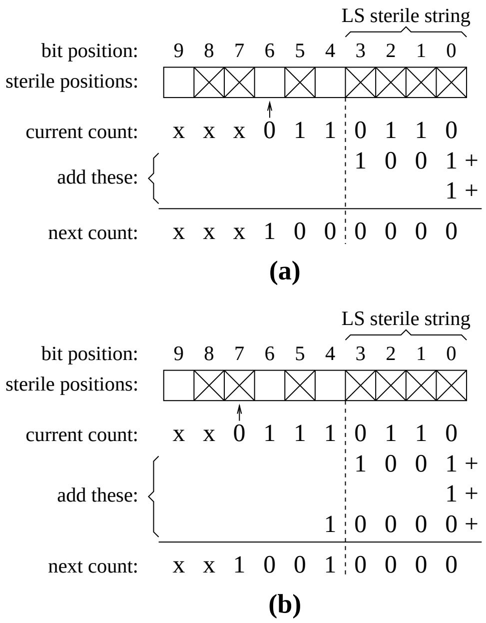

Figure 3.9: Incrementing the cycle counter so as to jump over sterile bit positions

Figure 3.9 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 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 3.7). 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.

3.6 The Cell Selection Mechanism

Until now the cell multiplexing mechanism has been described without considering the details of the actual circuit delays, and the priority levels of the multiplexed VCs. In this section we focus particularly on these details, by describing the algorithm according to which the scanning memory is searched by its FSM. As explained in the previous section, the 54 clock cycles (20 ns each) that it takes the current cell to be transmitted are available for locating the next VC to service, read the corresponding cell from the buffer memory and place it in the output buffer, or grant permission to a high priority cell to cut-through. We will refer to this time interval as a ''selection process period''. During the selection period, the selection process is in a given ''major'' state which is determined by:

(i) *The current priority class* that is searched for ready VCs. At any given time this is the highest priority class that has some ready (i.e. *full* and *not-stopped*) VCs in it. A transition to another priority class will occur, when a selection period starts and there are no more ready VCs in the current class, or at any point during a selection period if a higher priority VC becomes ready (either by a cell arrival or by the arrival of a flow control token).

(ii) *The current circular scan cycle identifier* – i.e the value of the ‘‘label counter’’ described in the previous section. In order to preserve the ‘‘search state’’ of each priority class, this value is saved in a special register whenever a transition to another priority level occurs, and is restored upon return to the same priority level. In addition to the label counter value, a flag bit indicating whether this value has ‘‘matched’’ during the corresponding scan cycle, is saved and restored between priority transitions. We will see in a while how this ‘‘label-counter-matched’’ flag is used.

(iii) *The value of the sterile mask* for the selection process currently in progress. Whenever such a selection process starts (i.e. immediately after the initiation of the transmission of a cell over the outgoing link), the register that holds the sterile mask is cleared. As the selection process proceeds, and various bit positions of the label counter are discovered to be sterile, the corresponding sterile mask register bits are set, in order for the cycle counter to bypass these bit positions.

In order to determine the next VC to service, in addition to the administrative information stored in the scanning memory, the scanning circuitry uses the information fed to it whenever one of the following events occurs :

(i) A cell is incoming from an input link, and it is destined to this outgoing link. In that case, the VC ID of this cell is fed to the scanning circuitry for determining whether this VC must be granted permission to cut-through – i.e. whether it is *not-stopped* by flow control and has *higher priority* than the current.

(ii) A flow control token from the downstream chip arrives into the switch. This VC ID is provided to the scanning circuit, in order to determine whether the corresponding VC has a buffered cell and a *higher priority* level than the current, and hence it must be serviced.

(iii) The current cell transmission is about to be finished, and hence a decision about which cell has to transmit next must be made immediately. The signal that notifies us of such an event is generated by the central controller that arbitrates the requests for accessing the buffer memory – see § 3.7.

The output port multiplexing circuit operates under the same clock as the buffer memory (see § 4.4); the cycle time of this clock is twice that of the chip interface (40 ns). Therefore, a selection will be terminated 21 to 24 clock cycles (of 40 ns) after the initiation of the current cell transmission, depending on the relative alignment of the cells transmitted over the four outgoing links. The major steps of the selection are depicted in figure 3.10. The first step is to determine the current priority class – i.e the highest priority class with some ready VCs in it. This operation takes only two 40 ns clock cycles. During the first cycle, a matching operation is performed to find all the VCs that are ready – i.e. *valid*, *not-stopped*, and *full*, regardless their *still-unvisited* and *frequency of service* values. During the second cycle, all the rows of the scanning memory

2 cycles 1) determine priority class

up to 12 cycles
2) loop
scanning memory matching access
check priority and state of incoming cell or token
in parallel
until VC selected for service

up to 11 cycles
3) loop
check priority and state of incoming cell or token
until central controller's notification

1 cycle 4) read out cell address and mark VC as visited, stopped, and empty

Figure 3.10: Major steps of the cell selection

that have matched during the previous cycle drive their (four bit decoded) priority-class bits onto the corresponding bit lines, which operate in a wired-OR fashion. In this way at the end of the second ‘‘ScanClass’’ cycle, all the ‘‘ready’’ priority classes are known; the highest of these becomes the current class for this selection cycle. This search results in a priority level, which, when compared to this of the previous selection may be: *(i)* higher, due to the arrival of higher priority cell(s), *(ii)* lower, when having run out of cells of the previous priority class, or *(iii)* the same, if none of the above conditions hold.

After determining the priority level of the current selection, the next step is to locate a ready VC whose weight value matches the value of the ‘‘label counter’’ for the current class. This is done by performing ‘‘label matching’’ accesses to the scanning memory (figure 3.8). If the access does not match a ready VC, thus signifying the end of the current circular scan (§ 3.5), the *still-unvisited* flags *only for the current class* are cleared, as well as the ‘‘label-counter-matched’’ flag and the counter is advanced to its next position. In such a case if the label-counter-matched flag is zero (before the clear), indicating that this position of the label counter is ‘‘sterile’’, the sterile mask is augmented to indicate this position as well. The above operations, which take one 40 ns clock cycle, are normally repeated (up to 12 times) until a ready VC in the current class matches the value of the label counter. When this happens this VC is selected for service.

During these iterative ‘‘label-matching’’ accesses to the scanning memory any special accesses (necessary for determining the priority and state of a VC that either has an incoming cell or becomes enabled by a token arrival) are performed in parallel with the ‘‘label-matching’’ accesses – the organization of the scanning memory that enables these parallel accesses is discussed in § 4.4. If such a special access indicates that the corresponding cell has *higher* priority than the current and it is ready, then this VC is selected for service and step 2 iteration terminates.

The selection of a VC, either by ‘‘label-matching’’ or by a special access, can be canceled later and thus the address of the selected cell is not fed to the buffer RAM immediately after the selection (unless our output port is currently idle). The access to the buffer memory for reading the next cell to transmit is postponed until the central coordinator notifies this FSM that the time has come to finalize the decision, since this link’s read operation out of the buffer RAM is imminent. In this way a higher priority VC that happens to become ready (either by cell or flow control arrival) will be selected during this delay period (step 3), altering the initial selection.

When the notification of the central controller arrives the selected VC is marked as *visited, not-full,* and *stopped,* and the address of the corresponding cell is read-out of the scanning memory and fed to the buffer RAM. In case the central controller’s timeout notification arrives while the FSM has not selected yet a VC for service, a VC in the current priority class is immediately selected *regardless its weight of service frequency value.* This mechanism, that keeps the outgoing link busy even if the label counter has not reached the proper count, is not necessary to be implemented in the current version of the switch chip. That is because, even in the worst case where the only ready VC has frequency of service value $1/(2^{12}-1)$, it takes only 12 clock cycles to the label counter to reach the proper value for ‘‘matching’’ this VC. Thus the address of the corresponding cell will be ready to be fed to the buffer RAM 15 clock cycles after the beginning of the selection period – 2 cycles for priority class determination, 12 cycles for the cycle counter to reach the proper value, and 1 cycle for reading the selected cell’s address out of the scanning memory. The ‘‘selection timeout’’ signal from the central controller will not asserted before 21 clock cycles pass from the beginning of the selection period, and hence the ‘‘urgent search’’ mechanism will be never activated. However, such a mechanism must be implemented in future versions of the switch chip, where the links will have larger throughput and the frequency of service weight field of the scanning memory may be wider.

3.7 Buffer Access Arbitration

As described in § 2.3 the bit parallel organization of the buffer memory offers more (average) bandwidth than that required for servicing the four input and output links. Nevertheless, for the achievement of full utilization of the available bandwidth, we still need a mechanism that schedules the memory accesses in such a way that an outgoing link never stays idle, provided that there are cells destined to it. Additionally this scheduling mechanism must ensure that a cell stored in a (double) input buffer will never be overwritten by the next incoming cell. A central controlling circuit that arbitrates the accesses to the buffer memory by the 4 input and the 4 output ports is responsible for satisfying the above requirements. This controller keeps track of when the transmission of the current cells over the outgoing links will be completed. In addition, it is notified whenever an incoming cell has completely arrived into the switch and has been copied into the second part of its corresponding double input buffer (i.e. it is ready to be written into the buffer memory). Based on the above information the central controller selects one of the following accesses to be performed on the buffer memory:

> *outgoing link accesses* – read a cell from the buffer memory and place it to the appropriate output buffer. As explained in the previous section the access for reading a cell selected for transmission over an outgoing link must be scheduled for the latest possible cycle, in order for higher priority cells, that arrive in the meanwhile to be given a chance to ‘‘cut-though’’. In order for a read access to be performed, the arbiter issues a timeout notification to the outgoing link FSM 4 (20 ns) clock cycles before the actual access to the buffer memory has to take place. In case the link is currently idle, the multiplexing machine issues a request and the arbiter grants access in the earliest possible cycle.
>
> *incoming link accesses* – write a cell from an input buffer to the appropriate row in the buffer memory. In order to avoid overwriting of the cell which is temporarily stored in the input buffer by the next incoming cell, these accesses can occur at most 51 clock cycles after the cycle during which the cell has been loaded into the second part of the input buffer. The arbiter is notified by the input buffer whenever a cell is ready to be written into the buffer memory, and allows that access to take place as soon as possible (but not before outgoing link accesses whose selection interval is about to expire).
>
> *refresh accesses* – read and subsequently write a buffer memory row specified by the contents of a special rotating address counter. Obviously this type of access is not time critical. Visiting one row every 54 clock cycles is adequate for visiting all the rows within a period well under 1 msec. The eight link accesses leave 19 free slots for refresh accesses and therefore cell data will never be corrupted due to infrequent refreshing.

Of key importance for the switch performance is the order in which the three types of accesses to the buffer memory take place. If incoming link and refresh accesses are given priority over the outgoing link accesses, this could result in an outgoing link staying idle for up to 18 cycles in the worst case – 8 cycles for incoming link accesses, 4 cycles for the refresh access and 6 cycles for the other 3 outgoing link accesses. On the other hand, giving higher priority to the read operations of the outgoing links, and thus delaying the write and refresh accesses, will *never* lead to starvation of the latter operations, or even to excessive delay, because it is guaranteed that the 4 read's can never consume more than 8 of the buffer RAM cycles in any 54-cycle interval.

Therefore, to achieve continuous and full utilization of the links' bandwidth, the central controller operates in the following way. Whenever an outgoing link is transmitting a cell, the corresponding access to the buffer memory is scheduled for the latest possible cycle. In a given memory cycle, if a multiplexing machine (attached to a currently idle outgoing link) requests immediate access to the buffer memory, and there is not another access already scheduled for this cycle, the outgoing link is permitted to read the selected cell − if there are some other read access(es) scheduled we will wait for them to be performed first. If there are no scheduled outgoing link accesses for the current cycle, and if there is a pending write access request, the write access is granted. If there are no scheduled read accesses or pending write accesses and there are no scheduled read accesses for the next cycle either, a refresh access is performed in the two subsequent memory cycles. When there are more than one pending requests of the same type, we make a random choice (currently the first of them is granted according to a fixed priority).

The part of the controller that grants access to the buffer memory according to the access type, is simple and can be implemented by using a small combinatorial circuit − e.g. a pseudo-NMOS PLA. The scheduling of the outgoing link accesses according to the ending cycle of the current cell transmission is more challenging. Ideally, reading a cell from the buffer RAM starts 2 clock cycles before the transmission of its first byte by the output buffer. The corresponding timeout notification to the multiplexing machine is send 4 clock cycles earlier − 1 scanning-memory cycle for the potential urgent scan and 1 scanning-memory cycle for reading the selected cell address. If however the transmission of the current cells over two or more outgoing links will complete in the same (or adjacent) clock cycle, some of the outgoing link access(es) has to be performed earlier. Whenever two or more cells that are being transmitted over an outgoing link are aligned in such a way, we will say that there is a buffer scheduling conflict. We have such a conflict whenever two outgoing links started transmitting their current cells with a time difference of 0 or 1 clock cycle (or when one of them was rescheduled owing to a conflict with a third link).

Figure 3.11 help us explain access conflicts and the way in which they are resolved. In what follows the word ‘‘cycle’’ refers to a buffer-RAM access cycle. In part **(a)** of figure 3.11 the four outgoing links have to initiate the transmission of a cell during the same cycle t. Ideally the memory accesses for all the four links would take place in cycle $t-1$. However this is not possible, so the access of links 3,2,1 are scheduled for the cycles $t-2$, $t-3$, $t-4$ respectively. Similar situations are depicted in cases **(b)** and **(c)** where three and two links respectively compete for accessing the buffer memory in the same access cycle. Parts **(d)** to **(h)** of figure 3.11 depict somewhat more complex situations. For example, in case (d) scheduling the accesses of links 1 and 2 for the cycles $t-2$ and $t-3$ forces the access of link 4 (that otherwise would take place in slot $t-2$) to be scheduled for the slot $t-4$. In general, an access that should normally take place in cycle t can be moved up to $n-1$ cycles ‘‘backwards in time’’, if there are n accesses that would

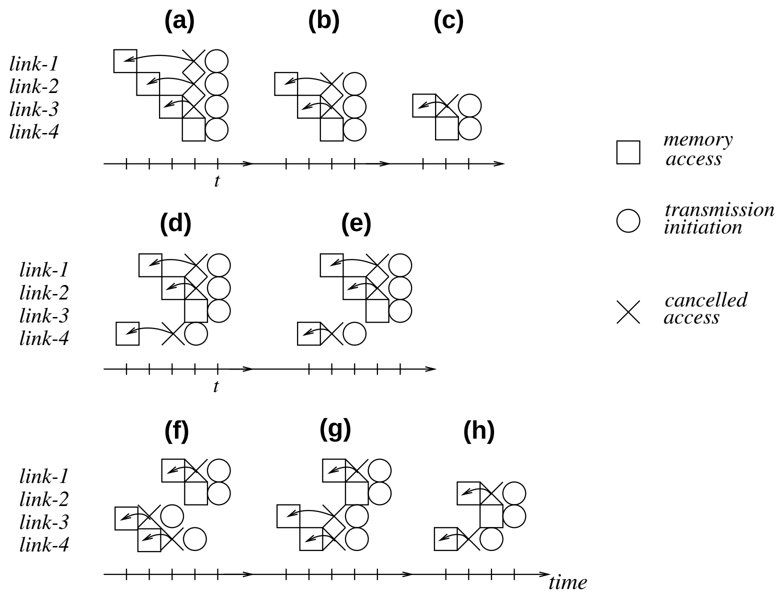

Figure 3.11: Access conflict resolution

also normally take place in cycle t $(1<n \leq 4)$ – see figure 3.11 parts (a) to (c). Any such a rescheduling of an access may affect the scheduling of other accesses which can be moved for m additional cycles backwards $(m \leq 2, m+n \leq 4)$ owing to buffer memory conflict with the rescheduled access(es) – see figure 3.11 parts (d) to (h). Note that figure 3.11 presents all the possible access conflict ‘‘patterns’’ that can occur in the current 4×4 switch implementation, if we consider that each case is representative for a class of equivalent cases that can be obtained by permuting the order of the links. Note also that case (b) is a subcase of (d) and (e), while case (c) is a subcase of (f),(g) and (h).

The scheduler uses information that is already maintained in the input and output buffers. Each buffer circuit incorporates a 53-bit shift register in which a single flag bit is shifted every clock cycle, enabling the loading or transmission of the corresponding byte stored in the buffer (see § 4.3). The distance between the bit currently holding the flag and the 53rd bit of the shift register shows how many clock cycles remain until the end of the current cell loading or transmission. These flag bits are used by the scheduler, in order to determine when an outgoing link access must take place. Figure 3.12 shows the main idea of how the scheduler circuit is designed. The circuit has four inputs (on the left) corresponding to the four outgoing links. Each of them is fed by a 2-input OR gate. The inputs of each OR gate are fed via multiplexers either from bits 38 and 39 of the output buffer shift register, or from bits 38 and 39 of the shift registers of the input buffers that are currently ‘‘cutting-through’’. The circuit operates under the same clock as the four multiplexing machines and the buffer memory, and its outputs are the four

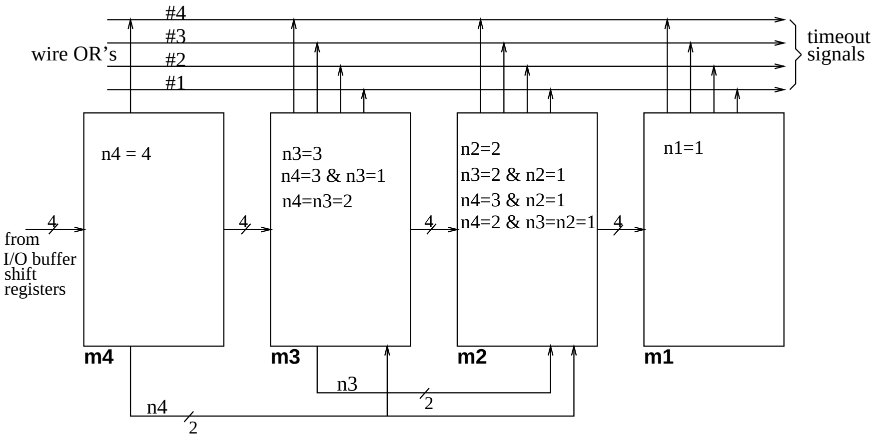

Figure 3.12: The scheduler circuit

''timeout'' signals fed to the multiplexing machines. Each of the modules consists of a register, a tally circuit that counts the number of 1's in the register (n_1 to n_4 in the figure), and a combinatorial circuit that uses the above numbers to decide whether one of the conditions depicted in each module is met. The first three modules also contain a 4 bit priority enforer. During each clock cycle, if the combinatorial circuit of each module indicates that its conditions are met, the first ''1'' bit in the register is selected (via the the priority enforcer) and the corresponding timeout signal is asserted. The first module can only activate the timeout of link-1, because it only detects the conditions of figure 3.11-(a). Each time a timeout signal is asserted the corresponding bit of the register is cleared and all the bits are transferred to the next module's register by the end of the current cycle. In this way, the appropriate outgoing link multiplexing machine is notified two cycles earlier than the cycle in which the access for retrieving the next cell have to be performed. The timeout signals are also fed to the arbiter part of the central controller, which latches them and consults them two cycles later to see if it can grant another access to the buffer memory.

The circuit described samples the state of the shift registers with a period twice the period that this state changes. Therefore it will consider that two outgoing links will complete the transmission of their cells in the same clock cycle t, even if they will actually complete in cycles t and $t+1$. Consequently, the memory access of the link that finishes its current cell transmission in cycle $t+1$ may be scheduled before this of the link that finishes in cycle t – an unfair scheduling. Resolving this problem requires an implementation of the scheduler with 8 modules operating under the same clock as the outgoing link shift registers – the combinatorial circuit of each module will be more complex as well. However we can see that such an unfair scheduling is not a major problem – it may result in at most 1 lost cycle for an outgoing link multiplexer.

3.8 Round-Trip Delay

Having described all the mechanisms involved in cell selection, transmission, and cut-through we can now calculate the round-trip delay between two neighboring switch chips. Since the switch chips handle VCs of different priority classes, the round-trip delay is defined as *the time delay between the initiation of transmission of two cells of the same (highest-priority) VC.* This delay varies 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. In the worst case there will be many low-priority/congested VCs that pass-through two switch chips, and only one high priority VC, which must be served as soon as possible whenever a cell belonging to this latter arrives in a switching node.

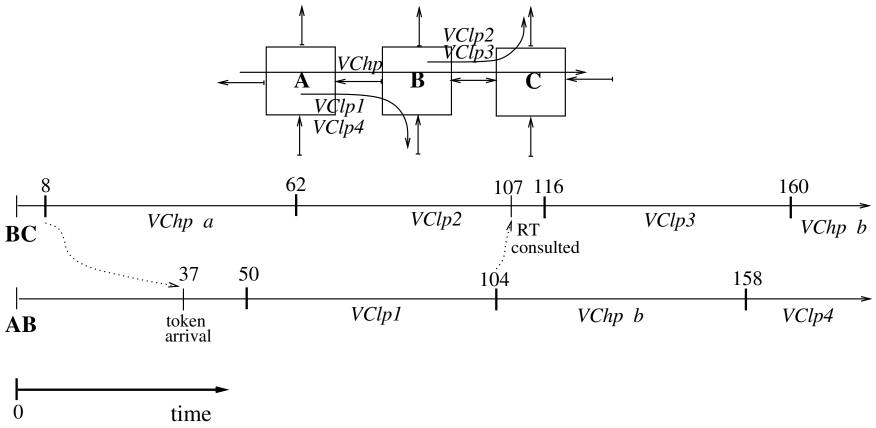

Figure 3.13: Worst case round-trip delay example

Imagine a VC_{hp} of highest priority that goes from switch chip A to chip B to chip C − see figure 3.13. Say that at t=8 switch B starts sending cell a of VC_{hp} to C; this transmission will last till t=62. As soon as B decides the above transmission it also tries to send a token for VC_{hp} to A. This could happen as late as t=4. However, two other multiplexing machines may have decided to transmit two other cells (which have arrived into B via $A \rightarrow B$) at t=0 and t=2 respectively†, and thus these tokens will be transmitted before the token of VC_{hp}. Furthermore, a fourth (null) token may have been already in transit over $B \rightarrow A$ when the first decision has been made. Thus, the worst-case arrival time of the token for VC_{hp} at A is t=37 (=8+3×9+2) . If we are unlucky, at t=37 the multiplexing machine of link $A \rightarrow B$ has just decided to read the cell of

† Recall that the buffer-RAM scheduler issues the timeout notifications to the four links always in different time-slots (§ 3.7).

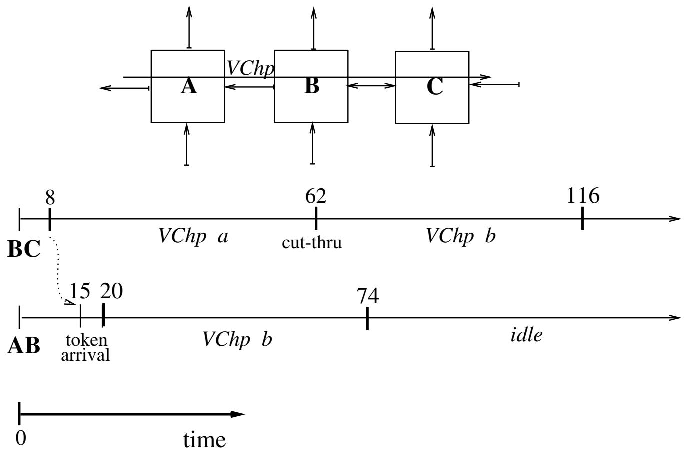

Figure 3.14: Best case round-trip delay example

another, low-priority VC_{lp1} from its buffer memory in order to forward it to B. This buffer-read operation may be required as early as 11 cycles before actual transmission starts, because of access conflicts from all 4 links to the single buffer RAM, and the decision for this read is made 2 cycles earlier, because of the scanning-memory delay. Thus, at time t=50 (=37+11+2) A starts forwarding a VC_{lp1} cell to B. The next cell that A forwards to B, at t=104 (=50+54), will obviously be cell b of VC_{hp}. Switch B sees the VC ID of b at t=105 – the time delay for crossing the chip boundary is 1 clock cycle – and consults its routing table; the answer that b is destined to C comes at t=107 – the access delay of the routing table memories is 2 clock cycles. The next thing to do is to notify the multiplexing machine of link $B \rightarrow C$ that a cell of VC_{hp} has arrived, before the decision of this machine about which cell to transmit next is made at t=112 – since the first decision was made at t=8 the next decisions will be made at t=58,112,166... because of the static nature of the buffer-memory scheduler. However, two other incoming links may as well request inspection of their incoming cell priority from the same FSM, in order to be given a chance to cut-through. If we are unlucky, the FSM will consider the other two requests at t=108 and t=110 respectively – reading a VC's priority out of the scanning memory takes 2 clock cycles. Cycles 112 and 113 are dedicated to cell address calculation, and thus cell b of VC_{hp} will loose the opportunity to ''cut-through'' at t=116. Replicating the VC's priority and flow control state in the routing table would not help much, since *(i)* an additional comparison would be required before notifying the outgoing link FSM, and *(ii)* it will further complicate the management of this duplicate information. So, in this worst case the notification for cell b will only result in scheduling this cell to be transmitted over $B \rightarrow C$ at t=169. However, this transmission will not be performed via ''cut-through'', since from t=158 another cell may have already started being transmitted over the link $A \rightarrow B$. Cell b will be transferred from the input buffer of link $A \rightarrow B$ to the output buffer of link $B \rightarrow C$ via the buffer memory bus. This will happen during the access cycle that is normally occupied for the link's $B \rightarrow C$ read access at t=156. From this example we conclude that the worst case round-trip delay is 3 cell times − 2 low priority cells may be transmitted between 2 cells of a highest priority VC.

The best case round-trip (figure 3.14) is much better: cell a starts being transmitted over the link $B \rightarrow C$ at t=8, the token arrives in A at t=15, the scanning-memory of A decides at t=17 to serve VC_{hp}, cell b is read from the buffer-memory of A at t=19, and its transmission to B starts at t=20. Switch B sees the VC ID of b at t=22, the routing-table access is complete by t=24. The multiplexing machine of link $B \rightarrow C$ has about 38 cycles to be notified and decide to allow cell b to cut-through, at our next chance at t=62. Thus, the best-case round-trip delay is only 1 cell time. We conclude that the average round-trip for tokens-cells must be about two cell times.

3.9 Composite Switch Initialization

Until now we have considered the switch chip operation only in the state where its routing tables have been set-up properly. In a real intergrated network, an efficient method for establishing VCs, i.e. for initializing and later updating these tables, must exist. In our case, the hierarchical nature of the network architecture implies that VC establishment will be made in a hierarchical way as well. Local network management software will be responsible for setting up VCs between end-points of local area broadband networks. Whenever a request for a remote connection is issued, the appropriate VC will be established by the local manager within the limits of its responsibility, while the request will be forwarded for further processing by the remote network managers. In either case the switching hardware must provide the necessary primitives for setting-up VCs over the route that has been selected by the network management software.

The first initialization mechanism that we have considered is to use ‘‘meta-signaling’’ VCs for this purpose [CCIT90]. The data carried by such special VCs will initialize the routing tables of a switch chip. These VCs should be treated as ordinary by other switches lying in the path between the controlling end-point(s) and the switch to be configured. When using this approach a composite switch of arbitrary topology is said to be initialized, if every chip in it: *(i)* is accessible from the controlling end-point(s) via one or more VCs, and *(ii)* knows which VCs to listen to for altering its routing tables. Obviously, a large number of special VCs will be required to initialize a relatively large composite switch or network. Furthermore, an additional mechanism (e.g. node-IDs) will be required for configuring each switch to listen to the appropriate meta-signaling VCs. These disadvantages lead us to adopt a different initialization mechanism to be described below.

The first experimental implementation of the switch chip will implement an initialization mechanism similar to the one used in the NECTAR network backplane [Arn89]. Compared to

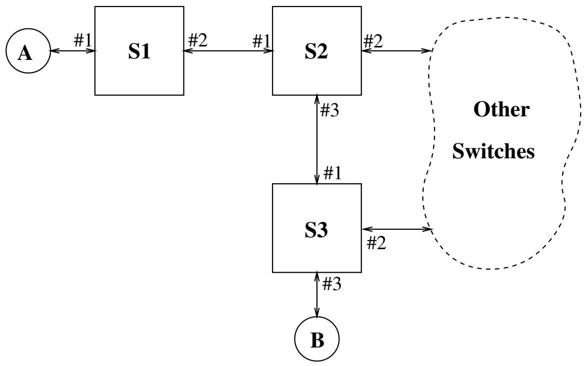

Figure 3.15: Part of a sample network topology

the mechanism just described, this scheme requires less VCs to be reserved for network initialization. However, it forces us to use a cell header format somewhat different from the CCITT standard. This difference in cell header is not a major problem, since a simple translation mechanism can be implemented in the interface chips, if compatibility with this standard is required. Four different types of cells are recognized by our switch chip: *(i)* normal cells carrying user payload, *(ii)* node-ID initialization cells, *(iii)* VC set-up cells, and *(iv)* signaling cells e.g. for carrying remote flow control messages (§ 2.3). The type of each cell is identified by the first two bits in its second byte. Each switch chip includes a 16-bit register that contains its node-ID – we will see in the example below how this register is initialized. Whenever a VC set-up cell specifying this chip's node-ID arrives into the switch, the routing tables of the chip are altered according to the information contained in this cell. Consider the composite switch topology illustrated in figure 3.15. Say that this composite switch is in the state just after a global reset has been issued. In this state the contents of the routing tables, scanning memories and node-ID registers of all switch chips are invalid. In order to initialize switches S_1,S_2,S_3 with their corresponding node-IDs, end-point A – which is a computer running the local network manager – sends the following cells over link $\#1$ of switch S_1:

VC ID	*TYPE*	*DATA*
VC_{any}	nodeID set-up	ID=1
VC_{any}	VC set-up	NodeID=1, OpenVC, Class=don't care, VCinit=VC_x, VCtrans=VC_y, InLink=1, OutLink=2

After the reception of the above cells switch S_1 knows that its node-ID is 1. Furthermore, its routing table attached to link $\#1$ is initialized so that when a cell of VC_x arrives through this link, the cell's VC ID will be changed to VC_y and the cell will be forwarded to link $\#2$. Switch S_2 is then initialized in a similar way when end-point A sends the following cells:

VC ID	*TYPE*	*DATA*
VC_x	node-ID set-up	ID=2
VC_x	VC set-up	node-ID=2, OpenVC, Class=don't care, VCinit=VC_y, VCtrans=VC_z, InLink=1, OutLink=3

When S_1 detects the node-ID set-up cell in its incoming link, it does *not* change its node-ID to 2, because it already has VC_x in its routing table. Switch S_3 is configured by sending similar cells again from end point A. To establish e.g. a class 3 connection with end-point B, A sends the following cells:

VC ID	*TYPE*	*DATA*
VC_{any}	VC set-up	node-ID=1, OpenVC, Class=C, VCinit=VC_{AB}, VCtrans=VC_{AB}, InLink=1, OutLink=2
VC_{AB}	VC set-up	node-ID=2, OpenVC, Class=C, VCinit=VC_{AB}, VCtrans=VC_{AB}, InLink=1, OutLink=3
VC_{AB}	VC set-up	node-ID=3, OpenVC, Class=C, VCinit=VC_{AB}, VCtrans=VC_{AB}, InLink=1, OutLink=3

In a similar way, a VC from B to A can be established. After this establishment end-point A can notify end point B that a connection is open between them. End-point B can subsequently use this connection to request establishment of other VCs to various network end-points which will be initialized in a similar way.

This initialization mechanism requires minimal hardware, while it offers to the network management software the necessary support for implementing any VC establishment policy. Some additional hardware may be required if we want to increase the reliability of the initialization mechanism – e.g. by including checksums or some other error detection field(s) in the set-up packets.

Chapter 4

Switch Chip
Implementation Issues

This chapter deals with the design and simulation of the critical (in speed and size) parts of the switch chip. After a discussion about the floor-plan and the overall simulation policy used (§ 4.1), it presents the implementation of the buffer memory (§ 4.2) and I/O buffers (§ 4.3). The critical implementations of the scanning memory are discussed next (§ 4.4) and the chapter concludes with a discussion about the other parts of the switch chip and the simulation experiments performed for the ES2 1 μm CMOS technology.

4.1 Chip Size and Performance Estimation

The feasibility of the switch architecture presented in the previous chapters depends mainly on the size of the resulting silicon implementation and the actual speed of the circuit. In order to prove that our architecture is feasible, we have laid out and simulated all the parts of the chip that are critical in determining its size and speed. Although the results of this effort are preliminary, they give a good idea for the chip's speed and size, and confirm our initial expectation that the single chip implementation of the proposed architecture is feasible. Our layout was done for a 1 μm full-custom CMOS VLSI technology with double metal and single polysilicon layers. We have used the MOSIS Scalable-CMOS design rules (λ=0.5μm) and the *magic* layout editor [Oust85].

The size-critical parts of the chip that require full-custom mask-level layout are the buffer memory, the scanning memories, the I/O buffers, and the routing table memories. The remaining parts of the chip (controlling PLAs, decoders, etc.) are not size-critical, and most of them can be designed by using a semi-automated layout approach. Table 4.1 lists the sizes of the critical blocks in μm per bit-cell, total number of bits, and total area.

Table 4.1: Sizes of the critical blocks

Block	Bit Size (in μm) w	Bit Size (in μm) 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	11,264	4.0
Scanning Mem.	21	23	39,936	19.2

Using these sizes, plus the estimated sizes of the memory decoders, priority enforcers, wiring etc. we arranged the floor-plan of figure 4.1-**(a)**. As expected the buffer memory and the scanning memories occupy the majority of the chip – 55 % of it's 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 other controlling circuits.

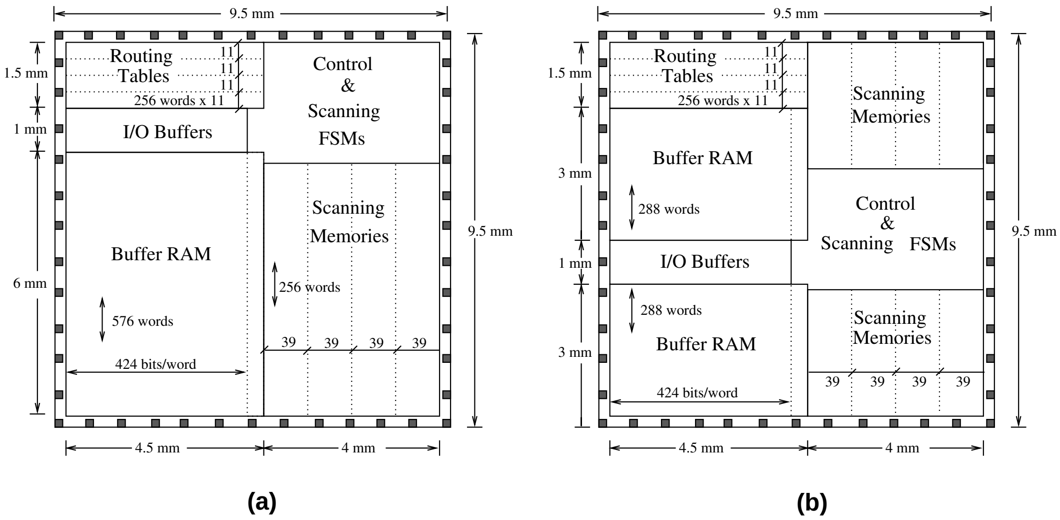

Figure 4.1: Switch chip floor-plan alternatives

An alternative floor-plan that can be used to achieve better performance, almost without affecting the total chip size, is shown in figure 4.1-**(b)**. Partitioning the buffer and scanning memories into two parts and placing the I/O buffer and control circuitry in between, can result in better memory access times with minimal additional hardware. Regardless of which floor-plan we will finaly choose we can conclude that the fact that the area requirements of the 4×4 chip version is inside the limits of today's conventional CMOS technology gives a satisfactory proof of the feasibility of the proposed switch architecture.

To estimate the performance of the time critical parts of the chip, we performed circuit level simulations with *spice* [Nage75]. Table 6.2 lists the parameters used in our simulations.

Table 6.2 : Spice parameters used in simulations

Device	LEVEL	PB	VMAX	TOX	KP
Both	3	0.7	17×10^{4}	200×10^{-10}	2×10^{-5}
	VTO	LD	XJ	NSUB	UO
Pmos	–0.8	0.10×10^{-6}	0.3×10^{-6}	5×10^{15}	220
Nmos	0.8	0.07×10^{-6}	0.1×10^{-6}	3×10^{15}	650
	CGSO	CGDO	CJSW	CJ	RSH
Pmos	3.44×10^{-10}	3.44×10^{-10}	2.31×10^{-10}	2.44×10^{-4}	20
Nmos	1.20×10^{-10}	1.20×10^{-10}	0.6×10^{-10}	1.89×10^{-4}	18
	ETA	THETA	KAPPA	NFS	MJ
Pmos	0.06	0.03	0.4	1×10^{10}	0.5
Nmos	0.14	0.06	0.4	1×10^{10}	0.5

The circuits that were laid-out were simulated based on the actual sizes and parasitic capacitances of their elements, assuming the floor-plan of figure 4.1-(a). Because the available technology file and the *magic* layout extractor were not trustworthy, we used automatically extracted netlists only for switch level simulation with *esim* [Term86]. In order to obtain accurate circuit level simulation results, we used hand-written *spice* input files in which we have included the FET sizes and parasitic devices based on the actual layout. The circuits that were not laid-out were simulated, again using hand written *spice* files in which we included ‘‘artificial’’ parasitic devices, whose sizes were intentionally overestimated in order to be on the safe side. The switch and behavioral level simulation of the latter circuits was done with *verilog* [Gate89].

Based on these simulation results, a preliminary clocking scheme can be devised. The external 50 MHz clock will be used to generate an internal clock of the same period with two non-overlapping phases. The two clock phases (ϕ_1,ϕ_2) will be 8 ns each, while their non-overlap period will be 2 ns. Internally to the chip, two other 2-phase clocks will be generated. The first one, with phases of unequal duration (χ_1=8 ns, χ_2=28 ns, 2 ns non-overlap), will be used for accessing the buffer memory. The second one, with phases of equal duration (ψ_1=ψ_2=18 ns, 2 ns non-overlap), will be used for the operation of the scanning hardware and the central controller. These two 25 MHz clocks will obviously be generated from the master 50 MHz clock, and they will be synchronized with it. Thus, the outputs of the various blocks of the chip that operate under different clocks will be synchronous with each other, and no inter-chip synchronization problem will occur, if clock generation and distribution is done carefully to minimize skews and phase differences.

4.2 The Buffer Memory Implementation

The considerable size of the buffer memory (≈245 Kbits) requires a dynamic type bit-cell in order to minimize its area. Ideally, the buffer memory should be build using the 1-transistor dynamic cell, allowing one to minimize the bit area, and thus increase the switch capacity and performance. However, using a 1-transistor DRAM requires deep design expertise, and access to special fabrication technologies. For that reason, we are unable to use this cell, at least for constructing the first experimental prototype. The two alternatives considered next were to use the 4-transistor and 3-transistor dynamic RAM cells. Laying out several versions of these cells resulted to a 4-transistor bit cell of size 182.25μm^2, and a 3-transistor bit cell with size 107μm^2. The only advantage of the 4-transistor cell is that it requires only 1 cycle time to be refreshed – reading the bits in a row results in restoring the internal bit values. The 3-transistor cell requires 2 cycles to be refreshed – a read followed by a write. In our case, dedicating two RAM cycles in every cell time for refreshing the buffer memory does not impose any particular problem. Thus the obvious choice is to use the 3-transistor cell for implementing the buffer memory.

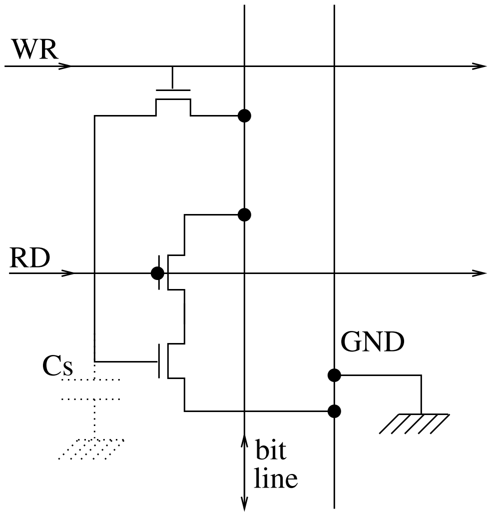

Figure 4.2: Three transistors dynamic RAM schematic

Figures 4.2 and 4.3 show the schematic diagram of the 3-transistor cell and the *cifplot* of the most compact version of its layout. Since performing simultaneous read and write accesses to the buffer RAM is not necessary, only one bit-line is incorporated. The bit and power lines are run in first level metal. In order to reduce the horizontal cell size, the ground line is shared between two adjacent mirrored cells. Furthermore, the word lines (*RD* and *WR* in figure 4.3) are run in polysilicon, but in order to reduce the delay of driving them which would result from their long length and the corresponding large RC, supporting word lines run on second level metal and are contacted to the polysilicon every 8 cells (*RDs* and *WRs* in figure 4.3).

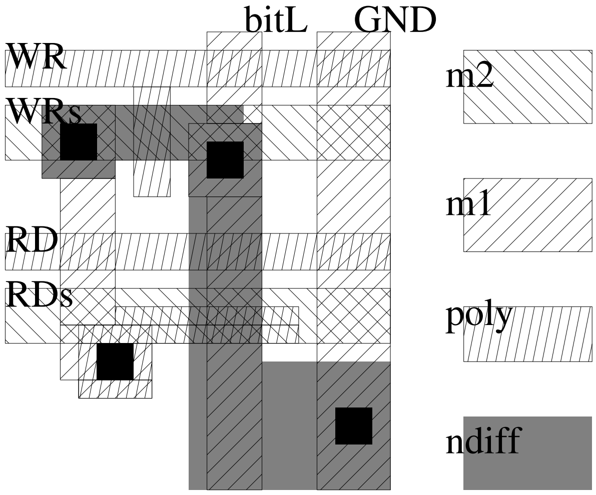

Figure 4.3: The dynamic-RAM cell layout

The memory decoder is a dynamic NOR decoder, precharged during χ_1, and evaluated during the non-overlap period χ_1-χ_2 [WeEs85]. We preferred to use this clocking scheme for the decoder – rather than a latched decoder evaluated during χ_1 – because it allows the address to settle relatively late, thus allowing the cell read access to take place only 1 clock cycle after the cell-address is read from the scanning memory. The only problem when using this decoder is that the decoding NOR gate must be designed carefully in order to be able to discharge in the short non-overlap period. Driving the long (≈ 6 mm) address-lines during the 8 ns χ_1 phase, does not impose any problem if strong drivers are used.

Both the buffer memory cell and the precharged address decoder were simulated. The simulation indicated that a 40 ns cycle time can be achieved without using any special sense amplifier circuit. The χ_1 8 ns time interval is more than adequate for precharging a 3 pF bit line. When reading from a cell by discharging the bit line, the output of a sensing inverter that amplifies the bit line voltage switches to high state within 27 ns, *including* the 8 ns delay for driving the word line high. Writing into the buffer RAM is not as time critical as reading; a logic-one value on the precharged bus has the entire 28 ns period available to charge the internal bit capacitor, while a logic-zero value can be written fast enough (within 18-20 ns) if the pull-down transistor of the bit line driver is properly ratioed. The simulation of the address decoder indicated that the 2 ns non-overlap period is adequate for discharging the decoding NOR gate through just one of it's pull-down transistors. Additionally, it indicated that a 2 pF word line can be pulled-up fast enough (5-8 ns) by the decoder output driver.

There is one more practical problem to be solved in the buffer RAM design. When a write access is performed, the written value may be over-written with a logic-one during the subsequent precharge phase, if the precharging transistor pulls-up the bit line before the word line has been pulled-down by the decoder driver. Two approaches to overcome this race condition exist. The first one is to arrange the W/L ratios of the precharging transistor and the word line pull-down transistor, so that the word line pull-down be faster than the bit line precharging. This solution, is simple and the simulation showed that it can be implemented easily. However, a safer solution is to use a self-timed precharging circuit. This circuit will turn precharging transistors on, only after a representative word-line has been driven low by the decoder driver. Due to lack of time such a circuit has not been simulated. However, its construction is relatively simple (see e.g. [AnAZ89] and [Horo87]). Furthermore, because the precharging operation is not time-critical, delaying it in this way will not increase the memory access cycle.

4.3 Input and Output Buffer Circuitry

The Input and Output buffer circuits are essential for the switch chip operation. Besides performing the word size conversion between the external links and the buffer RAM, they need to implement double buffering in order to retain the continuous flow of cells in and out of the switch. In addition, the input buffer needs to incorporate some additional circuitry to implement the ‘‘virtual cut-through’’ function.

The logic diagram of the input buffer is shown in figure 4.4. The 4-bit incoming link is converted to an internal 8-bit bus that operates at half the link rate, carrying 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. 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 or transferred to the appropriate output buffer. The right-most part of figure 4.4 illustrates the cut-through data-path, which operates 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 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.

Figure 4.5 illustrates the logic diagram of an output buffer. The transmission of the bytes of a cell from the upper latches is done in the same way as the cut-through function is implemented from the input buffers. The next cell to be transmitted is loaded into the lower latches from the

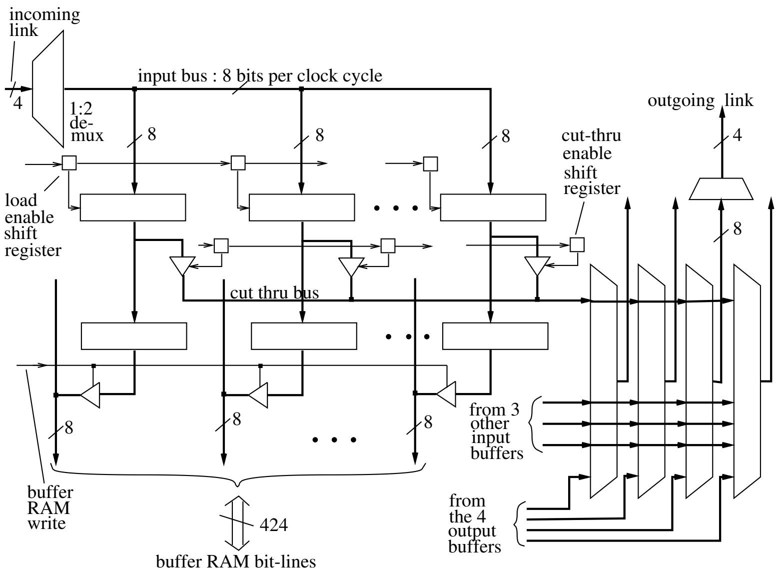

Figure 4.3: Input buffer logic diagram

buffer RAM bit lines, sometime between the 45th and 52d clock cycle of the current cell (see § 3.7). Note that the buffer RAM bit lines may be driven not only from a buffer RAM word, but also from the lower latches of an input buffer. On the 54th clock cycle of the current cell transmission (i.e. the delimiter byte cycle) the contents of the lower latches are copied into the upper latches, from where they will be transmitted during the next 53 cycles.

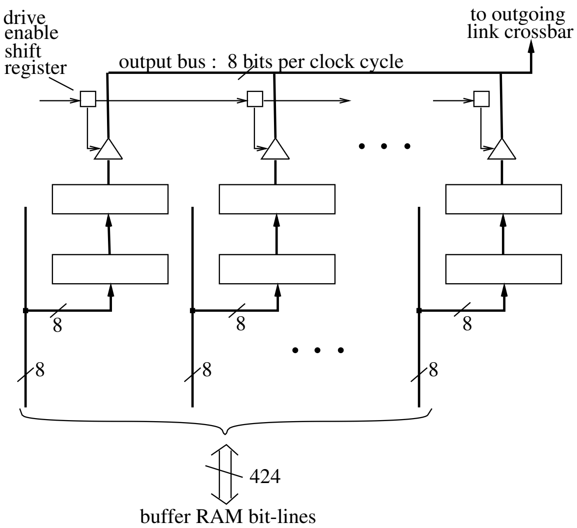

Figure 4.5: Output buffer logic diagram

There are two shortcomings in the input/output buffer operation described above. The first is a consequence of the presence of only one cut-through bus in each input buffer. Consider the situation illustrated in figure 4.6-(a) where two high priority VCs (VC_{hp1}, VC_{hp2}) pass through a switch chip and are destined to different outgoing links. Say that at t=0 cell a of VC_{hp1} arrives to the switch. Because cell c of low priority VC_{lp} is currently being transmitted over outgoing link #2 the cut-through of the current cell cannot start before e.g. t=20, and in that case it will last until t=74. Cell b of VC_{hp2} arrives into the switch at t=54, the routing table is consulted, and since outgoing link #2 is idle its transmission via cut-through could be started at about t=58. However, the cut-through bus of link #1 is currently used for transmitting cell cell a. Thus the transmission of cell b cannot be started before t=74. To avoid such situations each input buffer would need to incorporate two additional cut-through busses and shift registers. However we decided that the probably slight performance increase is not worth the increased area and complexity that the additional hardware would bring, and thus we will implement the input buffer with only one cut-through bus.

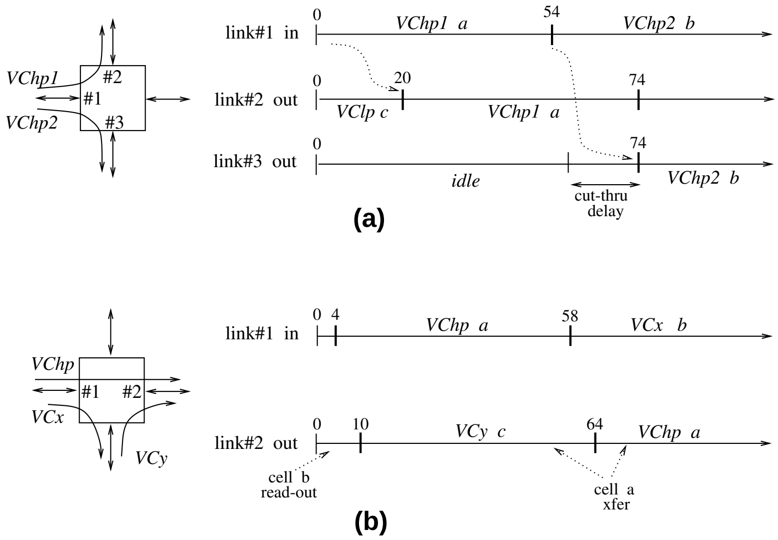

Figure 4.6: Race conditions on I/O buffer operation

The second shortcoming is a consequence of the inability of the outgoing link multiplexing machine to reverse its decision about which cell has to be transmitted next over the outgoing link. Consider the situation in figure 4.5-(b). Cell a of a high priority VC arrives into a switch after the decision for the cell to be transmitted next on link #2, has been made and even after the low priority cell has been read out of the buffer RAM at t=2. Obviously, cell a will not cut-through at t=10, but it will start being transmitted over the outgoing link over link #2 at t=64. This transmission however will not be performed via cut-through. At t=58 another cell will be incoming from link #1, thus occupying the upper latches of link's #1 input buffer. Thus cell *a* has to be transferred to the output buffer of link #2 during the 2 clock cycles that are normally occupied for the read access of outgoing link #2. However these clock cycles may be scheduled for as early as $t=54,55$ by the buffer memory access scheduler (see § 3.7). By this time not all the 53 bytes of cell *a* will have arrived into the switch and been stored in the input buffer.

One solution is to allow the decision of the scanning circuitry to be reversed. In such a case, cell *a* will cut-through at $t=10$ and the race condition will be avoided. However as explained in § 3.6 this would result in complicating the scanning memory and scanning FSM implementation and would better be avoided at least in the first experimental implementation of the switch. The second, less expensive solution is to transfer the first 24 bytes of the cell during $t=54,55$. The remaining 29 bytes will be transferred after $t=64$ and as late as $t=70$ during the memory slot that is normally reserved for the write access of incoming link #1. This requires that the first 24 bytes of the cell will be present in the input buffer's lower latches at $t=54$. This can be accomplished by copying the first 24 upper input buffer latches to the corresponding lower ones, when they have been loaded by the incoming cell data. The remaining 29 bytes will be transferred during the 54th clock cycle as described above. Note that copying the first 24 bytes of the cell to the lower input buffer latches cannot destroy the data of the previous cell, since the buffer RAM scheduler ensures that at most 16 cycles may pass between the clock cycle in which the cell is transferred to the lower input buffer latches, and the clock cycle when this cell is stored into the buffer RAM (4×2 cycles for possible read accesses, plus 3×2 cycles for possible other write accesses plus 2 cycles for the final write access). This modification to the input buffer operation does not complicate the circuit of figure 4.4 − two transfer-enable signals are needed which can be obtained by the output of 25th and 53d bits of the load enable shift register.

Figure 4.7 shows the schematic diagram of an input and output buffer cell along with their interface to the buffer RAM bit-lines. In order to achieve a 20 ns cycle time, the input, output, and cut-through busses are precharged during ϕ_1 and driven during ϕ_2. In this way, their operation is faster, and the size of the drivers in the buffer cell is smaller − that cell must be pitch-matched with the memory cell. The I/O buffer latches are dynamic, since the cell data need not stay in them for more than 1-1.5 μs. In order to retain the cell data polarity, the interface to the buffer RAM passes data as-is on write, while it inverts them on read (the buffer memory cell acts like as inverter). The simulation of the above circuits proved that they can operate with 20 ns cycle time. A 1.5 pF capacity on the input, output and cut-through busses can be precharged within 6 ns or less. A logic-zero value can discharge the input bus and get latched in the upper latches of the input buffer within 7 ns. A logic one value charges the internal node of the output buffer's first latch to 4.0 Volts within 4 ns − the above delays *include* a 1.5 ns delay for driving the control signals. A logic-one value in the input or output buffer latch can drive the cut-through or output bus low within the 8 ns ϕ_2 period. The value on the cut-through and output busses is

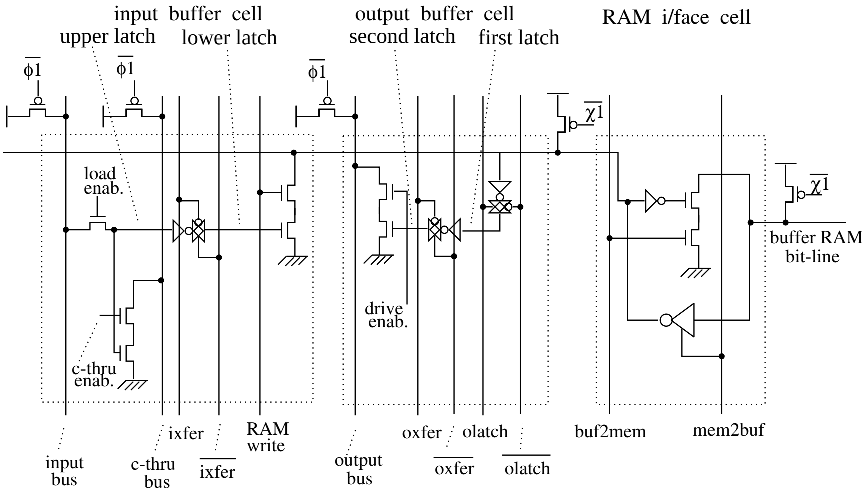

Figure 4.7: Input and output buffer schematics

latched at the end of ϕ_2 and the output crossbar operates during the subsequent cycle.

4.4 Scanning Hardware Implementation

The satisfactory function of our cell multiplexing mechanism depends mainly on the performance of the scanning memory. As described in chapter 3, the cell selection is made by content addressable accesses into the scanning memory. During a selection period (which may need to be as short as 800 ns) as many as 22 accesses to the scanning memory may be required – 6 accesses for checking the priority class of incoming cells and tokens, 12 accesses for locating a VC with a small frequency of service weight, and 4 accesses for determining the scanning priority level. This means that if the scanning memory is organized as a conventional content addressable memory, its cycle time has to be as short as 36 ns. This could be a major limitation for the multiplexing mechanism implementation since: *(i)* the content addressable memory access time is usually twice the access time of an ordinary memory, and *(ii)* in future versions of the switch chip with larger link throughput and total number of links, the available time interval for locating the next VC to service will be smaller and many more accesses due to incoming cell or token arrivals will be required. However, the peculiarities of the scanning algorithm and the design freedom provided by a full-custom VLSI technology, allow us to overcome these problems. The organization of the scanning memory solves problem *(i)* by effectively halving the content addressable memory access time. Furthermore, this organization allows the accesses required for checking the priority of incoming tokens or cells, to be performed in parallel with cell selection accesses – thus eliminating problem *(ii)* above. The basic content addressable memory cell used for the

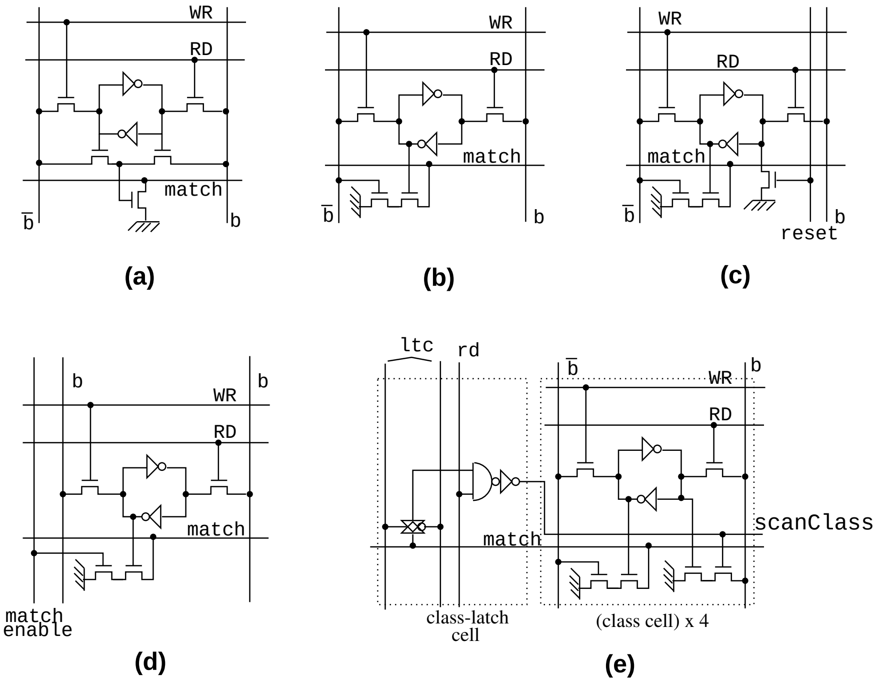

Figure 4.8: Content addressable memory cells: **(a)** conventional 9-transistors cell, **(b)** 8-transistors cell with 1 and don't care match capability, **(c)** with reset capability, **(d)** with concurrent match and write capability, and **(e)** with wired-OR read capability.

construction of the scanning memory is depicted in figure 4.8-**(b)**. We did not use the 9-transistor CAM cell of figure 4.8-**(a)** [Koo70], because the scanning algorithm always searches the content addressable memory only for logic one or don't-care logic values. Furthermore, the variation of the 9-transistor CAM cell for matching only logic-one and don't-care values occupies larger area than the one we use, because it requires an additional contact between diffusion and polysilicon.

The cell in figure 4.8-**(b)** can be used to build a content addressable memory, in which matching accesses can be performed in parallel with conventional read accesses – the *b* bit-line is not used for the match operation, and thus if the *RD* line is driven high by the memory decoder the value stored into the cell is output on the *b* bit line. This cell is used for the *frequency-of-service weight* field of the scanning memory.

Figure 4.8-**(c)** shows the same cell augmented with one more transistor which allows to clear all the bits in a column by asserting the *reset* line. This cell is used for the *valid* and *still-unvisited* fields of the scanning memory, which require such an asynchronous clear at system reset and at the end of the circular scan cycles respectively.

Figure 4.8-**(d)** illustrates a variation of the basic cell equipped with one additional *match-enable* line. Besides the capability for concurrent read and match operations this cell has the

capability for concurrent match and write operations. This cell is used for storing the *enabled (not-stopped)* and *full* fields of the scanning memory, where we need to write in parallel with matching accesses. There is a race condition in the operation of this cell. Whenever a matching access is performed in parallel with a write access, writing a logic-one value into the cell may result in partial discharging of the *match* line. This problem can be avoided either by making the bit-line drivers weak enough to write the new logic-one value after the match line has been discharged, or by using an additional cell which continues discharging the *match* line whenever a logic-one value is written in its adjacent *enabled* or *full* bits. Note that in such cases we are not interested in ‘‘write-through’’ match operations, since the logic-one value written in the cell can be considered during the next match operation.

Figure 4.8-**(e)** illustrates the variation of the basic cell which is used for the *priority-class* field. The use of this cell allows determining the current priority level in only 2 scanning memory cycles. During the first cycle a matching access is performed to find all the VCs that are *valid, enabled,* and *full,* regardless their *still-unvisited, frequency of service* and *class* fields. The values on the *match* lines are latched in special latches (figure 4.8-**(e)**, left). During the subsequent cycle the *scanClass* line is asserted, enabling all the lines that have matched previously to output their class value on the *b* bit line.

Figure 4.9 illustrates the organization of the scanning memory. The fields *valid, frequency-weight, priority-class, full,* and *enabled* are built from the CAM cells described above. The other blocks of the scanning memory are:

- *write-All*: 256 AND gates which drive the *WR* word lines of the fields left of this block. These fields of the scanning memory are written only when a VC is opened, while the fields to the right of this block can be written whenever a token or cell arrives into the switch and in parallel with the matching accesses.
- *NOR*: a 256-input pseudo-NMOS NOR the output of which indicates whether there is a match or not. We prefer to use a pseudo-NMOS NOR rather than a precharged one, because this implementation allows the NOR gate to operate in the same phase as the memory matching access.
- *PRIO*: 256 latches fed by the match lines along with a priority enforcer used to find the first matching entry, as described below.
- *DEC*: an 8-to-256 decoder with output enable used for accessing the scanning memory fields by address.
- *from-VCID, from-link*: a 256×8 and 256×2 6-transistor static memories.
- *cell-addr*: a memory which holds the addresses of the packets stored in the buffer RAM. The first 128 7-bit-wide words of this memory are built using ROM cells, and the rest 128 6-bit-wide words use 6-transistor static RAM cells (see § 3.4).

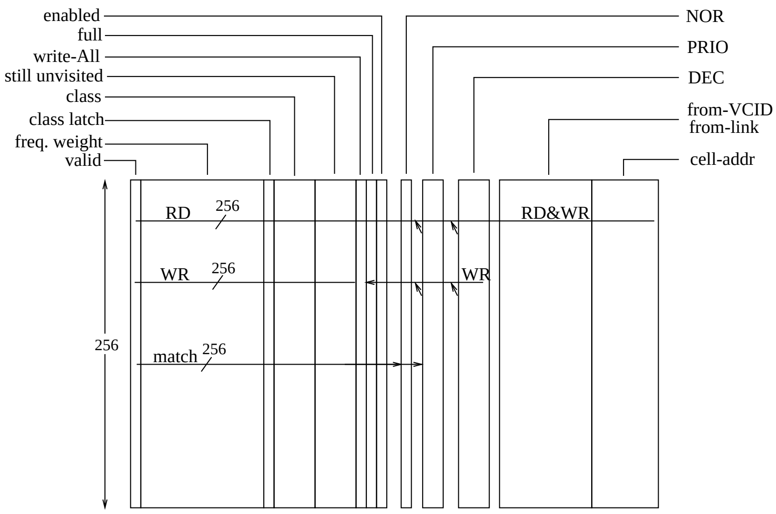

Figure 4.9: Scanning memory organization and floor-plan

Besides opening and closing VCs, three other types of accesses are performed to the scanning memory:

(i) whenever a cell selection is in progress the bit lines and the match-enable lines of appropriate fields are asserted by the scanning FSM and the scanning memory indicates whether no VCs or some VCs match.

(ii) whenever a flow control token or a cell arrives into the switch the decoder is used to read-out the priority of the corresponding VC and write or read its *enabled* and *full* bits.

(iii) as the last step of the selection process, the first matching VC is selected, using the priority enforcer, and its *cell-address, from-link,* and *from-VCID* fields are read out.

All but the last accesses are of type *(i)* and *(ii)*, performed *in parallel.* A single type *(iii)* access is performed at the end, as late as possible, when the memory access scheduler notifies this FSM that a decision must be made right away because this link's read operation out of the buffer RAM is imminent.

The simulation showed that type *(i)* and *(ii)* accesses can be repeated with a 40 ns cycle time. The 18 ns ψ_1 time interval is more than enough for precharging the 0.8 pF match lines and the 2 pF read bit-lines, in parallel with the scanning FSM and decoder operation. The ψ_2 phase and its subsequent non-overlap time interval (total 20 ns) is enough for performing a worst case (i.e. sterile) match access. The bit-lines or the match-enable lines are driven high, the match line discharges, and the output of an inverter (which amplifies the output voltage of the pseudo-NMOS NOR gate) switches within 18.5 ns. A read or write type *(ii)* access is performed within 15 ns. Type *(iii)* accesses involving the priority enforcer are more difficult. The heart of the

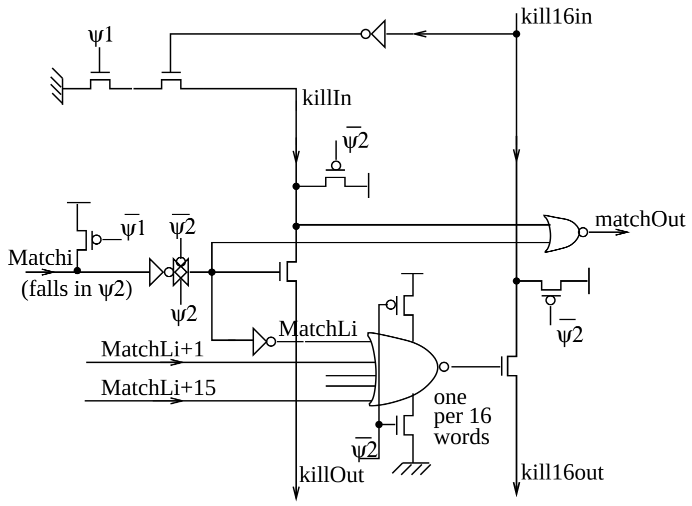

Figure 4.10: The scanning memory priority enforcer

priority enforcer 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 4.10 shows the main idea of the two-level Manchester chain that we use. It operates during ψ_1, i.e. 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 20 ns. This delay is comfortably short since it allows enough time for reading the cell address and writing the full and enabled bits of the matched VC during the subsequent ψ_2 phase.

Another important circuit for the implementation of the multiplexing mechanism is the cycle counter presented in § 3.5. The heart of this circuit is an incrementer which increments the current value of the cycle counter by 1 or 2 *beyond the LS sterile string* (see figure 3.9). Figure 4.11-**(a)** shows the basic cell of this incrementer. It consists of a manchester carry chain in which the carry is ''injected'' into the correct bit position beyond the LS sterile mask. The manchester chain is precharged during ψ_2 and operates during ψ_1. The signals ls_i and ls_{i-1}, – which are evaluated during the precharging phase as described below – indicate whether this bit is the first or second bit beyond the LS sterile mask. These signals are fed to the pull-down network of each cell and along with the add_1, add_2 (ψ_1 qualified) signals are used to inject the carry into the correct bit position. Note that in this implementation of the cycle counter the priority enforcer of figure 3.8 identifies the rightmost ''0'' digit. Thus when the add_2 signal is asserted the value of the first bit beyond the LS sterile string is always ''0'' and there is no danger for its C_{in} value to be affected by the pull down network of the next bit − it is obvious that we don't care if the C_{in} values of the LS sterile string are affected in any case.

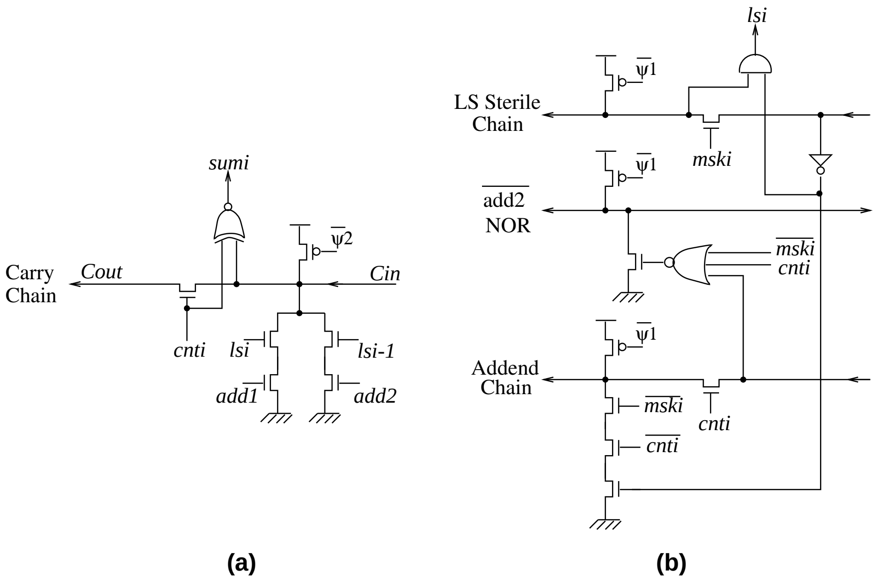

Figure 4.11: Cycle counter schematic diagram

Figure 4.11-**(b)** presents the circuit which determines the addend (1 or 2) and the position of the first bit beyond the LS sterile string. It consists of two manchester chains and a 12-bit precharged NOR gate which operate in a precharged domino fashion. The circuit is precharged during ψ_1. During ψ_2 the *LS Sterile String Chain* starts discharging from the least significant bit. All the nodes of the chain discharge till the first bit position beyond the LS sterile string − the msk_i signal is the value of the sterile mask at the corresponding bit position. If the cycle counter value at this position (cnt_i) is ''0'' the *addend chain* discharges until the next bit position with ''0'' value or the end of the chain. If this position is currently a sterile position this means that the next addend must be 2 in order for this position to be skipped. Thus, the transistor that pulls down the precharged NOR is turned on in order to assert the $\overline{add2}$ signal, which is latched and used during the subsequent ψ_1 phase.

The operation of the above circuit has been simulated with *verilog*. We have not performed *spice* simulations to determine its delay, but based on the results of the simulation of the priority enforcer we anticipate that a 12 bits cycle counter can easily operate in a 40 ns cycle time. Note that the circuits in figure 4.11 can be made significantly faster by using a carry skip circuit [ChSc90] and by substituting the three-transistor pull-down network of the *Addend Chain* through only one transistor controlled by a NOR gate.

The last block of the multiplexing hardware, the scanning FSM, will not present any implementation problem. A preliminary version of this FSM, which does not include the states required for VC-setup, has been simulated with *verilog*, along with a behavioral model of the scanning memory and a structural model of the cycle counter. The FSM will necessarily operate during phase ψ_1. It can be implemented either by a pseudo-NMOS or by a precharged self-timed PLA. To minimize its area and delay, the PLA will include only the logic required to implement state transitions – the controlling signals will be produced by random logic outside the PLA near the blocks that use them. To estimate its size, a pseudo-NMOS version has been built using *peg* [Hama86] and *mpla* [MaOb86]. The size that resulted is about $120^2 \mu m^2$, so we can infer that the final version of the PLA will be comfortably small and fast.

4.5 Implementation of Other Switch Chip Blocks

In this section we briefly present the blocks of the switch that have not yet been discussed in detail, along with some design alternatives and some reasonable estimates on their performance.

Of key importance for the switch performance are the routing table memories. The most suitable cell for their construction is the 6-transistor static RAM cell. The 3-transistor dynamic type cell can be used as well, but due to the refreshing overhead it requires it is not a very attractive alternative. However, using a dynamic type cell for the routing table memories must be carefully considered if larger VC capacity is required in future switch chip versions. The simulation of the 6-transistor cell we have laid out indicated that the routing table memory can easily operate with 40 ns cycle time. The bit lines are precharged during phase ψ_1, in parallel with the evaluation of the precharged address decoder, and the data in a cell are read during ψ_2, within 15 ns, *including* 3 ns for the word line pull-up. We can easily infer that the speed of the routing table memories will not impose any obstacle to the switch chip performance.

Due to lack of time, the outgoing link cross-bar has not yet been laid-out and simulated. However we anticipate that an implementation utilizing four precharged 4-to-1 8-bit multiplexers will not require more than a clock phase to operate. Furthermore, with careful design, the outgoing link cross-bar can be made to operate in domino fashion with the output and cut-through busses, thus minimizing the on-chip transmission delay. Although such a design may be feasible, it was considered too optimistic under the current two equal-phase clock discipline – only 8 ns are available for discharging the output buffer bus up to the outgoing link latch through the domino outgoing link multiplexer. Thus in the round-trip delay calculation (§ 3.8) we have conservatively considered 2 clock cycles of on-chip transmission delay.

Another circuit that has not been designed yet is the priority enforcer which implements the free list of the shared part of the buffer memory. This enforcer can be implemented using a circuit similar to the one used for the scanning memory priority enforcer. We anticipate that this 64 bit priority enforcer will have a cycle time shorter than the 40 ns cycle time of the buffer memory – it will be precharged during phase χ_1 and evaluated during phase χ_2, each time locating the next free row in the buffer memory.

Other minor blocks of the switch chip – e.g. the buffer RAM scheduler, the I/O buffer logic etc. – are not time and size critical, and can be implemented using standard cells or automatic logic synthesis. No effort has been made yet to determine the power and clock distribution scheme that will be used in the switch chip. However, the large size and operating frequency of the chip make these two issues very critical for its operation. The clock distribution must be designed very carefully in order to minimize phase differences between distant blocks of the chip. A tree structured clock distribution circuit is probably appropriate. The high signaling rate of the chip interface and the extensive use of precharging inside the chip will create serious switching noise problems. Providing two or more pins for separate pad power supply, and using an advanced package with reduced parasitic inductances will suppress switching noise generated from pad drivers. The switching noise generated by internal circuitry (which will consume about 0.5-0.7 Watts of power) can be bypassed by using an on-chip decoupling capacitor like the one demonstrated in [JoTa89].

4.6 Performance Simulation for the 1 μm technology of ES2

When this study was in its initial phase, there were no available simulation parameters for any actual 1 μm CMOS fabrication process. Consequently, considering a hypothetical but realistic 1 μm CMOS technology was the only way to prove the feasibility of the studied architecture. For the layout of the size critical blocks of the chip we have used the MOSIS Scalable-CMOS design rules assuming λ=0.5μm. This was a reasonable choice, since the SCMOS design rules can be scaled down to 1 μm. For the simulations with *spice* we have used the parameters of table 4.1, which were obtained by scaling down the MOSIS 2 μm typical process parameters. The use of such hypothetical process parameters, may create some doubts for the credibility of the results presented in this chapter. However, recently (June 91) European Silicon Structures (ES2) provided us the preliminary *spice* parameters of its 1 μm EPCD10 fabrication process†. This

† the parameters of this process cannot presented here because they are confidential

allowed us to verify the results presented in this chapter with the actual parameters of the technology for which our prototype is targeted.

The simulations of all the circuits have been repeated with the fast, typical, and slow ECPD10 *spice* parameters and the worst case parasitic device values. In summary, the simulations indicated that the circuits will be roughly 30 % faster in the best case, 5 % slower in the typical case, and 35 % slower in the worst case. Although that our initial *spice* parameters can thus be considered as slightly optimistic, our conclusions for the feasibility of the proposed architecture are not affected, since all the simulations were performed for the for the floor-plan of figure 4.1-**(a)**. Repeating the simulations, and using the floor-plan of figure 4.1-**(b)** for estimating the parasitic capacitances and resistances, indicated that this floor-plan arrangement is appropriate for achieving the desired circuit speed, even under the worst case fabrication process conditions. A buffer and scanning memory cycle time of 40 ns, can be trivialy achieved, since the bit-line capacitances will be almost halved. The routing table cycle time will be 40 ns even though the bit-line size is not changed in this floor-plan arrangement – precharging will be performed during χ_1 and access during χ_2 , thus increasing the time interval available for memory access. A more thoroughful calculation of the input, output, and cut-through busses (which are drawn in second level metal), indicated that the 1.5 pF parasitic capacitance value which was initially used was too pessimistic even with the worst case ECPD10 parameters. This capacitance will not be more than 0.8 pF and even with the worst case parameters it can be easily discharged within 6-8 ns by a slightly wider (1 μm) pull down transistor. The additional hardware and wiring required for this arrangement will be minimal. Doubling the number of buffer RAM bit line drivers requires negligible additional silicon area. Some more extra wiring and logic is required to the outgoing link multiplexing machines for handling the two distinct parts of the scanning memories, but we anticipate that the impact to the scanning hardware performance will be minimal.

Due to lack of the ECPD10 design rules we have not laid out the size critical blocks of the chip. However the layout of the buffer RAM and scanning memory cells using the ECPD15 1.5 μm design rules‡ (which as far as ES2 claims are almost compatible with the ECPD10 design rules), indicate that the chip size will be roughly the same.

In summary the fact that the implementation of the 50 MHz switch chip is feasible even with the ‘‘conservative’’ technology of ES2, constitutes a strong evidence for the feasibility and scalability of the proposed architecture.

‡ these cells were designed by G. Dimitriadis, and their layout is based on the SCMOS cells used to estimate the chip size

Chapter 5

Conclusion

Most of the existing approaches to build fast packet switches for BISDN have some or all of the following disadvantages: *(i)* they suffer from inherent scalability problems, *(ii)* they do not employ effective mechanisms for efficient and fair bandwidth allocation, or *(iii)* they do not support quality of service assurance.

In the context of the broadband network organization proposed in [Kate87], a switch chip implementing in hardware sophisticated buffer and bandwidth management techniques can be used a building block for constructing high speed ATM switching systems which solve to a large degree the above problems. Such switch chips will be connected via their bit parallel links to other identical chips or to fiber interface chips to form distributed or centralized switches of arbitrary topology and size. By adopting a buffer organization which provides dedicated and shared buffers to the virtual circuits, along with a back-pressure flow control mechanism, the chip can achieve continuous and full utilization of the available link throughput. By combining the above hardware organization with a weighted round robin cell scheduling policy (instead of a FIFO policy), the bandwidth allocation becomes fair and the network manager is provided with a powerful mechanism for efficiently combining traffic flows. Furthermore, using the cut-through facility, the communication latency of the delay sensitive traffic can be minimized.

The bandwidth requirements that a reasonable number of high speed links pose to the shared buffer memory, can be satisfied by using a wide word organization of this memory. The implementation of the sophisticated cell scheduling and queueing mechanism can be handled by using a content addressable memory, a special counter and an FSM implementing the selection mechanism. Arbitrating the link accesses to the common buffer memory can be done easily by utilizing a simple sequential circuit. The deterministic worst case round-trip delay of 3 cell times proves that the switch performance, even in the presence of congestion, is very satisfactory.

The layout and simulation of the critical blocks in terms of size and speed, clearly show that the proposed architecture is realizable with today's conventional 1 μm CMOS technology. More advanced fabrication technology (e.g. sub-micron CMOS or BiCMOS) can be used to implement larger and higher speed versions of the switch chip.

The goal of this work was to define and explore the feasibility of the switch chip architecture. Completing the chip design and layout, and comparing the results of testing the first prototypes with the simulation results presented here, will be a further way to prove the correctness of our approach. Extending the chip architecture with features such as multicasting and cell discarding in the shared buffer pool will substantially improve its performance, and it is necessary for the chip to become commercially interesting. More work is needed for implementing the fiber interface chips – especially their buffering and multiplexing mechanisms. The ideas presented here can apply as well in that case; the presence of external buffer memory, and the hierarchical grouping of VCs to VPs differentiates that design, though.

Much more work needs to be conducted for evaluating the performance of the proposed network organization. Some analytical results are presented in [KaSC91] but they need to be confirmed by detailed simulation experiments. Finally, designing and implementing the network management software which will take advantage of the mechanisms that this chip provides, is an area open for research.

We believe that through a coordinated effort the envisioned broadband network architecture can become a working reality.

REFERENCES

[AnAZ89] M.L. Anido, D.J. Allerton, and E.J. Zaluska, ‘‘A Three-Port/Three-Access Register File for Concurrent Processing and I/O Communication in a RISC-like Graphics Engine,’’ *IEEE Computer Architecture Symposium Proceedings*, May 1989.

[Arn89] E.A. Arnould, F.J. Bitz, E.C. Cooper, H.T. Kung, R.D. Sansom, and P.A. Steenkiste, ‘‘The Design of Nectar : A Network Backplane for Heterogeneous Multicomputers,’’ *ACM ASPLOS-III, Proceedings*, 1989.

[BCHW72] T.H. Beeforth, R.L. Crimsdale, F. Halsall, and D.J. Woolons, ‘‘Proposed Organization for Packet-Switched Data-Communication Networks,’’ *Proceedings of the IEEE*, vol. 119(12), pp. 1677-1682, December 1972.

[CCIT90] CCITT, *Recommendations on B-ISDN*, Geneve, January 1990.

[ChAT91] F.M. Chiussi, H. Amano, and F.A. Tobagi, ‘‘A 0.8-μm BiCMOS SEA-OF-GATES Implementation of the Tandem Banyan Fast Packet Switch,’’ *Proceedings of Custom Integrated Circuits Conference*, San Diego, May 12-15, 1991.

[ChSc90] P.K. Chan and M.D.F. Schlang, ‘‘Analysis and Design of CMOS Manchester Adders with Variable Carry Skip,’’ *IEEE Transactions on Computers*, vol. 39(8), August 1990.

[CoST88] J.-P. Coudreuse, W.D. Sincoskie, and J.S. Turner, ‘‘Guest Editorial in Broadband Packet Communications,’’ *IEEE Journal on Selected Areas in Communications*, vol. 6(9), pp. 1452-1454, December 1988.

[DeCS88] M. Devault, J. Conchennec, and M. Servel, ‘‘The ‘‘Prelude’’ ATD Experiment: Assessments and Future Prospects,’’ *IEEE Journal on Selected Areas in Communications*, vol. 6(8), pp. 1528-1537, December 1988.

[DeKS89] A. Demers, S. Keshav, and S. Shenker, ‘‘Analysis and Simulation of a Fair Queueing Algorithm,’’ *ACM SigComm Proceedings*, 1989.

[FrTa88] G. Frazier and Y. Tamir, ‘‘High Performance Multi Queue Buffers for VLSI Communication Switches,’’ *IEEE Computer Architecture Symposium Proceedings*, May 1988.

[Fuji83] R.M. Fujimoto, ‘‘VLSI Communication Components for Multicomputer Networks,’’ *CS Division Report No. UCB/CSD 83/136*, University of California, Berkeley, 1983.

[Gate89] Gateway Co., *Verilog-XL Reference Manual*, September 1989.

[GiLS90] J.N. Giacopelli, M. Littlewood, and W.D. Sincoskie, ‘‘Sunshine: A High Performance Self-Routing Broadband Packet Switch Architecture,’’ *Proc. of the International Switching Symposium (ISS’90)*, vol. 3(P21), pp. 123-129, Stockholm, Sweden, May 27-June 1, 1990.

[Hama86] G. Hamachi, ‘‘Designing Finite State Machines with PEG,’’ *Berkeley CAD Tools User’s Manual*, 1986.

[Horo87] M. Horowitz, P. Chow, D. Stark, R. Simoni, A. Salz, S. Przybylski, J. Henessey, G. Gulak, A. Agarwal, and J. Acken, ‘‘MIPS-X: A 20-MIPS Peak, 32-bit Microprocessor with On-Chip Cache,’’ *IEEE Journal of Solid-State Circuits*, vol. 22(5), October 1987.

[JoTa89] N.P. Joupi and J. Tang, ‘‘A 20 MIPS 32-bit CMOS Microprocessor with High Ratio of Sustained to Peak Performance,’’ *IEEE Journal of Solid-State Circuits*, vol. 24(8), October 1989.

[KaSC91] M. Katevenis, S. Sidiropoulos, and C. Courcoubetis, ‘‘Weighted Round-Robin Multiplexing in a General Purpose ATM Switch Chip,’’ *to appear in IEEE JSAC special issue on Large Scale ATM Switching Systems for BISDN*, Fall 1991.

[Kate87] M. Katevenis, ‘‘Fast Switching and Fair Control of Congested Flow in Broadband Networks,’’ *IEEE Journal on Selected Areas in Communications*, vol. 5(8), pp. 1315-1327, October 1987.

[KaVe89] G. Karlsson and M. Vetterli, ‘‘Packet Video and Its Integration into the Network Architecture,’’ *IEEE Journal on Selected Areas in Communications*, vol. 7(5), pp. 739-752, June 1989.

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

[KEOK89] H. Kuwahara, N. Endo, M. Ogino, and T. Kozaki, ‘‘Shared Buffer Memory Switch for an ATM Exchange,’’ *Proc. Int. Conf. on Communications*, Boston, MA, June 1989.

[Koo70] J.T. Koo, ‘‘Intergrated-Circuit Content-Addressable Memories,’’ *IEEE Journal of Solid-State Circuits*, vol. 5, pp. 208-215, Oct. 1970.

[KwTo91] T.C. Kwok and F.A. Tobagi, ‘‘The Tandem Banyan Switching Fabric: A Simple High-Performance Fast Packet Switch,’’ *Proceedings of Infocom’91*, Bal Harbour, Florida, April 9-11, 1991.

[LaTG90a] A.A. Lazar, A. Temple, and R. Gidron, ‘‘An Architecture for Intergrated Networks that Guarantees Quality of Service,’’ *International Journal of Digital and Analog Communication Systems*, vol. 3, pp. 229-238, April 1990.

[LaTG90b] A.A. Lazar, A. Temple, and R. Gidron, ‘‘MAGNET II: A Metropolitan Area Network Based on Asynchronous Time Sharing,’’ *IEEE Journal on Selected Areas in*

Communications, vol. 8(8), pp. 1528-1594, October 1990.

[MaOb86] R.N. Mayo and F.W. Obermeier, ‘‘Mpla - Technology Independent PLA Generator,’’ *Berkeley CAD Tools User’s Manual*, 1986.

[MaZa90] N.F. Maxemchuk and M. El Zarki, ‘‘Routing and Flow Control in High-Speed Wide-Area Networks,’’ *Proceedings of the IEEE*, vol. 78, pp. 204-221, January 1990.

[Mess90] D.G. Messerchmitt, ‘‘Synchronization in Digital System Design,’’ *IEEE Journal on Selected Areas in Communications*, vol. 8(8), pp. 1404-1419, Oct. 1990.

[Minz89] S.E. Minzer, ‘‘Broadband ISDN and Asynchronous Transfer Mode (ATM),’’ *IEEE Communications Magazine*, vol. 27, pp. 204-221, September 1989.

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

[Nage75] L. Nagel, ‘‘Spice2: A Computer Program to Simulate Semiconductor Circuits,’’ *ERL MEMO ERL-M520*, University California Berkeley, California, May 1975.

[OhON88] H. Ohnishi, T. Okada, and K. Noguchi, ‘‘Flow Control Schemes and Delay/Loss Tradeoff in ATM Networks,’’ *IEEE Journal on Selected Areas in Communication*, vol. 6(9), pp. 1609-1616, December 1988.

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

[Oust85] J.K. Ousterhout, G.T. Hamachi, R.N. Mayo, W.S. Scott, and G.S. Taylor, ‘‘The Magic VLSI Layout System,’’ *IEEE Design Test Comput.*, vol. 2(1), pp. 19-30, Feb. 1985.

[PfNo85] G.F. Pfister and V.A. Norton, ‘‘‘Hot Spot’ Contention and Combining in Multistage Interconnection Networks,’’ *IEEE Transactions on Computers*, vol. C-34(10), pp. 943-948, October 1985.

[SNSTI90] H. Suzuki, H. Nagano, T. Suzuki, T. Takeuchi, and S. Iwasaki, ‘‘Output-Buffer Switch Architecture for Asynchronous Transfer Mode,’’ *Proc. Int. Conf. on Communications*, Boston, MA, June 1989.

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

[Term86] C. Terman, ‘‘Esim - Event Driven Switch Level Simulator,’’ *Berkeley CAD Tools User’s Manual* , 1986.

[Toba90] F.A. Tobagi, ‘‘Fast Packet Switch Architectures For Broadband Intergrated Services Digital Networks,’’ *Proceedings of the IEEE*, vol. 78, pp. 133-167, January 1990.

[Turn88] J.S. Turner, ‘‘Design of a Broadcast Packet Switching Network,’’ *IEEE Transactions on Communications*, vol. 36(6), pp. 734-743, June 1988.

[Tyme81] L. Tymes, ‘‘Routing and Flow Control in TYMNET,’’ *IEEE Transactions on Communications*, vol. 29(4), pp. 392-398, April 1981.

[WaTo91] W. Wang and F.A. Tobagi, ‘‘The Christmas-Tree Switch: An Output Queueing Space-Divison Fast Packet Switch Based on Interleaving Distribution and Concentration Functions,’’ *Proceedings of Infocom’91*, Bal Harbour, Florida, April 9-11, 1991.

[WeEs85] N. Weste and K. Eshragian, *Principles of CMOS VLSI Design*, Addison-Wesley Publishing Co., 1985.

[YeHA87] Y.-S. Yeh, M. Hluchyj, and A. Acampora, ‘‘The Knockout Switch: A Simple, modular architecture for high performance packet switching,’’ *IEEE Journal on Selected Areas in Communications*, vol. 5(8), pp. 1274-1283, October 1987.