CHAPTER 6

SHARED-MEMORY SWITCHES

In shared-memory switches, all input and output ports have access to a common memory. In every cell time slot, all input ports can store incoming cells and all output ports can retrieve their outgoing cells (if any). A shared-memory switch works essentially as an output-buffered switches, and thereby achieving optimal throughput and delay performance. Furthermore, for a given cell loss rate, a shared memory switch using centralized memory management requires less amount of buffers than other switches. However, the switch size is limited by the memory read/write access time, within which N incoming and N outgoing cells in a time slot need to be accessed. As shown in the formula given below, the memory access cycle must be shorter than $1/(2N)$ of the cell slot, which is the ratio of the cell length and the link speed:

$$\text{Memory access cycle} \leq \frac{\text{cell length}}{2 \cdot N \cdot \text{link speed}}$$

For instance, with a cell slot of 2.83 μs (53-byte cells at the line rate of 149.76 Mbit/s, or 155.52 Mbit/s × 26/27) and with a memory cycle time of 10 ns, the switch size is limited to 141.

Several commercial ATM switch chip sets based on the shared memory architecture provide several tens Gbit/s capacity. Some people may argue that the memory density doubles every 18 months and memory saving by the shared-memory architecture is not that significant. However, since the memory used in the ATM switch requires high speed (e.g., 5–10 ns cycle time), they are expensive. Thus, reducing the total buffer size can considerably reduce the implementation cost. Some vendors' shared-memory switch chip sets can have the capability of integrating with other space switches to build a large-capacity switch (e.g., a few hundred Gbit/s). Although shared-memory switch has the advantage of saving buffer size, the buffer can be occupied by one or a few output ports that are congested

High Performance Switches and Routers, by H. Jonathan Chao and Bin Liu

and leaves no room to other cells destined for other output ports. Thus, there is normally a cap for the buffer size that can be used by any output port.

The following sections discuss different approaches to organizing the shared memory and necessary control logics, respectively. The basic idea of the shared-memory switch is to use logical queues to link the cells destined for the same output port. Section 6.1 describes this basic concept and the structure of the logical queues, and the pointer processing associated with writing and reading cells to and from the shared memory. Section 6.2 describes a different approach to implementing the shared-memory switch by using the content addressable memory (CAM) instead of the random access memory (RAM) as in most approaches. Although CAM is not as cost-effective and **as** fast as RAM, the idea of using CAM to implement the shared-memory switch is interesting because CAM approach eliminates the need of maintaining the logical queues. Since the switch size is limited by memory chip's speed constraint, several approaches have been proposed to increase the switch size, such as the space-time-space approach in Section 6.3, parallel shared memory switches in 6.4.3 and multistage shared memory switches in Section 6.4. Section 6.5 describes several shared-memory switch architectures to accommodate the multicasting capability.

6.1 LINKED LIST APPROACH

Figure 6.1 illustrates the concept of the shared-memory switch architecture. Cells arriving on all input lines are time-division multiplexed into a single stream, which is then converted to a parallel word stream and fed to the common memory for storage. Internally to the memory, cells are organized into separate logical queues, one for each output lines. Cells destined for the same output port are linked together in the same logical queue. On the other hand, an output stream of cells is formed by retrieving the head-of-line (HOL) cells from the output queues sequentially, one per queue; the output stream is then time-division demultiplexed, and cells are transmitted on the output lines. Each logical queue is confined by two pointers, the head pointer (HP) and the tail pointer (TP). The former points to the first cell of a logical queue, while the latter points to the last cell of a logical queue or to a vacant location to which the next arriving cell will be stored.

Figure 6.2 depicts a linked-list based shared-memory switch where a logical queue is maintained for each output to link all cells in the memory destined for the output. Each logical queue is essentially operated as a FIFO queue.

The switch operation is as follows. Incoming cells are time-division multiplexed to two synchronized streams: a stream of data cells to the memory, and a stream of the

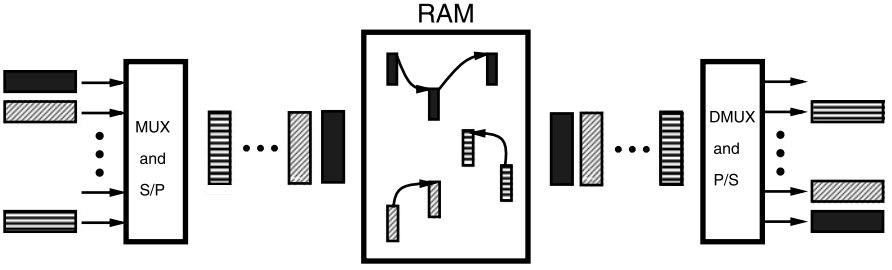


Figure 6.1 Logical queues in a shared-memory switch.

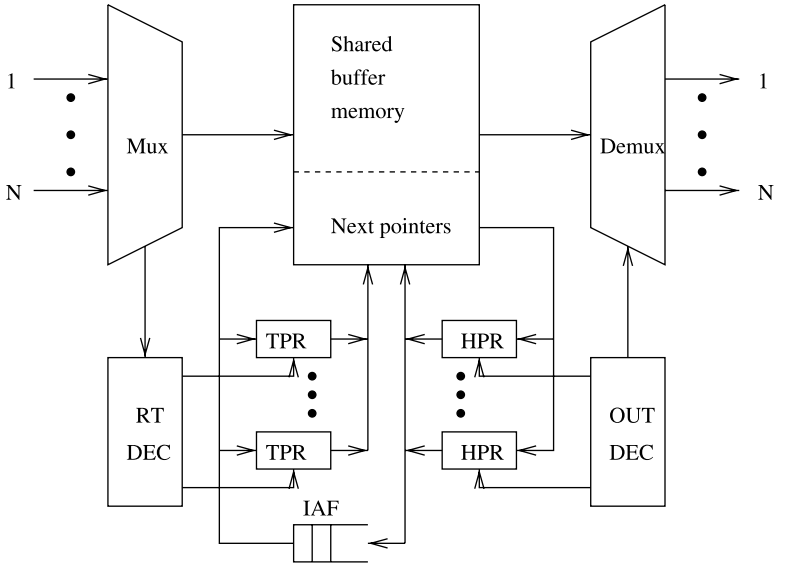


Figure 6.2 Basic structure of a linked list based shared-memory switch.

corresponding headers to the route decoder (RT DEC) for maintaining logical queues. The RT DEC then decodes the header stream one by one, accesses the corresponding tail pointer register (TPR), and triggers the WRITE process of the logical queue. An idle address of the cell memory is simultaneously read from an idle address FIFO (IAF) that holds all vacant cell locations of the shared memory.[1] This is the next address pointer in the cell memory as the address for storing the next arriving cell. This address is also put into the corresponding TPR to update the logical queue. Figure 6.3 shows two logical queues, where next pointers (NPs) are used to link the cells in the same logical queue. TPR 0 and HPR 0 (head pointer register) correspond to output port #0, where $(n-1)$ cells are linked together. TPR 1 and HPR 1 correspond to output port #1, The end of each logical queue indicates the address into which the next cell to the output will be written.

The READ process of the switch is the reverse of the WRITE process. The cells pointed by the HPRs (heads of each logical queue) are read out of the memory one by one. The resulting cell stream is demultiplexed into the outputs of the switch. The addresses of the leaving cells then become idle, and will be put into the IAF.

Figure 6.4 illustrates an example of adding and deleting a cell from a logical queue. In this example, the TP points to a vacant location to which the next arriving cell will be stored. When a cell arrives, it is stored in the cell memory at a vacant location pointed by

[1] Since FIFO chips are more expensive than memory chips, an alternative to implement the IAF is to store vacant cells' addresses in memory chips, rather than in FIFO chips, and link the addresses in a logical queue. However, this approach requires more memory access cycles in each cell slot than using FIFO chips approach and thus can only support a smaller switch size.

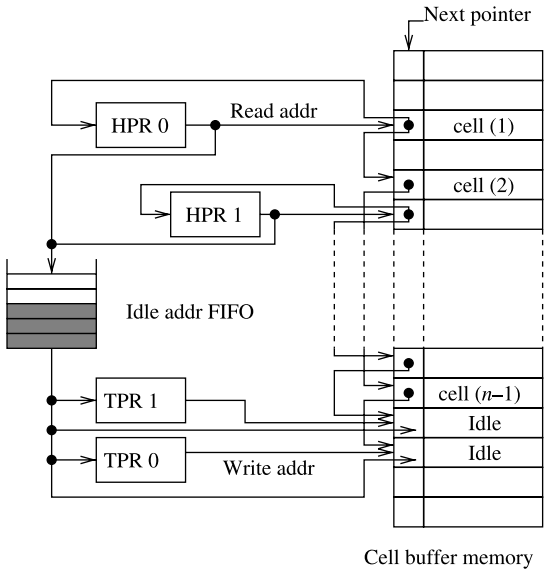


Figure 6.3 Two linked-list logical queues.

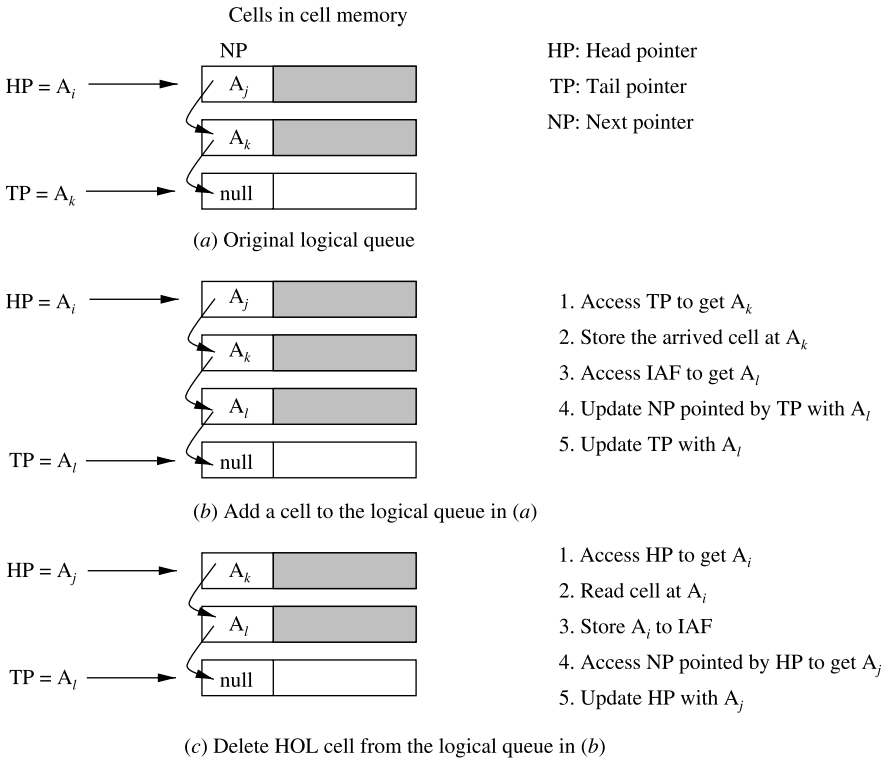


Figure 6.4 Adding and deleting a cell in a logical queue for the tail pointer pointing to a vacant location.

the TP (A_k in Figure 6.4*a*). An empty location is obtained from the IAF (e.g., A_l). The NP field pointed by the TP is changed from null to the next arriving cell's address (A_l in Figure 6.4*b*). After that, the TP is replaced by the next arriving cell's address (A_l). When a cell is to be transmitted, the HOL cell of the logical queue pointed by the HP (A_i in Figure 6.4*b*) is accessed for transmission. A cell to which the NP points (A_j in Figure 6.4*b*) becomes the HOL cell. As a result, the HP is updated with the new HOL cell's address (A_j in Figure 6.4*c*). Meanwhile, the vacant location (A_i) is put into in the IAF for use by future arriving cells.

Figure 6.5 illustrates an example of adding and deleting a cell in a logical queue for the tail pointer pointing to the last cell of the queue. The writing procedures are different in Figures 6.4 and 6.5, while the reading procedures are identical. While the approach in Figure 6.4 may have an advantage of better applying parallelism for pointer processing, it wastes one cell buffer for each logical queue. For a practical shared memory switch with 32 or 64 ports, the wasted buffer space is negligible compared with the total buffer size of several tens of thousands. However, if the logical queues are arranged on a per connection basis (e.g., in packet fair queueing scheduling), the wasted buffer space can be quite large as the number of connections can be several tens of thousand. As a result, the approach in Figure 6.5 will be a better choice in this situation.

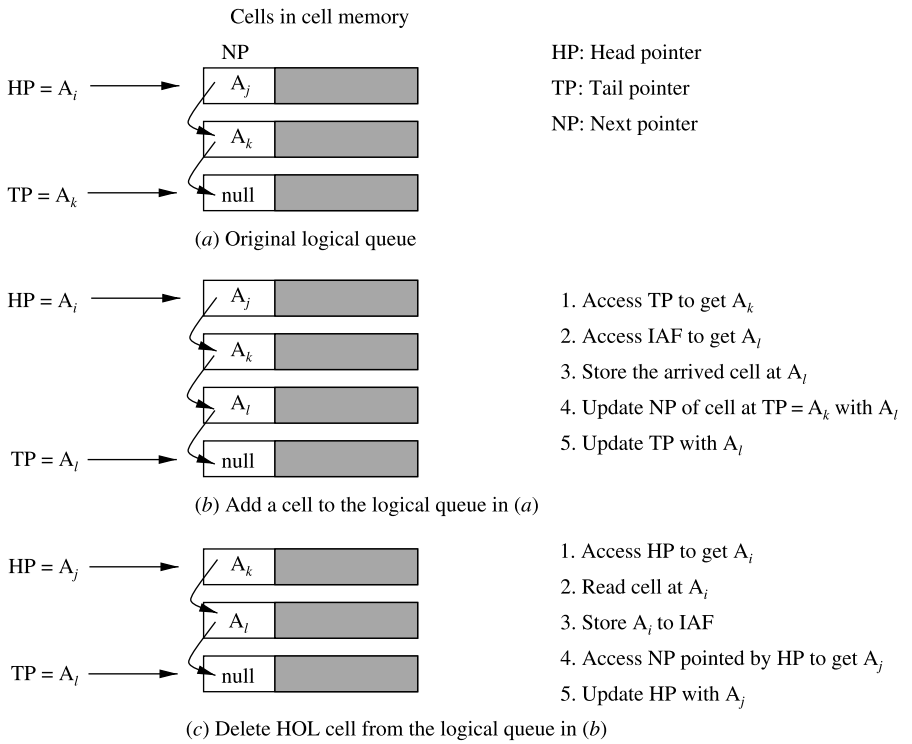


Figure 6.5 Adding and deleting a cell in a logical queue for the tail pointer pointing to the last cell of the queue.

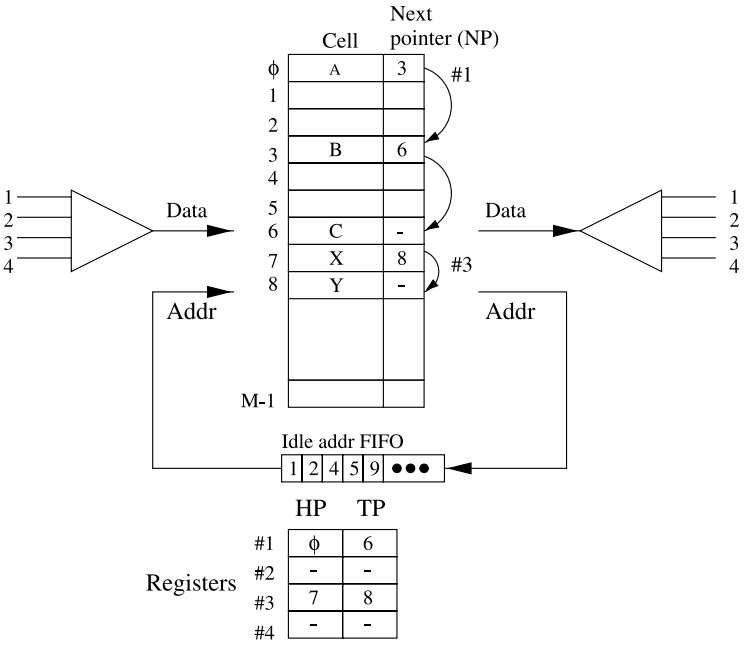


Figure 6.6 Example of a 4 × 4 shared-memory switch.

As shown in Figures 6.4 and 6.5, each write and read operation requires five memory access cycles.[2] For an $N \times N$ switch, the number of memory access cycles in each time slot is $10N$. For a time slot of 2.83 μs and a memory cycle time of 10 ns, the maximum number of switch ports is 28, which is much smaller than the number when considering only the time constraint on the data path (e.g., it was shown at the beginning of the chapter that using the same memory technology, the switch size can be 141. That is because we did not consider the time spent on updating the pointers of the logical queues). As a result, pipeline and parallelism techniques are usually employed to reduce the required memory cycles in each time slot. For instance, the pointer updating and cell storage/access can be executed simultaneously. Or, the pointer registers can be stored inside a pointer processor instead of at an external memory, thus reducing the access time and increasing parallelism of pointer operations.

Figure 6.6 shows an example of a 4 × 4 shared-memory switch. In this example, the TP points to the last cell of a logical queue. Cells A, B, and C destined for output port #1 form a logical queue with HP = 0 and TP = 6 and are linked by NPs shown in the figure. Cells X and Y destined for output port #3 form another logical queue with HP = 7 and TP = 8. Assuming a new cell D destined for output #1 arrives, it will be written to memory location 1 (this address is provided by the IAF). Both the NP at location 6 and TP1 are updated with value 1. When a cell from output #1 is to be read for transmission, HP1 is first accessed to locate the HOL cell (cell A at location 0). Once it is transmitted, HP1 is updated with the NP at location 0, which is 3.

[2]Here, we assume the entire cell can be written to the shared memory in one memory cycle. In real implementation, cells are usually written to the memory in multiple cycles. The number of cycles depends on the memory bus width and the cell size.

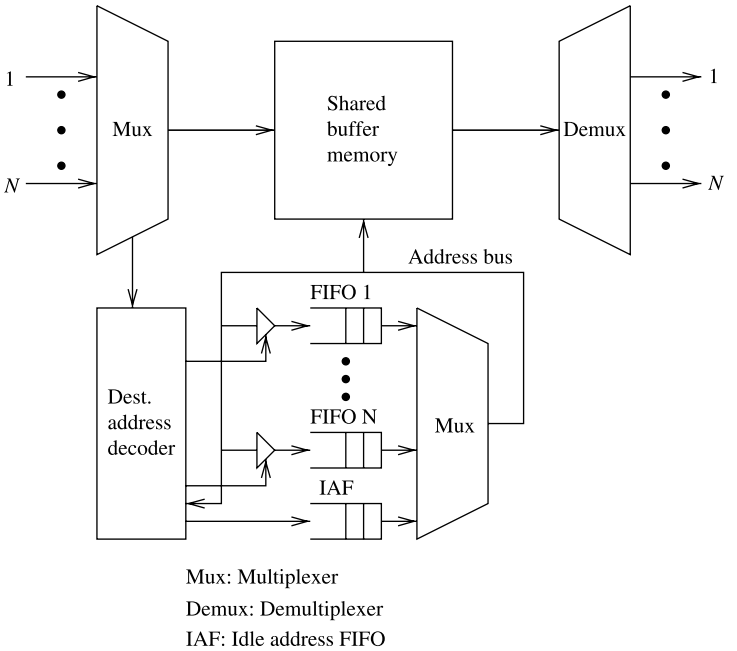
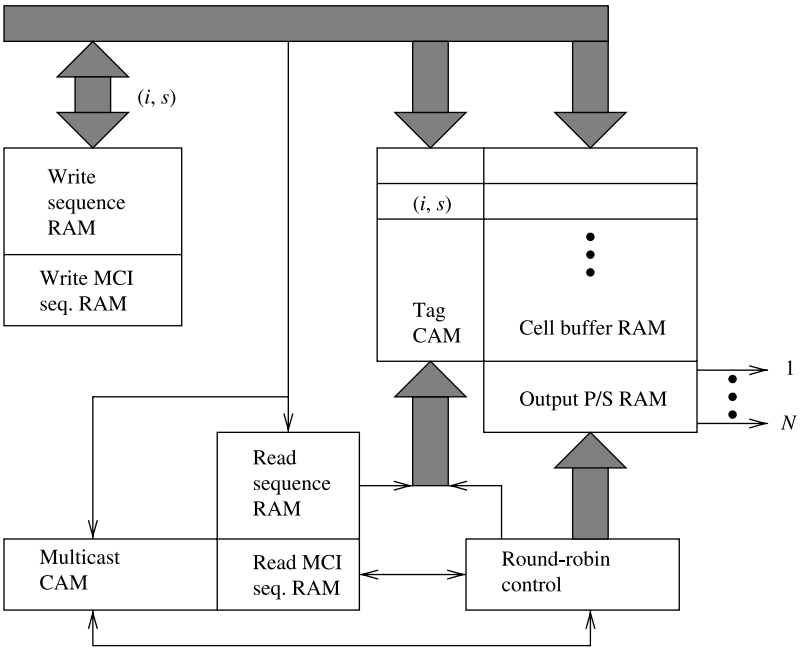


Figure 6.7 Using FIFOs to maintain the logical queues.

Alternatively, the logical queues can be organized with dedicated FIFO queues, one for each output port [1]. The switch architecture is shown in Figure 6.7. The performance is the same as using linked lists. The necessary amount of buffering is higher due to the buffering required for the FIFO queues, but the implementation is simpler and the multicasting and priority control easier to implement. For instance, when an incoming cell is broadcast to all output ports, it requires at least N pointer updates in Figure 6.2, while in Figure 6.7 the new address of the incoming cell can be written to all FIFO queues at the same time. The size of each FIFO in Figure 6.7 is $M \times \log M$ while the size of each pointer register in Figure 6.2 is only $\log M$, where M is the number of cells that can be stored in the shared memory. This approach provides better reliability than the linked-list approach (Fig. 6.2) in that if the cell address stored in FIFO is corrupted due to hardware failure, only one cell is affected. However, in the linked-list approach, if the pointer in the linked list is corrupted, cells in the remaining list will be either routed to an incorrect output port, or never be accessed.

6.2 CONTENT ADDRESSABLE MEMORY APPROACH

In the CAM-based switch [2], the shared-buffer RAM is replaced by a CAM/RAM structure, where the RAM stores the cell and the CAM stores a tag used to reference the cell. This approach eliminates the need of maintaining the logical queues. A cell can be uniquely identified by its output port number and a sequence number, and these together constitute the tag. A cell is read by searching for the desired tag. Figure 6.8 shows the switch architecture. Write circuits serve input ports and read circuits serve output ports, both on a round-robin



Tag: (Output address, sequence number), e.g., (i, s)

MCI: Multicast connection identifier

Figure 6.8 Basic architecture of a CAM-based shared-memory switch.

basis as follows:

For a **write**:

(1) Read the write sequence number $WS[i]$ from the write sequence RAM (WS-RAM) (corresponding to the destination port i), and use this value (s) for the cell's tag **i**, **s**.
(2) Search the tag CAM for the first empty location **emp**.
(3) Write the cell into the buffer (**B[emp]** = cell), and **i**, **s** into the associated tag.
(4) Increment the sequence number s by one and update $WS[i]$ with $(s + 1)$.

For a **read**:

(1) Read the read sequence number RS[j] from the read sequence RAM (RS-RAM) (corresponding to the destination port j), say t.
(2) Search for the tag with the value **j**, **t**.
(3) Read the cell in the buffer associated with the tag with the value **j**, **t**.
(4) Increment the sequence number t by one and update $RS[i]$ with $(t + 1)$.

This technique replaces link storage indirectly with content-addressable tag storage, and read/write pointers/registers with sequence numbers. A single extra 'valid' bit to identify if buffer is empty is added to each tag, effectively replacing the idle address FIFO (IAF) and its pointers. No address decoder is required in the CAM/RAM buffer. Moreover, since

TABLE 6.1 Comparison of Linked List and CAM-Access

	Linked List	CAM-Access
Cell storage (decode/encode) bits	RAM (decode) $256 \times 424 = 108{,}544$	CAM/RAM (neither) $256 \times 424 = 108{,}544$
Look-up bits	Link:RAM $256 \times 8 = 2048$	Tag:CAM $256 \times (4 + 7) = 2{,}816$
Write and read reference (queue length checking) bits	Address registers (additional counters) $2 \times 16 \times 8 = 256$	Sequence number registers (compare W and R numbers) $2 \times 16 \times 7 = 224$
Idle address storage (additional overhead) bits	IAF (pointer maintenance, extra memory block) $256 \times 8 = 2048$	CAM valid bit (none) $256 \times 1 = 256$
Total bits	112,896	111,840

Sample switch size is 16×16 with $2^8 = 256$ cell buffer capacity; seven-bit sequence numbers are used.

the CAM does not output the address of tag matches ('hits'), it requires no address encoder (usually a source of CAM area overhead). Both linked list address registers and CAM-access sequence number registers must be initialized to known values on power-up. If it is necessary to monitor the length of each queue, additional counters are required for the linked list case while sequence numbers can simply be subtracted in the CAM-access case. Table 6.1 summarizes this comparison, and provides bit counts for an example single-chip configuration. Although the number of bits in the linked list and CAM-access approaches are comparable, the latter is less attractive due to the slower speed and higher implementation cost for the CAM chip.

6.3 SPACE-TIME-SPACE APPROACH

Figure 6.9 depicts an 8×8 example to show the basic configuration of the space-time-space (STS)-type shared-memory switch [3]. Separate buffer memories are shared among all input and output ports via crosspoint space-division switches. The multiplexing and demultiplexing stages in the traditional shared-memory switch are replaced with crosspoint space switches, thus the resulting structure is referred to STS-type. Since there is no time-division multiplexing, the required memory access speed may be drastically reduced.

The WRITE process is as follows. The destination of each incoming cell is inspected in a header detector, and is forwarded to the control block that controls the input-side space switch and thus the connection between the input ports and the buffer memories. As the number of shared-buffer memories (SBMs) is equal to or greater than the number of input ports, each incoming cell can surely be written into an SBM as far as no SBM is full. In order to realize the buffer sharing effectively, cells are written to the least occupied SBM first and the most occupied SBM last. When a cell is written into an SBM, its position (the SBM number and the address of the cell in the SBM) is queued in the address queue. The read address selector picks out the first address from each address queue and controls the output-side space switch to connect the picked cells (SBMs) with the corresponding output

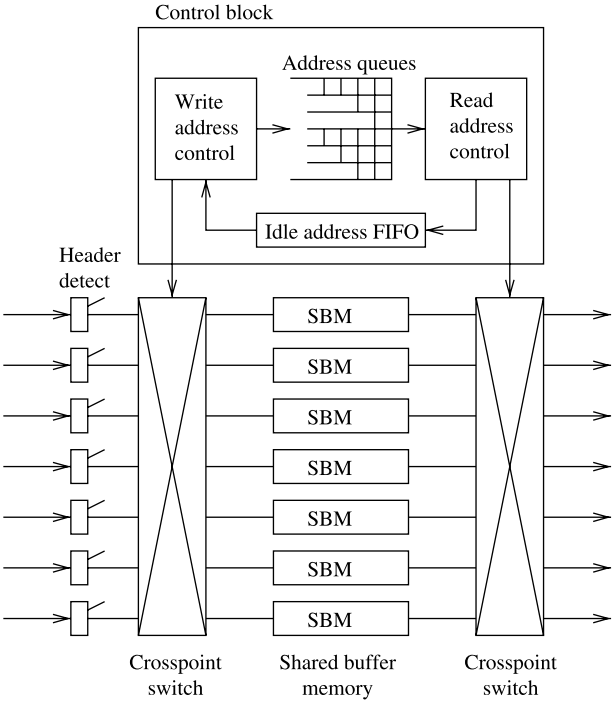


Figure 6.9 Basic configuration of STS-type shared-memory ATM switch.

ports. It may occur that two or more cells picked for different output ports are from the same SBM. An example is shown in Figure 6.10. Thus, to increase switch's throughput it requires some kind of internal speedup to allow more than one cell to be read out from an SBM in every cell slot. Another disadvantage of this switch is the requirement of searching for the least occupied SBM (may need multiple searches in a time slot), which may cause a system bottleneck when the number of SBMs is large.

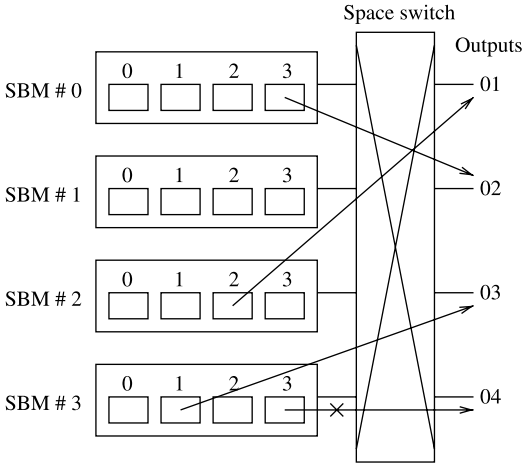


Figure 6.10 Blocking in STS-type shared-memory switch.

6.4 SCALING THE SHARED-MEMORY SWITCHES

Since single shared memory switch is hard to scale to much larger capacity, it is a feasible approach to connect multiple shared memory blocks together in a multi-stage topology. The most common multi-stage topology is the three-stage Clos network (Fig. 6.11). There, the blocks in adjacent stages are fully connected.

In practical, several switch architectures are proposed which interconnected small-scale shared-memory switch modules with a multi-stage network to build a large-scale switch. Among them are Washington University Gigabit Switch [4], Concentrator-based Growable Switch Architecture [5], Multinet switch [6], Siemens switch [7], and Alcatel switch [8].

6.4.1 Washington University Gigabit Switch

Turner proposed an ATM switch architecture called the Washington University Gigabit Switch (WUGS) [4]. The overall architecture is shown in Figure 6.12. It consists of three main components; the input port processors (IPPs), the output port processors (OPPs), and

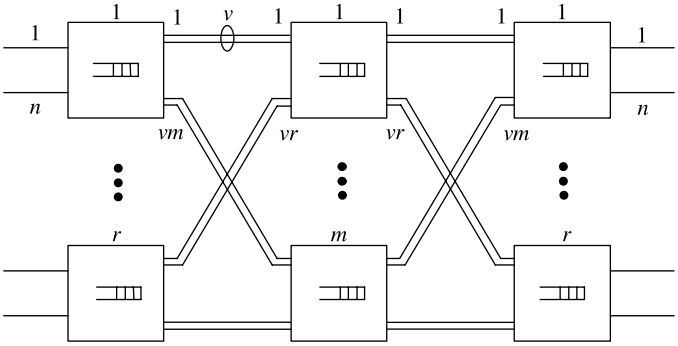


Figure 6.11 Three-stage switching fabric of shared-memory blocks.

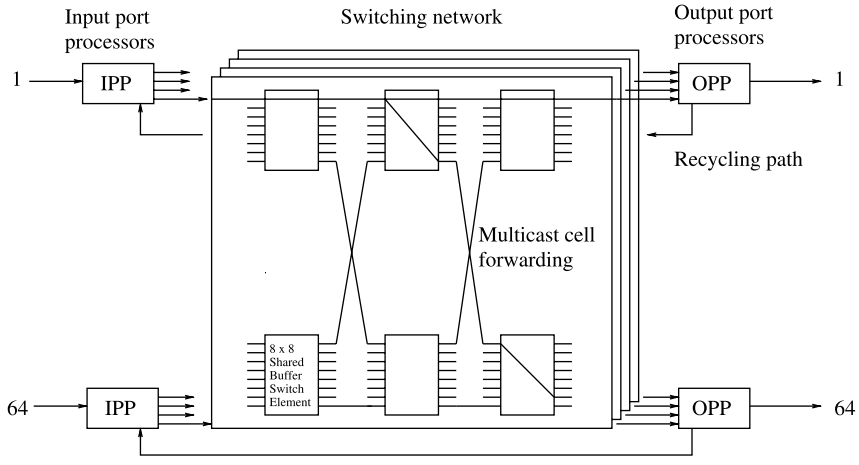


Figure 6.12 Washington university gigabit switch (WUGS) (©1997 IEEE).

the central switching network. The IPP receives cells from the incoming links, buffers them while awaiting transmission through the central switching network and performs the virtual path/circuit translation required to route cells to their proper outputs. The OPP resequences cells received from the switching network and queues them while they await transmission on the outgoing link. This resequencing operation increases the delay and implementation complexity. Each OPP is connected to its corresponding IPP, providing the ability to recycle cells belonging to multicast connections. The central switching network is made up of switching elements (SEs) with eight inputs and eight outputs and a common buffer to resolve local contention. The SEs switch cells to the proper output using information contained in the cell header or distribute cells dynamically to provide load balancing. Adjacent switch elements employ a simple hardware flow control mechanism to regulate the flow of cells between successive stages, eliminating the possibility of cell loss within the switching network.

The switching network uses a Benes network topology. The Benes network extends to arbitrarily large configurations by the way of a recursive expansion. Figure 6.12 shows a 64 port Benes network. A 512 port network can be constructed by taking eight copies of the 64 port network and adding the first and fifth stage on either side, with 64 switch elements a copy. Output j of the ith switch element in the first stage is then connected to input i of the jth 64 port network. Similarly, output j of the ith 64-port network is connected to input i of the jth switch element in the fifth stage. Repeating in this way, any large size network can be obtained. For $N = 8^k$, the Benes network constructed using eight port switch elements has $(2k - 1)$ stages. Since $k = \log_8 N$, the number of switch elements scales in proportion to $N \log_8 N$, which is the best possible scaling characteristic. In [4], it is shown that when dynamic load distribution is performed in the first $(k - 1)$ stages of the Benes network, the load on the internal data paths of the switching network cannot exceed the load on the external ports; that is, the network achieves ideal load balancing. This is true for both point-to-point and multipoint traffic.

6.4.2 Concentrator-Based Growable Switch Architecture

A concentrator-based growable switch architecture is shown in Figure 6.13 [5]. The 8×8 output ATM switches (with the shared-memory structure) are each preceded by an $N \times 8$ concentrator. The $N \times 8$ concentrator is preceded by a front-end broadcast network (i.e., a memoryless cell distribution network). The concentrators are preceded by address filters that only accept cells that are destined to its group of dedicated outputs. The valid cells in each concentrator are then buffered for a FIFO operation. In other words, each concentrator is an N-input eight-output FIFO buffer. The actual ATM cell switching is performed in the ATM Output Switch. Obviously, this is increasingly challenging as N becomes large. For instance, a 512×8 concentrator can be built using a column of eight 64×8 concentrators followed by another 64×8 concentrator. At 2.5 Gbit/s per port, a 64×64 switch has a capacity of 160 Gbit/s, and a 512×512 switch has a capacity of 1.28 Tbit/s.

6.4.3 Parallel Shared-Memory Switches

A parallel shared-memory structure is given in Figure 6.14 [9]. N input ports are divided into $\lceil N/n \rceil$ groups, with each group of n input ports. There are in total $\lceil N/n \rceil^2$ shared-memory switch modules, each with n ports, connected in parallel. $\lceil N/n \rceil^2$ switch modules are divided into $\lceil N/n \rceil$ groups with $\lceil N/n \rceil$ blocks in each group (as the dotted area in

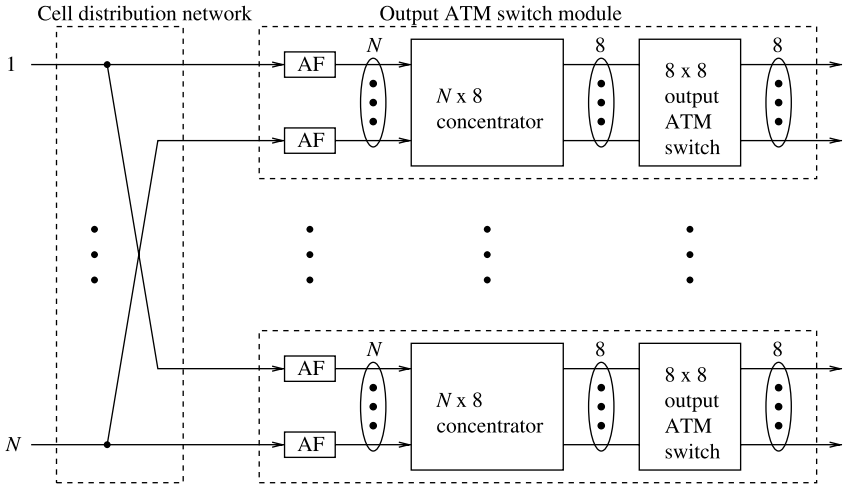


Figure 6.13 Concentrator-based growable switch architecture.

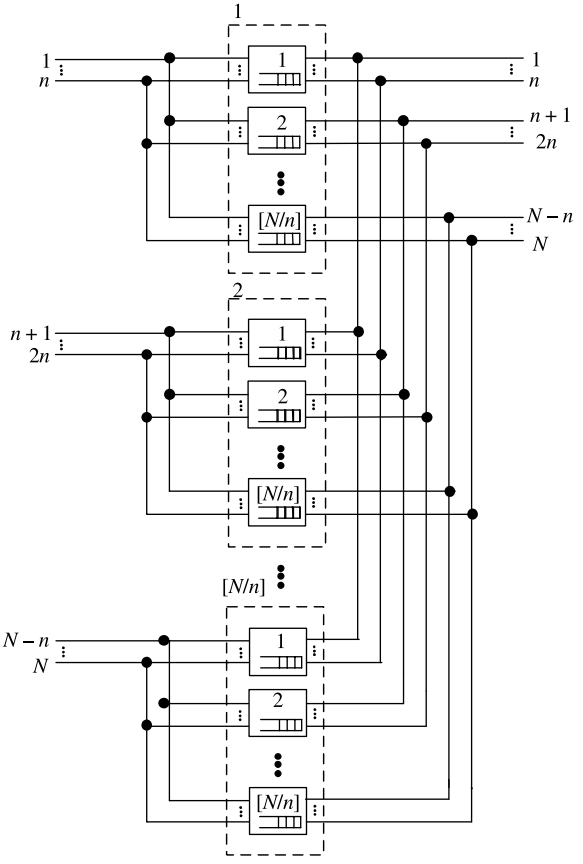


Figure 6.14 Switching fabric with parallel shared-memory switches.

Fig. 6.14). Each group of n input ports are connected to each of $\lceil N/n \rceil$ switch modules, as shown in the figure. At the output side, each output group (n ports) is connected to the outputs of $\lceil N/n \rceil$ switch modules, as shown in the figure.

In this parallel architecture, arrival packets are stored in one of its connected shared-memory switch modules that contain the output port that the packets are destined for. For instance, packets from the ith input port group destined for jth output port group are stored in the jth switch module in ith shared memory group. As a result, incoming traffic is partitioned into $\lceil N/n \rceil^2$ groups and each is handled by a switch module. This parallel switch architecture allows each shared-memory switch module operating $\lceil N/n \rceil$ times slower than a single $N \times N$ shared-memory switch. However, more buffer capacity is required to guarantee the same cell loss rate of a single $N \times N$ shared-memory switch due to less sharing efficiency.

6.5 MULTICAST SHARED-MEMORY SWITCHES

Of the various ATM switch architectures, the shared-memory switch provides superior benefits compared to others. Since its memory space is shared among its switch ports, it achieves high buffer utilization efficiency. Under conditions of identical memory size, the cell-loss probability of the shared-memory switches is smaller than output-buffered switches. Additionally, multicast operations can be easily supported by this switch architecture. In this section, some classes of multicast shared-memory switches are described along with their advantages and disadvantages.

6.5.1 Shared-Memory Switch with a Multicast Logical Queue

The simplest way to implement multicasting in a shared-memory switch is to link all multicasting cells in a logical queue as shown in Figure 6.15. The advantage of this approach is the number of logical queues that need to be updated in every cell slot is minimized. The multicast routing table stores routing information, such as a bit-map of output ports to which the multicast cells are routed. In this example, multicast cells always have higher priority than unicast cells. It is called strict priority, where unicast cells will be served only when the multicast logical queue is empty or the multicast HOL cell is not destined for the output port for which multicast cells are destined. Of course, there can be other service policies between unicast and multicast cells than the strict priority. For instance, they can be served in a round-robin manner, or a weighted round-robin manner with the weight depending on, for example, the ratio of unicast and multicast traffic load. However, the major disadvantage of this approach is there may be HOL blocking for the multicast logical queue when the schemes other than the strict priority are used. This is because when the multicast HOL cell is blocked because of yielding its turn to unicast cells, it may block other multicast cells that are behind it which could otherwise have been transmitted to some idle outputs.

6.5.2 Shared-Memory Switch with Cell Copy

In this approach, a multicast cell is replicated into multiple copies for all its multicast connections by a cell-copy circuit located before the routing switch. Bianchini and Kim [10] proposed a nonblocking cell-copy circuit for multicast purposes in an ATM switch.

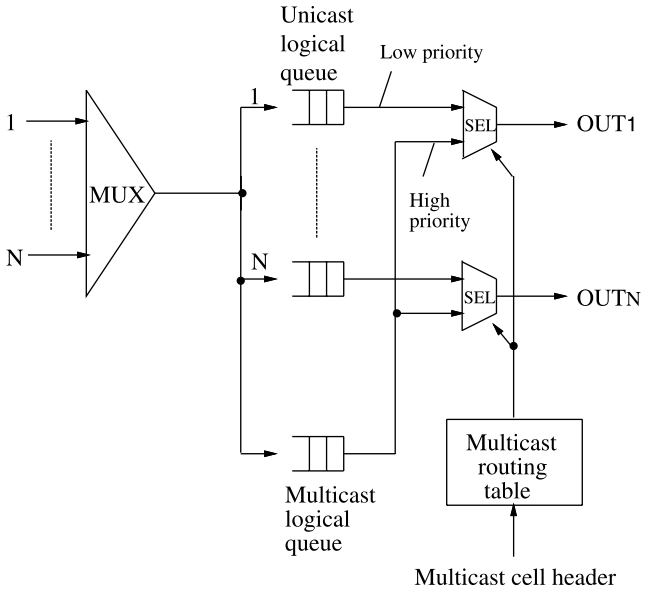


Figure 6.15 A shared-memory ATM switch with a multicast logical queue.

Each copy is stored in the SBM. The address for each copy is queued in output address queues (AQs).

Figure 6.16 shows an example of a shared-memory switch with a cell-copy circuit. A multicast cell arrives and is replicated into three copies for output ports 0, 1, and 3. These three

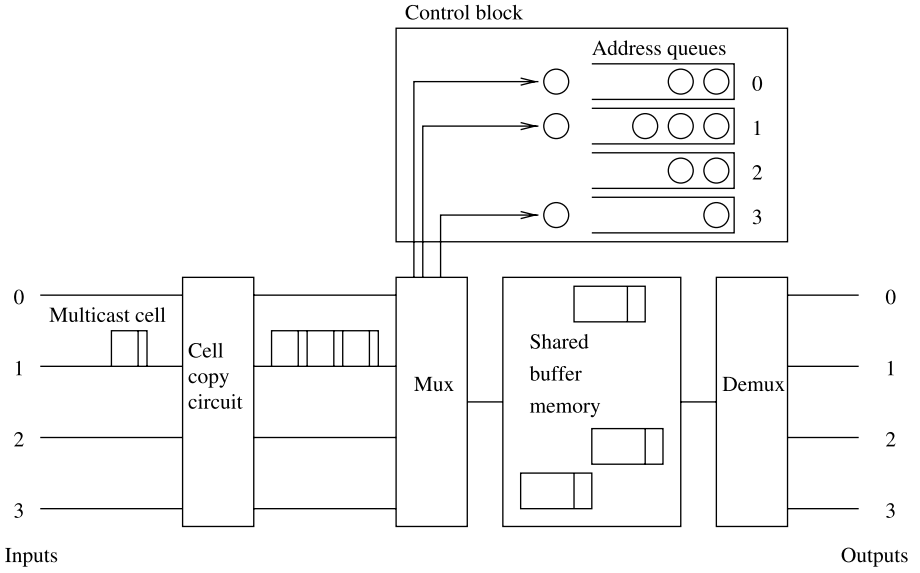


Figure 6.16 Shared-memory ATM switch with cell-copy circuit.

copies of the cell are stored at different locations of the shared buffer memory. Meanwhile, their addresses are stored at corresponding address queues. In this scheme, after replicating the multicast cell, each copy is treated the same way as a unicast cell. It provides fairness among cells. The major disadvantage of this approach is that, in the worst case, N cells might be replicated to N^2 cells. Thus, the number of replicated cells to the SBM in each switch cycle could be $O(N^2)$. Since only at most N cells could be transmitted, this would result in storing $O(N^2)$ cells in each cycle. For a finite memory space, it would result in considerable cell loss. The replication of multicast cells would also require $O(N^2)$ cells to be written to the SBM, which will further limit the switch size from the memory speed. Moreover, adding the cell-copy circuit in the switch increases hardware complexity.

6.5.3 Shared-Memory Switch with Address Copy

Saito et al. [11] have proposed a scheme that required less memory for the same cell loss probability. A similar architecture was also proposed by Schultz and Gulak [12] in the same year. In these schemes, an address-copy circuit is used in the controller. When a multicast cell arrives, only a single copy is stored in the shared buffer memory (SBM). Its address is copied by the address-copy circuit and queued into multiple address queues, as shown in Figure 6.17. The multicast cell will be read multiple times to different output ports. Once completed, the cell's address becomes available for the next arriving cells.

In this architecture, multicast cell counters (MCCs) are required to hold the number of copies of each multicast cell. The MCC values are loaded as new cells (unicast or multicast). They are decreased by one each time a copy of a cell is read out from the SBM.

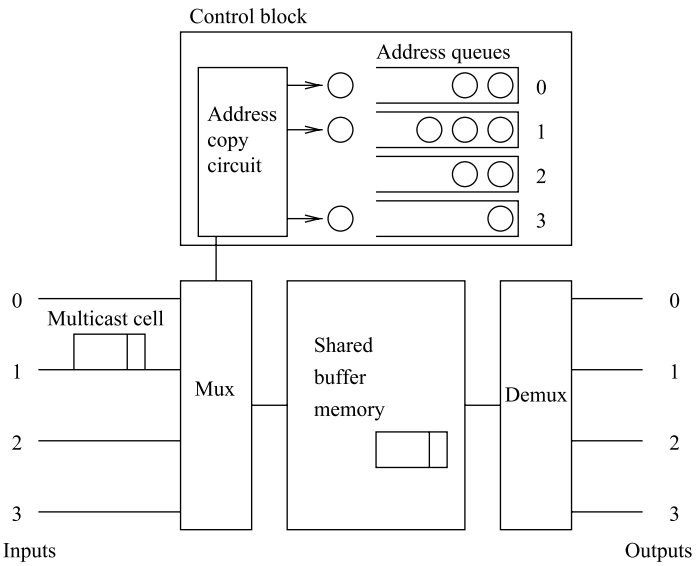


Figure 6.17 Shared-memory ATM switch with address-copy circuit.

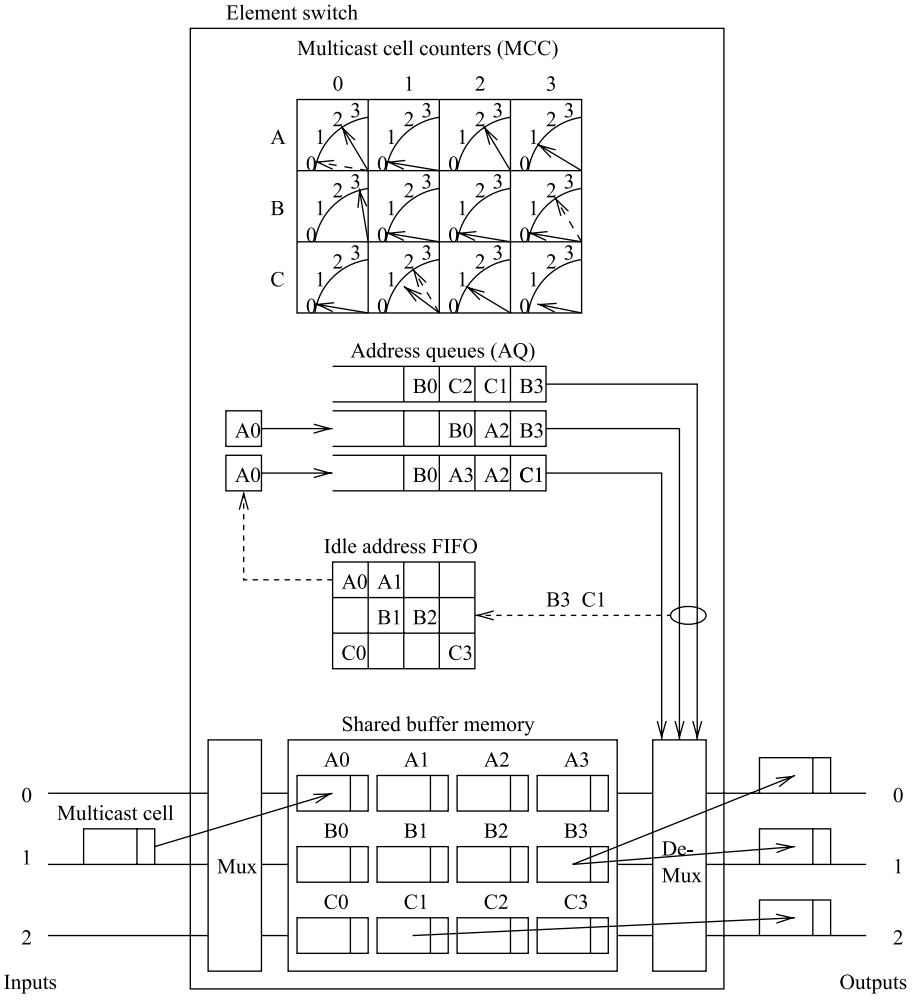


Figure 6.18 Example of multicast function in an address-copy switch.

Figure 6.18 illustrates an example of a 3×3 ATM switch. In this example, a multicast cell arriving at input port 1 is destined for output ports 1 and 2. The multicast cell is written into A0 in the SBM, and its address is provided by an Idle Address FIFO (IAF). Other vacant cells' addresses stored in the IAF are A1, B1, B2, C0, and C3. A0 is copied and queued in the Address Queues 1 and 2. At the same time, the MCC of A0 is set to two. Another multicast cell stored at B3 is destined for output ports 0 and 1. As it is transmitted to both output ports 0 and 1, the associated MCC becomes zero and the address B3 is released to the IAF. Within the same time slot, the cell stored at C1 is read to output port 2. Since it still has one copy to be sent to output port 0, its address C1 is not released until it is transmitted to output port 0.

Although in each time slot the maximum number of cells to be written into the buffer is N, the number of replications of cell addresses for incoming cells could be $O(N^2)$ in the

broadcast case, which still limits the switch size. More specifically, since the address queues or the MCCs are in practice stored in the same memory, there can be up to N^2 memory accesses in each cell slot, which may cause a system bottleneck for a large N.

REFERENCES

[1] H. Lee, K. H. Kook, C. S. Rim, K. P. Jun, and S. K. Lim, "A limited shared output buffer switch for ATM," in *Proc. Fourth Int. Conf. on Data Commun. Syst. and their Performance*, Barcelona, Spain, pp. 163–179 (June 1990).

[2] K. J. Schultz and P. G. Gulak, "CAM-based single-chip shared buffer ATM switch," in *Proc. IEEE ICC'94*, New Orleans, Louisiana, pp. 1190–1195 (May 1994).

[3] K. Oshima, H. Yamanaka, H. Saito, H. Yamada, S. Kohama, H. Kondoh, and Y. Matsuda, "A new ATM switch architecture based on STS-type shared buffering and its LSI implementation," in *Proc. IEICE'92*, Yokohama, Japan, pp. 359–363 (Mar. 1992).

[4] T. Cheney, J. A. Fingerhut, M. Flucke, and J. S. Turner, "Design of a Gigabit ATM switch," in *Proc. IEEE INFOCOM'97*, Kobe, Japan, vol. 1, pp. 2–11 (Apr. 1997).

[5] K. Y. Eng and M. J. Karol, "State of the art in gigabit ATM switching," in *Proc. IEEE BSS'95*, Poznan, Poland, pp. 3–20 (Apr. 1995).

[6] H. S. Kim, "Design and performance of multinet switch: a multistage ATM switch architecture with partially shared buffers," *IEEE/ACM Transactions on Networking*, vol. 2, no. 6, pp. 571–580 (Dec. 1994).

[7] W. Fischer, O. Fundneider, E. H. Goeldner, and K. A. Lutz, "A scalable ATM switching system architecture," *IEEE Journal on Selected Areas in Communications*, vol. 9, no. 8, pp. 1299–1307 (Oct. 1991).

[8] T. R. Banniza, G. J. Eilenberger, B. Pauwels, and Y. Therasse, "Design and technology aspects of VLSI's for ATM switches," *IEEE Journal on Selected Areas in Communications*, vol. 9, no. 8, pp. 1255–1264 (Oct. 1991).

[9] J. Garcia-Haro and A. Jajszczyk, "ATM shared-memory switching architectures," *IEEE Network*, vol. 8, no. 4, pp. 18–26 (July 1994).

[10] R. P. Bianchini, Jr. and H. S. Kim, "Design of a nonblocking shared-memory copy network for ATM," in *Proc. IEEE INFOCOM'92*, Florence, Italy, pp. 876–885 (May 1992).

[11] H. Saito, H. Yamanaka, H. Yamada, M. Tuzuki, H. Kondoh, Y. Matsuda, and K. Oshima, "Multicast function and its LSI implementation in a shared multibuffer ATM switch," in *Proc. IEEE INFOCOM'94*, Toronto, Canada, vol. 1, pp. 315–322 (June 1994).

[12] T. H. Lee and S. J. Liu, "Multicasting in a shared buffer memory switch," in *Proc. IEEE TENCON'93*, Beijing, People's Republic of China, pp. 209–212 (Oct. 1993).