Mitsubishi Electric Research Laboratories
Cambridge Research Center

Technical Report 94-17 September 30, 1994

A General Purpose Queue Architecture for an ATM Switch

Hugh C. Lauer
Abhijit Ghosh
Chia Shen

Abstract

This paper describes a general purpose queue architecture for an ATM switch capable of supporting both real-time and non-real-time communication. The central part of the architecture is a kind of searchable, self-timed FIFO circuit into which are merged the queues of all virtual channels in the switch. Arriving cells are tagged with numerical values indicating the priorities, deadlines, or other characterizations of the order of transmission, then they are inserted into the queue. Entries are selected from the queue both by destination and by tag, with the earliest entry being selected from among a set of equals. By this means, the switch can schedule virtual channels independently, but without maintaining separate queues for each one. This architecture supports a broad class of scheduling algorithms at ATM speeds, so that guaranteed qualities of service can be provided to real-time applications. It is programmable because the tag calculations are done in microprocessors at the interface to each physical link connected to the switch.

Presented at Massachusetts Telecommunications Conference,
University of Massachusetts at Lowell, Lowell, MA, October 25,1994.

Revision History:—

1. First version: July 28, 1994
2. Revised September 30, 1994

A General Purpose Queue Architecture for an ATM Switch

Hugh C. Lauer, Abhijit Ghosh,[†] Chia Shen

Mitsubishi Electric Research Laboratories
201 Broadway
Cambridge, MA 02139

Abstract

This paper describes a general purpose queue architecture for an ATM switch capable of supporting both real-time and non-real-time communication. The central part of the architecture is a kind of searchable, self-timed FIFO circuit into which are merged the queues of all virtual channels in the switch. Arriving cells are tagged with numerical values indicating the priorities, deadlines, or other characterizations of the order of transmission, then they are inserted into the queue. Entries are selected from the queue both by destination and by tag, with the earliest entry being selected from among a set of equals. By this means, the switch can schedule virtual channels independently, but without maintaining separate queues for each one. This architecture supports a broad class of scheduling algorithms at ATM speeds, so that guaranteed qualities of service can be provided to real-time applications. It is programmable because the tag calculations are done in microprocessors at the interface to each physical link connected to the switch.

Problem Statement

A modern high speed network needs to support hard real-time applications such as plant control, continuous media such as audio or video, as well as ordinary data communications. In order to do this, it is necessary to be able to schedule the real-time communications to meet quality of service goals and to protect them from the unpredictable demands of the non-real-time communications, all the while providing the remaining bandwidth to the non-real-time communications. There are two parts to the real-time scheduling problem:– an off-line part dealing with the setting up of connections (admission control); and an on-line part involving the actual scheduling of individual messages, packets, or cells as they arrive. A large body of theory about real-time scheduling has yielded many results predicting that certain combinations of off-line and on-line scheduling algorithms can guarantee certain qualities of service if certain *a priori* conditions are satisfied.

In real-time ATM networks, the challenge is the on-line part, i.e., cell-by-cell scheduling. For a network operating at 622 megabits per second, an ATM switch must make a scheduling decision for each physical link within each cell cycle of 680 nanoseconds. In the general case, there will be *n* virtual channels with cells awaiting transmission, each of which has its own scheduling requirements. Therefore, each decision involves selecting the first cell from the virtual channel with the highest priority, earliest deadline, most frequent rate, or other parameter.

[†] Address: Mitsubishi Electric Research Laboratories, 1050 E. Arques Avenue, Sunnyvale, CA 94086

This scheduling decision can be reduced to a comparison to select the minimum or maximum from among a set of *n* numerical values. There is no way around this *n*-way comparison. It is the fundamental complexity of real-time scheduling, and it must be done once during every scheduling interval. Obviously, if *n* is very small, it is not difficult. But if *n* is large, as is likely to be the case in a network with many real-time channels with widely varying characteristics, the comparison becomes a significant part of the design of the switch.

Shared-buffer Switch Architecture

At Mitsubishi Electric, ATM switches use a shared buffer architecture, an unfolded view of which is shown in Figure 1. The central fabric of the switch is a *2k*-port memory, where *k* is the number of physical links connected to the switch. During each cell cycle, *k* cells arrive at the *k* input sides of the physical links, shown on the left, and are concurrently written into the shared cell buffer memory. At the same time, *k* cells are read from the same memory for transmission to the *k* output sides of the links. Scheduling is merely a matter of selecting which cells to transmit at which time.

The advantage of this architecture over the more common input- or output-buffered switches is that less total memory is required because it can be statistically multiplexed among the physical links. The total bandwidth of the memory must be *2 * k * B* cells per second, where *B* is the bandwidth of the individual links. This is only slight more stringent than the buffers of input- and output-buffered switches, which require *k* separate buffers with bandwidth *(k + 1) * B* each in

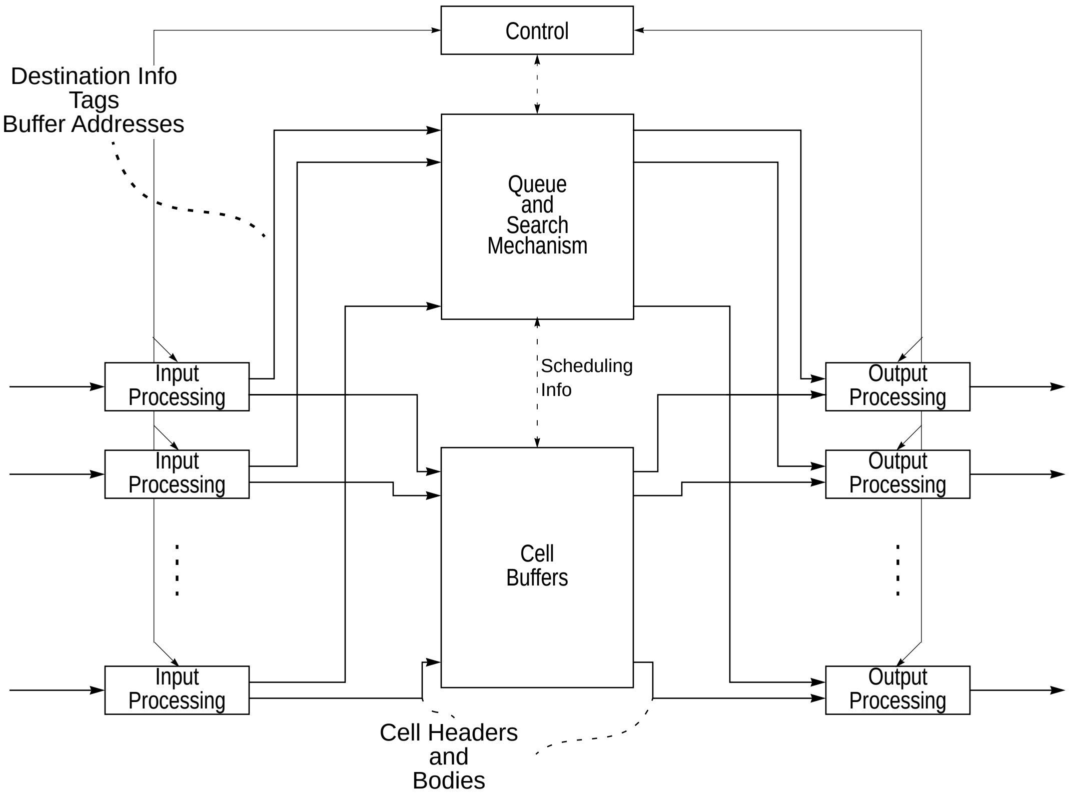

Figure 1 Shared-buffer ATM Switch Architecture

order to be non-blocking. Therefore, the shared-buffer architecture makes better use of the physical resources of the switch [Oshima92].

Selectable Queue Circuit

One way to do cell-by-cell scheduling is to use a self-timed FIFO with a search circuit such as that described in [Kondoh94] and illustrated in Figure 2. This circuit contains a linear array of registers, as many as there are cell buffers in the memory. Each register contains a cell address, a priority value, and a *k*-bit destination vector in which one bit represents each physical link. A value of one in a position of the destination vector means that the cell addressed by the entry waiting to be transmitted to that link. During each cell cycle, the circuit is searched *k* times, once for each link. Each search finds the first entry containing a one bit in the corresponding position of the destination vector — i.e., nearest the head of the queue (at the bottom of Figure 2). The cell address is retrieved, the cell at the buffer address is transmitted, and the bit of destination vector is reset. When all of the destination vector bits of an entry are zero, the entry is deemed to be vacant. Then the internal, self-timed circuitry propagates all subsequent entries forward to fill up vacant spaces. In this sense, it acts like a traditional FIFO circuit. New entries are always added to the tail of the queue (shown at the top in the Figure).

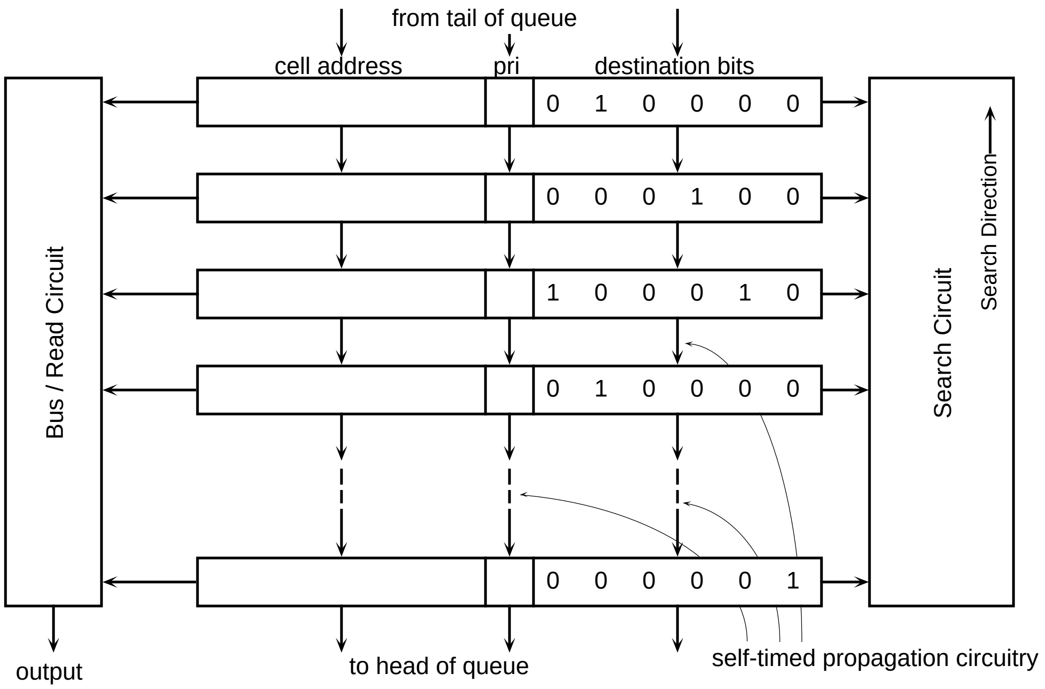

Figure 2: Self-time FIFO with search circuit

To support real-time communication, we use a modification of this circuit in which selection takes account of the priority information encoded in numerical tags. Instead of merely selecting the first entry with a particular destination bit set, this circuit selects the entry with both the smallest tag value and a destination bit of one. A block diagram of the circuit is shown in Figure 3.

This figure shows only one column of destination bits (corresponding to one physical link), a *tag* register for each entry, comparators between adjacent entries, and a simplified selection circuit. Not shown are the cell addresses or self-timed circuitry which causes entries to propagate from the tail to the head of the queue. The comparators all operate in parallel. Each compares the value of its tag register with the output of the previous comparator (towards the tail of the queue). If the destination bit is one, it returns the smaller of these two values for use by the comparator of the next entry (toward the head of the queue); otherwise, it simply propagates the value from the previous entry to the next entry. Each comparator also outputs a single bit, which has a value of one if and only if (a) the destination bit of that entry is one, and (b) the tag of that entry is not greater than any tag propagated from entries toward the tail of the queue. The selection circuitry on the right is identical to that of Figure 2 and finds the first such entry.

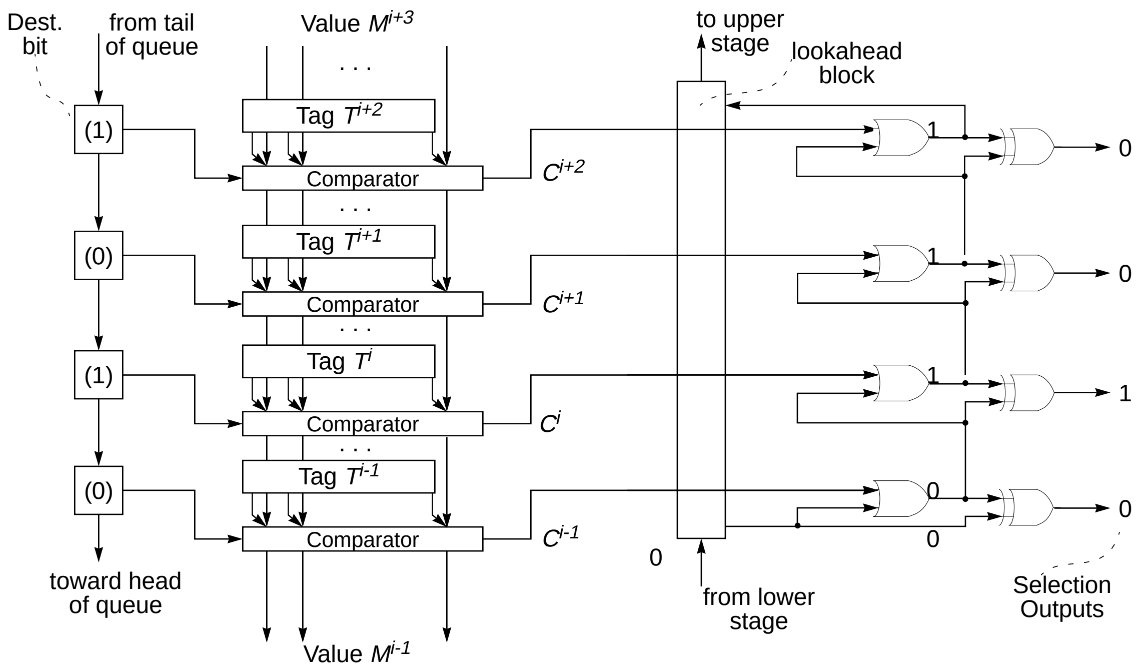

Figure 3 Tag-based Selectable Queue Circuit

The queue circuit of Figure 3 is an extremely simplified one. For a practical circuit in a *k*-by-*k* shared-buffer switch operating at 622 megabits per second, the circuit must search the queue *k* times in each 680 nanosecond cell cycle. If $k = 16$, for example, each search must be completed in about 40 nanoseconds. Even with look ahead, the ripple effect of the comparators takes too long. Therefore, we actually partition the search into sections, each one producing a minimum tag value. Then these compare the results of the sections hierarchically to select the minimum overall tag value. Experience with an evaluation circuit reported by Kondoh, plus estimates based on 0.8µ CMOS design rules, indicates that search can be completed in less than 30 nanoseconds for 512 queue entries with 16-bit tags.

Implementation of Real-time Scheduling Algorithms

This scheduling mechanism is extremely flexible and general and can support a broad class of real-time and non-real-time scheduling algorithms. The key feature is the ability to compute tags for individual cells when they arrive at the switch. For example,

- In rate monotonic systems, tags represent intervals between periodic messages. Tasks requiring messages with higher frequencies are assigned smaller tags. Tags can be pre-computed when virtual channels are set up, then assigned by table look-up when cells arrive.
- In the Virtual Clock algorithm [Zhang91], tags represent the logical arrival times of cells. These are computed by taking the actual arrival time of each cell, then adjusting them as necessary to reflect early arrivals, etc.
- In the Earliest Deadline First algorithm [Zheng94], tags represent deadlines by which time cells must be transmitted. They are calculated by adding pre-computed delay values to logical arrival times.

In addition, round robin, weighted fair queuing, and other algorithms can be supported by defining appropriate tag-calculation functions to be applied to incoming cells arriving at the switch. Moreover, multiple classes of traffic can be supported simultaneously by partitioning the space of tag values and assigning tags in the different subspaces by different algorithms.

In some of the algorithms, such as Virtual Clock and Earliest Deadline First, tag values increase monotonically for each virtual channel. Eventually, they will overflow the tag field of the registers in the queue. To cope with this, a control signal is provided which, when raised, causes each tag to be treated as having one additional high order bit. The value of this bit is the opposite of the actual high-order bit of the tag. Initially, all tags are zero. After a while, tags of newer entries in the queue have high-order bits of one.

Assuming that there are enough bits in the tag fields and that all entries are chosen within a predictable period of time, it will be the case that all tags in the queue will have high order bits of one. I.e., earlier entries will have been selected and transmitted and the tag values of new entries will not have wrapped around back to zero again. At this point, the control signal is raised. When the tag values eventually do wrap around back to zero, they will be treated as being larger than the earlier queue entries with ones in the high-order bits of theirs tags. Later still, all entries with ones in the high order bits of their tags will have been removed from the queue, at which time the control signal can be lowered again. Note that this depends upon the tag values having enough bits so that the interval between wrap around is greater than the lifetime of any entry in the queue.

Summary

A queue architecture for a shared-buffer ATM switch has been described. This architecture supports real-time communication traffic by allowing cell-by-cell scheduling according to a wide variety of scheduling algorithms. The central feature of the architecture is a tag-based FIFO queue circuit containing entries for all cells to all destinations. It can be searched for the cell with the smallest tag for a particular destination. In effect, it provides the necessary queuing without requiring separate queues for individual virtual channels. The queue circuit can be implemented in VLSI and can control a 16-by-16 switch at 622 megabits per second.

References

[Kondoh94]
H. Kondoh, H. Yamanaka, M. Ishiwaki, Y. Matsuda, and M. Nakaya, “An Efficient, Self-Timed Queue Architecture for ATM Switch LSI’s,” *Proceedings. of Custom Integrated Circuit Conference,* San Diego, May 1994.

[Oshima92]
K. Oshima, H. Yamanaka, H. Saito, H. Yamada, S. Kohama, H. Kondoh, Y. Matsuda, “A New ATM Switch Architecture based on STS-type Shared Buffering and its LSI Implementation,” *Proceedings of International Switching Symposium ‘92,* Yokohama, Japan, October 1992, pp. 359-363.

[Zhang91]
L. Zhang, “Virtual Clock: A new traffic control algorithm for packet-switched networks,” ACM Transactions on Computer Systems, vol. 9, No. 2, May 1991, pp. 101-124.

[Zheng94]
Q. Zheng, K. Shin, and C. Shen, “Real-time Communication in ATM Networks,” to appear in *19th Annual Local Computer Network Conference*, Minneapolis, Oct. 2 - 5, 1994.