ATM SWITCH BUFFER CONTROLLER BASED ON LINKED LISTS SUITED FOR HARDWARE IMPLEMENTATION

Elmar Uwe Kurt Melcher, Lírida Alves de Barros Naviner, Lavoisier Leite, William Ferreira Giozza, José Augusto Suruagy Monteiro(*)

Universidade Federal da Paraíba
(*)Universidade Federal de Pernambuco

Centro de Ciências e Tecnologia
Av. Aprígio Veloso, 882 - Bodocongó
58109-970 Campina Grande - PB

email: elmar@dee.ufpb.br

Abstract

In this paper we describe a new algorithm for a multi-class ATM (asynchronous transfer mode) buffer controller that performs differentiated treatment depending on the cell's service class and priority. The algorithm is based on multiple linked lists for each class and avoids time-consuming iterating searches in the buffer. The algorithm is being implemented in an application specific integrated circuit that will be part of an ATM switch.

Resumo

Este artigo descreve um novo algoritmo para controlador de *buffer* multiclasse em um comutador ATM (asynchronous transfer mode). Diferentes tratamentos são executados dependendo da classe de serviço e da prioridade das células. O algoritmo é baseado em múltiplas listas para cada classe e proporciona a redução do tempo de processamento através de pesquisas iterativas no buffer. Este controlador faz parte de um comutador ATM e está sendo implementado em um circuito integrado de aplicação específica.

1. Introduction

One of the most important features of ATM nets is their ability to provide adequate services for different types of information such as data, voice, and video [6].

Each type of information has its particular quality of service requirements [4]. For example, voice transmissions need low transfer delay whereas data need low loss rate to avoid retransmission.

The quality of service (QOS) can be defined by a set of parameters such as cell loss rate, cell misinsertion rate, cell transfer delay, cell delay variation, etc. In order to simplify operation management, a certain number of classes can be defined. Each class corresponds to a set of service quality parameter values. The network client may choose among the classes offered by the network provider when he negotiates a virtual channel for his transmission.

The efficiency of ATM networks depends on the source traffic characteristics and the strategies for network control, switching, and transmission. Given a determined physical channel bandwidth, the parameters that have the highest impact on network efficiency are strongly associated to the buffering mechanisms used [7]. Thus, the overall performance of the ATM switch is largely determined by the intelligence of the buffer controller.

Several buffer management strategies have been proposed, based on buffers with separate queues assigned to each class of traffic [6].

- head of line priority

Cells belonging to a higher class are always served first. Cells belonging to a lower class must wait until all the cells of the higher classes have been served.

- weighted round-robin

The queues are served in a cyclic fashion. Queues belonging to higher classes are served more often during one cycle than queues belonging to lower classes.

- push-out

A high-priority cell can enter a queue that is already full, expelling a low-priority cell.

- partial sharing

When a queue is filled above a certain threshold, only high-priority cells can enter the queue.

The implementation described in this paper uses head of line priority and push-out (see section 3).

In ATM, data are divided in cells of 53 bytes. Each cell that enters an ATM buffer has an associated class and the buffer must treat the cell accordingly. The buffer controller must decide what to do with the cell before the next cell arrives. At high data rates (155 Mb/s or 620 Mb/s) this decision must be taken in a few microseconds or in even less than one microsecond. This speed requirement is very difficult to fulfill with a general purpose processor: either the buffer input cell rate must be limited or the buffer controller must be simplified. A high buffer input cell rate and a sophisticated buffer controller can only be implemented in application specific integrated circuit (ASIC).

The buffer controller described in this paper is part of an 8x8 ATM switch being implemented in the ProTeM II/COMATM project [3]. The switch consists of a general purpose crossbar router surrounded by 8 ASIC companions. The crossbar router called RCube is developed by an European research group including the MASI laboratory, University of Paris VI, France, and BULL, France [8]. The companion chip called ABSE (ATM Basic Switching Element) is being developed by researchers from USP, UFPE, and UFPB.

In the following section the exact specification for the COMATM buffer is described. In section 3 we show the algorithm that implements this specification and in section 4 we briefly describe the hardware implementation of the algorithm. The last section gives some conclusions about the potentials of the algorithm and the hardware implementation.

2. Buffer Controller Specifications

In the COMATM switch the buffers are located at its outputs ports. This alternative presents the best delay and throughput figures [5]. Data enter the buffers coming from the RCube crossbar. The output of each buffer goes to a physical interface circuit using the UTOPIA [1] interface protocol.

The maximum input bit rate of the buffer is 480 Mb/s for an RCube chip working at 60 MHz clock speed. This corresponds to a minimal input cell period of 0.93 ms. The output bit rate is 155 Mb/s, corresponding to a minimum output cell period of 2.89 µs.

The buffer must be able to hold up to 2000 cells.

The implemented controller must support 3 classes even if it may be extended to 5 classes as recommended by the ATM Forum [9]. Each class has two priorities: high priority (CLP=0) and low priority (CLP=1).

When a cell comes into the buffer, the controller assigns it a free address in the buffer if there is any space left. If the buffer is full, the controller discards a cell.

When the buffer contains only cells that belong to a higher class than the incoming cell, the incoming cell is discarded. When the buffer contains cells of the same class as the incoming cell but no cells of a lower class and the incoming cell has low priority, again the incoming cell is discarded. In all other cases, the controller discards a cell from the buffer.

The controller searches for the cells in the buffer that have the lowest priority class. Among them it looks for cells having low priority. If there is at least one, it discards the one that entered the buffer most recently. If the lowest priority class has only cells with high priority (CLP=0), it discards the cell of that class that entered the buffer most recently.

When a cell can be sent out of the buffer (signal TxClav=1 from the UTOPIA interface) the controller looks for the cell of the highest priority class that first entered the buffer. The cell loss priority (CLP) has no influence on the choice of the outgoing cell. This is because cells from the same virtual channel can have high or low priority and the order of the cells of the same virtual channel must not be changed by the ATM switch.

3. Buffer Organization

Studying the specifications, it is easy to see that any algorithm suited for implementation must avoid any iterating searches throughout the buffer because it is technically impossible to do up to 2000 access/compare cycles in only 0.93 ms.

One way to avoid iterative searches is to allocate physical memory of 2000 cells for each class. This aproach would result in a very simple buffer controller but in a threefold increase in buffer memory.

A better solution is to organize the buffer using linked lists [2]. When the lists are correctly organized, the right cell address can be obtained by one direct access/compare operation. However, one or two further modify accesses are needed for certain operations in order to re-organize the list, for example when a cell is discarded.

The system of linked lists proposed in this section not only allows the implementation of the head of line strategy for different classes [2], but also the push-out strategy for low-priority cells. In the following, we will first re-formulate the specifications and then propose an adequate structure of lists.

Let us increase the buffer size by one cell and at the same time let us assume that there is always space left in the buffer for at least one cell. Thus, the effective buffer size is not altered.

When a cell comes into the buffer, it is directly written in that free space.

Immediately after the cell has been received, the controller checks if there is still free space left in the buffer so that the next incoming cell could be written. If the buffer is full, one cell is immediately discarded from the buffer. The cell that must be chosen is the one that has the lowest priority class, having low priority if available, and having entered the buffer most recently. Note that exactly under the conditions mentioned in section 2, the cell that has just been received is discarded.

As far as outgoing cells are concerned, the specification remains unchanged.

One can see that the re-formulation makes the specification simpler and easier to implement, at the expense of physical storage space for one more cell.

In the following the set of lists that are used to organize the buffer are described.

The free cell addresses are linked together by a one-threaded list that starts at pointer F and ends at address 0 (this address can not be used to store any cell).

There is a pair of lists for each class. The first list contains all the cells of the class. The second list is a subset of the first. It contains only the cells that have low priority. Both lists have two threads: a forward thread and a backward thread. Both lists have the same start address. However, the start address must not be used to store any cell.

The buffer controller has direct access only to the start and the end of each list and sublist. Program 1 and program 2 describe the functionality of the list controller.

```
link next_cell_address to end of all_priorities of class d
if CLP=1 then
    link next_cell_address to end of low_priorities of class d
end if

if free is not empty then
    take address from end of free and assign to next_cell_address
else
    for lowest priority to highest priority class
        if all_priorities of class is not empty then
            if low_priorities of class is not empty then
                take address from end of low_priorities of class
                and assign to next_cell_address;
                using forward thread of all_priorities of class,
                link preceding address to following address;
                using backward thread of all_priorities of class,
                link following address to preceding address;
            else
                take address from end of all_priorities of class
                and assign to next_cell_address;
            end if
            exit for
        end if
    end for
end if
```

Program 1: Procedure executed when cell of class d and priority bit CLP enters the buffer

```
for highest priority to lowest priority class
     if class is not empty then
          take address from start of all_priorities of class
          and link it to end of free;
          send out cell at address now at start of all_priorities;
          exit for;
     end if
end for
```

Program 2: Procedure executed when a cell is sent out of the buffer

In figures 1 to 5 a buffer with 2 classes and an effective capacity of 4 cells is shown in different states. The buffer has physical storage space for 8 cells. Figure 1 shows the initial state of the buffer: all the 4 free cells are in the list of free cell addresses and the lists of classes A and B are empty. Cell address 7 is reserved for the next incoming cell.

While an incoming cell is being written at the reserved address, it is linked into the class and priority it belongs to. Immediately after the cell has been completely received, an address is taken from the end of the free address list to be ready for the next cell.

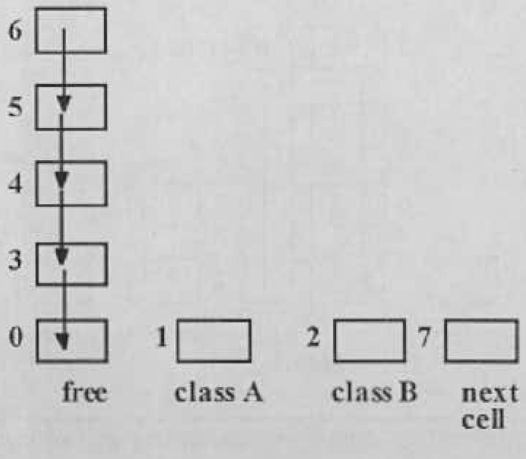

Figure 1: Initial state of the buffer.

In figure 2, one can see the state of the buffer after reception of:
- a cell for class A, CLP=0 (address 7),
- another cell for class A, CLP=0 (address 6),
- a cell for class B, CLP=0 (address 5),
- a cell for class A, CLP=1 (address 4).

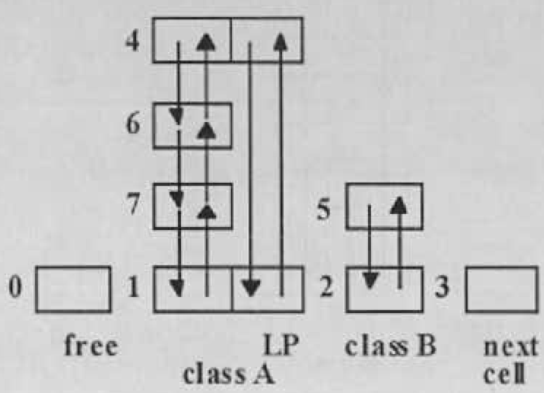

Figure 2 : State of the buffer after reception of cells.

Now suppose that we are enabled to output a cell from the buffer. As long as there are cells in the highest priority class (class A), the cell to send is taken from the start of this class.

Figure 3 shows the state of the buffer after one cell (address 7) has been read out. Address 7 is now used to mark the start of the lists for class A and address 1 is free.

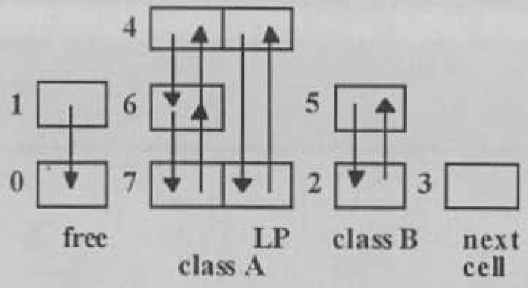

Figure 3 : State of the buffer after one cell (address 7) has been read out.

Figure 4 shows the buffer after one more incoming cell of class A. The next cell was taken from the free address list and the buffer is now full.

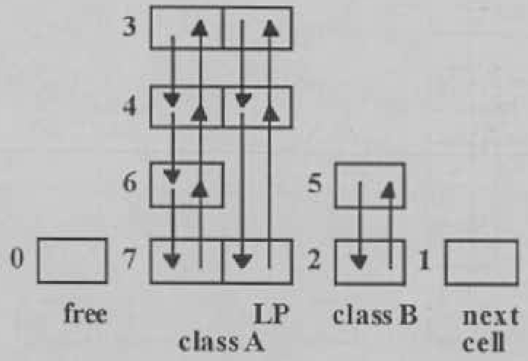

Figure 4 : State of the buffer after one more incoming cell of class A and low priority has been written.

Figure 5 shows the buffer after one more incoming cell of class A, high priority, has been written. Note that in order to find a new address for the next cell, the last cell of class B (address 5) had to be discarded. If the cell that was written had been of class B instead of class A, that same cell would have been discarded. This is exactly what the initial specification demands: if an incoming cell has equal or lower class and priority than the cells already in the buffer, this incoming cell is discarded.

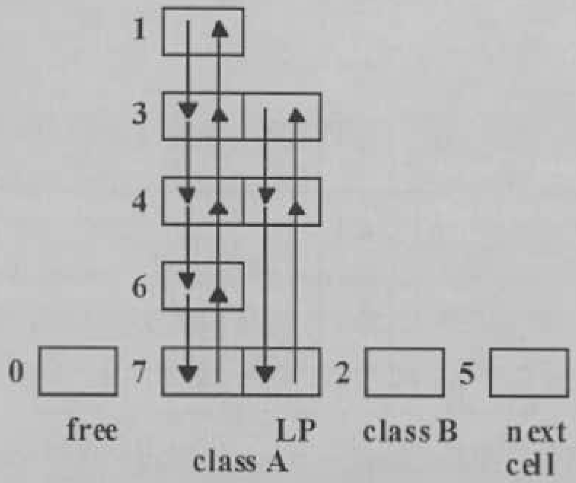

Figure 5 : State of the buffer after one more incoming cell of class A and high priority has been written.

In figure 6, another incoming cell of class A, high priority, has been written. This time, a cell from class A had to be discarded because class B was empty. As there are cells with low priority (CLP=1) in class A, the last one of them (address 3) is discarded in order to become the next cell. Note that the pointers between address 1 and address 4 had to be readjusted so that the threads of the list are not broken.

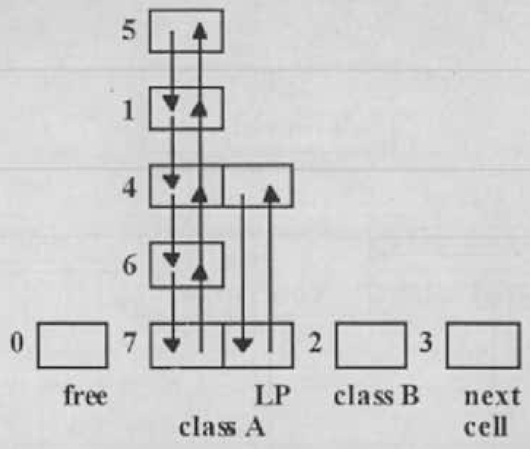

Figure 6 : State of the buffer after another incoming cell of class A and high priority has been written.

In this section, the lists that are used to organize the buffer have been presented and an example with a reduced number of classes and cells has been presented. It has been shown that all necessary operations can be performed by direct accesses from the start or the end of the lists, thus enabling real time operation of the hardware implementation.

4. Hardware Implementation

In the hardware implementation, the lists are physically separated from the cells. The cells are stored in a pool off chip [2], while the lists are stored in on-chip RAM (random access memory). A simplified schematic of the circuit is shown in figure 7.

The interfaces and the list controller were implemented in the hardware description language VHDL (very high speed hardware description langage). The description language is similar to a software programing language, but has special constructs that allow to express parallelism (process) and timing (wait until clock'event). This description was simulated for validation and synthesized automatically in the Cadence framework. The standard cell library used for VLSI (very large scale integration) implementation is ECPD07 from European Silicon Structures. Table 1 lists some characteristics of the implementation.

The functional correctness of the description was validated using a test bench that contains every possible test cenario. This test bench was also written in VHDL and is independent of the internal circuit architecture and the abstraction level of the buffer controller. The behavioral description of the buffer controller and the test bench were carried out by different designers, both starting from the specifications. This working strategy improves the quality of the test.

Before fabrication of the VLSI chip, the buffer controller will be put together with the other parts that form the ABSE.

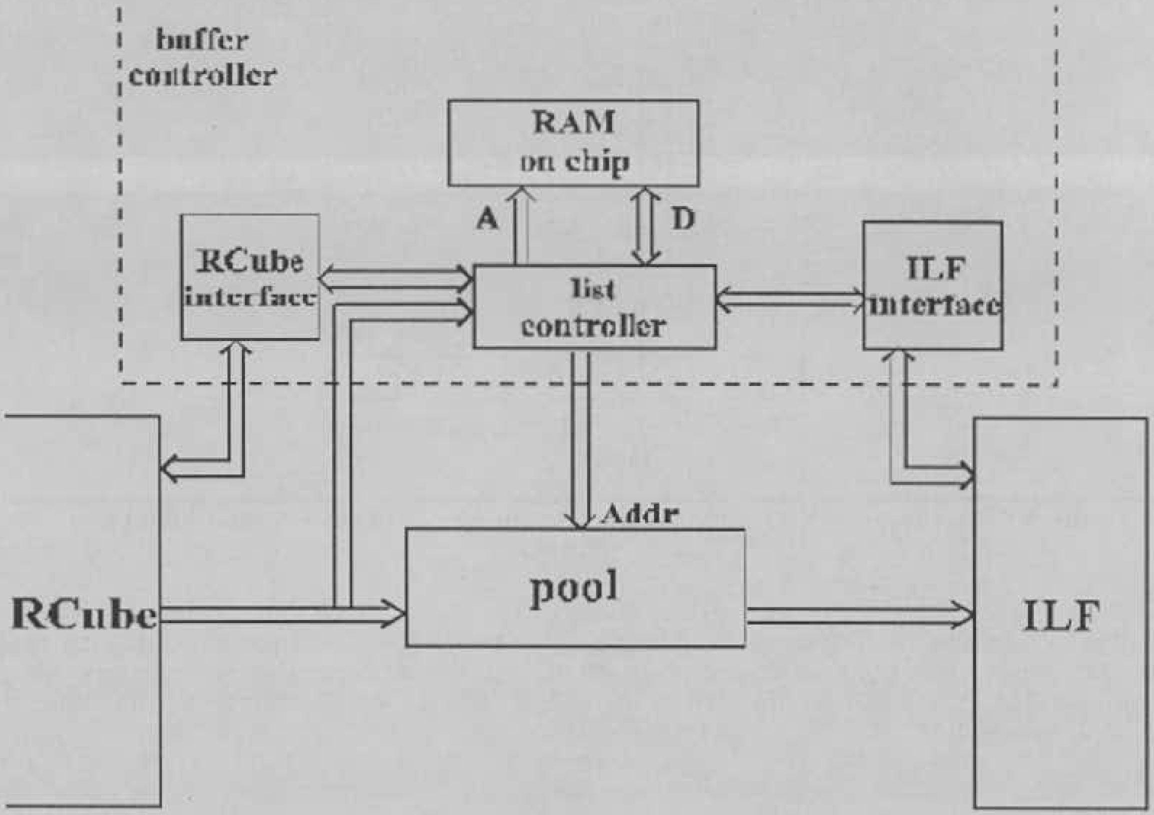

Figure 7 : Schematic of the circuit.

maximum clock frequency	64 MHz
on-chip RAM area	26.9 mm^2
cell area of list controller	1.2 mm^2
number of logic gates	1282
number of registers	246
number of transistors	17582

Table 1 : Characteristics of the chip.

5. Conclusion

The buffer controller architecture proposed in this paper is able to improve overall switch performance and link utilization factors through more sophisticated buffer management because it supports multiple classes as well as low/high priority within each class. Using a high-level synthesis tool, the complex operations involved could be implemented in dedicated hardware observing the tough timing requirements for high speed ATM switches.

Acknowledgements

The work presented in this paper was financed by CNPq project ProTeM II/COMATM. Lavoisier has a student grant from PIBIC/UFPB. The maintenance support and training for the Cadence synthesis tool was provided by QuickChip, São Paulo.

References

[1] ATM Forum, "Utopia ATM Physical Interface Specification Level 1, Version 2.01", Mar. 1994.

[2] H.J. Chao, "A novel Architecture for queue Management in the ATM Network", IEEE Journal on selected areas in communications, vol. 9, no. 7, Sept. 1991.

[3] Projeto COMATM, Relatório técnico Rt-01/95 Versão 3.0 - Arquitetura Básica e Descricão funcional do Comutador ATM, Feb. 1995

[4] ITU-T, Recommendation I.371: Traffic Control and Congestion Control in B-ISDN, Mar. 1994.

[5] M.J. Karol et al., "Input versus output queuing in a space-division packet switch", IEEE Transactions on Communications, vol. COM-35, Dec. 1987.

[6] José Augusto Suruagy Monteiro, Rede Digital de Serviços Integrados de Faixa Larga (RDSI-FL), IX Escola de Computação, Recife, Jul. 1994.

[7] P. Plaza, L. Merayo, J.C. Diaz, and J.L. Conesa, "A 2.5 Gb/s ATM switch Chip Set", IEEE Transactions on VLSI Systems, vol. 4, no. 3, Sept. 1996.

[8] Hamilton S. Silva, R. Ferreira, A.C. Cavalcanti, and W. Giozza, "Implementação de um Comutador ATM usando como matriz de comutação o roteador de uso geral RCube e um conjunto de ASICs companions", IX SBMicro, Aug. 1994.

[9] The ATM Forum, Traffic Management Specification Version 4.0, Apr. 1996.