

# Design of an Adaptive Flow Control Algorithm for ATM Networks

This paper presents the design of an Adaptive Flow Control Algorithm for ATM Networks using hardware/software implementation. The renewed interest in such implementation is driven by advances in technologies that support both hardware and software parts in order to handle high complexity and to optimize the performance/cost factor. Exploration of the design architecture has been performed in order to have more efficiency and flexibility. A solution that correctly implements the system functionality while meeting real-time requirements is produced. After simulation and synthesis steps, an ASIC has been implemented by using a CMOS 0.6  $\mu\text{m}$  technology.

## INTRODUCTION

The Asynchronous Transfer Mode (ATM) is a fast packet switching network that supports high-speed integrated services by splitting all communications into equal 53-byte cells. A key distinguishing feature of ATM networks as compared to current packet networks is that they offer multiple Qualities of Service (QoS)[1].

Flow control in ATM networks is based on the philosophy of preventing congestion rather than reacting to it. In principal it does that by sending a feedback signal from the destination point to the source informing it of the congestion and throttling the input traffic. In this paper we focus our study on the Adaptive Leaky Bucket (ALB) algorithm in order to implement it using a VHDL-based methodology. The basic idea consists in an exchange of control cells (feedback) between two nodes of the network implementing the ALB mechanism. We made some parts of the design in software, while time critical parts are realized in hardware. The design of the hardware component starts from VHDL-description that has come to be recognized as an integral part of the design process of complex circuits. The high level description allowed describing the design in a purely behavioral form, void of any implementation details. This specification allowed to describe and design a microchip's functionality with a behavioral description that is void of any timing, engineering or technological information, and then synthesize them automatically into gate-level components.

This paper is organized as follows. The ALB algorithm is described and analyzed in section II. Section III out-

lines and explores experiments consisting of the design methodology, the architecture of the design, and the results of the hardware/software exploration. Finally, section IV draws some conclusions and outlines future directions.

## THE ADAPTIVE LEAKY BUCKET

The capability of ATM networks to provide large bandwidth and to handle multiple quality of service guarantees can only be realized by preparing effective traffic management mechanism [2]. The Leaky Bucket (LB) mechanism (cf. figure 1) is one way to ensure that sources do not exceed the negotiated rate. Schematically, it can be represented as a token pool with a capacity  $M$  which can be seen as the maximum allowable burst length. A cell can enter the network only when it can draw a token from the pool; if the token pool is empty, then the cell is discarded, put in queue, or marked and transmitted. Tokens are generated at a certain admission rate  $R$ . After filling the pool, additional tokens are discarded. The LB is a common example of a traffic control at the cell level. In [3], the authors studied the LB destined for variable bit rate services in three forms: a simple LB, a LB with tokens and a dynamic control LB.

Our approach [4] consists in representing the LB as a "credit" mechanism, since it exercises a control over the sender restricting the maximum amount of data that might be transmitted at a certain instant of time. The ALB uses a counter, which is initially equal to a maximum counter value, the so-called bucket size. Each time a cell is sent into the network, the counter value is decremented by one. So the counter value is

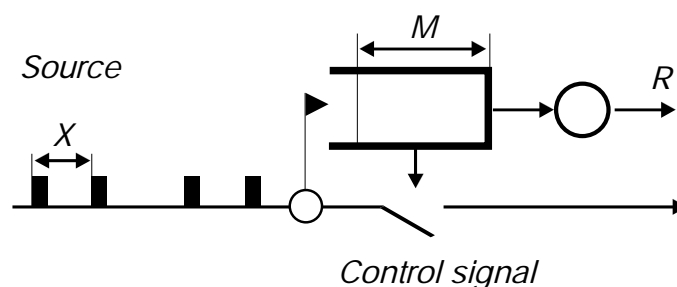


Figure 1. Leaky Bucket mechanism.

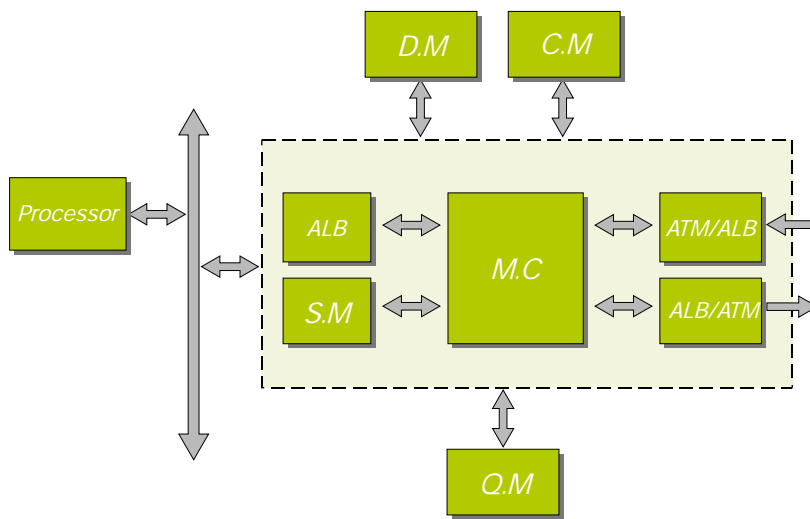


Figure 2. Design architecture.

considered as the available tokens. New tokens are generated by periodically incrementing the counter value by a certain amount. The ALB mechanism is then described by the initial number of tokens, an update function and an algorithm for the regeneration of the tokens.

The Data-cells are buffered in a data memory using a dynamic buffer management method. It consists on creating a free list of the data memory addresses where the incoming cells will be buffered. The Control-cells are stocked in a QoS memory. The Control-cell fields used to achieve the ALB mechanism are: the PCR (Peak Cell Rate), the MCR (Minimum Cell Rate), the FTN (Free Tokens Number), the MTN (Max. Token Number), the CRN (Cell Received Number) and the PST (Period to Send a Token). The ALB uses these parameters to schedule cells transmission at the rate sustained for every VC.

The ALB has two behaviors simultaneously: it receives Data-cells, from the node upstream, then after a period specified in the QoS it sends control cells containing the fields described above. The sending node will generate a Control-cell after forwarding TX Data-cells belonging to the VC. This cell will contain the number

of free buffers for this VC on node. The receiver node to regenerate its Control-cell will use this value. The Control-cell reaches the receiving node, which regenerates its Control-cell by subtracting a value, corresponding to the number of cells sent during the last RTT (Round Trip Time), from the value carried by the Control-cell.

These operations have the cell-time as a real-time constraint. The cell processing must be achieved before that the following one is received. This constraint is 12  $\mu$ s for the rate 34 Mbit/s, 2.73  $\mu$ s for 155 Mbit/s and 680 ns for 622 Mbit/s. Starting from the algorithm partitioning, we can specify two local constraints for the transfer rate 155 Mbit/s. The first is related to the header and parameters bytes reception which is ( $C_1 = 867$  ns). The second one is the remaining cell-time, considered as local constraint for the other operations presented by the scheduling of the received cell and the parameters process ( $C_2 = 1836$  ns).

## EXPERIMENTS

### A. The ALB Design Architecture

The design architecture requires three memory modules: a DM (Data Memory) in which data will be

Part. / Sol.	$S_1$	$S_2$	$S_3$	$S_4$	$S_5$	$S_6$
H&P Rec.	S	H	S	S	H	H
Sched	S	S	S	H	S	H
Par. Proc.	S	S	H	S	H	S
Time	10.5	5.52	71	71	3.92	2.12
GC Viol	Yes	Yes	Yes	Yes	Yes	No
$C_1$ Viol	Yes	No	Yes	Yes	No	No
$C_2$ Viol	Yes	Yes	Yes	No	Yes	No
Cost	404	397.47	1864.36	2711.55	1983.83	2705.02
Funct.-cost	3.98	2.13	3.05	3.25	1.91	1.41

Table 1. Hardware/Software solution exploration.

<i>Component</i>	<i>Number of cells</i>	<i>Number of gates</i>
<i>ATM/ALB</i>	205.47	501.14
<i>ALB/ATM</i>	76.88	187.51
<i>M.C</i>	3891.12	9890.53
<i>ALB/ALG</i>	30.80	75.12
<i>Global</i>	4204.68	10255.31

Table 2.

buffered, a CM (Control Memory) containing a free list of DM addresses, and a QM (QoS Memory) where the QoS fields will be stocked. This architecture (see figure 2) contains two interfaces with the network, a Memory Controller (MC) and the ALB module.

The MC performs a dynamic memory management using addresses stocked in the CM to extract addresses where Data-cells will be buffered for every VC. It uses the VC value as address for the QM to stock QoS parameters. The ALB module performs operations specified in the algorithm described in the previous section such as processing the Control-cell fields and scheduling of Data-cells transmission priority. The design contains two interfaces with the ATM network. The ATM/ALB interface receives the cell fields, identifies the cell kind (data or control) then activates the MC to store the cell. The ALB/ATM interface receives the payload bytes from the MC, reassembles them with the suitable header then activates its transmission on the network. The design contains SM (Scheduling Memory) used for scheduling of emission and reception data. This architecture decomposition permits to easily handle the complexity of the algorithm and also gives more flexibility of the design.

### **B. Hardware/Software exploration**

We have studied the implementation of the ALB algorithm using a hardware/software methodology combining the partitions executing time and cost. Hardware partition cost is obtained by the gate number, and the software partition cost is calculated using required bytes number. The execution times are deduced using synthesis tool for the hardware partition and the microprocessor clock for the software one [5].

Table 1 gives all possible solutions with time execution, the cost-function and the viol possibilities for the local and the global constraints. These results show that S2 and S5 solutions do not violate the local constraint C1 but they violate the global constraint GC. Also we note that S4 solution does not violate the local constraint C2 but it violates the global constraint GC. The S6 solution is the most interesting solution because it satisfies the global and local constraints. This solution has the minimal function-cost (1.41).

In the next section we focus our study on the design of the hardware part of our system using a VHDL-based methodology.

### **C. Design methodology**

In general, the process of designing a system will pro-

ceed from a behavioral to a physical representation, gaining implementation details along the way. High level synthesis converts a behavioral specification of a digital system into an equivalent RTL design that meets a set of stated performance constraint [4]. The designer describes his system with a high level specification at one of abstraction levels. This description with a HDL (Hardware Description Language) is synthesized using existent synthesis tool allowing passage to the next abstraction level until reaching either integration in ASIC or implementation in FPGA.

After design verification, a design compiler is used to perform logic synthesis. The result of synthesis is a gate level description of the system. The logic synthesis tool starts with two kinds of information: an RTL specification given in VHDL and a functional unit library that can include complex functional units. The RTL description accesses these function blocks through VHDL procedure calls. For each procedure or function used, the library must include at least one functional unit able to execute the corresponding operation [6].

### **D. Results**

The design is simulated and synthesized using Synopsys. The resulting netlist is used as input to Cadence in order to perform mapping and routing with a 0.6  $\mu\text{m}$  CMOS technology. The results of these operations are presented in table 2 and figure 3.

The final ASIC has been implemented using a CMOS 0.6  $\mu\text{m}$  technology and its area is about 19 mm<sup>2</sup>. This circuit operates with clock frequency of 80 MHz and

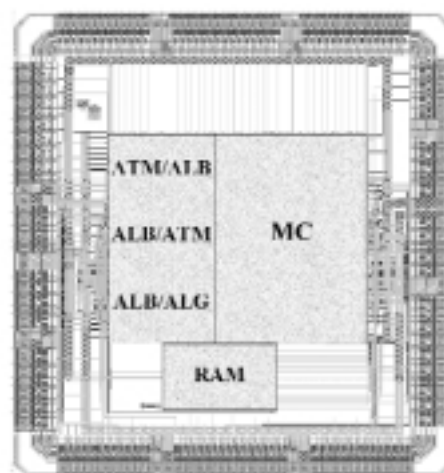


Figure 3. layout of the design.

allows the cells transfer on 155 Mbit/s and 622 Mbit/s. It can process 512 ATM connections and requires three types of memories: the first one has 256 K-Bytes size, the second one 16 K-Bytes size and the third contains 4 K-Bytes. The circuit also contains a cache memory of 256 Bytes used for scheduling of emission and reception data.

## CONCLUSIONS

In this paper we follow a design methodology in order to implement the Adaptive Leaky Bucket studied as an example of traffic management. This algorithm uses feedback cells as Control-cells carrying parameters of QoS for every VC and permitting a dialogue between two nodes of the network.

Hardware/Software exploration has been performed in order to have more efficiency and flexibility. A solution that correctly implements the system functionality while meeting real-time requirements is produced. Simulation, synthesis and routing of the design are achieved using a 0.6 CMOS technology. The ASIC area is about 19 mm<sup>2</sup>. This circuit operates with clock frequency of 80 MHz and allows the cells transfer on 155 Mbit/s and 622 Mbit/s.

This work will be completed by the use of a hardware/software interface environment in order to test our architecture. This architecture will be integrated in a system allowing real-time image processing using wavelet transformation ■

*Mohamed ATRI is Ph. D. student at Faculty of Sciences, Monastir, Tunisia. He is a member of Laboratory of Electronics and Micro-Electronics. He obtained his DEA in physics (Option Micro-Electronics) at 1996. His researches interest, high level design using VHDL language, Hardware/Software Co-design and flow control for ATM networks in collaboration with the TIMA-CMP Laboratory, I. N. P. Grenoble, France.*

## REFERENCES

1. ATM Forum, " ATM Traffic Management Specification Version 4.0", April 1996.
2. P. Newman, "Traffic management for ATM local area network", IEEE Communications Magazine, pp. 44-50, August 1994.
3. M. Lemerrier & G. Pujolle, " Contrôle d'accès dans les réseaux ATM Etude des performances des 'Leaky Buckets', Réseaux et informatique répartie, Vol.3 - n 3/1993, pp 267-286.
4. M. Atri, R. Djemal, M. Abid, and R. Tourki, "An Adaptive Flow Control Algorithm for ATM LANs", in Proc of IEEE, IMACS CESA'98, 1-4 April, 1998.
5. M. Abid, A. Changuel, A. A. Jarraya, "Exploration of Hardware/Software design space through a Codesign of Robot Arm Controller", EURO'96, Sep. 16-20 1996, Geneva, Switzerland.
6. Standard VHDL Language reference manual, New-