VHDL-FORUM
for CAD in
EUROPE

IFIP
WG 10.2 / 10.5
ECIP
Esprit 2072

VHDL-Forum for CAD in Europe, Fall '94 Meeting
Grenoble, France - September 19, 1994

at EURO-DAC '94 with EURO-VHDL '94, Alpexpo and Alpes Congrès, Grenoble, France,
September 19-23, 1994

Proceedings
User Paper Session

Sponsors:
IFIP WG 10.2 / 10.5
ECIP Esprit 2072

System-Level Modeling of an ATM Node in VHDL

M. Cornero, M. Marchese, M. Chirico, F. Curatelli
DIBE – University of Genova
Via Opera Pia 11A, 16145 Genova, Italy

Abstract

The design of Broadband Integrated Service Digital Networks requires the investigation of specific techniques to guarantee the different requirements of the supported traffic classes. In this context, a unified environment for the simulation and the design of an ATM node is a useful approach for validating the effectiveness and the implementation aspects of different congestion control strategies. In this paper we describe how this approach can be followed using VHDL. In particular, a complete ATM node has been modeled in VHDL in a modular and flexible way, so that different alternatives in the node architecture can be easily verified. Compared to previous works concerning the software simulation of networks our approach is more focused on higher protocol layers, and on the functionality of a single network node.

1 Introduction

In the near future new telecommunication services, like videotelephony, videoconferencing, high-speed data, HDTV etc. will be added to existing services, such as voice and low speed data. With the evolution of telecommunication techniques, and in particular with the advances in optical transmission technologies, higher transmission speeds can be achieved, so that the support of different services on a unified Broadband Integrated Services Digital Network (B-ISDN) has become feasible.

Due to the different requirements of each communication service a new transfer mode has been defined for broadband networks. In particular, the Asynchronous Transfer Mode (ATM) technique was selected in 1987 by CCITT to be the transfer mode of the future B-ISDN. ATM is based on advanced technologies (optical transmission and VLSI) and is defined as a specific POTM (Packet Oriented Transfer Mode) where the information is statistically multiplexed.

The feasibility of some critical parts of an ATM node has been already demonstrated. For example the realization of an ATM switch is presented in [1], and an ATM layer chip, performing typical ATM functions, such as cell assembly and disassembly, is presented in [2]. However, due to the integrated support of different services, further investigation is required as concerns higher layers in the B-ISDN protocol model for ATM. In particular the satisfaction of each specific service requirement must be guaranteed by a *congestion control* mechanism. This topic has received a great deal of attention in the literature (see, for instance, [3] for a survey and [4] for specific papers). In particular, several works have addressed the issue of admission control ([5], [6], [7], [8] and [9] among others), as a means of guaranteeing quality of service to the connections in progress.

The purpose of the presented work is to build an experimental bench to 1) validate theoretical models of congestion control in ATM nodes through simulation, and to 2) study hardware and software architectures well suited for the implementation of those models. In particular we have modeled a complete ATM node in VHDL by adopting a modular approach, so different alternatives in the node architecture and functionality can be easily verified. Since the main purpose of the work is to study congestion control techniques, the other functions of the node, e.g. switching, are modeled at an high-level of abstraction in order to avoid unnecessary complications and to achieve a high simulation efficiency. The effectiveness of congestion control mechanisms is strongly related to the statistical parameters of the traffic sources applied to the network, so that information sources have been modeled very carefully, in order to reflect realistic conditions as much as possible.

Compared to previous works concerning the software simulation of networks ([10], [11], and [12], [13] for more recent works on distributed simulation techniques) our approach is more focused on higher protocol layers, and on the functionality of a single network node. Moreover, a unified environment for the simulation and the design of an ATM node, as presented in this paper, is a useful approach for validating different congestion control strategies and to evaluate, at the same time, their influence on the system architecture.

The paper is organized as follows. In Section 2 the basic characteristics of ATM are outlined. The basic data types used in the VHDL specifications are described in Section 3, while in Section 4 we illustrate our model of traffic sources. An overview of the complete ATM node description is presented in Section 5, while in Section 6 we present a congestion control technique that we have investigated. Some concluding remarks are illustrated in Section 7.

2 Overview of ATM

The basic characteristics of the ATM technique are [14], [15]:

1. The multiplexed information flow is organized in fixed size units (53 bytes), called cells. Each cell consists of a user information field (48 bytes) and a header whose main function is to guarantee the proper routing of each cell in the network (an error control procedure is applied on the header content);

2. No error protection or flow control on a link-by-link basis is provided, thanks to the high reliability of optical transmission. In this way a much higher rate than usual packet switching systems can be achieved, as the operations performed by the network on every single cell are considerably limited;

3. The switching principle adopts a connection-oriented routing, i.e. before information is transferred from the terminal to the network a logical/virtual connection setup phase must allow the network to do the reservation of the necessary resources, if these are available; if no sufficient resources are available the connection is refused to the requesting terminal. At call setup it is checked whether statistically enough resources are available. This means that with a certain probability it can be guaranteed that resources will be available and no queue overflow will occur. The probability of overflow can be dimensioned by queuing theory.

In addition to user cells, some other information, such as communication set-up signals, is transmitted over separate channels.

An important characteristic of ATM is that it takes advantage of the *bursty* nature of most traffic sources. Bursty sources generate information at a non-fixed rate, i.e. they alternate active periods, during which information is produced at near-peak rate, and idle periods, during which no information is produced. To take advantage of bursty sources ATM uses the statistical multiplexing technique which is more efficient than a static resource allocation (for example circuit switching) as regards the bandwidth utilization and the number of accepted connections.

However, a congestion control mechanism is needed in order to satisfy the Quality of Service (QoS) requirements of each service. Typical QoS requirements are expressed in terms of cell loss and cell delay rates. In general different services require different QoS; for example a relatively high cell loss rate but a small cell delay rate is tolerated in voice transmission, while the contrary holds for data transmission.

3 Basic data types

A part of the `ATM` package, containing type definitions, constant declarations ecc. which are used in the VHDL specifications, is illustrated in figure 1. As explained in the following sections the whole system is characterized in terms of the supported services, defined by means of the enumeration type `class_type`. Every constant associated to a specific service class is defined using a constant array, so that parametrized descriptions can be easily specified.

User cells and signalling are represented by means of records, as shown in Figure 1. As it is clear from cell record structure, cells do not contain any information field, since it is useless for our purposes. The cell record is an abstraction of the information contained in each cell header, relevant to congestion control. In particular each cell is uniquely identified by the node input `link` where the cell comes from, the cell traffic `class`, and the connection which generated the cell, specified by means of the `connection_id` field.

As regards the signalling record, a limited set of signals, defined by the `signalling_message` enumeration type, has been specified.

```
package atm is
    type class_type is (TELEPHONE, VIDEO, SLOW_DATA, FAST_DATA);
    type real_class_array is array(traffic_class) of real;
    type natural_class_array is array(traffic_class) of natural;
    -- constants
    constant PEAK_BAND : real_class_array :=
                         (TELEPHONE => 0.064,
                          VIDEO => 150.0,
                          ........          );
    ....
    -- cell and signaling types
    type cell_rec is record
        link : natural;
        class : class_type;
        connection_id : natural;
    end record;

    type signalling_message is (conn_req, conn_ack, conn_end);
    type signalling_rec is record
        link : natural;
        class : class_type;
        connection_id : natural;
        message : signalling_message;
    end record;
    ....
end atm;
```

Figure 1: *The ATM package.*

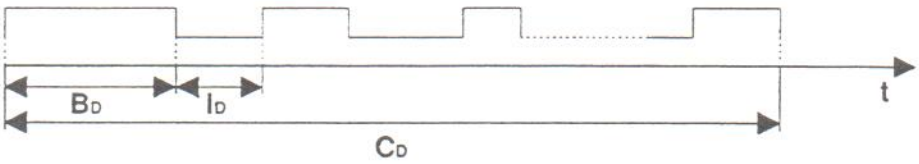


Figure 2: *A bursty connection.*

4 Modeling traffic sources

As already said, ATM takes advantage of *bursty* sources, characterized by active periods during which information is produced at peak rate, and idle periods during which no information is produced, as shown in figure 2. The information traffic is generated by a number of *connections*. A connection of a given traffic class h is completely statistically described by the *mean connection duration* C_D^h, the *mean burst duration* B_D^h and the connection *burstiness* b^h defined as the ratio between the peak and the mean bandwidth of traffic class h. In order to model the connection behavior, it is necessary to calculate the mean duration of idle periods (I_D^h) as follows: $I_D^h = B_D^h * (b^h - 1)$. During active periods, a connection produces information at the peak rate of its traffic class, which is of course smaller than the total channel capacity C_T.

In addition to the parameters characterizing a single connection, the behavior of a traffic class h is determined by the *mean connection generation* rate (C_G^h), i.e. the rate at which new connections requests of traffic class h are submitted to the network.

Link source. The information traffic supported by a single network link is model with the `link source` entity, whose structure and hierarchical decomposition is illustrated in figure 3. The information relative to traffic class h is generated by the `class source` entity with input generic parameter h. The basic block of our traffic source model is the `connection` entity with input generic parameter h, which generates the information of a single connection of traffic class h. As shown in figure 3 MAX_CONN connections are instantiated in each `class source` entity.

Class source. The `class source` entity is composed of the `connection activator` component which interacts with MAX_CONN connections. The VHDL description of the `class source` entity is shown in figure 4 a). The

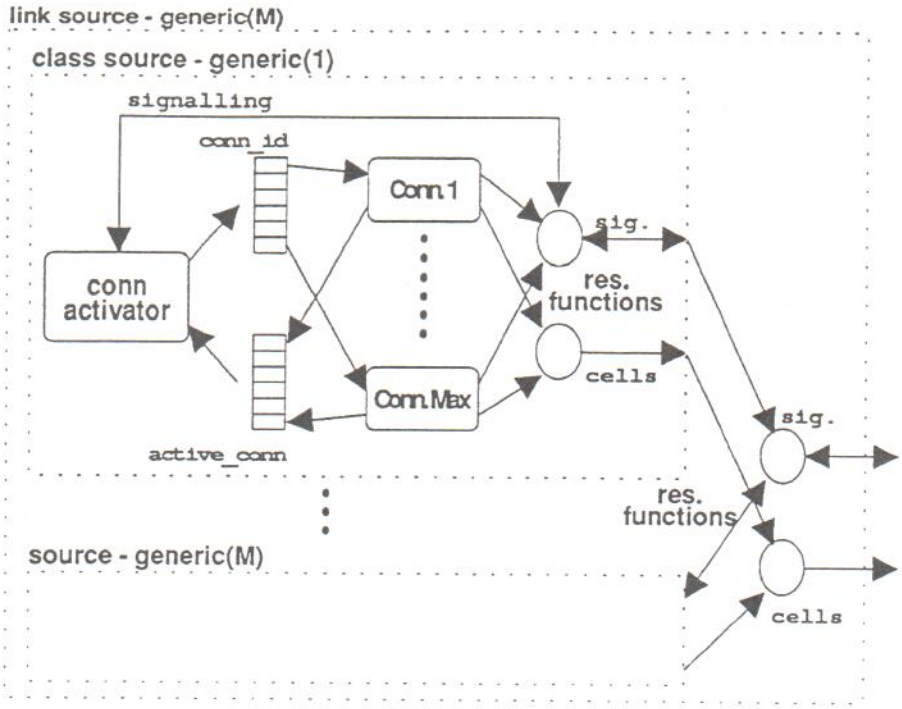


Figure 3: *The structure of the* `link source` *entity.*

interaction between the `connection activator` and the `connections` is mediated by the two arrays `conn_id` and `active_conn`. Initially every connection C_i is inactive, and this is indicated by `'0'` 's in all the items of the `active_conn` array. The `connection activator` generates connection requests, which are sent to the network by the signal `signalling`, at rate C_G. If the connection request signal $Creq_i$ of connection C_i is accepted by the network (see next sections), a connection acknowledgment signal, $Cack_i$, is sent back to the `connection activator`, which activates a connection as follows: an inactive `connection` is selected by looking for an item j in the `active_conn` array such that `active_conn[j] = '0'`; `conn_id[j]` is then set to i, an event that activates `connection` j. Once activated, connection j sets `active_conn[j]` to `'1'` and then starts the generation of its cells. When connection j terminates it sends a `connection_end` signal to the network, and it sets `active_conn[j]` to `'0'`, notifying its inactive state.

In order to obtain a realistic model of the traffic statistics a relevant number of `connections` (e.g. 500 - 1000) must be instantiated for each `class source`. As a consequence the number of input signals to the resolution functions illustrated in figure 3 is very large, causing a considerable degradation in the simulation run time, also taking into account that the number of generated cells is very large too. A solution to this problem is provided by the 'process' statement specified inside the 'generate' statement of figure 4 a), which *disconnects* all the drivers of the `cells` and `signalling` signals whose `connection_id` field is `0`. The disconnection is obtained by means of 'null' assignments and by declaring the `cells` and `signalling` signals as *bus*'s, as highlighted in figure 4 a). This approach works well since the `connection_id` field of the cell signals produced by connection C_i is set to i only for a very small time interval (1 ns), and to '0' for the rest of the time (see below). This optimization is essential as it yealds a dramatic improvemenmt in the simulation times.

Connection. The behavioral description of the `connection` entity is illustrated in figure 4 b). The `Conn` process waits until `connection_id > 0`, an event caused by the `connection activator`, as already explained. The `Conn` process generates statistically the connection length. Bursts are generated by the `Burst` loop, and cells are generated at the source peak rate by the `Cell` loop. Note that one nanosecond after each cell production the `connection_id` field of the `cells` signal is set to zero in order to resolve multiple drivers at the higher hierarchical level, as described in the previous paragraph. Every statistical parameter is generated using the `exponential` distribution, which is well suited

```
entity class_source is
    generic(class: class_type);
    port(cells : out resolved_cell_rec bus;
         signalling : out resolved_sig_rec bus);
end class_source;

architecture arc of class_source is
signal conn_id : natural_array(1 to MAX_CONN);
signal active_conn : bit_vector(1 to MAX_CONN);
signal cell_tmp : cell_array(1 to MAX_CONN);
signal sig_tmp : sig_array(1 to MAX_CONN);
....
begin
REQ: conn_generator
    generic map(class)
    port map(conn_id,active_conn,signalling);

G:  for I in 1 to MAX_CONN generate
        CONN: connection
        generic map(class)
        port map(conn_id(I), active_conn(I),
                 cell_tmp(I),sig_tmp(I));

        process (cell_tmp(I), sig_tmp(I))
        begin
            if cell_tmp(I).connection_id > 0 then
                cells <= cell_tmp(I);
            else
                cells <= null;
            end if;
            if sig_tmp(I).connection_id > 0 then
                signalling <= sig_tmp(I);
            else
                signalling <= null;
            end if;
        end process;
    end generate;
end arc;
```

a)

```
entity connection is
    generic(class: class_type; link : natural);
    port(connection_id : in natural;
         active_flag : out bit;
         cells : out cell_rec;
         sig_out : signalling_rec);
end connection;

architecture arc of connection is
gen: process
....
constant cell_time : real :=
    (1/PEAK_BAND(class)) * T_cell;
begin
  wait until connection_id > 0;
  active_flag <= '1';
  conn_len := exponential(MEAN_CONN_LEN(class));
Burst: while conn_len > 0 loop
      burst_len := exponential(MEAN_BURST_LEN(class));
Cell: while burst_len > 0 and conn_len > 0 loop
        cells <= transport
              (link => link class => class,
               connection_id => connection_id),
        cells.connection_id <= 0 after 1 ns;
        wait for cell_time * 1 us;
        burst_len := burst_len - cell_time;
        conn_len := conn_len - cell_time;
      end loop;
      idle_len := exponential(MEAN_IDLE_LEN(class));
      if idle_len > conn_len then
         idle_len := conn_len;
      end if;
      wait for idle_len * 1 us;
      conn_len := conn_len - idle_len;
  end loop;
  sig_out <= (link => link, class => class,
              connection_id => connection_id,
              message => conn_end);
  sig_out.connection_id <= 0 after 1 ns;
  active_flag <= '0';
end process;
end arc;
```

b)

Figure 4: *The VHDL specification of a) the* `class source` *entity and b) the* `Connection` *entity.*

for communication applications.

Connection activator. The `connection activator` entity is composed of two concurrent processes `req_generator` and `conn_activator`. The `req_generator` process generates the connection requests signals at rate C_G^h, while connection acknowledgments are processed by the `conn_activator` process.

5 The ATM node model

Our model regards an ATM node with L_{in} incoming and L_{out} outgoing links, whose input traffic is generated by M different traffic classes per input link, as explained in the previous section. The hierarchical VHDL decomposition of the complete system, obtained by means of entity declarations and component instantiations, is shown in Figure 5. In this section we describe the functionality of the complete ATM node by analyzing each block separately, except the `Access Controller` and `Bandwidth Allocator` blocks which are the subject of section 6.

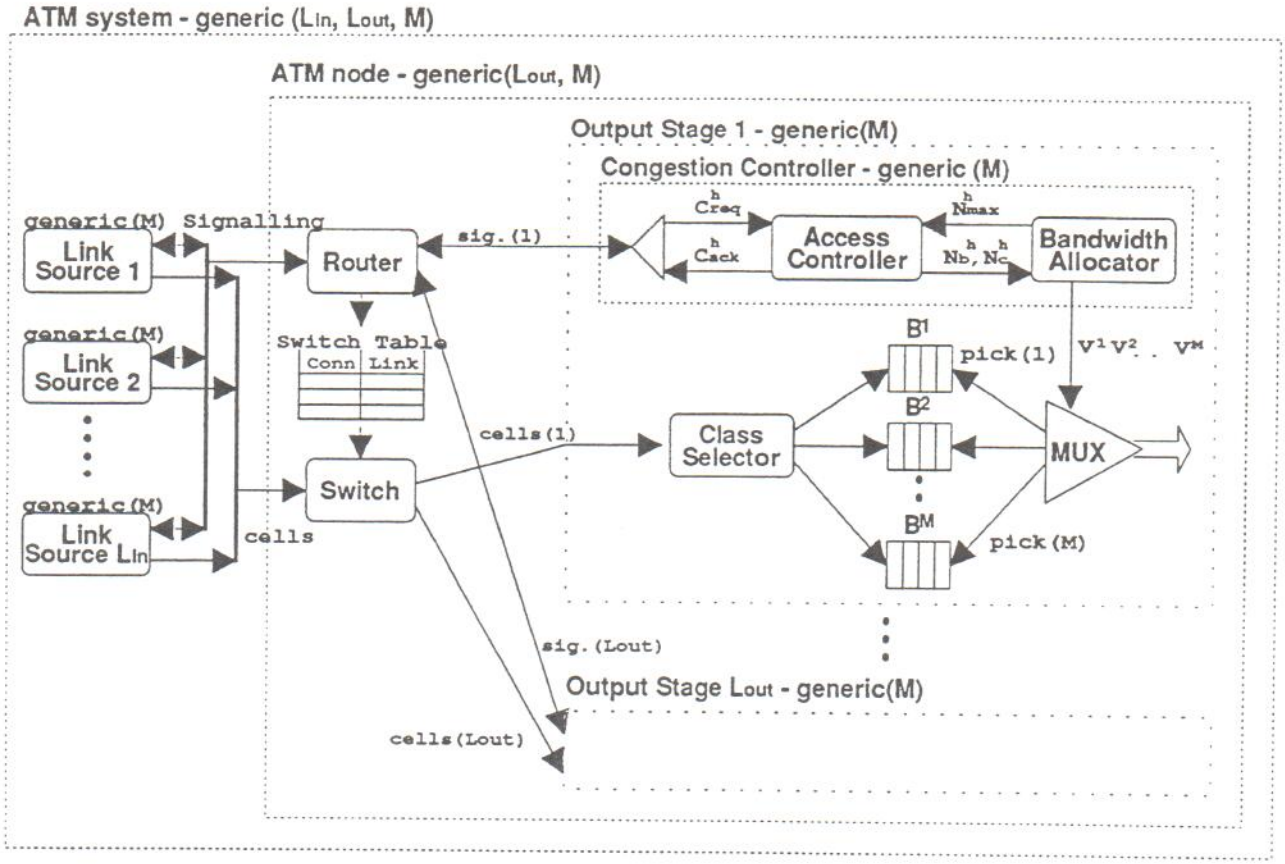


Figure 5: *The ATM node model.*

5.1 ATM system

At the highest hierarchical level, the complete system is modeled as an entity with three generic input parameters L_{in} and L_{out} and M, indicating the number of input and output links and the number of traffic classes respectively. As shown in figure 5, L_{in} input link sources, each composed of M traffic classes, are instantiated in order to model the node input traffic. The global cell and signalling traffic is conveyed into the two signals `cells` and `signalling` by means of the same resolution functions illustrated in the previous section.

Modularity in the system description is achieved by means of *generics*, combined with generate statements for component instantiations, as in the case of the L_{in} input link sources. An extensive use of this specification style results in a general and flexible description of the compete system, characterized in terms of M traffic classes, L_{in} input links and L_{out} output links.

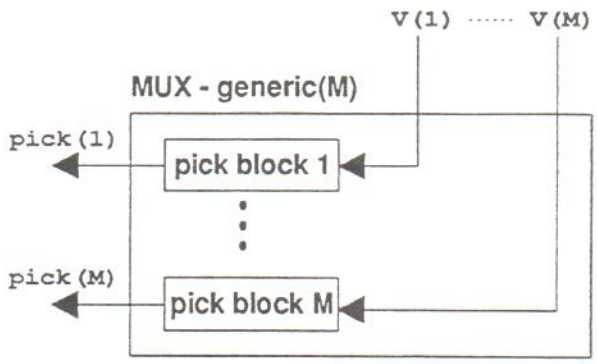
5.2 ATM node

The ATM node is modeled as an entity with two input signals `cells` and `signalling`, and two generic input parameters L_{out} and M, as indicated in Figure 5. As already said ATM is *connection-oriented*. As a consequence all the cells of a connection follow the same path through the network. The path is determined at the connection set-up by means of signals which are processed by the `Router` element contained in each node. In this paper we do not deal with the routing strategy, which is described in detail in [16]. Concerning the functionalities addressed in this paper, the `routing` component is only used to address connection requests to the proper `Access Controller` component, which decides whether to accept or reject connections by following the congestion control strategy described in section 6. After a connection C_i has been accepted the router stores the association between that connection and the selected output link L_j in the `Switch_Table`, so that every cell belonging to the connection C_i will be sent to output the link L_j by the `Switch` component.

```
entity pick_block is
    port (V_h : in real;
          pick : out bit);
end pick_block;

architecture arc of pick_block is
begin
P: process
variable t: real;
begin
    t := (C_T / V_h) * T_cell;
    wait for t*1 us;
    pick <= transport '1';
    pick <= transport '0' after 1 ns;
end process;
end arc;
```

a)



b)

Figure 6: *a) The pick and b) the* Mux *entities.*

Switch. Although the switching element is an essential part of an ATM node, it does not affect the congestion control mechanism in our model, so that we have modeled it as an ideal component, i.e. it does not introduce any delay and it does not loose any cell. Each incoming cell is sent to the proper output link by looking up the `Switch_table`, whose entries are filled in by the router at the connection set-up.

5.3 Output stage

Each output stage is composed of a `Congestion Control` component, which is explained in the next section, a `Class Selector`, M `Output Buffers` and a `Mux`, as indicated in figure 5.

Class Selector. An important characteristic of our model is that each output stage is composed of a separate buffer for each traffic class. The `Class Selector` component sends the information cells to the proper `Output Buffer`, according to the traffic class of each cell.

Output buffer. In the node model output buffers are FIFO queues whose size is determined off line on the basis of performance requirements and the declared traffic intensity of the corresponding class. A cell is *lost* if it is sent to a full buffer, while a cell is considered *delayed* if the delay introduced by the FIFO exceeds a given threshold T_d^h.

As we explain in the next section, the efficiency of the congestion control strategy is measured by the number of accepted and rejected connections, while the correctness of the strategy is verified by checking whether the quality of service requirements (QoS), in terms of cell loss and cell delay rates, are satisfied. This is done by maintaining two counters in each `Output Buffer` component, indicating the total number of lost and delayed cells respectively. No other parameters are relevant concerning congestion control. In particular, since we are not interested in network simulations, but only in the analysis of a single node, cells do not need to be sent out of the node. Therefore each output buffer is simply modeled with a state variable `FIFO_state`, indicating the number of cells present in the FIFO. More precisely, each time a cell is queued `FIFO_state` is incremented by one, while each time a cell is picked out `FIFO_state` is decremented. If `FIFO_state` equals the buffer size and a cell enters the buffer, the cell is lost. In this case the `lost_cells` counter is incremented instead of `FIFO_state`. To check weather an incoming cell is delayed, and so to update the `delayed_cells` counter, the delay of the cell is estimated as follows:

$$cell\_delay = FIFO\_state * \frac{C_T}{V^h} * T_{cell},$$

where the parameter V^h, available from the `Bandwidth Allocator` (see section 6), is the bandwidth allocated to class h and T_{cell} is the time required to transmit a single cell.

```
entity access_controller is
    generic(M : natural);
    port(N_max : in natural_class_array;
         sig_in : in signalling_rec;
         sig_out : out signalling_rec;
         Nc : inout natural_class_array;
         Nb : inout natural_class_array);
end access_controller;

architecture arc of access_controller is
begin
P: process (sig_in)
begin
    if sig_in.message = conn_end then
        Nc(sig_in.class) <= Nc(sig_in.class) - 1;
    else if sig_in.message = conn_req then
        if N_max(sig_in.class) = Nc(sig_in.class) then
            Nb(sig_in.class) <= Nb(sig_in.class) + 1;
        else
            Nc(sig_in.class) <= Nc(sig_in.class) + 1;
            sig_out <= (link => sig_in.link,class => sig_in.class,
                        message => conn_ack,
                        connection_id = sig_in.connection_id);
        end if;
    end if;
end process;
end arc;
```

Figure 7: *VHDL specification of the* `Access Controller` *entity.*

Mux. The `Mux` entity sends the `pick` signals to the output buffers, indicating the consumption of a cell. `Pick` signals are sent to the buffer B_h at a rate which depends on V^h. Similarly to the modeling of traffic sources, a sequential specification of such a behavior would require an unnecessary overhead, while the exploitation of the VHDL capabilities to express concurrency provides a straightforward solution. The `pick` signal for class h is generated by the simple process illustrated in figure 6 a). The `pick_block` component is then instantiated M times, once for each output buffer, in the `Mux` entity as indicated in figure 6 b).

6 Congestion control

As already said, in our model we suppose that the traffic in the network is divided into M classes of service, each of one characterized by statistical parameters, like peak and average transmission rate, as well as by QoS requirements, like cell loss probability and cell delay.

The QoS requirements are guaranteed by the `Access Controller`. This component decides weather a connection request for traffic class h can be accepted or not, by checking if the total number of already accepted connections N_c^h, plus the new one exceeds a threshold value N_{max}^h. More precisely, as indicated in Figure 7, the following condition is checked

$$N_c^h + 1 \leq N_{max}^h. \tag{1}$$

If condition 1 is satisfied the connection is accepted, otherwise it is rejected. Two counters N_c^h and N_b^h are maintained in order to keep track of the number of active and blocked (rejected) connections respectively.

The values N_{max}^h is calculated by the `Bandwidth Allocator`, and is given by

$$N_{max}^h = min\{N_{max,D}^h, N_{max,L}^h\} \tag{2}$$

where $N_{max,D}^h$ is the maximum number of connections the output link can support as regards the cell delay requirement and $N_{max,L}^h$ is the maximum number of connections the link can support as regards the cell loss requirement for traffic class h.

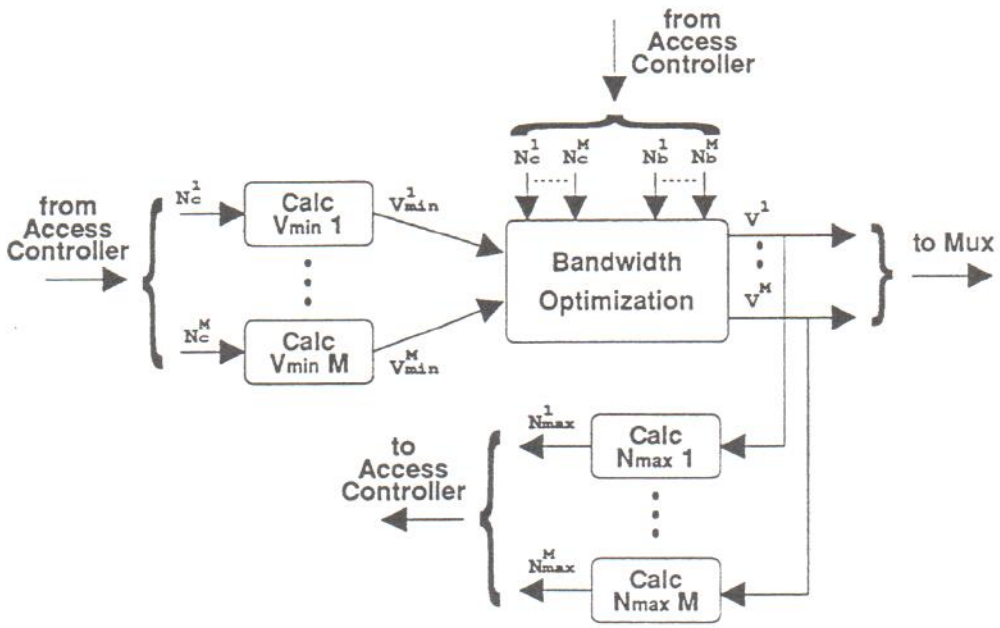


Figure 8: *Computation flow and hierarchical decomposition of the* `Bandidth Allocator`.

The computation of $N^h_{max,L}$ and $N^h_{max,D}$ is based on a statistical approach derived from the queuing theory, and it constitutes the kernel of the congestion control strategy. More precisely,

$$\begin{array}{ll} N^h_{max,L} & = max_N\{N \mid \sum_{n=0}^{N} Ploss^h(n) v^h_{n,N} \leq \epsilon^h\} \\ N^h_{max,D} & = max_N\{N \mid \sum_{n=0}^{N} Pdelay^h(n) v^h_{n,N} \leq \delta^h\} \end{array} \tag{3}$$

where $Ploss^h(n)$ ($Pdelay^h(n)$) represents the cell loss (delay) rate of traffic class h given that there are n active connections (i.e. not idle), and $v^h_{n,N}$ is the probability of having only n active connections out of N accepted ones. ϵ^h and δ^h are the QoS requirement of traffic class h.

The computation of N^h_{max} depends on the bandwidth allocated to traffic class h. The flow of the required computation and the hierarchical VHDL decomposition of the `Bandwidth Allocator` is depicted in Figure 8. The aim of the bandwidth allocation is to partition the output link capacity C_T among the different traffic classes in order to minimize the total number of rejected connections, $\sum_{h=1}^{M} N^h$. The capacity partitions V^h_m, are dynamically computed by the `Bandwidth Optimization` at discrete time instants $m = 0, k, 2k, ...$, where k is the length of the allocator intervention period. The optimization phase must be preceded by the determination of the minimum bandwidth V^h_{min} which assures service quality (QoS) for the connections *already in progress* for each traffic class traffic, i.e. N^h_c. The set of inequality constraints

$$V^h_m \geq V^h_{min} \tag{4}$$

and the equality constraint

$$\sum_{h=1}^{M} V^h_m = C_T \tag{5}$$

fix the boundaries of the feasibility space for the new capacity partitions. As illustrated in figure 8, each $\{V^h_{min}\}$ can be calculated independently from each other. In particular, similarly to equations 3, V^h_{min} is defined as follows:

$$V^h_{min} = min_V\{V \mid \sum_{n=0}^{N^h_c} Ploss^h(n,V) v^h_{n,N} \leq \epsilon^h \; and \sum_{n=0}^{N^h_c} Pdelay^h(n,V) v^h_{n,N} \leq \delta^h\} \tag{6}$$

where the dependency from V has been explicitly indicated. The evaluation of the quantities referenced in equations 3 and 6 requires complex computations based on the queuing and probability theory. Similar quantities must be evaluated in the bandwidth optimization procedure. The details on the required computations can be found in [9].

7 Concluding remarks

Compared to other modeling alternatives, e.g. using another programming language like C, VHDL offers several advantages, such as a the possibility to exploit architectural decomposition, in addition to functional decomposition, through the use of component instantiation statements. By means of this feature, the adoption of a modular specification style, which is mandatory in case of complex system specifications, is greatly encouraged. Another significant advantage is the support of concurrent specifications, which has simplified the system description to a great extent.

Furthermore, through the use of commercial analysis and synthesis tools, VHDL provides a path to hardware implementation, which is the final objective of our work, at least as concerns the congestion control strategy. Although we have described the complete system at behavioral level, we are already able to investigate different architectural alternatives by means of a detailed decomposition, together with the behavioral VHDL specifications of all the computational blocks. For example, as illustrated in figure 8, different computations can be performed in parallel (e.g. the set of V^h_{min} and the set of N^h_{max}) so that serial/parallel implementation trade offs can be analyzed. Moreover we have located a common set of critical operations to be performed in every block, which suggest to design a hardware (VLSI) accelerator to speed up execution. However, the complexity of the required computations precludes the possibility of a VLSI implementation of the complete congestion control system, so that we expect that a mixed hardware/software architecture will probably be the most appropriate solution.

References

[1] P. Barri and J. A. O. Goubert, "Implementation of a 16 to 16 switching element for ATM exchanges", *IEEE Journal on Selected Areas in Communications*, vol. 9, pp. 751–757, June 1991.

[2] C. A. Johnston and H. J. Chao, "The ATM layer chip: An ASIC for B-ISDN applications", *IEEE Journal on Selected Areas in Communications*, vol. 9, pp. 741–750, June 1991.

[3] T. Hong and T. Suda, "Congestion control and prevention in ATM networks", *IEEE Network Magazine*, vol. 5, pp. 10–16, July 1991.

[4] *Special Issue on Congestion Control in High-Speed Packet Switched Networks*. IEEE Journal on Selected Areas in Communications, 1991.

[5] F. C. Schoute, "Simple decision rules for acceptance of mixed traffic streams", *in Proc. ITC*, Torino, Italy, September 1988.

[6] G. Galassi, G. Rigoglio, and L. Fratta, "ATM: Bandwidth assignment and bandwidth enforcement policies", *in Proc. GLOBECOM '89*, pp. 1788–1793, Dallas, Texas, November 1989.

[7] T. Kamitake and T. Suda, "Evaluation of an admission control scheme for an ATM network considering fluctuations in cell loss rate", *in Proc. GLOBECOM '89*, pp. 1774–1780, Dallas, Texas, November 1989.

[8] R. Bolla, F. Davoli, A. Lombardo, S. Palazzo, and D. Panno, "Adaptative bandwidth allocation by hierarchical control of multiple ATM traffic classes", *in Proc. IEEE INFOCOM '92*, pp. 30–38, Florence, Italy, May 1992.

[9] R. Bolla, F. Danovaro, F. Davoli, and M. Marchese, "An integrated dynamic resource allocation scheme for ATM networks", *in Proc. IEEE INFOCOM '93*, March 1993.

[10] H. T. Mouftah and E. A. Krause, "Computer-aided design and performance evaluation of communication controllers with mixed traffic", *IEEE Journal on Selected Areas in Communications*, vol. 6, pp. 180–189, January 1988.

[11] V. S. Frost, W. Larue, and K. S. Shanmugan, "Efficient techniques for the simulation of computer networks", *IEEE Journal on Selected Areas in Communications*, vol. 6, pp. 146–157, January 1988.

[12] A. Bhimani and S. Ghosh, "Modeling and distributed simulation of complex broadband ISDN networks under overload on loosely-coupled parallel processors", *in Proc. IEEE Int.l Conf. on Communications*, pp. 1280–1284, 1992.

[13] A. Chai and S. Ghosh, "Modeling and distributed simulation of a broadband-ISDN network", *IEEE Computer*, vol. 26, pp. 37–52, September 1993.

[14] M. de Prycker, *Asyncronous Transfer Mode Solution for Broadband ISDN*, Ellis Horwood Limited, 1991.

[15] M. Listanti and A. Roveri, "Integrated services digital networks: Broadband networks", *European Transactions on Telecommunications*, vol. 2, Jan-Febr 1991.

[16] R. Bolla, F. Davoli, and M. Marchese, "A distributed routing and access control scheme for ATM networks", *in to be published in Proc. ICC '94*, 1994.