A VLSI Sequencer Chip for ATM Traffic Shaper and Queue Manager

H. Jonathan Chao, Member, IEEE, and Necdet Uzun

Abstract—The asynchronous transfer mode (ATM) technique provides a standardized and flexible scheme to transport and switch traffic effectively for different services. To provide satisfactory quality of service (QOS) to all users on the network, it is necessary to control users' traffic so that network resources, such as transmission bandwidth and buffer capacity, can be efficiently and fairly utilized by all users while still meeting the individual QOS requirement. However, due to the natural randomness of the broad-band traffic (e.g., data file transfer and variable bit-rate video communication), it is difficult to control users' traffic effectively so that network congestion is prevented or, at least, occurs rarely. In this paper, we propose to control users' traffic at two places in the network: at the user-network interface (UNI) by a traffic enforcer, and at the network-node interface (NNI) by a queue manager. The traffic enforcer adopted in our work contains a buffer to delay and reshape the violating cells that do not comply with some agreed-upon traffic parameters, and thus is also called a traffic shaper. The queue manager manages the queued cells in network nodes in such a way that higher priority cells are always served first, low-priority cells are discarded when the queue is full, and any interference between same-priority cells is prevented. We present in the paper our proposed architectures for the traffic shaper and the queue manager. We have implemented and tested a key component, called the Sequencer chip, to realize both architectures. The Sequencer chip is implemented using 1.2-μm CMOS technology. It contains about 150K transistors, has a die size of 7.5 mm × 8.3 mm, and is packaged in a 223-pin ceramic pin-grid-array (PGA) carrier.

I. INTRODUCTION

BROAD-BAND integrated service digital networks (B-ISDN's) provide end-to-end transport for a wide range of broad-band services in a flexible and efficient manner via the asynchronous transfer mode (ATM) technique [1]. Due to the natural burstiness of the broad-band traffic, congestion control is required to effectively and fairly allocate the shared network resources (e.g., transmission bandwidth and buffer capacity) so that satisfactory quality of service (QOS) to all network users can be provided. Congestion in the ATM network arises when the offered load exceeds the capacity of the network. A suitable set of congestion controls for ATM networks includes admission control, traffic enforcement, queue management, and reactive flow control [2].

The function of admission control is to decide whether a new virtual channel connection should be admitted to the network (or rejected) based on the knowledge of the current network status (e.g., loading and available resources), the connection's traffic descriptor (including parameters such as average and peak bit rates, and maximum burst lengths), and performance objectives (such as cell loss probability, cell transfer delay, and cell delay variation). A new connection will be admitted only if the QOS can be met and the service quality of calls in progress will not be affected. Several schemes for the call admission control have been proposed [3], [4].

Reactive flow control alleviates the instantaneous overload condition in the network so that the cell loss in the network is reduced and the users' QOS is maintained. Two reactive flow control schemes have been suggested: backward congestion notification (BCN) [5] and forward congestion indication (FCI) [6]. For both schemes the source terminal is, either directly or indirectly, informed by the network to adjust its data rate when the network is congested. However, because of the large product of the transmission bandwidth and the round-trip delay, many cells will still be in transit between the source terminal and the congested node, and may be lost by the time the source terminal receives the congestion information and starts to regulate its traffic flow.

A traffic enforcer/shaper, as shown in Fig. 1, monitors (or polices) each virtual connection to ensure that its traffic flow into the network conforms to the traffic descriptor, which could be specified at call setup. If the user's traffic does not conform to the traffic descriptor, some action has to be taken against the violating traffic. For instance, the violating cells could be dropped [2], [7], [8], temporarily stored in a reshaping buffer [9]–[11], or transmitted to the network anyway, but with the cells tagged with a lower priority [12]–[14]. The last option implies that some sort of priority mechanism has to be implemented within the network. We have adopted a traffic enforcement scheme that has a buffer to delay and reshape the violating cells. If the buffer is full, some cells are transmitted to the network but are tagged with a lower priority. A novel architecture to implement the traffic enforcer (or a traffic shaper at the customer premises as shown in Fig. 1) has been proposed in [15]. This architecture is capable of performing the traffic enforcement for a large number of virtual channels (e.g., a few thousand) on each input line.
Since a future network node will have to deal with traffic having different requirements, the use of multiple priorities and switch control functions serves as a possibility for distinguishing among different traffic types. A queue manager as shown in Fig. 1 manages the queued cells in a network node in such a way that higher priority cells are always served first, low-priority cells are discarded when the queue is full, and any interference between same-priority cells is prevented. By assigning a departure sequence number to every cell, the effect of long-burst traffic on other cells is avoided. We have proposed an architecture to implement the queue manager using a mechanism called VirtualClock [16]. The architecture has the capability to support thousands of priority levels [17].

To implement the traffic enforcer/shaper and the queue manager, we apply the concepts of fully distributed and highly parallel processing to schedule the sending sequence or discarding sequence of ATM cells. A VLSI chip (called the Sequencer), which contains about 150K CMOS transistors, has been implemented in a regular structure so that the number of virtual channels or priority levels can grow flexibly. Sections II and III depict the architectures for the traffic enforcer/shaper and the queue manager, respectively. Section IV describes the detailed design of the Sequencer chip, and Section V gives the conclusion.

II. TRAFFIC ENFORCER/SHAPER

Fig. 2 illustrates an example showing three different actions applied to the violating cells: discarding, tagging, and reshaping them by delaying (adopted in this paper). Whether or not the incoming cell can immediately be sent out usually depends on its current “token” count (or credit). The token count can be, for example, incremented periodically with the average arrival rate. This token count has a maximum limit, similar to the “credit line” in the bank credit account. The number of permissible transmitted cells in a burst (namely, continuous transmission at its peak bit rate) cannot exceed this predetermined value, i.e., maximum burst length. When there is no token, the newly arriving cell will be discarded, tagged, or delayed. Besides the constraint on the burst length, the peak bit rate of each virtual channel (VC) must also be limited.

In this example, we have assumed the average rate to be 30 Mb/s on a 150-Mb/s channel, equivalent to the average arrival interval of five cells. Tokens are thus generated every five cell time slots as shown in Fig. 2. The peak bit rate is assumed to be 75 Mb/s, or the minimum departure interval (MI) is two cells. The number of tokens for the VC is initially assigned the maximum burst length (three). Let us also assume that six cells, a through f,
arrive at time slots 6, 7, 8, 10, 12, and 14, respectively. When cell \( a \) arrives, the token count is three, and is then decremented to two after cell \( a \) is immediately sent out. Since cell \( b \) arrives at time slot 7 and violates the minimum departure interval (two), it will be discarded, tagged (the shaded box in Fig. 2), or delayed by one cell time slot and sent out at time slot 8. When cell \( f \) arrives at time slot 14, there are no more tokens left. Therefore, it is either discarded, tagged, or delayed until a token is generated at time slot 20. The basic idea of performing credit management is to keep a real time clock and assign a departure time (DT) to every arrived cell based on its current token count. The algorithm of assigning a DT to every arrived cell according to each VC's traffic descriptor (average and peak bit rate, and maximum burst length) can be found in [15].

Fig. 3 shows an architecture to implement the traffic enforcer/shaper. It consists of a cell pool, an idle-address FIFO, and a credit manager, which includes a memory, a processor, and a Sequencer. The processor executes the algorithm by assigning DT's to arrived cells. Upon a cell's arrival, if the cell pool is not full, the cell is stored in the cell pool with a writing address given by the idle-address FIFO, which contains the addresses of current vacant cell locations in the pool. If the cell pool is full (i.e., the idle-address FIFO is empty), the address field of the right-end entry of the Sequencer (see below) is used to read out the corresponding cell in the pool, whether or not its departure is due. The affected cell is then tagged and transmitted to the network. The vacant cell location will then be filled by the newly arrived cell. To avoid a malicious user queuing up too many violating cells in the cell pool, the number of backlogged cells for each VC is monitored. If the number exceeds a predetermined value, a warning message is sent back to this connection's source to slow down its traffic flow, or its following cells are simply discarded.

Before the cell is written into the cell pool, its virtual channel identifier (VCI) is first extracted by the processor in order to access the corresponding intermediate data stored in the memory (with \( N \) entries for \( N \) VCs) to calculate the arriving cell's DT. The cells' departure times are stored with the cell's addresses in the Sequencer in descending order (the smaller value is on the right of the larger one). As the real time clock ticks, the smallest value of the DT (at the right end of the Sequencer, \( DT' \)) is compared to the real time value. If the \( DT' \) is equal to (or less than) the real time, meaning that the departure is due (or overdue), its corresponding cell stored in the pool is accessed and sent out. In the meantime, the address of the available vacant location is written back to the idle-address FIFO for the next arriving cell. The architecture is not restricted by the number of the virtual channels \( (N) \) on the input line, nor by the size of the cell pool, because the processor will only assign a departure time to the cell when it arrives. With the assistance of the Sequencer, the processor only needs to check the departure time once in each cell time slot (2.83 \( \mu \)s).

Each entry in the Sequencer has two values: the cell's DT and the cell's physical address \( A \) in the pool. The operation of the Sequencer is illustrated in Fig. 4. Assume that the value of \( DT' \) is less than that of \( DT_{n-1} \). When a new cell with departure time \( DT_n \) arrives, all pairs on the right of \( A_i \), including the \( A_i \) itself, remain at their positions while others are shifted to the left, and the vacant position is inserted with the pair of the new cell's departure time (\( DT_n \)) and address (\( A_n \)).

### III. QUEUE MANAGER

By assigning a departure sequence number (DS) to every cell based on its average arrival rate (AR), which could be negotiated at the call setup, one can provide fairness among all virtual channels and set up resource firewalls to prevent interference among them. The queue manager always serves the cell that has the smallest value
of the DS first. The DS assignment algorithm that is based on the VirtualClock concept [16] is depicted below:

1) upon the arrival of the first cell of connection $i$, its $DS_i = \text{real time}$, where the real time can be the value of a counter incremented per cell time ($2.83 \mu s$);
2) upon receiving every cell from connection $i$, its $DS_i = \text{maximum} \{ \text{real time}, DS_i + 1/AR_i \}$.

Fig. 5 shows an example of assigning the DS to the arrived cells and achieving fairness. Let us assume that the input X’s AR = 0.1 cell/time slot, and the input Y’s AR = 0.2 cell/time slot. Initially, the real time is reset to zero. Immediately following the reset, four consecutive cells arrive from input Y, and then two consecutive cells arrive from input X, as shown in Fig. 5(a). The DS of the first cell from the Y input is assigned the value of the real time (zero), and the cells that follow are assigned the values 5, 10, and 15, respectively. When the first cell of the X input arrives, the real-time value is 4, which is then assigned to the cell. For the X’s second cell, its DS is set to maximum [5, 4 + 10], or 14. Fig. 5(b) shows the DS’s of all six arrived cells. Based on these DS values, the server will transmit cells with smaller values sequentially, as shown in Fig. 5(c).

Fig. 6 shows an architecture to implement the queue manager. Cells from L inputs are time-division multiplexed first and then written into the cell pool at idle addresses retrieved from a FIFO (only valid cells are stored in the pool). L is the number of cells that are routed through an ATM switch network and arrive simultaneously at an output port (as shown in Fig. 1). It has also been proven that, with uncorrelated traffic from input ports uniformly directed to all output ports, the probability of more than 12 cells destined for any particular output port in each cell time interval is very low (e.g., $10^{-10}$) [18]. The write/read controllers generate proper control signals for all other functional blocks. A pair that is composed of a cell’s DS and its corresponding address is stored in the Sequencer in such a way that smaller DS’s are always at the right of larger DS’s. Therefore, cells with smaller DS’s will be accessed earlier by the read controller. Once the pair has been accessed, the address is used to read out the corresponding cell in the cell pool, and the address of the available vacant location is written back to the idle-address FIFO. For traffic that has different service classes, the priority field of each cell can be constructed from the combination of the service class and the DS. The priority field in the cells that are routed in the internal ATM switch network can be arranged by cascading the service-class bits (most-significant bits) and the DS (least-significant bits). Consequently, the Sequencer will sort the cells’ priority levels rather than the DS’s.

When the cell pool is full (i.e., the idle-address FIFO is empty), the priority field at the leftmost position of the Sequencer (e.g., $P_a$) is compared to that of the newly arrived cell ($P_a$). If $P_a$ is smaller than $P_a$, the pair of $P_a$ and $A_a$ is pushed away from the Sequencer as the new pair ($P_a$ and $A_a$) is inserted in the Sequencer. Meanwhile, the cell with address $A_a$ in the pool is overwritten with the new cell. However, if $P_a \geq P_a$, the new cell is discarded instead.
IV. SEQUENCER CHIP IMPLEMENTATION

This section describes the implementation of a general-purpose Sequencer chip. After describing the basic functions, we show how the major building blocks, such as the subtractor and the register, are designed to reduce the area and the power. A chip summary and the testing results are given.

A. Basic Functions

Both the traffic shaper’s architecture (Fig. 3) and the queue manager’s architecture (Fig. 6) require a Sequencer to sort the cells’ DT’s or DS’s in descending order. We have implemented the Sequencer with a VLSI chip, which is essentially a 256-word sorting-memory chip. Due to its general sorting function, it can also be used for other scheduling algorithms and priority assignment procedures. Fig. 7 shows the building block of the chip, where the circuit in the dashed box is a module and is repeated 256 times in the chip. Each module has a 24-b register, which stores the 14-b DT/DS values and the 10-b address. A single chip can accommodate a cell pool capacity of up to 256 cells and DT/DS values (or the number of priority levels in some applications) up to 2\(^{14} - 1\). This provides the DT/DS ranging from 0 to 4095, with the ability to handle services with bit rates equal to or higher than 33 kb/s, or (48 bytes $\times$ 8 b/byte)/(2.83 $\mu$s/cell $\times$ 4096 cells), if each cell’s 48-byte payload is filled with service information. Any services with bit rates lower than 33 kb/s will require a larger DT/DS range. By cascading multiple Sequencer chips in series or in parallel, a larger cell pool (e.g., a few thousand cells) or a larger DT/DS value can be supported.

Since every module is identical, let us examine the operations of an arbitrary module, say module $i$. When a new pair of the DT/DS and the address field, denoted by $Z_{0,23}$, is to be inserted into the Sequencer, it is first broadcast to every module. By comparing the DT/DS values ($Q_{0,13}$) of module $(i-1)$ and module $i$, and the new broadcast value ($Z_{0,13}$), the controller generates proper signals, $cp$, $cx$, $cz$, and $clk$, to shift the broadcast value ($Z_{0,23}$) into the 24-b register in module $i$, shift module $(i-1)$’s $Q_{0,23}$ to the register, or retain the register’s original value. Table I lists these three possible actions performed by the controller, where $X_{0,13}$ is the module $(i-1)$’s $Q_{0,13}$. The $b_{out}$ is the borrow-out of ($Z_{0,13} - Q_{0,13}$), and the $b_{in}$ is the borrow-out of ($Z_{0,13} - X_{0,13}$). Since the smaller DT/DS is always on the right of the larger one, the case where $Q_{0,13} < X_{0,13}$, or $b_{out}b_{in} = 01$, will not happen.

When a cell with the smallest DT/DS value is to be transmitted, its corresponding address will be shifted out from the Sequencer chip, and the data of all registers will be shifted one position to the right. For instance, the $Q_{0,23}$ in module $i$ will be shifted to the register in module $(i-1)$. Fig. 8 shows the connection of signals between two cascaded Sequencer chips. Note that the $P_{0,23}$ of the left Sequencer chip is connected to all ones; $X_{0,23}$ and $b_{in}$ of the right Sequencer chip are all connected to zeros. The superscripts of $l$ and $r$ indicate, respectively, the module at the most left and the most right of the Sequencer chip. At the initialization, all the registers inside the chip are loaded with the largest DT/DS values, i.e., all ones, so that new arrival cells with DT/DS values between 0 and $2^{14} - 1$ can be inserted into the Sequencer.

The initialization is done by asserting the init and sre signals and setting $Z_{0,23}$ to 11 $\cdots$ 11, which is the initialization mode in Fig. 9.

Fig. 9 also shows that two values, $A_{0,23}$ and $B_{0,23}$, are
TABLE I

<table>
<thead>
<tr>
<th>Cases</th>
<th>$b_{out}$</th>
<th>$b_{in}$</th>
<th>Action performed by the controller</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a) $X_{0,13} \leq Z_{0,13} &lt; Q_{0,13}$</td>
<td>1</td>
<td>0</td>
<td>Module $i$ shifts its content to the left, and $Q_{0,23} = Z_{0,23}$</td>
</tr>
<tr>
<td>(b) $Z_{0,13} &lt; X_{0,13} \leq Q_{0,13}$</td>
<td>1</td>
<td>1</td>
<td>Both modules $i$ and $(i - 1)$ shift their contents left, and $Q_{0,23} = X_{0,23}$</td>
</tr>
<tr>
<td>(c) $X_{0,13} \leq Q_{0,13} \leq Z_{0,13}$</td>
<td>0</td>
<td>0</td>
<td>Retain the $Q_{0,23}$</td>
</tr>
</tbody>
</table>

Fig. 8. Interconnection signals of two cascaded Sequencer chips.

![Circuit Diagram](image)

Fig. 9. Timing of the initialization, write-in, and readout operations.

written into the chip by asserting the slck signal (write-in mode). They appear at the $Q_{0,23}$ output of the right chip. If we assume that $A_{0,13} > B_{0,13}$, the readout signal will be read out before $A$. The readout mode is operated by asserting the srck and sr signals simultaneously. The shifting clock’s period is equal to one cell time (2.83 µs) divided by the number of total inputs and outputs of the traffic shaper or the queue manager. For example, if the number of inputs plus outputs is 13 (for the queue manager application), the clock period will be about 200 ns (2.83 µs/13). So, the chip’s required operation frequency is less than 5 MHz. It is important to notice that the operation speed for the traffic enforcer/shaper and queue manager is not affected by the number of cells in the pool, nor by the possible values of DT/DS (or priority levels) due to the broadcast mechanism and the distributed processing.

The logic equations of the controller are listed below,

and its detailed circuit is shown in Fig. 10.

$$cp = sr \quad cx = \overline{sr} \cdot b_{in} \cdot init \quad cz = \overline{sr} \cdot (b_{in} + init)$$

$$clk = (b_{out} \cdot slck) \cdot \overline{srck}$$

Several buffers are required to provide enough driving capability for the large fan-out of the control signals. The $nx (n = 2, 3, 6, \text{or } 16)$ in Fig. 10 represents the associated gate widths as $n$ times that of the smallest gate. A pulse is generated on the clk signal when the Sequencer is in the initialization mode (both srck and init are asserted), in the write-in mode (slck is asserted and either case (a) or (b) in Table I occurs), or in the readout mode (both srck and sr are asserted). The register is clocked at the falling edge of the slck or srck signals. If it is clocked at the rising edge (i.e., $clk = b_{out} \cdot slck + srck$), the $b_{out}$ may change again after the register has been loaded with a new DT/DS value. This may cause undesired transitions on the clk signal during the write-in mode, and a wrong value may be latched into the register.
B. Subtractor

The 14-b subtractor, consisting of 14 full subtractors cascaded in series, has been designed to minimize the number of transistors. Fig. 11 shows the detailed circuit and lists the logic equations for the full subtractor. Since only the borrow-out is needed in our application, the circuit for generating the difference is omitted and is not shown in the circuit diagram. Notice that the EXCLUSIVE-OR function is implemented with six transistors. To avoid the long chain of pass transistors, the borrow-out at every stage is inverted. If we name the full subtractor of the least-significant bit as an even one, and the next one as an odd one, the 14-b subtractor then consists of even and odd subtractors alternately and repeatedly [19]. Through the SPICE simulation, we found that the maximum delay for the 14-b subtractor is 12 ns, which occurs when the least-significant bit changes from high to low or low to high.

C. Register

The 24-b register in each module is implemented with static, rather than dynamic, D-type flip-flops (DFF's) because the chip is operated in an asynchronous manner where, during idle cell slots, there are no write operations for the chip. When one designs a DFF, several considerations need to be taken into account, such as speed, area, power consumption, race problem, and noise margin. In our applications, speed is not a concern because the operating speed is relatively low (about 5 MHz). Instead, area and power consumption are important because there are more than 6000 DFF's (24 x 256) in the chip. We have also adopted a single-phase clocking scheme for the DFF to eliminate the race problem [20], [21].

Here, we have laid out three different static DFF's and compared their area, power consumption, noise margin, and speed. The comparison of dynamic DFF's on their speed and robustness against the race problem can be found in [22]. DFF1, shown in Fig. 12(a), was proposed in [23]. DFF2, shown in Fig. 12(b), has four inverters and four transmission gates, and is commonly used. DFF2 uses a two-phase clock scheme (ck and ćk signals) and may cause the race problem if the overlap between ck and ćk is large. This normally occurs when clock signals are distributed in the entire chip and drive large capacitance loads. DFF3, shown in Fig. 12(c), is adopted in the chip. It is similar to DFF2 except that the transmission gates are replaced with pass transistors, and a single-phase clock is used. Table II summarizes the properties of each DFF, obtained from SPICE simulations with 1.2-μm CMOS typical parameters. Although DFF2 has the largest noise margin and the highest speed, DFF3 has the smallest area and power consumption, making it the most appropriate choice for the Sequencer chip. Note that the area of DFF2 is almost twice that of DFF3. Also notice that the power consumption of DFF1 is almost twice that of DFF3. This
Table II
Comparison of Three Different DFF's

<table>
<thead>
<tr>
<th>Design</th>
<th>Area (μm²)</th>
<th>Power (mW) @ 100 MHz</th>
<th>Noise Margin</th>
<th>Speed Figure (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DFF1</td>
<td>1848</td>
<td>1.48</td>
<td>0.8</td>
<td>2.1</td>
</tr>
<tr>
<td>DFF2</td>
<td>3465</td>
<td>0.88</td>
<td>2.5</td>
<td>2.0</td>
</tr>
<tr>
<td>DFF3</td>
<td>1800</td>
<td>0.75</td>
<td>0.9</td>
<td>2.1</td>
</tr>
</tbody>
</table>

is because DFF1's clock signal (ck) is connected to two more transistor's drains, which causes some current flowing to ss during the transient periods and thus wastes the power.

Noise margins are measured in the worst case at DFF’s internal points, by the absolute difference between the node voltage and an inverter’s threshold voltage (2.5 V). For instance, the highest voltage for logic one in DFF3 is at node w (see Fig. 12(c)) and the value is 1.6 V, while the lowest voltage for logic zero occurs at node x and the value is 3.5 V. Thus, the noise margin is equal to minimum {[(1.6 - 2.5), [3.5 - 2.5]], or 0.9 V. Referring to Fig. 7 in combination with Fig. 12(c), the multiplexer (MUX) in front of the 24-b register is a collection of simple n-transistors controlled by cp, cx, and cs signals. Cascading the n-transistor with the p-transistor in the DFF does not affect the logic zero’s noise margin at node w, but it does cause the voltage for logic one at node w to drop below 5 V. However, this voltage level is still greater than that at node x, so the logic one’s noise margin is not affected by cascading the n- and p-transistors. In each module, a 3.7-pF bypass capacitor between the vdd and vss was implemented to reduce the ripple voltage on the power rails (caused by the inductance between vdd and vss), compensating for the smaller noise margin in DFF3.

The speed figure of a DFF is determined by the sum of a DFF’s delay time τd and set-up time τs. A way of determining a DFF's speed figure is shown in Fig. 13. A divide-by-2 counter is formed by feeding a DFF output q to an odd-number inverter chain (nine in our example) and back to the DFF input. The frequency of the ck signal is increased to where the divide-by-2 function fails, which corresponds to the testing circuit’s maximum operating frequency. To have a correct operation in a synchronous system, the following condition must be satisfied:

\[ T = t_d + \tau + t_s \]

where T is the clock signal’s period, and \( \tau \) is the propagation time of a circuit between a DFF’s. Since the divide-by-2 circuit is simulated up to the clock signal’s maximum frequency, the equality in the above equation holds. A ring oscillator consisting of a nine-inverter chain was simulated and \( \tau \) was found to be 1.5 ns, half of the ring oscillator’s period. By subtracting the nine inverters’ delay (\( \tau \)) from the T in the above equation, the speed figure of each DFF can be obtained.

D. Chip Summary and Testing

The Sequencer chip was laid out with a custom symbolic layout tool (called DASL [24]). It has 182 pads and is packaged in a 223-pin ceramic pin-grid-array (PGA) carrier. It has low bonding-wire inductance due to a ground plane on the chip carrier and thus reduces the ripple voltage on the power rails inside the chip. The chip’s die size is about 7.5 mm \( \times \) 8.3 mm, and its photograph is shown in Fig. 14. The chip was simulated at circuit level to operate at a clock rate of 50 MHz. The circuit-level simulation tool adopts a table look-up approach to calculate the voltages and currents of internal nodes inside the chip, rather than solving nonlinear equations such as in the SPICE simulation program. Therefore, the tool can simulate very complex chips, such as the Sequencer that contains about 150K CMOS transistors, in a reasonable amount of time. For instance, it took about three days to run 20 test evens (1600 ns) for the chip simulation on a 27-MIPS workstation.

The chip was tested in three different ways: P-test, X-test, and Z-test, depending on where test vectors were loaded into the chip. The P-test (shifting data to the right) and the X-test (shifting data to the left) allow us to find out if there is any bit stuck at logic zero or one. Because of the modular structure of the chip, we can monitor the outputs shifted out of the chip and determine which module inside the chip causes errors. These preliminary tests help us identify bad chips quickly. For those chips passing the P- and X-tests, we further tested them by entering test vectors through the Z bus and operating the chips at three different modes: initialization, write-in, and read-out. Let us denote the time period needed for initialization, readout, or write-in as a clock cycle.
For the P-test, we first loaded the chip with 24 words from \(P_{23}^{0,23}\) input by asserting the sck and sr signals (read-out mode). The value of the Z bus can be arbitrary. These words all have 23 b of logic one and 1 b of zero; the 0 bit appears diagonally in the 24 words. After 232, or 256 – 24, clock cycles, we should see the first-loaded word appear at the \(Q_{64}^{24}\) output. If a certain bit in a module, say module \(k\) (module 1 is on the most right), is stuck at zero, we will see the output of that bit stay zero after \(k\) clock cycles. To detect stuck-at-zero errors, all modules have to be initialized to all zeros instead of to all ones before the same tests are applied.

For the X-test, the same 24 words are loaded to the chip from \(X_{63}^{23}\) input by asserting the sck signal (write-in mode), setting the \(b_{64}^{0}\) to logic one, and applying all zeros to the Z bus. As these 24 words are shifted to the left, their vacant positions are filled with all zero values. Any stuck-at-zero error in module \(k\) causes that bit position to stay zero at the output \(Q_{64}^{23}\) after \((256 - k)\) clock cycles. However, since the chip must not be initialized to all zeros in this test (otherwise, no write-in operations can be performed), the stuck-at-one error can be detected, but cannot be located.

For the Z-test, mixed-order numbers are written into the chip through the Z bus after the initialization. These numbers are then read out serially and examined to see if they are in an ascending order. For those test vectors that have the same DS/DI value, the ones written earlier will be read out earlier.

Chips were tested and operating correctly up to 33 MHz, limited by the pattern generator’s speed. This speed provides a sufficient operation margin for normal applications (about 5 MHz). The delay of the output is 10 ns from the falling edge of the sck or sckl signal. The setup time for the inputs is 15 ns before the rising edge of the sck or sckl signal. The holding time is 1 ns after the falling edge. The worst case of the chip power dissipation is about 2.3 W, when the clock is coached at 33 MHz in the initialization mode, and the values of all zeros and all ones are alternated on the Z bus at this rate. This causes all 6144 (24 × 256) DFF’s to toggle simultaneously at this rate.

V. CONCLUSIONS

We proposed to control the users’ traffic at two places in the network: at the user-network interface (UNI) by a traffic enforcer, and at the network-node interface (NNI) by a queue manager. The users’ traffic must first be enforced to conform to a traffic descriptor, which may include the parameters such as average and peak bit rates, and maximum burst lengths. If the users’ traffic does not comply with the traffic descriptor, some action has to be taken against the violating traffic. We have adopted an enforcement strategy for the traffic enforcer (also called a traffic shaper) by providing a buffer to delay and reshape the violating cells. If the buffer is full, cells are transmitted to the network but are tagged with a lower priority. Once users’ cells are allowed to enter the network either with or without reshaping, they may be queued up in the buffers between network nodes. A queue manager is required to manage the queued cells in such a way that higher priority cells are always served first, low-priority cells are discarded when the queue is full, and any interference between same-priority cells is prevented. By assigning a departure sequence number of every arrived cell based on its average arrival rate agreed upon at the call setup, the queue manager can easily and effectively perform the queue management function.

Both implementation architectures for the traffic shaper and the queue manager apply the concepts of fully distributed and highly parallel processing to schedule the sending or discarding sequence of ATM cells. To support these two architectures, a VLSI chip (the Sequencer), which contains about 150K CMOS transistors, has been implemented with 1.2-μm CMOS technology in a regular structure so that the number of virtual channels or priority levels can grow flexibly. The Sequencer chip has been tested and operated correctly up to 33 MHz, providing a sufficient operation margin for normal applications (about 5 MHz).

REFERENCES


H. Jonathan Chao (S'83-M'84) received the B.S.E.E. and M.S.E.E. degrees from National Chiao Tung University, Hsinchu, Taiwan, in 1977 and 1980, respectively, and the Ph.D. degree in electrical engineering from Ohio State University, Columbus, in 1985.

From 1977 to 1981 he worked at Taiwan Telecommunication Laboratories, where he was engaged in the development of a digital switching system. Since 1985 he has been a Member of Technical Staff at Bellcore, Red Bank, NJ, where he was involved in the research of architecture designs and VLSI implementations for SONET/ATM-based broadband networks. He was a key architect to design a high-speed computer/network interface for the Nectar Gigabit Testbed. From 1985 to 1991, he was an Adjunct Associate Professor in the Electrical Engineering Department of Polytechnic University, Brooklyn, NY, where he became a full-time Associate Professor in 1992. His research interests include scalable ATM switches, high-speed computer networks, and congestion/flow controls in ATM networks. He holds nine patents with five more pending, and has published over 30 journal and conference papers in the above areas.

Nedel Uzun received the B.S.E.E. and M.S.E.E. degrees from Istanbul Technical University, Turkey, in 1983 and 1986, respectively. He is currently a Ph.D. student in the Electrical Engineering Department of Polytechnic University, Brooklyn, NY.

Since 1990 he has been a part-time contractor in the Transwicching Technology Research Department at Bellcore, Red Bank, NJ. His current research interests include VLSI designs and digital subband codings.