FAST: An FPGA-Based Simulation Testbed for ATM Networks

Dimitrios Stiliadis
Anujan Varma

UCSC-CRL-95-47
September 8, 1995

Baskin Center for
Computer Engineering & Information Sciences
University of California, Santa Cruz
Santa Cruz, CA 95064 USA

ABSTRACT

Simulation of ATM switches and networks is a computationally demanding problem as compared to simulation of conventional packet-based networks, owing to the large number of cell events that need to be simulated in the former. To address this problem, we are developing a flexible hardware testbed for simulation of ATM-based networks. The testbed, called FAST (FPGA-based ATM simulation testbed), uses high-density field-programmable gate arrays (FPGAs) to allow implementation of the key simulation components such as traffic generators, switching fabric, buffer management, traffic scheduling, congestion control mechanisms, and statistics recording in hardware. In the first version of the testbed (FAST-1), each board consists of 13 Altera FLEX devices (including 4 multichip modules), providing a total of 336,000 usable gates. Each board can be used to simulate an ATM switch. Multiple boards may be interconnected to simulate large ATM networks. Software tools haven been developed for specifying the components of the underlying simulation model, such as the switch structure, traffic model, traffic scheduling algorithm, and congestion control mechanisms; synthesizing the specifications into the individual FPGAs; controlling and monitoring the simulation; and collecting and reporting statistics.

Keywords: ATM switch scheduling, field-programmable gate array (FPGA), simulation testbed

---

This research is supported by NSF Young Investigator Award MIP-9257103, faculty research funds granted by the University of California, Santa Cruz, Altera Corporation and a software donation by Mentor Graphics.

1 Introduction

Broadband networks based on Asynchronous Transfer Mode (ATM) are enabling the integration of traffic with a wide range of characteristics within a single communication network. In these networks, all communication at the ATM layer is in terms of fixed-size packets, called "cells" in ATM terminology. Routing of cells is accomplished through packet switches over virtual circuits set up between endpoints.

ATM technology places stringent demands on the underlying switching system. In order to support a wide variety of applications, ATM networks will need to provide guarantees on bandwidth, delay, jitter, and cell loss rate. Implementation of these quality-of-service (QoS) guarantees requires the use of appropriate traffic scheduling algorithms in the switches so that the available resources are properly allocated to the individual traffic streams. A large number of such traffic scheduling algorithms have been proposed in recent literature [1, 2, 3]. In addition, congestion control mechanisms may need to be incorporated in the individual switches for the transport of best-effort traffic. A number of disparate approaches to congestion control in ATM networks have been proposed and are currently being discussed for standardization [4].

Evaluating the performance of traffic-scheduling algorithms and congestion control mechanisms in the ATM switches is a challenging problem. Although analytical techniques can provide valuable insight into the operation of the system, these are often inadequate for modeling the switch and the algorithms at the needed level of detail. In addition, the characteristics of the traffic streams carried by the switch can vary over a wide range, making analysis difficult. Simulation is often the only alternative available to evaluate the performance. Even when analytical modeling is feasible, simulation is needed to verify the validity of the simplifying assumptions used in modeling.

Simulation of ATM switches and networks using the conventional software approach is time-consuming. Because of the small size of the ATM cell, a large number of cell events may need to be simulated to reach satisfactory confidence levels. One approach to improve the speed of the simulation is to resort to parallel or distributed simulation, but this is a relatively expensive option. In addition, the speedup obtained by parallelizing event-driven simulators may be small because of their inherently serial nature [5]. Communication and synchronization bottlenecks also limit the achievable speedup.

To address this problem, we are developing a flexible hardware testbed for the simulation of

ATM switches and networks at the University of California, Santa Cruz. The testbed, called FAST (FPGA-based ATM simulation testbed), uses high-density field-programmable gate arrays (FPGAs) to allow implementation of the key simulation components in an ATM network such as traffic generators, switching fabric, buffer management, traffic scheduling, congestion control mechanisms, and statistics recording in hardware. Our approach consists in implementing a functional model of the switching system in hardware, using field-programmable gate arrays (FPGAs). FPGAs offer relatively fast and inexpensive means for prototyping hardware systems. Although FPGAs typically provide much lower densities (gates/chip) as compared to gate arrays, they are currently reaching the density levels needed to model complex subsystems. For example, the Altera FLEX family currently offers devices with densities as high as 16,000 usable gates in a single chip. In addition, developments in the multichip-module and interconnection technologies have made available devices with as many as 48,000 usable gates, such as the Altera FLEX 8050 multichip module [6].

In the first version of the testbed (FAST-1), each board consists of 13 Altera FLEX devices (including 4 multichip modules), providing a total of 336,000 usable gates on each board. Each board can be used to simulate a single ATM switch with its associated algorithms. Multiple boards may be interconnected to simulate large ATM networks. Software tools are being developed for specifying the components of the simulation model, such as the switch structure, packet arrival process, traffic scheduling algorithm, and congestion control mechanisms; synthesizing the specifications into the individual FPGAs; controlling and monitoring the simulation; and collecting and reporting statistics.

FPGAs are ideally suited to building reconfigurable hardware systems. Devices such as the Altera FLEX family and the Xilinx FPGAs use RAM-based lookup tables as their basic logic element, thus allowing in-system configurability [6, 7]. FPGA-based prototyping aids are a valuable tool in hardware development. For example, the QuickTurn system, based on Xilinx FPGA devices, is widely used in the industry for hardware prototyping [8]. Several other efforts have been reported in the literature for building reconfigurable hardware systems for prototyping or emulation of complex systems such as SIMD architectures [9, 10], MIMD parallel processors [11], neural networks [12], accelerators for scientific computation [13], and general-purpose coprocessors [14, 15].

In addition, advances in high-level hardware description languages and synthesis tools have significantly reduced the time for hardware system prototyping [16]. The FAST-1 testbed plans to use VHDL as the high-level modeling language. Commercial hardware synthesis tools will then be used for producing the final logic that is automatically mapped to the FPGAs. The logic produced by such tools is of course not optimal, but allows the completion of complex designs in a very short time. In the case of our testbed, hardware performance, although important, is not critical since the model does not need to run in real time.

FAST is a functional emulation system, not a prototyping system. Most of the reconfigurable systems built so far using FPGAs are designed to serve as either hardware prototyping platforms or general purpose coprocessors. A general-purpose FPGA-based prototyping system would not be efficient for our application since their internal organization is not usually optimized for simulating an ATM switch architecture. In addition, a prototyping system is usually used to implement the complete hardware of the target system. The same hardware design could then be used to manufacture the system, often using ASIC technology. In our simulation testbed, however, the FPGAs are used as the building blocks for the functional simulation of the system. This allows different parts of the system to be modeled at different levels of detail. For example, the size and format of ATM cells used in the simulation may be different from that specified by ATM standards. The cells may include only fields that are relevant to the simulation model; cells may carry no data, or may contain only part of the data field essential to the protocols implemented in the simulation. The testbed is intended for modeling at the ATM layer and above; details of the physical layer are not modeled. However, delays at the physical layer can be incorporated in the simulation model. A host processor is used to guide the simulation and run parts of it that are not critical to performance.

The remainder of this paper is organized as follows: Section II describes the architecture of the FAST system and its key components. Section III discusses an example implementation of a traffic scheduling algorithm based on weighted round-robin scheduling on the FAST-1 board and compares its performance with conventional software simulation on a workstation. Section IV concludes the paper with a summary of the current status and future directions for this research.

2 FAST-1 Architecture

The first version of the testbed, called FAST-1, uses a printed-circuit board consisting of thirteen Altera FPGA devices as its building block. The board provides a total of 336,000 usable gates for implementing the simulation model of the target system, and up to 17 Mbytes of static RAM. The FAST-1 board is designed such that a single board can be used to simulate an

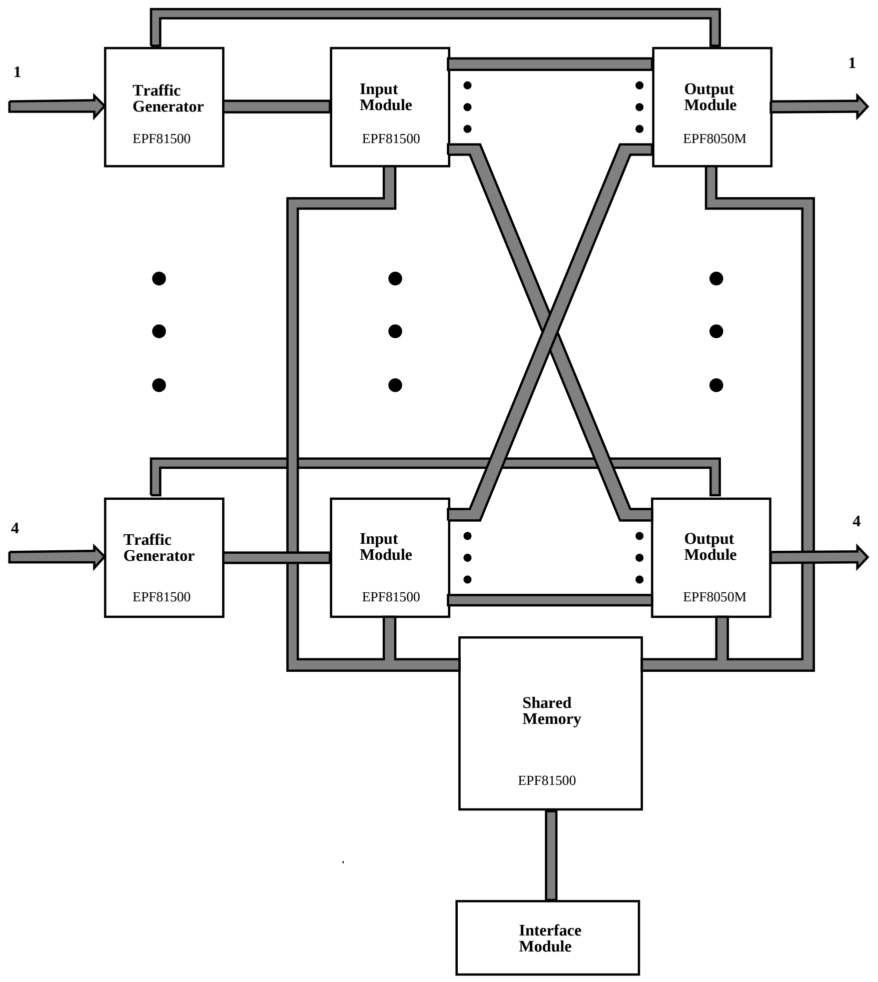

Figure 1.1: Architecture of the FAST-1 board.

ATM switch, and multiple boards can be interconnected via available connectors to simulate more complex switch fabrics or an entire ATM network consisting of multiple switches. The FPGAs allow implementation of the key simulations components in hardware. The testbed is currently interfaced through the ISA bus to a PC serving as the host system; however, since the interface logic of the testbed is implemented using programmable hardware, it can be interfaced to other busses by reprogramming the interface logic. Software tools control the programming of the FPGAs and the running of the simulations.

Many different architectures for designing ATM switches have been proposed in the literature. These include shared memory, bus, crossbar, and multistage networks (for a survey, see [17]). The architecture and interconnection structure of the FAST-1 board have been chosen so that any of these architectures can be mapped efficiently on the board. Different buffering approaches such as input and output buffering can be accommodated. In addition, the board allows implementation of traffic-scheduling algorithms and congestion control mechanisms associated with the switch to study their behavior. Finally, traffic sources for simulations can be implemented on the board in hardware.

A block diagram of the testbed is shown in Figure 1.1. It consists of four *traffic generator modules*, four *input modules* and four *output modules.* Each of the modules consists of an FPGA device and local memory in the form of static RAM. The input and output modules are interconnected with each other via 18-bit wide paths forming a full bipartite graph. Each traffic generator module is connected to the corresponding input and output modules through 16-bit wide busses. Thus, the input and output modules together can efficiently simulate the function of a 4×4 crossbar switch with buffering at the input or output, and the traffic generator modules can be used to drive the switch model with the desired traffic distribution during the simulation. An additional module, called the *shared-memory module* is connected through a shared bus to all the input and output modules. The shared-memory module is also connected to all the traffic generator modules. This module can be used to implement a globally-accessible memory for simulations of switch architectures employing shared memories. In addition, this module is used for coordinating the loading of the local memories from the host system for setting up a simulation run.

Each of the input modules, traffic generator modules, and the shared-memory module uses an Altera FLEX 81500 as the programmable device. The 81500 device provides and equivalent of 16,000 usable gates and up to 200 I/O pins. Each of the four output modules employs a denser FPGA device, the FLEX 8050.* The 8050 is a multichip module (MCM) providing a total of 48,000 usable gates and 360 I/O pins. Each 8050 combines four FLEX 81188 FPGAs and an Aptix Field-Programmable Interconnect Chip (FPIC) in one package [18]. The FPIC is a passive device that can be programmed to interconnect the pins of the 81188 devices in a flexible manner, with a maximum of five pins being connected together [19]. All devices are SRAM-based allowing easy reconfiguration.

The use of FLEX 8050 devices allows more functions to be implemented in the output modules. This configuration is ideally suited to the mapping of simulation models where the output modules need to implement substantially more logic as compared to the input modules. For

*The FLEX 8050 is currently the densest FPGA device available commercially.

example, traffic-scheduling algorithms are often implemented within the output module; the use of the MCM device in the output modules allows us to simulate a much wider range of scheduling algorithms than that would be feasible with a single-chip FPGA device. The configuration, however, does not restrict us to simulation models where the majority of functions are implemented in the output modules. Since the architecture of the board is symmetric, the functions of the input and output modules can easily be reversed, thus allowing the MCM devices to function as the input modules.

Each of the traffic generator modules on the board consists of one FLEX 81500 FPGA and up to 1 Mbyte of static RAM organized in 16-bit words. Its function is to generate the input traffic for the simulations. The traffic generator uses a hardware algorithm to first generate a uniform distribution of random numbers. Such a sequence can be converted to any other distribution by a table lookup and interpolation. The local memory of each traffic generator module can be used for storing the lookup tables. More complex traffic models, for example a video stream, that can be modeled using Markov chains can be synthesized by implementing the Markov chain in hardware.

In addition, if a more realistic traffic model is desired, traffic can be injected into the testbed from an external source. Each traffic generator module can be connected to an external source via a 20-bit connector, and can be programmed to implement the handshaking protocol for interfacing to the traffic source. The local memory can now be used for temporary storage of incoming ATM cells before they are forwarded to the input modules. For this purpose we plan to utilize the "CPU Design Kit", that can be used as a general purpose interface to an ISA-based computer [33]. This board consists of six FLEX 81500 devices and can be programmed to provide a 20-bit interface to the FAST-1 board.

For example, in order to evaluate the performance of the switch under MPEG video traffic, the ATM cell sequence that corresponds to an MPEG video source can be produced in a PC and forwarded to the testbed through the interface boards. Notice, that it is not necessary to transmit the whole data part of the cells. The video sequence can be stored in both the source and destination PC. A sequence number is sufficient to reconstruct the data on the destination PC and determine the effect of cell-losses on the quality of the image.

The traffic-generator modules can also be used to simulate delays in the physical link and interface. This is achieved by implementing a pipeline of ATM cells in the traffic-generator module, causing cells to be delayed for a constant amount of time before they are forwarded to the input module. The pipeline can be implemented by two sliding pointers a constant distance apart. The first pointer shows the cell currently being serviced and the second the memory location where a new cell can be stored. At each cell-time, both pointers are advanced by one cell in memory (with cyclic wrap-around), to simulate the passage of time.

The input modules are typically used to emulate the functions at the input ports of the switch. Each input module consists of a FLEX 81500 and local memory of up to 1 Mbyte of static RAM organized in 16-bit words. There is a dedicated set of wires connecting each input module to each of the output modules. Each of these paths is 18 bits wide with two additional lines for signaling. The input module receives ATM cells from the traffic generator modules, determines the destination port of the cell (performing a translation from the virtual channel number to the outgoing port address, if necessary), and forwards the cell to the corresponding output module. If input buffering is used, the cell may be also buffered in the local memory associated with the input module. In a switch architecture based on shared memory, the cell can be forwarded to the shared-memory module through the shared bus. Some dedicated logic is also required for keeping statistics of various events at the input module.

The output modules are typically used for multiplexing cells arriving from the different inputs as well as scheduling their transmissions to the output port. Assuming that the board is used to simulate a 4×4 output buffered switch, the output module must be able to receive up to four cells per cell-time, buffer them, and schedule the next cell for transmission. Each output module consists of a FLEX 8050 and local memory of up to 2 Mbytes organized in 32-bit words. The local memory is also accessible as 16-bit words. The output modules often perform the most complex functions in the switch, and therefore the use of high-density MCM devices to implement them is justified. However, as pointed out earlier, the functions of the input and output modules can be swapped if necessary.

The shared-memory module consists of a FLEX 81500 device and up to 1 Mbyte of static RAM organized in 16-bit words. The function of the shared-memory module is to provide an efficient means for emulating shared memory in switch architectures. The shared-memory module is connected to the input and output modules through a 34-bit wide common bus; in addition, there are 6 dedicated lines from each input module and each output module to the shared-memory module that can be used for arbitrating accesses to the bus.

In addition to providing the shared-memory function, the shared-memory module is designed to serve also as a controller. This module is used to connect the testbed to the host processor; hence, all data from and to the host pass through it. Since it is connected to all the input and output modules, the shared-memory module can be used to coordinate their actions during the simulation; if, for example, a shared-memory switch architecture is being simulated, the shared-memory module can provide the arbitration function for accesses to the shared buffers. In addition, the shared bus can be used to load the local memories during the programming phase. Finally, some of the logic in the shared-memory module can be used to augment the logic in the interface module, if required, to implement the bus interface to the host system.

The final part of the FAST-1 board is the interface module. This module is responsible for providing the interface function to the host-bus and for controlling the programming of the other FPGAs on the board at startup time. The interface module consists of a FLEX 8820 device, a clock generator, and a small programmable logic device (PLD). The PLD enables the board to be accessed by the host at startup by decoding its base address in hardware. Using this basic addressing capability, the host processor first programs the interface FPGA; once the interface is configured, the programming of the rest of the FPGAs, as well as the loading of the memories, is done under control of the interface chip. This method was used for portability of the interface of the testbed. Use of an FPGA device to implement the interface function allows the board to be interfaced to any workstation- or PC-bus by reprogramming the FPGA. Although our testbed is currently connected to the ISA bus of a PC, any other bus could be accommodated by simply reprogramming the interface chip. Note that the ISA interface currently utilizes only 10% of the available logic in the FLEX 8820; thus, a more complex interface can easily be accommodated in the device. The clock generator provides a basic 40 MHz clock that can be divided within the interface FPGA to provide the clock(s) for the testbed. A special bus is used to distribute up to four different clocks from the interface FPGA to all the other modules. These lines are connected to the four pins designated to serve as clock inputs on each of the 81500 devices.

Programming of the FPGA devices on the board is accomplished through a programming bus. All the traffic generator modules can be programmed either independently or in parallel. The latter capability is useful when the traffic generator modules need to be programmed with identical designs. Similarly, all the input modules and output modules can be programmed either in parallel or individually. During normal operation the programming lines may be used as global control lines between the host processor and the testbed.

FAST-1 provides the ability to cascade multiple boards. Each output module is connected to an interface connector on the board, providing 20 signal lines to connect externally. The output modules can send out cells to another board or to an external system through this interface. The 20 lines are intended to be configured as a 16-bit datapath plus 4 control signals. A simple handshaking protocol can be then used to send data from an output port of one testbed to an input port of another testbed, thus enabling the emulation of a network of ATM switches or multistage switch fabrics. A global simulation clock is required to synchronize the multiple boards. One of the boards is determined as the master and the rest are the slaves. The master board will determine the starting time for the processing of a new cell. The processing will not finish until all boards have completed their operation.

A simulation model on the FAST-1 board may use either centralized or distributed control. With centralized control, the control function is performed by the interface module and/or the shared-memory module. After the programming of the board is complete, the interface module can be reprogrammed to implement the global controller function. The programming bus from the interface module to other modules can be used to provide the necessary connectivity between them. With distributed control, the programming bus can again be used for exchanging information among the different modules.

A photograph of the FAST-1 board is shown in Figure 2.1. The chips are laid out on a 10-layer printed-circuit board of size $16'' \times 17''$. The board is currently connected to the ISA bus of a PC via a ribbon cable. A separate interface board is used to connect the ribbon cable to the ISA bus.

2.1 Mapping Alternatives

The main goal of the design of the testbed architecture was to provide the means to emulate a broad range of ATM switching fabrics and provide the insights of how the hardware implementation can affect the performance of higher level protocols. In the previous section we provided a description of the testbed mainly in the context of output buffering switches.

However it is easy to see that the same architecture could be used for mapping other approaches. For example in an input buffering switch the input modules can also do the buffering of packets. The shared module can be then used as a controller driving the crossbar switch that

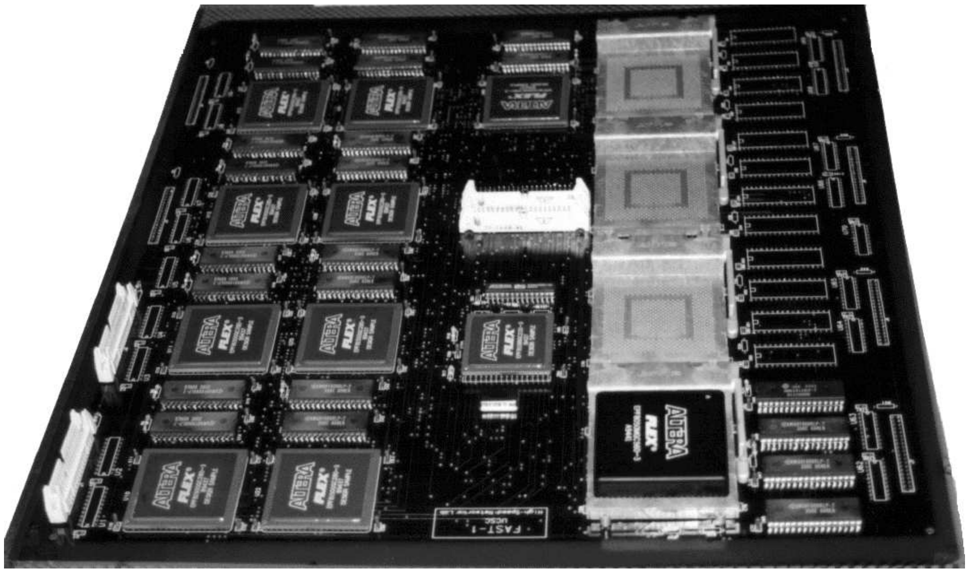

Figure 2.1: The FAST-1 board.

will send the packets from the inputs to the outputs. The dedicated lines between the input and output modules can represent the non-blocking crossbar switch.

In this case the main bottleneck of the logic become the input modules and it may be argued that there is not enough logic for these functions. A mapping we propose is to invert the functionality of the modules. Note that the traffic modules are connected directly to the output modules through a 40-bit wide bus. We can therefore interchange the functionality of the input and output modules. The FLEX 8050s will implement the input port functions and the FLEX 81500s will implement the output port functions.

In the case of a shared-bus or shared memory architecture the shared bus between all the input and output modules provide us with the necessary paths. If buffering is required in both the inputs and the outputs of the switch, the local SRAMs available to each module can be used for this purpose.

2.2 Design Process

Designs of different architectures are entered using either VHDL or Verilog. A library of commonly used modules has already been developed. Among others we have developed modules for a FIFO controller, random number generators, memory interface unit, as well as multiplexing

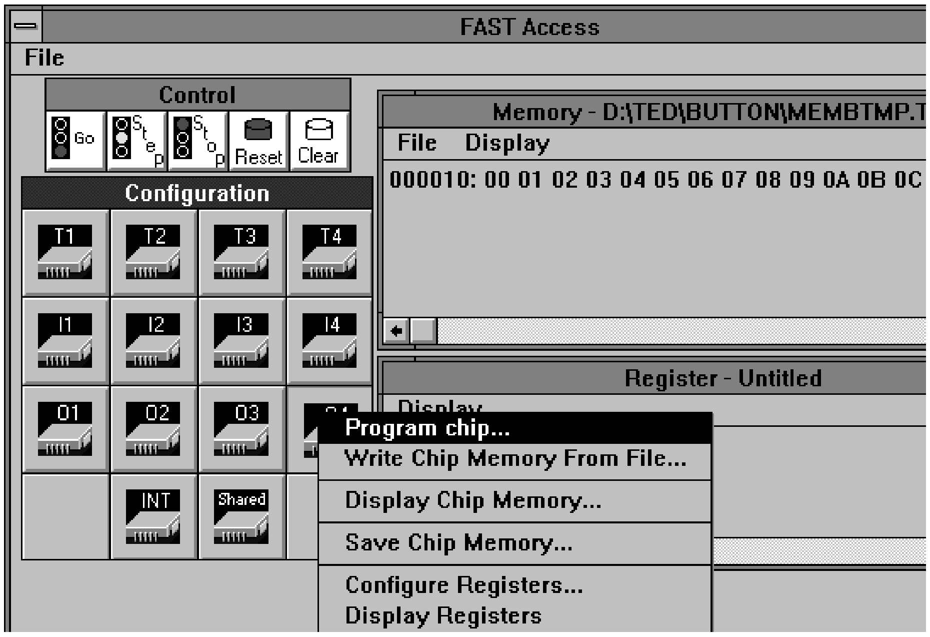

Figure 2.2: Graphical user interface of the FAST-1 board.

and distribution modules. These designs are optimized in order to better utilize the resources of the target FPGA architecture. Higher level designs will use this library as a basic tool to accelerate the prototyping process. We are currently in the process of designing different schedulers and buffer management modules.

A functional simulation of the design is performed using commercial tools [20]. High-level tools are also used for synthesizing the behavioral design into the target FPGA technology. Software interfaces between the tools are available from Altera and Mentor Graphics, and the whole design process is completed in a simple and efficient way [21, 22].

Furthermore, we have developed a graphical user interface that facilitates the programming of the FPGAs and provides the necessary mechanisms for debugging the designs. The software tools allow access to the memories and the internal registers of the designs (see Figure 2.2) as well as clock and control signals. The software can be customized to debug different architectures.

3 Design Example

In this section we present an example simulation model we have developed to demonstrate the use of the FAST-1 testbed and describe its implementation on the FAST-1 board. The model we

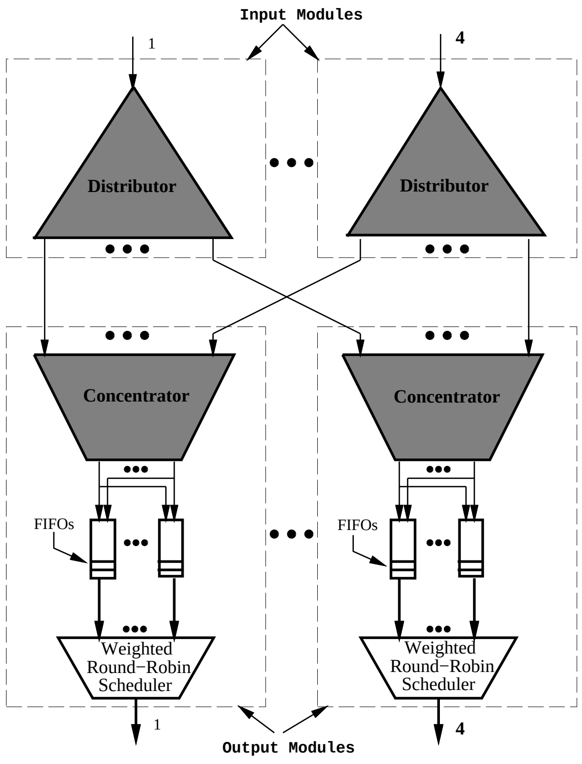

Figure 3.1: General model of an output-buffered ATM switch.

consider is that of a 4×4 output-buffered ATM switch with weighted round robin scheduling [23]. Each output port in the model can support up to 32 virtual channels and has sufficient amount of memory for buffering up to 32K cells. Since we are performing a functional simulation of the target system, the data fields of the ATM cells are not represented in the model and only parts of the header that are essential to the simulation are used.

The general architecture of the system is shown in Figure 3.1. The switch consists of two main stages, a distribution stage and a concentration stage. The distribution stage routes incoming cells to the output ports based on their virtual channel identifiers. Since multiple cells can be destined to the same output port at the same time, a multiplexing function is required. The concentration stage performs this function. The scheduling function follows the concentration stage. The scheduler buffers the incoming cells and schedules them for transmission based on their relative priorities.

A simple and efficient approach to scheduling is the *weighted round robin* algorithm (Figure 3.2). This algorithm can be used to provide bandwidth guarantees for individual flows passing through the switch. In our model of the algorithm, incoming cells at an output port are stored

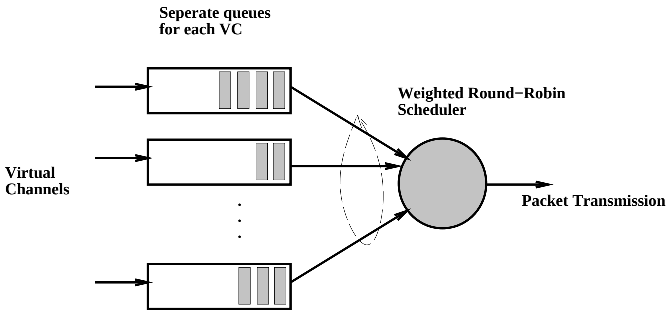

Figure 3.2: Weighted round-robin scheduling.

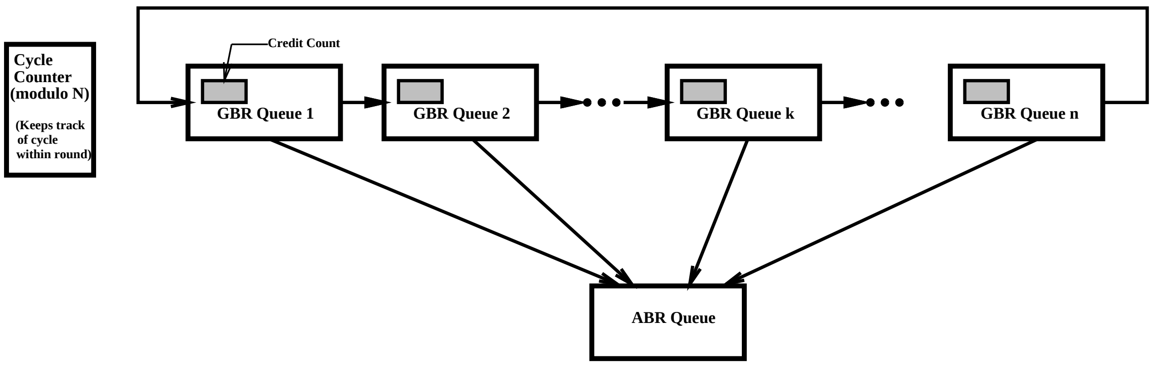

Figure 3.3: Implementation of weighted round robin scheduling.

in different FIFO queues depending on their Virtual Channel Identifier (VCI). During each cell time, the algorithm selects a cell for transmission in a round-robin fashion. Since different virtual channels may require different portions of the output bandwidth, a credit mechanism is employed to allocate bandwidth to the individual VC. With this approach, scheduling of cells is organized in frames such that a maximum of N cells can be sent during each frame. A credit n_i is associated with each VC i and is renewed at the beginning of the frame. Each time a cell is sent from a virtual channel, its credit is decremented. A VC can only participate in the round-robin selection process if it has available credits. During a frame period, if a_i cells from virtual channel i arrived at an output port, and s_i cells were sent, then

$$s_i \geq \min(a_i, n_i).$$

That is, each virtual channel is allocated a portion of the bandwidth equal to n_i/N, where N is the size of the frame.

Bandwidth guarantees are usually required only for real-time traffic flows. Data traffic requires only best-effort service. Traffic requiring no bandwidth guarantees is referred to as *available-bit-rate* (ABR) traffic. The instantaneous bandwidth left over after allocating to real-time flows can be used to transmit cells belonging to ABR traffic. ABR cells are selected for transmission only when no real-time flow with valid credits has a cell to send.

The implementation of the scheduling algorithm in the switch is illustrated in Figure 3.3. Incoming ATM cells belonging to a virtual channel with guaranteed bandwidth is added to one of the n queues based on its Virtual Channel Identifier (VCI). We refer to the queue corresponding to VC i as *GBR queue i* (for *guaranteed bit rate*). Cells belonging to ABR traffic enter a separate ABR queue.

During the start of each frame, each of the GBR queues is assigned a credit count corresponding to the bandwidth allocated to it. The credit count is the maximum number of cells that can be transmitted from the queue during the frame period. Cell transmissions from the queues to the outgoing link during the frame period are scheduled by an enable signal, which we call *token*. Any GBR queue with non-zero credit count that has a cell to send may block the token and transmit a cell. Let us assume that a cell from GBR queue k is being transmitted during the current cell cycle. The controller from queue k injects a token to the next queue. The token propagates through the chain until it is blocked by a queue that has a non-zero credit count and a cell to send; this queue may then transmit the next cell. If none of the queues with non-zero credit count has a cell to send, the token returns to GBR queue k and the next cell can be transmitted from queue k if it is non-empty and has available credits. In the case when none of the GBR queues with available credits has a cell to send, the next cell is selected from the ABR queue. If the ABR queue is also empty, a second round of token propagation is initiated. During this round any non-empty GBR queue can block the token regardless of its credit count. The second round is required to keep the output link busy when there are cells queued in the system (that is, to make the system *work conserving*).

The mapping of this model to the FAST-1 board is straightforward. The distribution function is conveniently implemented in the input modules and the concentration and scheduling functions in the output modules. The traffic modules are used to inject traffic to the switch model. This partitioning is indicated in Figure 3.1.

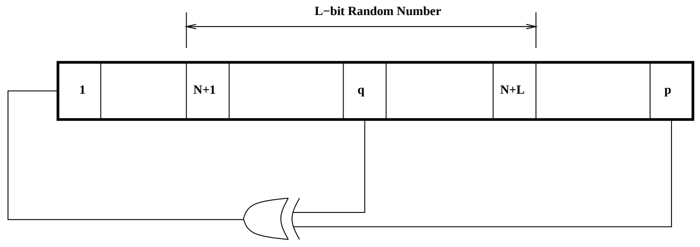

Figure 3.4: A simple implementation of Tausworthe random number generator.

3.1 Traffic Generation

The traffic generator modules can be used to implement the traffic sources needed for the simulations. Using a uniform random number generator in conjunction with lookup tables, they can be programmed to produce various traffic distributions. In an ATM simulation environment, the random number generator must satisfy several criteria such as independence of the distributions and long period of sequences to be able to simulate the target system for a long time. For example, if the target system needs to be simulated for 10^9 cells (approximately 45 minutes with a 155 Mbits/sec link speed), the random number generator must produce a non-periodic sequence of at least 10^9 numbers. In addition, the random-number generator needs to be invoked multiple times for generating cells from the different virtual channels sharing the same input port.

The random number generator we used is based on the algorithm proposed by Tausworthe [24]. Arbitrary long sequences of random numbers can be generated from linear shift-register sequences based on the primitive trinomials $X^p + X^q + 1$, over GF(2). Tausworthe proved that number sequences formed by L consecutive bits spaced any α bits apart along a sequence of bits produced by such a trinomial form a sequence of uniformly distributed random numbers with good statistical properties.

A straightforward implementation of such a random number involves a linear-feedback shift register (LFSR) as illustrated in Figure 3.4. The characteristic polynomial of the shift register is $p(x) = 1 + x^q + x^p$. The shift register has L outputs which provide an L-bit random number; producing a new random number requires shifting the LFSR L times. This approach, although simple, is relatively slow as it requires at least L cycles to generate a new random number. In our design we followed the method proposed in [25, 26] that parallelizes the L shifts by splitting the shift register into L smaller shift registers as shown in Figure 3.5. In our implementation, we used $L = 16$, $p = 127$, and $q = 1$, thus producing 16-bit random numbers. One shift operation of this

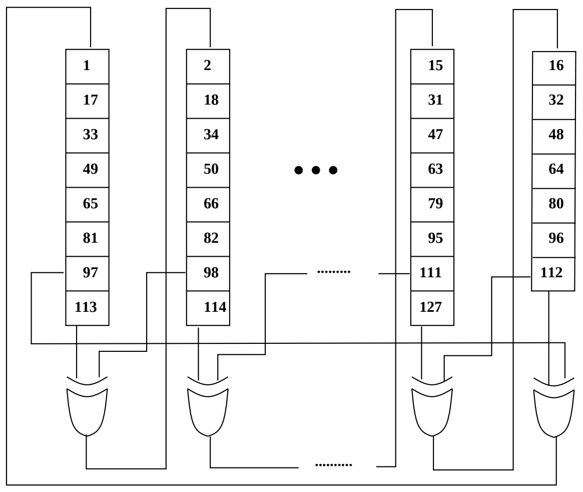

Figure 3.5: Parallel implementation of a Tausworthe random number generator ($p = 127, q = 1, L = 16$).

structure is equivalent to sixteen shifts of the simple linear-feedback shift register. Therefore, a new 16-bit random number can be produced in each cycle.

Uniformly distributed random numbers can be used in conjunction with the *alias method*, that is described in Appendix A, to produce random numbers from arbitrary distributions. Our current design of the traffic source, including the interface from the host processor to the local memory of the traffic-generator modules, runs at a maximum clock speed of 15 MHz and utilizes less than 35% of the FPGA. A total of six cycles is required for generating a cell. Note that, even if the target system were running in real-time at a link speed of 155 Mbits/sec, approximately 2.5 μseconds would be available for generating a new cell; thus the traffic generator is currently capable of producing traffic at a much higher rate than is needed by the system.

More complex traffic sources can be accommodated in the traffic generator modules. For example, an ON-OFF traffic source can be implemented using three lookup tables [30]. In this model, the ON and OFF intervals of the source are exponentially distributed. While in the ON state, it generates a burst of packets whose size is drawn from a geometrical distribution. Two lookup tables are required to determine how long the source stays in each of the two states and one lookup table is required to determine the size of the burst.

Models of video sources can be designed using Markov chains [31]. Implementation of these Markov chain models in hardware also involves the use of multiple lookup tables. A uniform random number is used to select the initial state of the Markov chain. Packets are sent at a rate determined by the current state of the Markov chain. A transition vector is associated with each state that determines the next state with different probabilities. Each of these transition vectors needs to be implemented by means of a separate lookup table. However, because the number of transitions with non-zero probabilities are small, the set of lookup tables needed can be accommodated with the available memory. In addition, adjacent states in the Markov chain can be aggregated to reduce the amount of memory needed without affecting the accuracy significantly. As we mentioned earlier, more accurate models of video traffic can be produced by injecting real MPEG video sequences through the interface board.

3.2 Distribution Stage

The distribution function of the switch is easily mapped to the input modules. A global simulation clock signals the start of a new cell cycle. Each input module reads a cell through the bus connecting it to the corresponding traffic-generator module, together with a flag indicating whether the cell is valid. The translation of the Virtual Circuit Identifier (VCI) in the cell to its output port is done through a lookup table that is stored in the local memory. The cell is then forwarded to the proper output port. In our current model, a cell is represented by just 9 bits, 8 bits for the VCI and one bit for the flag. Thus, only half of the available datapath between the modules is utilized. The rest of datapath can be used for expanding the design to support more complex functions. The current design of the distribution stage utilizes approximately 30% of the FPGA in the input module.

3.3 Concentration and Scheduling

A block-level diagram of the logic implemented in the output modules is shown in Figure 3.7. The local memory is partitioned into two regions, a *control-memory* used for storing the control information associated with the scheduling queues and a *cell memory* that is used as a buffer-pool for storing the incoming cells. The design consists of a number of modules. The receiver module receives cells from the input modules and adds them to the scheduling queues depending on their VCI values. The selection logic operates in parallel with the receiver; its function is to select the queue from which the next cell will be transmitted. Once the cell is selected, the sender module drives the necessary logic to reflect the transmission of a cell and removes the cell from the queue. The sender module is also responsible for updating a number of counters used to maintain statistics.

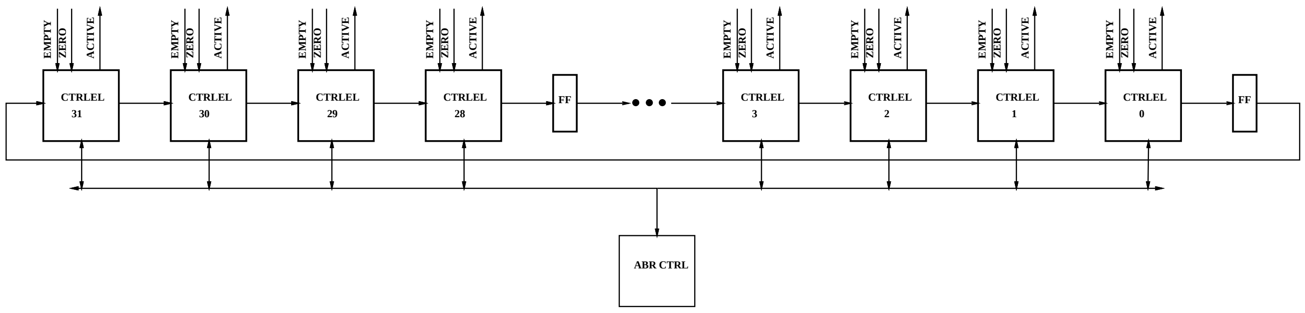

Figure 3.6: Block diagram of the selection module.

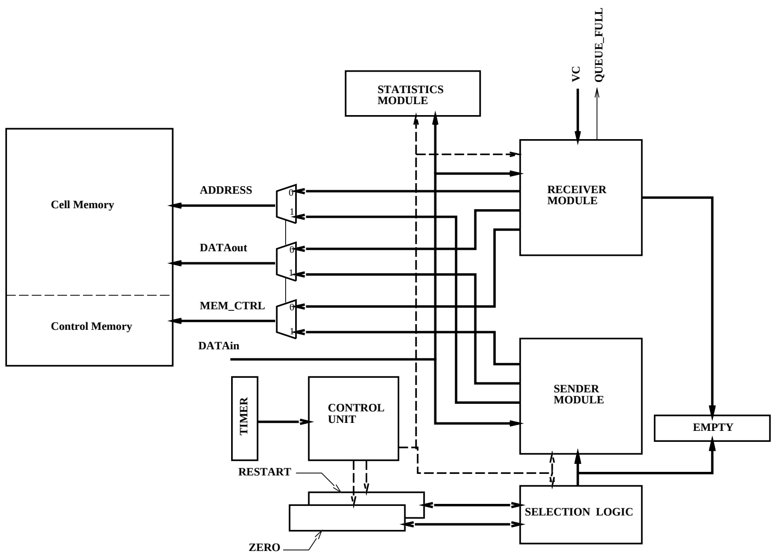

Figure 3.7: Internal architecture of the scheduler.

The control unit implements the state machine that coordinates the action of all the modules. The statistics module operates in parallel with the sender module and updates the event counters that keep track of the queueing delays of cells.

The receiver module can accept up to four new cells during each cell-time. In the worse case, the incoming cells from all the four inputs may be destined to the same output port. A separate queue is associated with each VC. A *head* and *tail* pointer for each VC are stored in the memory and are accessed immediately after the VCI is decoded. The queue for each VC is implemented as a linked list. All available buffers are shared through a free-buffer pool. Once a cell is received, a new buffer is allocated from the free pool, the cell is timestamped and added to its corresponding queue. This allows determination of the queueing delay of the cell at the time of its transmission. Finally, the head and tail pointers are updated and stored back in the control memory. The use of linked lists for implementing the queues allows full sharing of the available memory among the queues.

The selection logic uses two registers EMPTY and ZERO to select the queue from which the next cell is transmitted. The EMPTY register stores one bit for each of the queues indicating whether the queue is empty; similarly, the ZERO register consists of one bit per queue indicating whether there are available credits for each queue. The register is reset at the end of each frame to indicate the availability of credit for all queues. The RESTART register stores one bit per queue; each bit indicates whether the corresponding credit counter has been reloaded after a new frame has started. Use of this register avoids the need to reload all the counters at the beginning of each frame, allowing the counters to be stored in the control memory. TIMER is a simple down-counter that keeps track of the frame time.

A block diagram of the implementation of the weighted round robin algorithm by the selection logic is shown in Figure 3.6. There is a maximum of 32 distinct queues of VCs representing guaranteed bit-rate traffic and an additional queue for available-bit-rate (ABR) traffic. There is one control element (CTRLEL) assigned to each of the 32 queues associated with guaranteed bit-rate traffic. The CTRLEL that sent a cell during the previous cell-period initiates the selection process by activating its CARRY_OUT line to propagate the token signal through the chain. The next CTRLEL in the chain checks if its corresponding queue has a cell to transmit and if there are available credits. These conditions can be checked by simply checking the corresponding bits of the EMPTY and ZERO registers. If both conditions are true, it changes state by setting the FIRST_ROUND flip-flop and propagates a carry of zero to the next element. Otherwise, a carry of one is propagated to the next control element, allowing it to be selected.

The process is similar to that of a simple carry-chain adder. However, the logic involved in each stage is more complex. As a result, the carry propagation through the entire chain of 32 control elements can not be completed during one cycle. Instead, the control elements are organized in groups of four, as in a carry lookahead adder. All control elements belonging to the same group complete their operation during the same cycle. In the end of the cycle, the carry-out information from the last control element is stored in a flip-flop. Subsequent groups can complete their operation during subsequent cycles. The information from the last flip-flop is sent back to the first control element. Note that the control element that initiates the process is not necessarily the first one. After 8 cycles a complete decision cycle has been completed. If the carry has propagated back to the initial control element, no other queue was selected to send.

If, during the first round, a control element is selected to send, its FIRST_ROUND flip-flop would be activated. In this case the propagation of a carry would have stopped. If no control element was selected in the first round, the ABR queue is checked. If it has available cells, the cell at its head is selected for transmission and the propagating carry is set to zero. Otherwise, a second round of selection process is initiated. During this round, the first queue that has a cell to send is selected without checking its bit in the ZERO register and clearing the propagating carry to zero. In any case, the whole selection process is completed in at most 16 cycles when at least one of the queues has a cell to send.

After the selection logic has determined the queue from which the next cell will be transmitted, the sender module reads the corresponding head and tail pointers. The first cell is removed from the queue and the available buffer is added to the free list. The timestamp of the cell is used to update the counters in the statistics module. Finally, the new head and tail pointers are stored back in the control memory and the EMPTY register is updated.

The total logic implemented in the output module occupied two of the four FLEX 81188s within the FLEX 8050 multichip module, with a utilization of approximately 70% for one chip and 50% for the other. The maximum achievable speed for the output module was approximately 14 MHz. This was achieved by the use of pipelining in the processing of cells within the module. At this speed, the logic required approximately 3 μsecs to process an ATM cell. Note that this is within 20% of the time required to transmit an ATM cell on a 155 Mbits/sec link.

3.4 Connection Setup

So far, we did not mention anything about how a new VC connection is setup or how a connection is removed. Note that the goal was to design a system that will have enough logic to emulate a complete ATM switch. In each module there is some logic available for accessing the local memory through the host-processor. Note also, that all the crucial information like VC tables and credits are stored in the local memories of the modules.

The host processor is responsible for monitoring the simulation. When the software determines that a new virtual channel needs to be added to the system, the simulation is stopped and the host processor can access the local memories of the modules. In order to add a new virtual channel, the virtual table of the corresponding input module has to be updated with a new entry.

After the operation is completed the simulation can continue. In a similar way we can specify the bandwidth allocated to each virtual channel in the output link. The host-processor will access the local memory of the corresponding output module and update the value of the maximum number of credits available to the VC. Removing a VC is done in the same way by deleting the corresponding entries from the local memories. The host processor will function as a global controller of the ATM switch. Since the operations of adding or removing a virtual channel are not done very frequently, the speed of the simulation will not be affected by the speed of the processor or the interface bus.

3.5 Performance

To determine an estimate of the speedups offered by FAST-1 over conventional software simulation, we measured the running time for an example simulation of the switch model described in the previous paragraphs on the FAST-1 and compared with the running time of a simulation of the same model in a workstation. The results are summarized in Table I. The table provides the actual times taken to simulate the switch for 1 million cell-times.

The software simulations were run on a DEC Alpha 3000/400 workstation with 92 Mbytes of main memory using a simulator written in CSIM [32]. We then measured its running time by varying the number of virtual-channels used by the round-robin scheduling algorithm from 4 to 32. Note that the running time of the simulation on FAST-1 is not affected by the number of VCs, the allocation of credits among the VC, or the frame size; in the case of software simulation, however, the simulation time increased with the number of VCs owing to the sequential nature of the simulation.

The speedup of FAST-1 over software simulation varied over the range 140–180 in our example. With a more complex simulation model, the speedup of the hardware testbed would be even more dramatic. Note that the hardware approach does not incur the overheads of scheduling events, context switches, etc., that typically arise in conventional event-driven simulations.

The necessary time to program the FPGAs is also very low. Notice that multiple modules can be programmed in parallel. The software supports programming of all the traffic and input in parallel. Since in most architectures the design of these modules is the same, the total programming is completed in less than four seconds. An additional time of 5 to 10 seconds may be required for downloading the memories with the traffic distributions and control information. Notice also, that

Number of VCs	Simulation time (seconds)	
	CSIM	FAST-1
4	410	3
8	434	3
16	470	3
32	540	3

Table 3.1: Times for simulating the example 4×4 ATM switch model for 1 million cell-times using a software simulator based on CSIM running on a DEC Alpha workstation and on the FAST-1 board.

updates in the configuration of the switch are usually performed by selectively writing to memory locations; this operation requires negligible time.

4 Conclusions

Two problems are of fundamental importance in realizing the promise of ATM broadband networks: (i) Traffic scheduling algorithms in ATM switches to provide guarantees on bandwidth, delay, jitter, and cell loss rate; and (ii) congestion control algorithms to allow effective utilization of the capacity of these networks. In an ATM network, these functions will need to be implemented in hardware. A serious difficulty in evaluating traffic scheduling and congestion control algorithms in ATM networks is the lack of a hardware testbed where the algorithms could be implemented and simulated. Conventional software simulations are often inadequate for studying the behavior of these algorithms in ATM networks, owing to the large number of cell events that need to be simulated. The FAST project is an attempt to address this problem by developing a hardware testbed for functional simulation of ATM switches and networks. The testbed would allow us to evaluate the algorithms by simulating them at speeds several orders of magnitude over software simulation, and will serve as a valuable tool in our ongoing research in traffic scheduling algorithms [34, 35] and congestion control [36]. In addition to estimating the performance of the different algorithms, the testbed enables us to evaluate the hardware complexity of the algorithms and the effect of any simplifications made during implementation.

The FAST project is currently in its first phase: The hardware for FAST-1 is operational; a graphical user interface for accessing the board, programming the FPGAs and accelerating debugging has been also developed. We are currently in the process of enhancing our software tools to support the collection and presentation of the simulation data. Furthermore, we are working on expanding our library of modules with more sophisticated scheduling algorithms. Based on our experiences with FAST-1, we are also defining the architecture of a second version of the testbed that would allow simulation of entire ATM networks in real time under traffic generated by higher-level protocols running on a set of workstations. As the FPGA technology improves both in density and speed — Altera and Xilinx have already announced single-chip devices providing as many as 100,000 gates and speeds of 75 MHz — it will be possible to simulate systems of significantly more complexity and provide even higher speedups in the future.

References

[1] A.Demers, S. Keshav, and S.Shenker, "Analysis and simulation of a fair queueing algorithm," *Journal of Internetworking Research and Experience*, vol. 1, pp. 3–26, September 1990.

[2] L. Zhang, "VirtualClock: a new traffic control algorithm for packet switching networks," *ACM Transactions on Computer Systems*, vol. 9, pp. 101–124, May 1991.

[3] D. D. Clark, S. Shenker, and L. Zhang, "Supporting real-time applications in an integrated services packet network: Architecture and mechanism," in *Proc. ACM SIGCOMM '92*, pp. 14–26, August 1992.

[4] P. Newman, "Traffic management for ATM local area networks," *IEEE Communications*, pp. 34–50, August 1994.

[5] R. Fujimoto, "Parallel event-driven simulation," *Communications of the ACM*, vol. 33, pp. 30–53, October 1990.

[6] Altera Corporation, *FLEX 8000 Handbook*, July 1994.

[7] Xilinx, Inc., *The Programmable Logic Data Book*, 1994.

[8] S. Walters, "Computer-aided prototyping of ASIC-based systems," *IEEE Design & Test of Computers*, pp. 4–10, June 1991.

[9] M. Gokhale, W. Holmes, A. Kopser, S. Lucas, R. Minnich, D. Sweely, and D. Lopresti, "Building and using a highly parallel programmable logic array," *IEEE Computer*, no. 24, pp. 81–89, 1991.

[10] J. M. Arnold, D. A. Buell, and E. G. Davis, "Splash 2," in *Proceedings of the 4th Annual ACM Symposium on Parallel Algorithms and Architectures*, pp. 316–324, June 1992.

[11] L. Barroso, S. Iman, J. Jeong, K. Oner, K. Ramamurthy, and M. Dubois, "The USC multiprocessor testbed project: Project overview," Tech. Rep. 15, University of Southern California, 1994.

[12] C. Cox and W. Blanz, "GANGLION - a fast field-programmable gate array implementation of a connectionist classifier," *IEEE Journal of Solid-State Circuits*, vol. 27, March 1992.

[13] S. Monaghan and P. Noakes, "Reconfigurable special-purpose hardware for scientific computation and simulation," *Computer & Control Engineering Journal*, vol. 3, pp. 225–234, September 1992.

[14] P. Bertin, D. Roncin, and J. Vuillemin, "Programmable active memories: a performance assessment," in *Proc. International ACM/SIGDA Workshop on Field Programmable Gate Arrays*, pp. 57–59, February 1992.

[15] D. A. Thomas, T. Petersen, and D. Van den Bout, "The Anyboard rapid prototyping environment," in *Advanced Research in VLSI, Proceedings of the 1991 UC Santa Cruz Conference*, pp. 356–370, 1991.

[16] G. D. Micheli, *Synthesis and Optimization of Digital Circuits*, McGraw-Hill Series in Electrical and Computer Engineering, 1994.

[17] F. A. Tobagi, "Fast packet switch architectures for broadband integrated services digital networks," *Proceedings of the IEEE*, vol. 78, pp. 133–167, November 1990.

[18] Altera Corporation, *FLEX 8050M Data Sheet*, August 1994.

[19] Aptix Corporation, *Aptix System Data Book*, 1993.

[20] Mentor Graphics Corporation, *QuickSim User's Manual*, 1995

[21] Mentor Graphics Corporation, *Autologic II Reference Manual*, 1995

[22] Altera Corporation, *Mentor Graphics & MaxPlus II Logic Design*, 1994, Application Note 32.

[23] M. Katevenis, S. Sidiropoulos, and C. Courcoubetis, "Weighted round-robin cell multiplexing in a general-purpose ATM switch chip," *IEEE Journal on Selected Areas in Communications*, vol. 9, pp. 1265–79, October 1991.

[24] R. Tausworthe, "Random numbers generated by linear recurrence modulo two," *Mathematics of Computation*, vol. 19, pp. 100–119, 1965.

[25] J. Saarinen, J. Tomberg, L. Vehmanen, and K. Kaski, "VLSI implementation of Tausworthe random number generator for parallel processing environment," *IEEE Proceedings-E*, vol. 138, May 1991.

[26] M. Barel, "Fast hardware random number generator for the Tausworthe sequence," in *Proc. 16th Annual Simulation Symposium*, pp. 121–135, March 1983.

[27] A. M. Law and W. Kelton, *Simulation Modeling & Analysis*. McGraw-Hill, Inc., 1991.

[28] A. Walker, "An efficient method for generating random variables with general distributions," *ACM Transactions on Math. Software*, vol. 3, pp. 253–256, 1977.

[29] R. Kronmal and A. Peterson, "On the alias method for generating random variables from a discrete distribution," *Am. Statistician*, vol. 33, pp. 214–218, 1979.

[30] R. Jain and S. Routhier, "Packet trains — measurements and a new model for computer network traffic," *IEEE Journal on Selected Areas in Communications*, vol. 4, pp. 986–995, September 1986.

[31] D. Heyman, A.Tabatabai, and T. Lakshman, "Statistical analysis and simulation study of video teleconference traffic in ATM networks," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 2, pp. 49–59, March 1992.

[32] H. Schwetman, "CSIM Reference Manual," Tech. Rep. ACT-ST-252-87, Rev. 16, Microelectronics and Computer Technology Corporation, 1992.

[33] L. Kalampoukas, A. Varma, D. Stiliadis and Q. Jacobson, "The CPU Design Kit: An Instructional Prototyping Platform for Teaching Processor Design," Workshop on Computer Architecture Education, Int'l Symposium in Computer Architecture, June 1995.

[34] D. Stiliadis and A. Varma, "Providing bandwidth guarantees in an input-buffered crossbar switch," in *Proc. IEEE INFOCOM '95*, April 1995.

[35] D. Stiliadis and A. Varma, "Frame-based fair queueing: A new traffic scheduling algorithm for packet-switched networks," Tech. Rep. UCSC-CRL-95-39, U.C. Santa Cruz, Dept. of Computer Engineering, July 1995.

[36] L. Kalampoukas, A. Varma and K.K. Ramakrishnan, "An efficient rate allocation algorithm for packet-switched networks providing max-min fairness," in *Proc. High-Performance Networks*, September 1995.

A Traffic Generators

Once a uniform random number is generated, it can be used as an index into a lookup table to generate any other discrete distribution of random numbers. The lookup table is stored in the local memory of the traffic modules. The uniform random number is used as the address of the lookup table. The straightforward way to produce a discrete random variate X that has any distribution F is by using the inverse transform method [27]. If $p(x_i), i = 1, 2, \ldots, n$, is the probability mass function of the distribution, its distribution function is given by

$$F(x) = prob(X \leq x) = \sum_{x_i \leq x} p(x_i).$$

Without loss of generality we can assume that $x_1 < x_2 < \cdots x_n$. Then we can generate a random variate with distribution F by generating a uniform random variate U and returning the $X = x_j$, where j is the smallest positive integer such that $U \leq F(x_j)$. A binary search is required to determine j in the second step of the algorithm. Thus, although the space requirements of the inverse transform method is only $O(n)$, the time complexity is $O(\log n)$. This approach is therefore unsuitable for hardware implementation.

In our system we used the *alias* method [27, 28, 29]. The alias method can be used to generate any discrete random variate having a finite range of values. If the range of the distribution is $\{0, 1, \ldots, n\}$, the algorithm requires maintaining two tables of length $n + 1$ each. The first table contains the *cutoff values* F_i and the second table the *aliases* L_i. Once these tables are generated, we need to calculate two uniform and independent random variates, one from a continuous distribution over $(0, 1)$ and the other a discrete number from $\{0, 1, \ldots, n\}$. Let U be the former and I the latter. If $U \leq F_I$, that is the value obtained from the first table by indexing with the random number I, then I is returned as the desired random variate; otherwise L_I, the value from the second table indexed by I, is returned. Thus, the cutoff values represent the probabilities of returning I instead of its alias. By a suitable choice of cutoff values and aliases, the distribution produced by this method can be matched to the desired one.

The alias method requires the generation of two random numbers, one comparison and two accesses to the memory. Thus, it is ideally suited to hardware implementation. Note also that the Tausworthe random number generator allows us to generate multiple random numbers in parallel by using different distances between the selected bits of the LFSR. The main limitation of the alias-method is that it requires $2(n + 1)$ words of storage for the lookup tables. In our case, however, the available memory is not a bottleneck for even large values of n.

There are several algorithms that can be used to generate the tables of cutoff values and aliases. The tables produced by these algorithms, in general, are not identical, but they produce the same distribution. A method proposed by Walker generates the tables in $O(n \log n)$ time [28]. The most time consuming operation in this algorithm is an initial sorting of the $p(i)$'s. A more efficient algorithm was proposed by Kronmal and Peterson [29], based on simple operations on sets. However, if the sum of the probabilities $p(i)$ does not add exactly to one, the latter algorithm could suffer from roundoff errors. It is reported in [27] that the latter method failed when the sum of the probabilities differed from 1 by as little as 10^{-5}. Note that the run-time efficiency of the algorithm is not critical in our case as the tables are computed only once by the host processor and are not modified during the simulations.