An IRAM-Based Architecture for a Single-Chip ATM Switch

I. Papaefstathiou[†], A. Brown[*], J. Simer[*], D. Sobel[*], J. Sutaria[*], S.Y. Wang[*],
T. Blackwell[*], M. Smith[*], W. Yang[*]

Abstract

We have developed an architecture for an IRAM-based ATM switch that is implemented with merged DRAM and logic for a cost of about $100. The switch is based on a shared buffer memory organization and is fully non-blocking. It can support a total aggregate throughput of 9.6 Gbits per second, organized in any combination of up to 32 155 Mb/sec, eight 622 Mb/sec, or four 1.2 Gb/sec full-duplex links. The switch can be fabricated on a single chip, and includes an internal 4 MB memory buffer capable of storing over 85,000 cells. When combined with external support circuitry, the switch is competitive with commercial offerings in its feature set, and significantly less expensive than existing solutions. The switch is targeted to WAN infrastructure applications such as wide-area Internet access, data backbones, and digital telephony, where we feel untapped markets exist, but it is also usable for ATM-based LANs and even could be modified to penetrate the potentially lucrative Fast and Gigabit Ethernet markets.

1 Introduction

Asynchronous Transfer Mode (ATM) is a circuit-based switching technology that meets the ever-increasing market demands for network bandwidth. ATM networking technology provides support for Broadband Integrated Services Digital Networks (B-ISDN), the high-speed transfer of voice, video and data over a single network. In addition, ATM technology is an excellent choice for network backbones that carry sizeable amounts of traffic. In the near future, we may expect to see ATM switching technologies deployed to provide local and wide-area internetworking. While the cost of currently-available ATM networking solutions has hindered its adoption and deployment on a large scale, we believe that a low-cost solution and a number of recent market developments promise to create a sizeable demand for ATM switching technology.

ATM switches in recent years have been moving out of research testbeds and into commercial use. The recent adoption of several ATM signalling standards by the ATM Forum increases the potential for interoperability of different brands of ATM switches with different capabilities and opens up the ATM switch market to competition. We believe that the large-scale deployment of fiber-based Wide Area Networks (WANs) by telcos and cable companies will lower both the cost of fiber-interface circuitry and the cost of fiber deployment. As the costs associated with installation of fiber networks decrease, we believe that technologies such as ATM and Gigabit Ethernet will become more viable Local Area Network (LAN) technologies. Additionally, as cable companies and telcos begin to deploy advanced, wide-area, high-bandwidth digital services over these new fiber networks, they will require advanced switching technology that is both small and inexpensive; we feel that an ATM network using our switch will provide that technology.

2 Architectural Overview

The switch we have designed is in most respects a 32x32 port fully-interconnected output-buffered switch capable of operating at OC-3 (155 Mb/sec). The switch has 35.5 Mbits of buffer space for cells (85 kCells of space), its buffer is organized in a pipelined fashion as proposed in [?] and used in [?], and cells queued for a single OC-3 output port can occupy as much as one half of the buffer at any given time. Thus, while the buffer is not fully shared, it mimics the behavior of a shared buffer in most traffic conditions. Modifications to our core 32x32 architecture have been made to allow ports to be "bundled" and function together as a single port servicing a higher data rate. For instance, our switch can be configured seamlessly to function like a fully-interconnected 8x8 OC-12 (622 Mb/sec), or 4x4 OC-24 (1.2Gb/sec) switch. Any bundled port can have access to the entire memory space, and thus for any non-OC3 port, the cell memory is effectively fully shared. A detailed description of the bundling scheme can be found in [?]. A block diagram of our switching chip can be found in Figure 1.

[2]Cambridge University, Supported by a Marie Curie Research Training Grant under TMR activity 3
[1]Harvard University

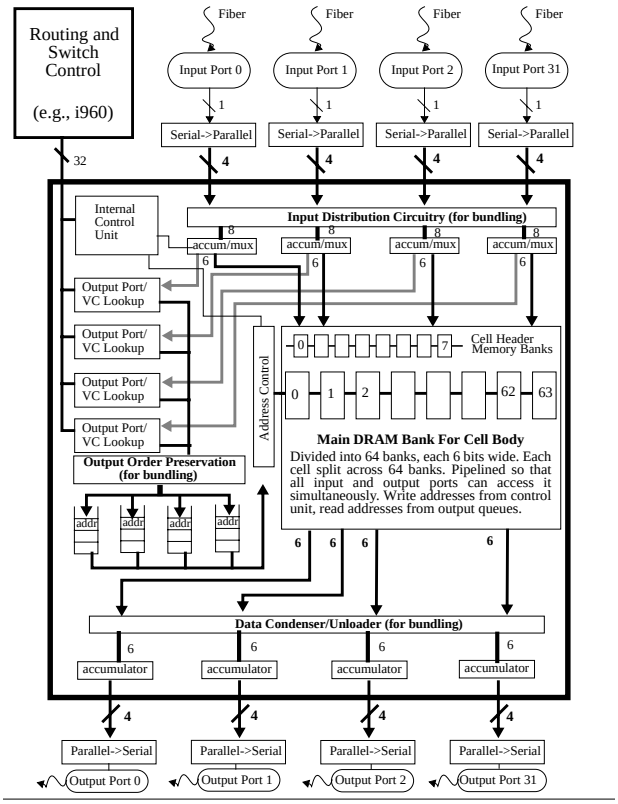

Figure 2: Block diagram of single-chip ATM switch. The diagram shows a switch with 32 full-duplex ports.

2.1 Getting Cells Onto and Off the Chip

Serial data arriving on fiber-optic transmission lines is converted from light signals to electrical signals by readily-available off-chip hardware. External logic is used to convert the serial data stream into a wider parallel stream and to perform necessary SONET-protocol framing procedures on the incoming cells. Such hardware is readily available for OC-3 and OC-12 data rates. While there is a paucity of OC-24 support circuitry available, we envision that this technology will become readily available as ATM technology matures. At that time, our ATM switch will be still be capable of handling these faster data rates.

We must briefly mention the functionality of the existing support hardware. Each bi-directional port must be supplied with two distinct pieces of hardware. One is a optical/electrical conversion chip, which handles all the light generation and detection as well as clock recovery. This chip, which is independent of cell framing, hands off the electrical data to a SONET-based cell detection processor (such as the one made by PMC-Sierra), which processes and frames each cell and performs some error detection. Of note is the fact that these chips will pad out the header to 6 bytes, with the sixth byte containing error detection information, and thus bring the total cell length to 54 bytes. For reasons that will become clear later, this extra byte padding makes the control of our switch far simpler. Also, these support chips are bi-directional, so data flow in the opposite direction occurs at the output ports.

Our ATM switch can be set up in several different configurations, and each configuration has a different number of "virtual ports" operating at a different data rate. Thus, the pin width for each "virtual port" is naturally dependent on the type of port being serviced. For an OC-24 data port (typically used in a 4x4 fashion), the pin width is 32 pins per port (or 64 pins per bi-directional port). If our chip is to be used as a 8x8 OC-12 switch, then the pin width is halved to 16 pins per port. Likewise an OC-3 port will require 4 pins.

In all of these configurations the pins will be clocked at an external speed of roughly 40 MHz, a speed easily realizable on a PC board.

It is important to note that there is nothing preventing our switch from being setup in some "hybrid" configuration. For instance, if our switch were to be used at the terminal point of some larger network, where it was to serve as a connection between several workstations and the network backbone, our chip could be setup to handle 6 OC-12 bi-directional ports to connect to 6 workstations and a single OC-24 bi-directional port to connect to the high-speed backbone.

Once inside the switch, the data stream is parsed via the accumulators (one per port) into 6 bit wide words and routed to the one of the 32 internal ports as appropriate. The 6- bit word size was chosen since the 48-byte body of an ATM cell can be broken up into 64 6-bit words, and we will be using 64 stages in the pipelined memory buffer. Note that a 53-byte ATM packet is one byte short of 72 full 6-bit words. However, since the serial-to-parallel SONET framing circuitry pads the header from five to six bytes, the cell divides nicely into 72 6-bit words. (The header comprises 8 words, and the payload comprises 64.) This division simplifies the header analysis within the chip. Cells leave the switch in the same manner: the 6-bit internal words are broken up or accumulated into their appropriate pin width, and are then transmitted off-chip via the dedicated data busses. Once off-chip, these words are converted to a serial signal, and then to light pulses on the fiber lines.

We chose to place the optical-to-electrical conver-

sion circuitry off-chip so that it was not necessary to worry about performing the light generation and detection on-chip. Also, the serial-to-parallel framing circuitry must run with an extremely fast clock, so pushing it off-chip simplifies our design significantly, and does not significantly affect the complexity of an entire switch built with our chip at its core. Also, these chips are readily available on the market.

2.2 Cell Headers: Routing Lookup, Handling, and Storage

Let's now examine how the headers are stored and updated. An ATM cell header is 5 bytes (40 bits) long, and contains exactly one 24-bit field (bits 4-27) which must be updated by the switch (the others can be left untouched). This is the virtual circuit/path identifier (VCI/VPI). The switch must maintain state that maps an input pair (input_port, input_VCI/VPI) to an output pair (output_port, output_VCI/VPI). The input_port is known (since the switch knows on which bus the cell arrived), and the input_VCI/VPI can be determined from the cell header. The fields of the (output_port, output_VCI/VPI) pair must be computed for each arriving cell, as the output_port field determines the output port for which the cell is destined, and the output_VCI/VPI must be written into the cell's header before it is sent out.

The translations between (input_port, input_VCI/VPI) and (output_port, output_VCI/VPI) are initialized externally by the routing control unit. The switch maintains 32 lookup tables, one per input port, in its internal DRAM; these tables are used to map the input_VCI/VPI to an output port and output VCI/VPI (since each port has its own table, the input port need not be specified in the mapping).

We would like to be able to support many virtual circuits through the switch, ideally as many as 256K (or 8K circuits per OC-3 port). However, there are 224, or 16M, possible VCI/VPI identifiers. Keeping a table of 224 entries per port is prohibitive in terms of memory used. Even using a 8K-entry open-hash table is not ideal, as the worst-case probe time for such a table is O(8K), which would severely impact the latency and synchronization of the switch. Luckily, since the problem only affects the input VCI/VPI (since we can easily store entire 24-bit output VCI/VPIs), we can merely specify a certain 14-bit subset of the VCI/VPI field which will be used by our switch (and only use those VCI/VPIs when establishing VCs or VPs). Then we need only address a 8K-entry linear table of (output_port, output_VCI/VPI) entries with these 14 bits in order to determine the destination and output header for each incoming packet. For a 32- port switch, each entry in this table takes 29 bits (24 for output VCI/VPI and 5 bits to select from 32 output ports), so each port's table takes up 232 kbits, and the space overhead of the 32 routing tables is 7.2 Mbits.

Because we only store cell bodies in the main memory bank, and because the cell header arrives in the switch in its own set of eight 6-bit words, we can perform the tasks of writing the cell body to memory in parallel performing the header lookup and storage. Furthermore, the cell headers will be stored in a 8-stage pipelined memory in order to maximize data throughput.

Each output port has associated with it a FIFO queue. Each entry in the queue is a 17-bit word containing the address of the cell in the main memory bank and the header in the header memory (they share the same 17-bit address). These queues are filled as follows. When a cell header arrives at an input port, the cell is assigned an address from the free address FIFO. The header is then sent to the look-up table for its input port, where it is latched into a register. The address assigned to the cell is also latched by a register associated with the look-up table, and is also sent to the pipelined memory. The table lookup and translation has been completed by the time the cell payload arrives at the switch, and thus the payload and header are written to their respective pipelined memories during the same clock cycles. Meanwhile, the address of the header and payload in memory is forwarded to the appropriate output port and stored in that port's output address queue.

Once an output port sees that its queue is nonempty, it uses the stored address to access the pipelined memory banks and to stream the cell header and body that it retrieves out to the external data bus to be sent on the fiber. It continues doing this until the queue is empty again, at which point it waits for more packets. Thus it is the output port that drives its own transmission, and which retrieves the data from memory, so there should be no unnecessary output blocking in our switch design. There is some centralized control that synchronizes and orders the ports' accesses to memory so that contention is not an issue.

2.3 Memory System: Size and Refresh

It was decided that an ATM switch was well suited to IRAM-implementation because of the "memory-heavy" nature of an ATM switch. It should be fairly clear to the reader that ATM switches seem to be little more than a stand-alone memory subsystem with small amounts of extra ATM-specific processing logic. Previous implementations of ATM switches have re-

Part of Switch	Memory Contents	Memory usage
Payload buffer	85k cells 48 bytes each	32Mbits
Header buffer	85k cells 8 bytes each	3.3Mbits
Lookup Tables	256k entries 29 bits each	7.2 Mbits
Output Address FIFOs	32 FIFOs 43K * 17 bits each	22 Mbits
Free Address FIFO	85k entries 17 bits each	1.4 Mbits
	Total Memory Usage	65.9Mbits

Table 1: Memory usage of the switch. Notice that the payload and header buffers together account for 55% of the memory, while the look-up tables use an additional 12%. The remaining 33% of memory is in the output address and free address FIFOs; this 33% is overhead beyond the basic cell storage requirements.

lied primarily on SRAM cells as the basis of its core memory, but our switch will be built using DRAM-based memory. The motivation for using a DRAM-based architecture is clear: DRAMs are smaller and cheaper than SRAMs, so a switch using DRAM can have significantly more memory in a similarly-sized package. With more memory, an ATM switch can perform remarkably better under heavy traffic loads.

One of the most difficult problems, however, in building a DRAM-based ATM switch–and probably a significant reason why one has never been built before–is in handling the memory refresh that DRAM requires. ATM switches require constant access to memory, and building a switching network around the predetermined refresh schemes of commodity DRAM is likely to require numerous stalls as the memory refreshes itself. This is one of the reasons IRAM integration is so appealing. We can have the chip internally schedule the necessary refresh cycle so as to completely eliminate conflict between memory requests and internal refresh.

Our switch would have an internal refresh scheduler controlling the refresh cycles for each major piece of DRAM memory: header and payload buffer memory, VPI/VCI look-up tables, free address FIFO, and output address FIFO. The refresh scheduler would be a simple counter cycling through the address space of each of these components and perform a read operation in order to refresh the individual DRAM cells. Our refresh schedulers will perform refresh cycles during the naturally occurring idle time of the various memory components. Thus, refresh can be handled invisibly without any performance degradation whatsoever.

3 Conclusions

We can implement a 9.6Gbits/s ATM switching fabric at low cost through the use of Intelligent DRAM technology and an integrated, pipelined architecture. We believe such a product can be utilized in ATM switches that offer substantially better cost/performance ratios than those currently available from ATM vendors. In addition, the widespread deployment of high-speed WANs by telecommunications companies will increase demand for low-cost, high-bandwidth networking solutions. We believe that our switching core is ideally suited to such applications because of its large feature set, its flexibility, and its small size. Not only is our design well-suited to WAN networking solutions, it is also appropriate for high-speed LANs. While the costs associated with adoption of high-speed ATM networking have been prohibitively high, we believe that large-scale deployment of such networks will provide an economy of scale. LAN switches based upon our integrated switching fabric will be well-positioned to capture sizeable market share. Furthermore, through the use of appropriate "filter" circuitry, we believe that our switching fabric can be used to support network technologies other than ATM–Fast Ethernet or Gigabit Ethernet for example. In conclusion, we believe that this design for an IRAM-based network switching fabric provides a high-performance, low-cost product that is well-positioned to provide significant profitability in the high bandwidth networking market.

References

[1] M. Katevenis, P. Vatsolaki, A. Efthymiou, *Pipelined Memory Shared Buffer for VLSI Switches*, Proceedings of SIGCOMM, June 1995.

[2] Katevenis M., Serpanos, D., and Vatsolaki, P., *ATLAS I: A General-Purpose, Single-Chip ATM Switch with Credit-Based Flow Control*, Proceedings of the Hot Interconnects IV Symposium, January 1996.

[3] Brown A., I. Papaefstathiou, J. Simer, D. Sobel, J. Sutaria, S. Wang, T. Blackwell, M. Smith, W. Yang. *An IRAM-Based Architecture for a Single-Chip ATM Switch*, Technical Report TR-07-97, Center for Research in Computing Technology, Harvard University, 1997.