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

Aaron Brown, Ioannis Papaefstathiou, Joshua Simer, David Sobel, Jay Sutaria, Shie-Yuan Wang
Harvard University

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 1.2 gigabytes 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

1.1 The Problem of Networking

We are today in the midst of a new era of computing: the era of internetworking. The past few years have seen unprecedented growth of networking, in LANs, WANs, Intranets, and especially the Internet. Connected computers have become the norm; it is becoming more and more rare to find a computer without at least a modem link to some sort of network. All indications are that this trend will continue.

However, today's mainstream network technology is barely capable of handling the existing demands of network traffic, let alone the new bandwidth demands of transmitting increasingly-popular multimedia content. With the explosion of popular interest in the Internet and its glamorous and multimedia-heavy World Wide Web, the demand for high-bandwidth connections across networks as wide in scope as the telephone and cable TV systems has become critical. Current mainstream network technologies are not capable of handling the new bandwidth demands: systems such as Ethernet degrade quickly under the heavy traffic load on the Internet and do not have the capability to reach over long distances, for example from a user's house to his ISP or phone company. Traditional high-end solutions to wide-area networking, including ISDN and dedicated digital telephone lines do not provide enough bandwidth or switching capability to justify their high expense.

1.2 The ATM Solution to Wide Area Networking

One solution to the problem of delivering bandwidth in wide area networks (WANs) is to build them on top of a network infrastructure based on ATM (Asynchronous Transfer Mode) technology. When run over fiber-optic lines with SONET framing, ATM protocols can deliver bandwidth in excess of 1.2 gigabits per second over distances of several miles. ATM is also well-suited to new wide-area multimedia applications (including digital telephony and videoconferencing) as it supports a small (48-byte) data transmission unit (a "cell"), reducing the impact of dropped cells on perceived audio and video.

Not only is ATM technically suited for WAN applications, but the economics of WAN networking make ATM an attractive choice. There are three major costs in building ATM networks: the fiber-optic cabling, the expensive switches to which each computer must directly connect, and the host adaptor hardware/protocol software bundle needed on each networked machine. Unlike in LANs, where systems such as Fast Ethernet allow ATM-like speeds with existing copper cabling, in WANs, the cost of rewiring with fiber optics is almost a given, as any existing copper wires do not support the bidirectional high-speed data flow necessary to support high-bandwidth services over long distances. Also, most Internet backbones already run on fiber-optic links, and currently many cable TV and telephone systems are rewiring their networks with fiber-optic digital trunks. Additionally, in WANs, the bulk of the connections are between switches and routers, and thus there are few host adaptors needed, minimizing the impact of the high cost of ATM adaptors.

Thus in the WAN market the greatest expense in converting to the high bandwidths promised by ATM is in the network switches. The switches are, after the cabling, the most critical component in a WAN, as they provide the routing which moves the data from its source to its destination. In addition, switches provide the necessary connection between lower-bandwidth subscriber lines and high-bandwidth trunk lines, so they must support flexible port configurations with multiple bandwidth capabilities. Unfortunately, most currently-available commercial ATM switches that have these features are shockingly expensive: a Cisco offering costs well over $5,000 per port (e.g. per machine connected to the switch). Additionally, these switches are large in physical size, making it hard to place them, for example, on telephone poles for a digital telephone or cable TV network.

1.3 Using IRAM for WAN ATM Switches

Thus in a WAN ATM switch, there are four especially desirable features: low cost, small size, high speed, and flexibility in port configuration. By using single-chip IRAM technology, we can provide all of these.

The low cost and small size come from the single-chip design of our switch. Because the entire switching core plus the memory buffer are combined on a single approximately-300-pin chip, expensive high-speed memory busses and external memory banks are unnecessary. External circuitry is needed only for implementing a framing protocol and doing any needed optical-to-electrical conversion, and for switch initialization. Our entire switch can easily be packaged on a single, small circuit board, and thus, unlike conventional switches, are small enough that they could be placed on each telephone pole to route digital transmissions between houses and the central office in a large-area WAN. Additionally, the switch cost itself is low because it fits on one chip that can be mass-produced almost as easily as DRAM itself. The only additional cost is that of the external circuitry, and we will demonstrate below how this may be inexpensive for certain applications.

The features of small size and low cost are not unique to IRAM solutions; there are other single-chip ATM switches, most notably the ATLAS switch [1], that also provide small size and low cost. However, what distinguishes our chip is the basis of IRAM in a DRAM process. Traditional DRAM technologies cannot provide the gigabyte-per-second throughput needed to support a high-speed store-and-forward ATM switch. Thus most traditional switch designs (as well as the existing single-chip designs) have used small SRAM buffers (13.5 KB for ATLAS, about 256 cells) and have dedicated large amounts of logic to fancy flow-control schemes to prevent buffer overruns. In contrast, IRAM allows us to place a very large DRAM buffer on chip (at least 4 MB in size, enough for more than 85,000 cells) and to access it with almost the speed of SRAM. The large buffer can easily absorb spikes in network traffic, making it unnecessary to implement a complex credit-based flow control scheme (which do not perform well in WANs in any case, as the cell transmission time is too great). Finally, because the DRAM can be organized especially for storage of ATM cells, and because it is so close to the logic, it can supply the 1.2 GB/s bandwidths we require for fast switching. We thus obtain the high performance that is desired in a WAN switch: the aggregate memory throughput allows for 4 full-duplex OC-24 (1.2 Gb/sec), 8 full-duplex OC-12 (622 Mb/sec) ports, or 32 full-duplex OC-3 (155 Mb/sec) ports. In addition, because the higher-bandwidth ports are provided by cell-level bundling over several lower-bandwidth switching paths (see Section 3.5), the switch can support any combination of these port configurations subject to the total bandwidth constraint (for example one OC-24 port and 24 OC-3 ports). This provides the desired flexibility in port configuration. Note however that this flexibility comes at some cost in latency (see Section 3.6); however, in a WAN where cells already take several microseconds just to travel across the fiber, this latency increase is an acceptable trade-off.

So now let us examine a possible scenario for which our single-chip ATM switch is ideal. Imagine that a cable company wants to provide speedy (at least 50 Mb/sec) Internet access to its subscribers (this is not an unreasonable scenario—for example, Continental Cablevision has done this already in several suburbs of Boston). They have already committed to replacing their aging and insufficient coaxial trunk lines with fiber optics. They can run OC-24 ATM with SONET framing over fiber from their distribution points to each block or collection of houses; the OC-24 trunk enters our ATM switch at the top of a telephone pole (this requires one relatively expensive SONET decoder circuit plus optical-to-electrical converters). The other 24 OC-3 ports on the switch each connect via existing coaxial cabling to a house on that block; they use a simple framing protocol like HIC/HS, obviating the need for the expensive optics and SONET encoder/decoder. Inside the house, a similar inexpensive HIC/HS decoder (or "cable modem") either connects directly to the PC or converts the ATM framing to fast Ethernet framing, allowing existing inexpensive network adaptors and software to be used. The cable modem could also implement simple rate-based flow control or data-rate throttling to provide different levels of service and to balance the load on the switches (since 24 fully-utilized OC-3 connections can easily overload the one OC-24 trunk link). Thus beyond the initial cost of the fiber optic trunk cabling, each house that wishes to be connected at a very high speed adds at most a new switch plus a "cable modem," both of which should be inexpensive to produce. Finally, our switch is small enough that it is feasible to put it atop a telephone pole or in a customer's basement.

If our switch were adopted for such a scheme, the market would be immense—consider the number of switches that would be sold if one were placed atop each telephone pole in just one city. And this market would be assured as long as the Internet continues its expansion and its users crave more speed, both of which are likely to be true for at least several years (witness the demand for ISDN, satellite data delivery, and cable modems).

1.4 IRAM-based Switching for LANs

Despite being targeted to WAN applications, our ATM switch is a competent LAN switch as well, and can handle up to 32 switched 155 Mb/s links (or 8 switched 622 Mb/s links) per switch. In the LAN domain, our switch's biggest advantage is price, as it lacks some of the high-end features present in some commercial ATM switches (such as credit-based flow control). Each OC-3 port on a switch based around our IRAM core has an estimated parts cost of $90; even allowing for an excessive 500% markup on our chip, this is still a factor of three cheaper than Cisco's offering. We believe that the price advantage outweighs the lack of built-in credit flow-control features, as flow control can still be obtained with our switch through end-to-end rate-based schemes.

However, it is important to remember the economics of the LAN market. ATM has been available in the LAN market at the OC-3 (155 Mb/sec) level for several years, and yet has made little penetration into the market. The primary reason for this is price. Recall from above that the three components of the price of an ATM network are the cost of the fiber-optic cabling, the switch cost, and the cost of the host adaptors and software. Unlike WANs, in most LAN environments, there is already a significant investment in copper twisted-pair Ethernet cable that is capable of carrying 100 Mb/sec Fast Ethernet (100Base-T), especially when technologies such as Cogent’s T4 systems allow existing Category 3 and 4 wiring to support 100 Mb Fast Ethernet. Additionally, Fast Ethernet switches and host adaptors are significantly less expensive than ATM switches and adaptors, and existing TCP/IP software runs essentially unmodified over Fast Ethernet. Thus ATM needs a compelling reason to justify the high cost it imposes in new cabling, switching, and interfaces, and 155 Mb/sec speeds are not reason enough for the LAN market. Even if switches were as cheap as we feel we can make them, the cost of fiber-optic wiring and new host adaptors and software far outweighs the bandwidth advantages in the mass market.

A potentially more profitable approach to the LAN market might involve dropping ATM entirely, and using the switching technology described herein as the core for a sub-$500 100Base-T Fast Ethernet switch: since the cabling and host-retrofitting costs for fast Ethernet are cheap, a small, inexpensive fast Ethernet switch could gain the ubiquity that the $50 desktop 10Base-T Ethernet hub has today. It seems quite reasonable to produce an Ethernet switch for this price, as the expensive SONET framing chips and optics required by ATM are not needed in Ethernet; it may even be possible to put a cheap “preprocessor” chip on either side of our ATM switch and convert Ethernet to ATM and back. This is certainly an avenue to consider before committing further resources to pure ATM switching, as a mass consumer market (such as that for Fast Ethernet, and even the forthcoming Gigabit Ethernet) might well be easier to penetrate than a relatively esoteric (but potentially more lucrative) market like that for digital WANs.

1.5 Roadmap

The remainder of this paper is organized as follows. Section 2 describes the high-level chip architecture, including features and design decisions. Section 3 presents the full architectural details of the switch design. Section 4 then presents some benchmarking and simulation results; Section 5 offers an analysis of the cost and market potential for the switch. Finally, we conclude in Section 6.

2 High-Level Architecture: Features and Design Decisions

In this section we present some of the features of our single-chip ATM switch and the design decisions that we made in establishing the architecture.

2.1 Overview of Design Decisions

In approaching the task of designing this switch we had to make several important high-level architecture decisions before we could even begin to make a paper design. As described in the introduction, ATM has not been widely accepted in many traditional networking markets, in particular the LAN market, due to the extraordinary high price/associated start-up costs of installing the ATM network and fiber. Therefore, we chose to primarily aim our switch, not at the small and saturated LAN market, but at the new and expanding market for high-speed WANs. Cable television and telephone companies are among those looking for high-speed backbone networks to handle rapidly increasing network traffic demands. We decided to design our switch to offer comparable performance to existing architectures at a substantially lower price. This approach not only gives us a new market which we could enter, but also could provide us with a stepping-stone into the LAN market, which is a possibility we will discuss at length later on.

Another way in which we sought to increase the marketability of our switch is to make it as flexible as possible, so the same basic design can support a large number of configurations. We have achieved this by designing a single chip which can serve as a 32x32 OC-3 switch, an 8x8 OC-12 switch, or a 4x4 OC-24 switch. A telephone company, for example, could use our chip to have an expensive OC-24 (or higher) backbone split off into several slower and less expensive cables to provide links to several homes or businesses. The only difference among these configurations would be a few control lines. A retailer could stock large amounts of identical parts, and it would be a simple matter to reconfigure the hardware based on what sort of demand there was from consumers. This would serve to reduce retailer costs and, in turn, the cost to consumers.

We also believe that with some modification we could configure our chip to service Gigabit ethernet. This would require that we either design a similar but distinct chip for this purpose (with the added manufacturing costs) or that we design additional “translator” chips. Still, it is well within the realm of possibility if it turns out to be worth the time and effort. This would further enhance the flexibility of the basic design, increase the potential market, and reduce the cost of both chips by spreading out the R&D costs further, since very little additional R&D would be needed for the Ethernet chip.

Therefore we believe that we have reached the point on the price/performance curve that we were aiming for: comparable performance at a substantially lower cost. Our chip represents an excellent choice for WAN users due to its cost and flexibility, while still providing adequate performance to serve LAN users. For retailers, it give enormous flexibility in stocking, since they only would need to stock one part, and will reduce inventory costs. Thus, we believe that this architecture not only has the potential to do well in the current market, but its production could result in a significant expansion of that market by making ATM in general a more attractive option to consumers.

2.2 Memory System

The memory system is the key to an ATM switch, as ATM switching is essentially a memory-bound application. The design space for a switch’s memory buffer has two essential dimensions: how the buffer is allocated to cells and input/output ports, and how the buffer is structured to obtain maximum bandwidth. We now will consider each of these dimensions in detail, and show where our switch fits in the design space.

Allocating the Buffer to Cells and Ports

There are several different ways in which the memory buffer can be allocated to ports or cells:

- *Input Queueing:* Memory is divided into queues with one queue allocated per input port. Incoming cells are stored in the queue associated with their input port. This is the simplest but worst performing architecture, as it causes head-of-line blocking, which occurs when two input ports both wish to deliver cells to the same output port: the front cell from one queue collides with the front cell from another queue and as a result it has to wait and so do all the packets behind it in its queue, even if the latter are destined to currently idle outputs.
- *Non-FIFO Input Buffering:* Non-FIFO input buffering (essentially input buffering with random-access queues) provides better performance than input queueing, but it requires a more complicated scheduler due to the fact that the scheduling of each output depends on the scheduling of the other outputs. The output link utilization is not optimal, for some outputs can remain idle because all buffers that contain packets for it are busy with forwarding other packets to other outputs.
- *Output Queueing:* Output queueing, a scheme in which cells are queued at their destination output ports rather than at the inputs, achieves optimal link utilization but requires high-throughput buffers. The reason for the latter is because each buffer is associated with one outgoing link and thus it must be able to accept, in the worst case, cells arriving simultaneously from all inputs.
- *Crosspoint Queueing:* Crosspoint queueing achieves optimal link utilization by using one buffer for each pairing of input and output ports, but has the disadvantage of needing a large number of non-shared buffers.
- *Shared Buffering:* Shared (centralized) buffering uses one large buffer between the input and output ports. All incoming cells are stored in the shared buffer and are read out by their

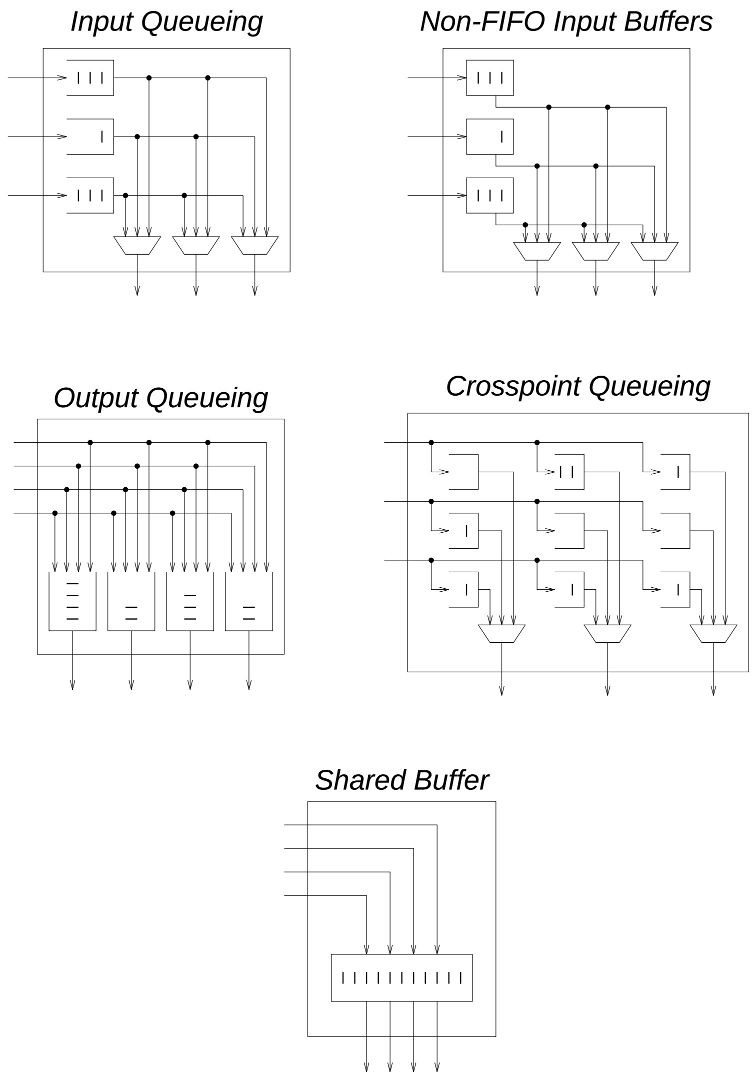

Figure 1: Design Space for Memory Buffer Organization. We chose the shared buffer arrangement for our switch, as it is the most flexible, offers the best utilization, and because our IRAM technology can easily accommodate its bandwidth demands.

associated output ports. The shared buffer must have a throughput at least as much as the aggregate throughput of all incoming and all outgoing links. It also achieves optimal link utilization and in addition, it achieves the best buffer memory utilization of all widely known architectures. In particular, the other four architectures mentioned above are quite wasteful with their memory utilization.

These architectures are depicted graphically in Figure 1.

For our switch, we selected the shared buffer organization, as it clearly provides the best use of memory resources, and our IRAM technology addresses the need for high memory bandwidth very effectively.

Structuring the Memory Buffer for Optimum Performance

There are several possible ways to structure a shared memory buffer to obtain the throughput needed (recall that the memory buffer must support a throughput equal to the aggregate input and output bandwidths of the switch). The most important of these are:

- *Wide Memories:* Wide memories support a “word” size of from 48 to 53 bytes, i.e. the size of an ATM cell. With this organization, an entire cell can be read from or written to memory in one operation. However, this memory must be multiported so that it can be accessed by all ports in parallel. These are difficult and expensive to build.
- *Interleaved Memories:* Interleaved memories spread accesses across multiple banks to compensate for long memory latencies (one bank can be read while others are recovering/readdressing). However, interleaved memories require complex control to take advantage of the interleaving, and are also very large to build, as they require independent addressing circuitry for each bank.
- *Pipelined Memories:* A pipelined memory is like an interleaved memory, except that the address used by a bank is the same as that used by the previous bank in the previous cycle; the pipelined memory thus requires simpler control and only one address decoder for then entire memory.

We chose to use pipelined memory in our ATM switch implementation, as it is simple and well-suited for a VLSI implementation such as ours. Because it requires little fancy control circuitry, pipelined memory costs about the same per bit of storage as an input-buffered memory, while providing much better performance and buffer utilization at the same buffer capacity. Relative to interleaved or wide memories, it simplifies control, reduces input and output datapath, and offers the same performance. In particular it simplifies control relative to the interleaved case, and it significantly reduces the size of the peripheral circuitry relative to the wide memory. Finally, its VLSI implementation is simpler and therefore much faster than the other options due to reduced control and addressing decoder logic. The only potential disadvantage of pipelined memory is that it requires a fixed-size unit of storage; since we are switching ATM cells which are of a fixed length, this disadvantage is moot.

We have designed our pipelined memory to use 64 pipelined DRAM banks. In all of its various configurations, ours is a 32x32 switch; the only difference among configurations is in the bundling of the various ports. To operate efficiently, each of these input and output ports should be able to access the shared buffer simultaneously; hence, the need for 64 banks (32 input plus 32 output ports).

We did look at using off-chip memory such as EDO or RAMBUS for the shared buffer, but we simply do not have enough pins on the chip to give us the necessary memory bandwidth to keep 32 OC-3 output ports supplied with data. Additionally, to support the data rates provided by our switch architecture, we need over 1 gigabyte per second of memory throughput. This is currently impossible with off-chip memory, but even if it were possible we would still need IRAM for the flexibility it gives us in memory organization. Supporting our architecture requires that all 32 input ports be able to write to memory, and 32 output ports be able to read, all simultaneously. In each clock cycle, 384 bits are read from or written to memory. In other words, to use off-chip memory we would need 384 pins for data alone. This is simply not practical. IRAM allows up us to merge the memory and logic to give us a compact, single chip design as well as enormous performance advantages.

2.3 Output Address FIFOs

The next design question that we faced concerns how each output port knows where to look for its next cell. In solving this problem, we found that we still needed to place queues at each output port, much like an output-buffered switch. However, our switch is still primarily a shared-buffer design, and with the other aspects of our architecture we minimized the problems associated with output queues.

Each output port has a FIFO queue. This queue does not contain the full cell; instead, it only contains the addresses in memory of cells which are waiting to use that port. The pipelined architecture eliminates the problem of what happens when multiple inputs try to access the same output: only one input has access to the output address FIFO during each clock cycle, since the input port clockings are staggered to line up with the pipelined memory cycles. The problem of wasted space when an output is left idle for a while is severely reduced because instead of holding a 53-byte cell, this output queue only holds a 17-bit address—a 96% reduction in space.

The length of each output FIFO was an important consideration. This is still a flexible part of our architecture, but currently we believe that it is not necessary that each OC-3 port be able to access the full buffer; in fact, if each of the 32 output ports has access to only half of the memory, that would certainly be more than adequate, and would use significantly less chip area. We thus decided to make each FIFO large enough to hold one-half of the total number of different addresses available, when running as a 32x32 OC-3 switch. Our simulation results (in Section 4) illustrate that we can still handle large traffic spikes despite this reduced queue size. Finally, note that bundled ports bundle their queues together, and thus have access to the full shared buffer.

2.4 VC/VP Lookup Tables

The VC/VP lookup tables, along with the cell buffer, complete the heart of our architecture. The lookup tables contain the complete list of all the virtual circuits currently active, and tabulate which output port is associated with cells belonging to a given VC. The lookup tables are not particularly unusual, except for the fact that we implement them in DRAM; this allows us to make them extremely large, capable of supporting as many as 256K active VCs at once. This is substantially more than the currently-available ATM switches for which we could obtain figures.

The routing control (that is, setting up and tearing down VCs) is not handled on-chip. This is a function which is not needed very often—only every few hours. Therefore we decided it would not be worth the space and complexity required to put these functions on our chip. Instead, we plan to use a separate, inexpensive, general-purpose microprocessor to handle these functions. Since routing changes are relatively infrequent, even a slow, cheap microprocessor like an i960 or i386 would suffice. See Section 3.8 for details on how the microprocessor interfaces with the switch chip.

2.5 Flow Control

The last significant design issue that we faced in formulating our architecture was whether or not to support a sophisticated flow-control scheme. There are a large number of ATM flow control proposals that have been propounded by industry and by researchers. We decided that the strict and complex policies of a flow control scheme such as credit-based flow control could not be suitably integrated into our design philosophy. The reasons for this decision are three-fold. First, credit-based flow control is not suitable for the long-haul links of a wide-area network, our target market. The long round-trip transmission time for the credit cells limits the bandwidth utilization of the switch, as it is forced to stall while waiting for a credit cell to arrive. Secondly, if we were to be implementing credit-based flow control, the large amounts of extra logic circuitry would make an IRAM implementation less appealing. Furthermore, the necessary per-VC queueing of credit-based flow control would prevent us from utilizing our shared, pipelined memory system. By pushing flow-control over to software, we are better able to exploit the memory-focused advantages of IRAM-based design. Additionally, the ATM has selected other flow-control methodologies as the ATM standard. While we do not accept the ATM Forum's word as "technological gospel," the fact remains that their opinion is highly representative of our target market. The flow-control scheme endorsed by the ATM forum is the Available Bit Rate (ABR) rate-based flow control, and while we have not fleshed out the a detailed implementation of ABR, we have left it open as something that we could add. We already have a general-purpose microprocessor which we expect will be idle the vast majority of the time. We could, without much difficulty, use this spare processing power to implement the new ABR flow control standard.

Augmenting our microprocessor interface to transmit the needed cells and FIFO lengths should be relatively trivial. Whether ABR is used could also be a user or retailer-configurable option, based on whether the consumer felt flow control was important for their purposes.

Once again, the main reason we have left flow control as an option—and one to be handled off-chip at that—is that the market for which we are primarily aiming (WAN backbones) is not one for which strict flow control is as critical. The long distances involved introduce a lot of latency. One mile of cable, for example, introduces 5 microseconds of latency, and we expect our chip could be running with cable segments longer than a mile. With this kind of delay with sending usage information, it is far more important to be able to handle flow spikes than to aggressively implement flow control. And as mentioned earlier, our switch is better equipped than any to handle spikes of data.

3 Architectural Details

The switch we have designed is in most respects a 32x32 port fully-interconnected output-buffered switch capable of operating at OC-3, or 155 Mb/sec/link. The switch has 35.5 Mbits of buffer space for cells (85 kCells of space), 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 16x16 OC-6, 8x8 OC-12, or 4x4 OC-24 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 block diagram of our switching chip can be found in Figure 2.

3.1 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 (for example, PMC-Sierra's PM5348 S/UNI-DUAL and PM5355 S-UNI 622 circuitry). 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 mentioned above), 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. If the chip is to be used in a 16x16 configuration OC-6, each port will be allocated 8 pins. Likewise an OC-3 port will require 4 pins.

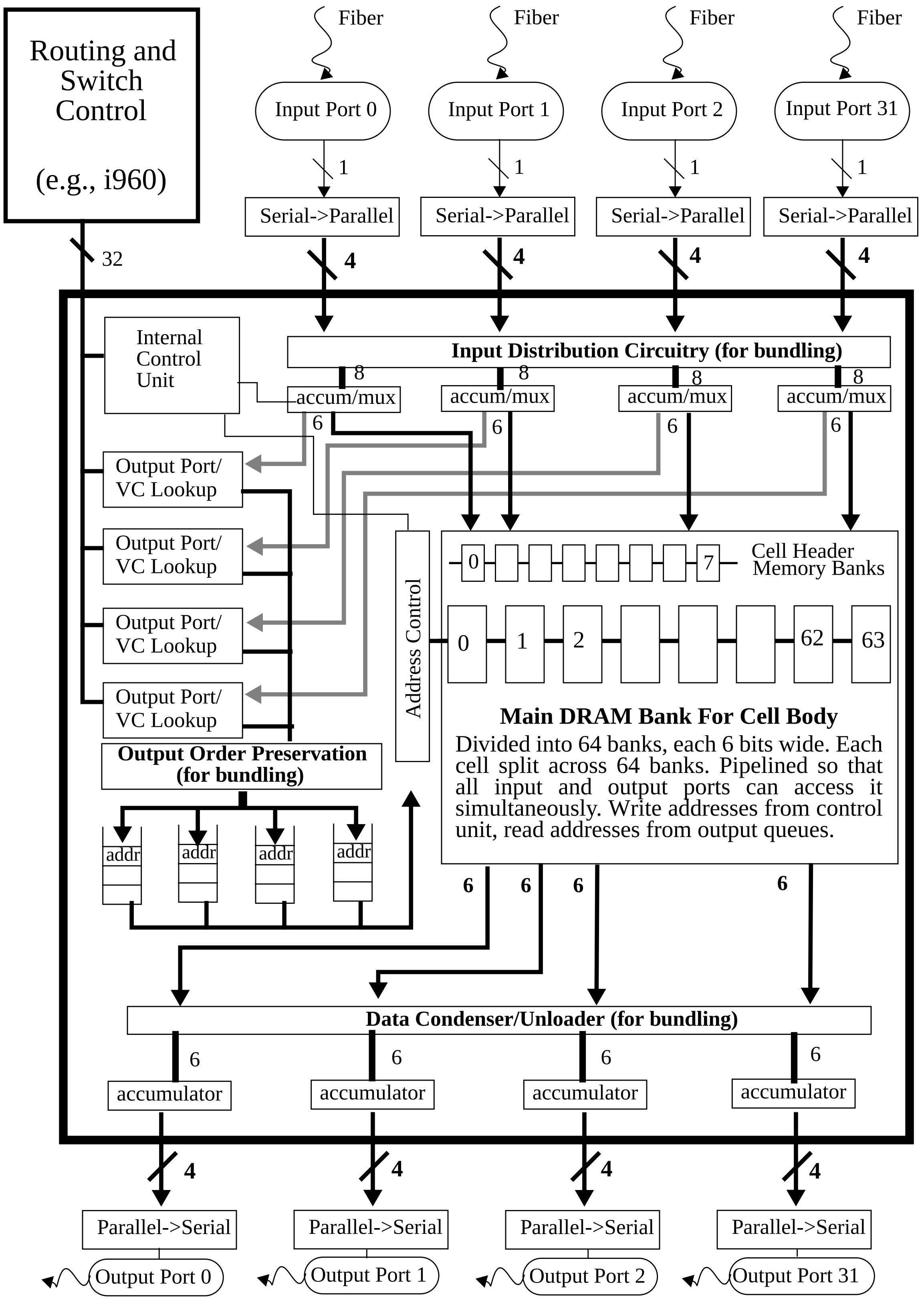

Figure 2: Block diagram of single-chip ATM switch. The diagram shows a switch with 32 full-duplex ports.

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. As will be explained below, 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 conversion 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.

3.2 Cell Headers: Routing Lookup, Handling, and Storage

Now that we have described how the data payloads of ATM cells are stored in the memory buffer, we must proceed to 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 as described in Section 3.8. 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 2^{24}, or 16M, possible VCI/VPI identifiers. Keeping a table of 2^{24} 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 (or perhaps a circular buffer). 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.

The cycle-level timing of cell processing will proceed as follows (each cycle refers to an internal 30nS clock cycle):

- *Cycles 1–2:* First and second 6-bit words of cell header accumulated in SRAM buffer
- *Cycle 5:* Fifth 6-bit word of cell header (and all VPI/VCI info) received
- *Cycle 6:* Look-up table referenced with VPI/VCI info and header changed as appropriate.
- *Cycle 7:* Cell memory address written to appropriate output port or cell dropped if no space available.
- *Cycle 8:* Last 6-bit word from header received
- *Cycle 9:* Cell payload starts being written to payload buffer *and* cell header starts being written to header buffer.
- *Cycle 10:* Cell ready to begin output.
- *Cycle 16:* Header completely written to memory.
- *Cycle 72:* Payload completely written to memory.

As can be seen from the above timings, it is not necessary for the switch to have received a particular cell in its entirety before it can be output. This phenomenon, known as cut-through, can substantially reduce the latency of our switch. Our methods of managing cut-through will be discussed in more detail below. Also, since our switch does not implement strict flow-control policies (such as credit-based flow control), it is possible that incoming cells may have to be dropped if no space is available in memory. As can be seen above, the switch can determine whether the cell should be dropped before it is written to memory, so no stray memory space will be lost to cells unaccounted for. A further discussion of cell dropping and its probability is included in Section 4.

3.3 Memory System Organization

As described above, we have decided to organize our memory in a pipelined manner. The pipelined memory is like an interleaved memory where the address used by a bank is the same as that used by the previous bank in the previous cycle. Using the same address for subsequent banks in subsequent cycles simplifies control relative to the interleaved case, makes the VLSI implementation faster than wide memory, and significantly reduces the size of the peripheral circuitry relative to the wide memory.

Figure 3 shows our shared-buffer switch, using pipelined memory. The number of pipeline stages is equal to the number of incoming plus outgoing links (64). As a cell arrives, it is written into the corresponding input buffer registers, one word at a time. During some cycle after the arrival of the first word, W_0, of this cell, and before the arrival of the first word of the next cell, W_0 is written into the first memory stage, M_0. In the sixty-three subsequent cycles, the next sixty-three words of the cell will be written into the next sixty-three memory stages (M_1, M_2, . . . , M_{63}) at the rate of one word per cycle. All words of the same cell are written into the same address in each of the memory stages. Because the tail of the cell may be written into the last memory stage after the head of a new cell has entered into the first input buffer register, there is no need for double buffering of the input.

Output operation is as follows. The first word of the outgoing cell is read out of M_0 and placed in the leftmost output buffer register. In the next cycle, this register drives the desired outgoing link and the next word of this cell is read out of M_1, and placed into the corresponding output register. The wave of reading from the memory stages and transmitting successive words proceeds left to right, at the rate of one word per cycle. Only one row of output buffer registers, shared among all the links, is used on the output side (unlike on the input side). This imposes the restriction that no two outgoing links can start sending out cells in the same cycle; by staggering the output port start-transmission times this restriction is easily met.

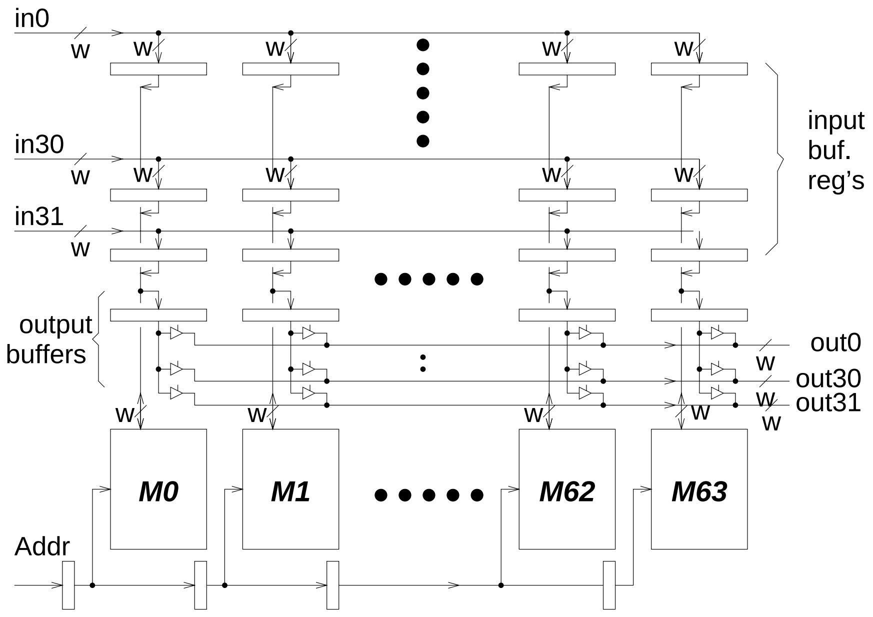

Figure 3: Organization of Shared Buffer as a Pipelined Memory. This figure shows the details of the pipelined memory organization for our shared memory buffer. *w* is the ATM word size, 6 bits in our 32x32 OC-3 switch. Cells arrive on lines In_x, are stored in memory banks M_i, and leave on lines out_y.

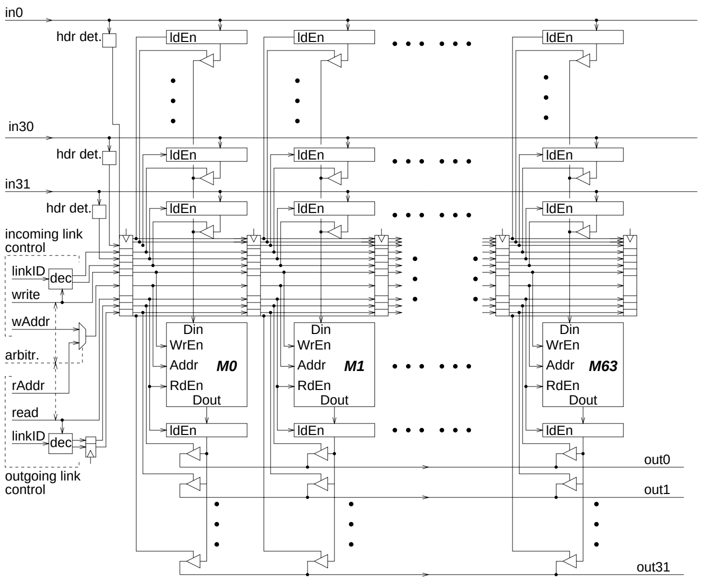

Figure 4: Control Circuitry for Pipelined Memory. Notice that the bulk of the control can be set up once for the first bank and then propagated from bank to bank along with the memory address.

Control of the pipelined memory and associated I/O circuits is simple and straightforward. In every cycle a new read or write operation can start from M_0. Each pipeline stage performs exactly the same operation as the previous stage in the previous cycle, and thus we only need to generate the control signals for the first memory stage. Figure 4 shows a possible implementation of the memory control. We can see that the control signals for stages $M_1, \ldots, M_{30}, M_{63}$ are identical to those for stage M_0, albeit delayed by the appropriate number of clock cycles. We can easily see that the circuits that provide these signals—in particular the buffer (address) management circuits—are independent of the pipelined memory.

Finally, Figure 5 illustrates how the pipelined memory can be implemented in VLSI such that only one address decode unit is necessary: as can be seen in this figure, the arrays (banks of our memory) touch each other, and thus the decoded address can be transferred from one stage to the next directly, without being re-decoded. As a result our memory has only one address decoder and several stages of flip-flops, which are small, fast circuits in comparison to the address decoder.

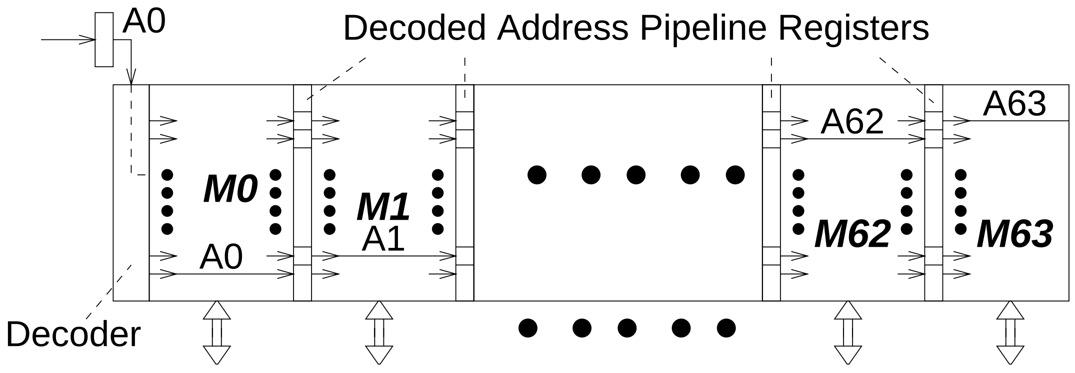

Figure 5: VLSI Implementation of Pipelined Memory. Notice that the address decode must be done only once; by placing the memory banks next to each other, we can pass the decoded address from bank to bank without having to duplicate the decode logic on each bank.

3.4 Memory System Details: Size and Refresh

The following table summarizes the memory usage of the switch:

Part of Switch	**Memory Contents**	**Memory Usage**
Payload Buffer	85k cells @ 48 bytes each	32 Mbits
Header Buffer	85k cells @ 40 bits each	3.3 Mbits
Lookup Tables	256k entries @ (24+5) bits each	7.2 Mbits
Output Address FIFOs	32 FIFOs with 42k entries @ 17 bits each	22 Mbits
Free Address FIFO	85k entries @ 17 bits each	1.4 Mbits
	Total Memory Usage	**65.9 Mbits**

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.

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 relied 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 pre-determined 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.

For the header and payload memory, it can be observed that a cell (at OC-3 rates) takes 72 internal (30ns) cycles to enter a switch. Thus on any given input or output port, a new cell will be initiated through the 6-bit bus every 72 cycles. It can also be noted that with 64 total ports (32 input, 32 output), our pipelined memory will only be in use 64 out of those 72 total cycles. Thus for every 72 cycles, there are 8 cycles in which our memory resource is not in use. Thus with appropriate scheduling, every 9th cycle can be used to send a “refresh bubble” down the pipelined memory. Therefore, the needed retention time for our DRAM cell in order to be able to cycle through all 85k cell entries is approximately 25 msec[1]. One could argue that there is no need to refresh at all, as we are virtually guaranteed that a particular cell will not be delayed in a single switch for nearly that long. Still, in the interests of robustness, our buffer memory will be refreshed, and it will not cause any internal stalls.

The VPI/VCI lookup tables must be refreshed, as they are to exist in a particular switch for long periods of time. Still, it is noted that refreshing this component is quite trivial. The lookup table for each port can be referenced independently, and of the 72 internal cycles comprising a single cell, a particular lookup table is required for fewer than five (allowing for such activities as connection establishment and teardown). Thus, refreshing all 8k entries of a particular look-up table can occur in under one millisecond. Even when ports are to be bundled together—and thus look-up tables shared—the required retention time for such a setup is under 3 msec. Similarly, one can refresh the output address FIFO’s in approximately 5 msec.

Similar to the cell buffer, the free address FIFO is utilizing 64 out of the 72 internal cycles per cell. If we use a simple 1-T DRAM cell structure (with appropriate SRAM buffering at either end), the necessary retention time is 25 msec. As this FIFO can remain unchanged and unread for long periods of time (in idle time), such a retention time requirement may undesirable if our process is highly susceptible to soft errors. In this case, we can construct this FIFO from a naturally dual-ported 3-T DRAM cell. Using the 3-T, on idle cycles when no cells arrive (and thus no free address is required), the FIFO can read the address at the queue and simultaneously write it to the end of the queue. Since it is of no consequence what address is used—as long as it is a free one—this reordering of free addresses has no consequences on performance. The necessary retention time then drops to approximately 5 msec. While the 3-T approach requires more chip area, the relatively small size of the free address FIFO makes such a modification relatively inexpensive.

3.5 Bundling Ports to Service Higher Data Rates

One of the most novel features of our design is the fact that our chip can be configured to service data rates higher than OC-3. In fact, our switch can be configured to “bundle” several OC-3 ports together in order to function at a higher data rate (i.e. two OC-3 ports can be bundled into an OC-6 port, four into an OC-12, and eight into an OC-24). With bundling, the fully-interconnected topology of our switch is not lost. Furthermore, it is not required by our switch that all the “virtual ports” seen by the external interfaces be of the same data rate. For instance our 32x32 OC-3 switch can be configured to act like a fully-interconnected switch with one OC-24 port and six OC-12 ports.

The concept of bundling is quite simple. Since *n* distinct OC-3 ports can service an aggregate throughput of *n*OC3_data_rate*, these same *n* ports used together have the throughput capability to service a single OC-<3**n*> data line. It was noticed, however, that simply using these *n* OC-3 data lines in parallel (i.e. byte interleaved) to form a wider internal data bus would require major architectural modifications to our core 32x32 switch. Instead, it was observed that bundling could be handled quite simply if it was implemented in a cell-interleaved fashion, as has been done in the ATLAS ATM switch [1]. For instance, suppose 4 OC-3 links are bundled as a single OC-12 link. In this case, 4 cells will arrive over the OC-12 link in the time each OC-3 link can process one cell. The first cell on the incoming OC-12 link will

1. Commodity DRAM parts are rated to have retention times of between 32 and 64 milliseconds, so our requirement is easily realizable.

be sent down the first OC-3 link; the second cell down the second link; and so on. By the time the fifth OC-12 cell arrives, the first link will have processed the very first cell, and it will be ready to process this fifth cell. Similar modes of operation are in effect at each output port.

The architecture for the input distribution circuitry is quite simple (see Figure 6). A series of flip-flopping demux's with some small amount of buffering can guarantee an optimally non-blocking in order delivery of incoming cells to each of the bundled OC-3 ports. After a cell passes through a particular demux in the input distribution circuitry, the state of that demux is flipped such that the next incoming cell will be routed through the other demux output. Thus, the incoming cells will be even distributed amongst the bundled OC-3 ports. Furthermore, as the cells arrive in order, they are guaranteed to exit in order. Since the pipelined memory and look-up tables service each port in order, we are guaranteed that the cells will arrive at the bundled output ports in chronological order.

The bundled input ports share each other's VCI/VPI look-up tables, thus preserving an aggregate of 256k possible VCI/VPI combinations. Even with the shared lookup tables, resource contention is not an issue as the incoming cells are staggered amongst the bundled ports.

The architecture for bundling on the output port "side of the world" is similar to the input distribution. It can be separated into two distinct parts, dubbed the Output Order Preservation circuitry and the Condensing/Unloading circuitry. The condensing/unloading circuitry is essentially a serial-to-parallel accumulator that accepts the several incoming OC-3 data streams, concatenates the data, and outputs it as a single faster data stream.

The condenser sequentially takes memory addresses in sequential order from the output address FIFOs to access its data streams. Without prior precaution, however, this operation could result in out-of-order cell delivery. Let's imagine that we are bundling our ports in sets of two, so that input ports 1 and 2 function as aggregate port *A*, input ports 3 and 4 function as aggregate port *B*, and input ports 5 and 6 are bundled as aggregate port *C*. Let's assume both these input ports are routing cells (thus sending memory addresses) to aggregate output port *D* (output ports 11 an 12). The look-up tables are setup such that input ports 1, 3, and 5 route to output port 11, and input ports 2, 4, and 6 route to output port 12.

So, let us assume that in the first round of port accesses, input port *A* sends two cells, a_1 and a_2, through the switch to output port *D*. The address of a1 is written in output port 11's address fifo, and the address of a_2 is written in output port 12's FIFO. So far everything is OK, as the unloader, when granted memory access, will fetch a_1 followed by a_2.

While still in this first round of port accesses, input port *B* has only 1 cell, b_1, to send to port *D*. The cell is routed through input port 3 by the input distribution circuitry, and the table lookup dictates that the address of b_1 should be written to output port 11 (behind a_1).

Next, input port *C* is granted port access. It has two cells, c_1 and c_2, to route to output port *D*. As it is unaware of the actions of input ports *A* and *B* (input ports do not communicate with one another), and nothing has been sent across input port *C* earlier, it routes c_1 to input port 5 and c_2 to input port 6. According to the lookup table's entries, the address of c_1 is then written to output port 11 (behind b_1), and the address of c_2 is written to output port 12 (behind a_2).

Now, when we examine the actions of the unloader, we realize that cells will be delivered out of order. During its first round of unloading, everything will be fine as output port *D* will deliver cells a_1 and a_2. In the next round, output port *D* will deliver cells b_1 and c_2. In the third round, output port *D* will deliver c_1. Thus cell c_1 is delivered *after* c_2. Therefore a problem arises, as ATM switches are not supposed to deliver cells out of order.

Thus the need for the circuitry we call the output order preservation circuitry. This circuitry is also a nested set of demux's that guarantee, no matter which of the bundled ports a cell address arrives at, it is routed in order to the next available bundled address FIFO. It works as follows. Every time an address passes through one of the demux's, that demux and all of the demux's it is tied to flip to the other routing path. Since the input distribution circuitry guarantees that in-order cells are delivered in-order to the outputs, the output order preservation can guarantee that these cell addresses are ordered correctly in the address FIFOs. Thus, the unloading circuitry will deliver the cells in order to the next switch downstream. Note that the output ordering circuitry only passes cell addresses and not actual cell data. These addresses are only 17-bits and can traverse these layers of demux's in one cycle without any buffering or latency. (See Figure 6 for the overall layout of the bundling circuitry.)

Furthermore, this single set of bundling circuitry can be configured to handle any 2^n link bundling. As can be seen by comparing the two parts of Figure 6, a 4-to-1 bundling setup can easily be configured as two 2-to-1 bundling setups. In order to do so, only the following changes must be made: in the input distribution circuitry, the top-level demux becomes transparent, so that the left-most two data paths always traverse the left-hand data route, and the right-most two data paths always traverse the right-hand data route. In this way, each set of two incoming data paths can be treated as an individual 2-to-1 bundled link. In the output ordering circuitry, the state-preserving control line of connecting the top set of demux's must be broken in order to allow each set of two demux's to switch independently of the other two. Also, the second row of demux's are made transparent, without any switching mechanism. Lastly, the 4x unloading circuitry is "split down the middle" into two 2x unloading blocks. This can be done trivially as the unloading blocks are merely serial-to-parallel accumulators. Thus, these four OC-3 ports can easily be changed from a single 4-to-1 bundled link to two 2-to-1 bundled links. Similar modifications can be made to configure them as separate, unbundled ports, or bundling 8 of these ports together in an 8-to-1 fashion. (The 8-to-

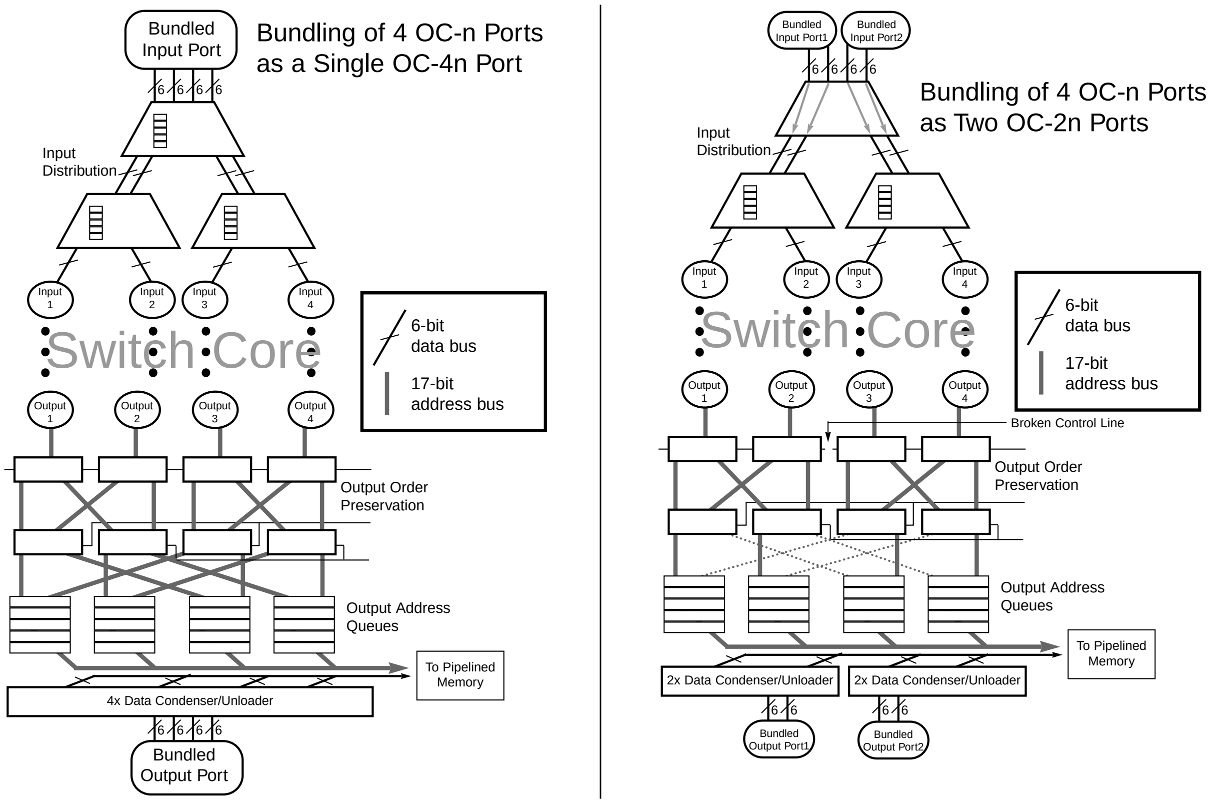

Figure 6: Configurations of Bundling Logic for Two-Way and Four-Way Cell-level Bundling. Notice that the two-way bundling (right) can be built from the four-way bundling (left) by breaking up the condenser/unloader and some control latches.

1 bundling is not shown here for the sake of easy graphical representation.) The total additional latency induced by such bundling operations is 1/2 of an OC-3 cell time (an OC-3 cell time is about 2.72 µs) for 2-to-1 bundling, 3/4 of an OC-3 cell time for 4-to-1 bundling, and 7/8 of an OC-3 cell time for 8-to-1 bundling.

3.6 Cut-Through and Latency Minimization

When the switch is under heavy traffic loads, the latency of a single cell is a function of the output port's buffer utilization. Since output port blocking is a phenomenon that one cannot work around, the latency of a cell through a busy switch is one that cannot easily be reduced through engineering design.

When the switch is under lighter traffic loads, latency is something that can be worked around. Cutting down on latency is important in our switch, because all the latencies of our switch are in reference to OC-3 data rates. Thus, cutting an additional OC-3 cell time (2.72 microseconds) of latency is equivalent to removing 8 cell times of latency while running at OC-24 data rates. In particular, if several ports are idle, and an incoming cell is routed to one of those idle ports, that particular output port should be given immediate access to the data and allow the cell data to cut-through. It should not have to wait for its pre-assigned access slot. In other words, under low utilization, a cell should not have to be entirely written to memory before it can be output to the next switch downstream. As can be seen in cell-timing chart in Section 3.2, while a cell takes 72 clock cycles to be completely written/read to/from memory, it can start being output as early as the tenth cycle. Our pipelined memory has some control logic to allow it to detect the presence of idle ports, and it will dynamically re-assign the port access order to allow for cut-through operation. By the law of averages, cut-through will save one-half of an OC-3 cell time of latency, or approximately 1.36 microseconds during low utilization. It is recognized that cut-through is of no benefit during high-traffic scenarios, but it has significant effects during low utilization, especially for bundled ports servicing higher data rates.

One might be concerned that cut-through and the dynamic re-ordering of port accesses might induce out-of-order cell delivery. This is not the case. Clearly, during unbundled operation, each port is independent of one another, and the relative ordering between them is inconsequential. During bundled operation it is not so clear that port re-ordering will not occur. Due to the nature of the bundling circuitry, however, it is impossible for one bundled output port to cut-through and end up "in front" of another port with which it is bundled. Thus, while non-bundled ports may change order relative to one another, bundled ports by nature stay in relative order, so all cells are delivered in order, even in cut-through situations.

3.7 Scalability of the Architecture

One nice feature of the single-chip ATM switch is that it can be easily cascaded into larger switches without having to purchase all of additional expensive support circuitry. For instance, four of our switches, running in 8x8 OC-12 mode, can easily be configured on a single board to function as a 16x16 OC-12 switch.

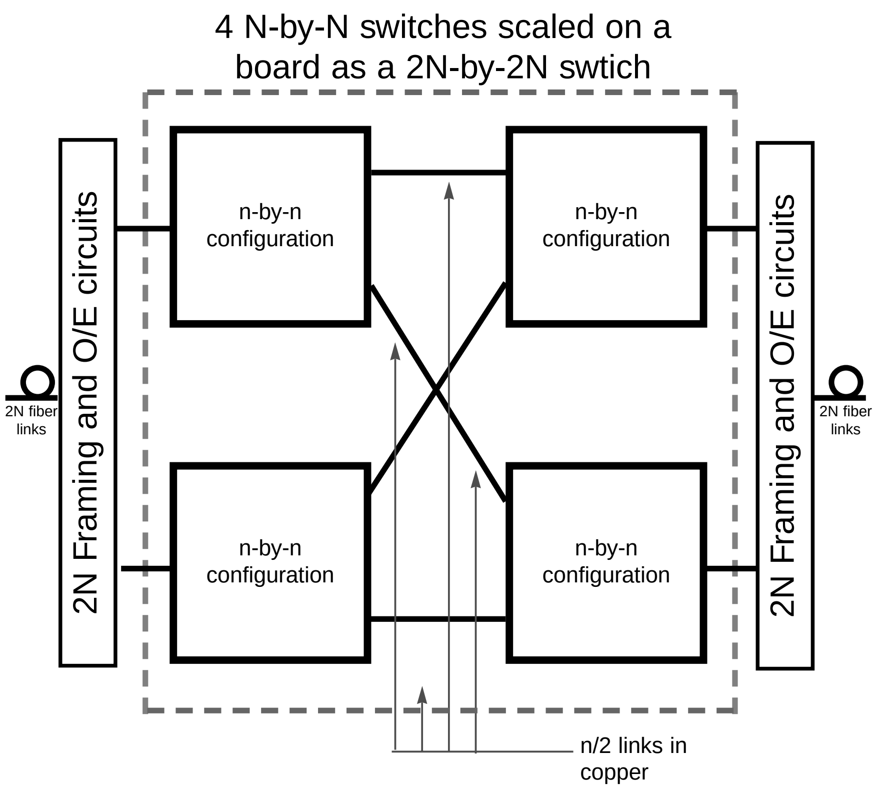

Figure 7: Four *N*x*N* Switches Scaled on a Board as a 2*N*x2*N* switch. This architecture approaches fully-interconnected behavior at a reduced cost, as less support circuitry is needed.

As can be seen in the Figure 7, two switches in parallel comprise the 16 input ports, and the other two switches in parallel comprise the 16 output ports. The internal layer of ports are hidden to the outside user, and are run in metal on a PC board, thus removing the need for the expensive framing or opto-electrical hardware. For each input switch, four of the output ports are connected to four of the input ports of the top output switch, and the other four output ports are connected to four of the input ports of the bottom output switch. Furthermore, using the output ordering circuitry from the bundling modifications, outgoing cells can be optimally distributed among these bundles of four links such that the overall performance of the chip-set comes close to mimicking full-interconnect behavior.

3.8 Microprocessor Interface

General Supported Operations

The internal control unit provides the interface between our chip and the external processor. This processor will be used to load our routing tables and to control the switch operation; it can also be used to support ABR flow control. In order to handle all these tasks, the processor must have the ability to write our routing tables, to receive cells sent by the neighbors of our switch, and to send some cells to these neighbors (these cells might contain flow control data or information about the network, needed in order to set up connections correctly). As a result, our the interface must support three operations:

- Receive a cell from the processor and write it to the internal memories so as to be ready to be transmitted in the network
- Read a cell and send it to the processor
- Write a particular entry in the VC/VP lookup tables.

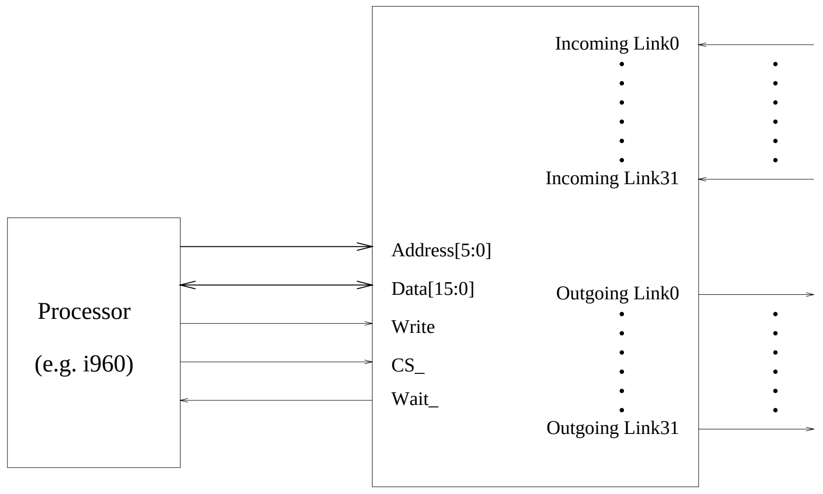

Figure 8: General Overview of Microprocessor Interface.

The Interface

The general interface (shown in Figure 8) consists of three control signals and two buses. The control signals are: *Write/Read_*, *CS_*, and *Wait_*, while the buses are the data bus (16 bits wide) and the address bus (6 bits wide). The *CS_* signal is connected point-to-point to the processor, while all the others are shared among all the switches which are controlled by the same processor. The reason for using this point-to-point connection is that we want a single processor to be able to control an arbitrary number of our switches in order to aid scalability. But if we use another “address” bus (or an extension of the address bus we have now) in order to choose which switch is selected for each operation, it is obvious that the number of the switches that can be connected to our processor would be limited by the width of this additional “address” bus. The point-to-point chip-select signal can be decoded from the processor bus, however, so we do not require *n* direct connections to the CPU when it is controlling *n* switches.

Although we have to address 32 tables of 8K entries each, we use only 6 address lines because we do the following trick: in the address bus the processor specifies just the table to which it wants to write (since we have 32 tables we just need 5 bits). The actual address where the data must be written is transmitted on the data bus, before the actual data (in other words we multiplex the address and the data so as to minimized the required address lines (and of course address pins)). The sixth bit is used in order to distinguish which of the 3 operations we have. A 1 represents a cell operation (one of the first two listed above), and a 0 represents a lookup table operation (the third listed above).

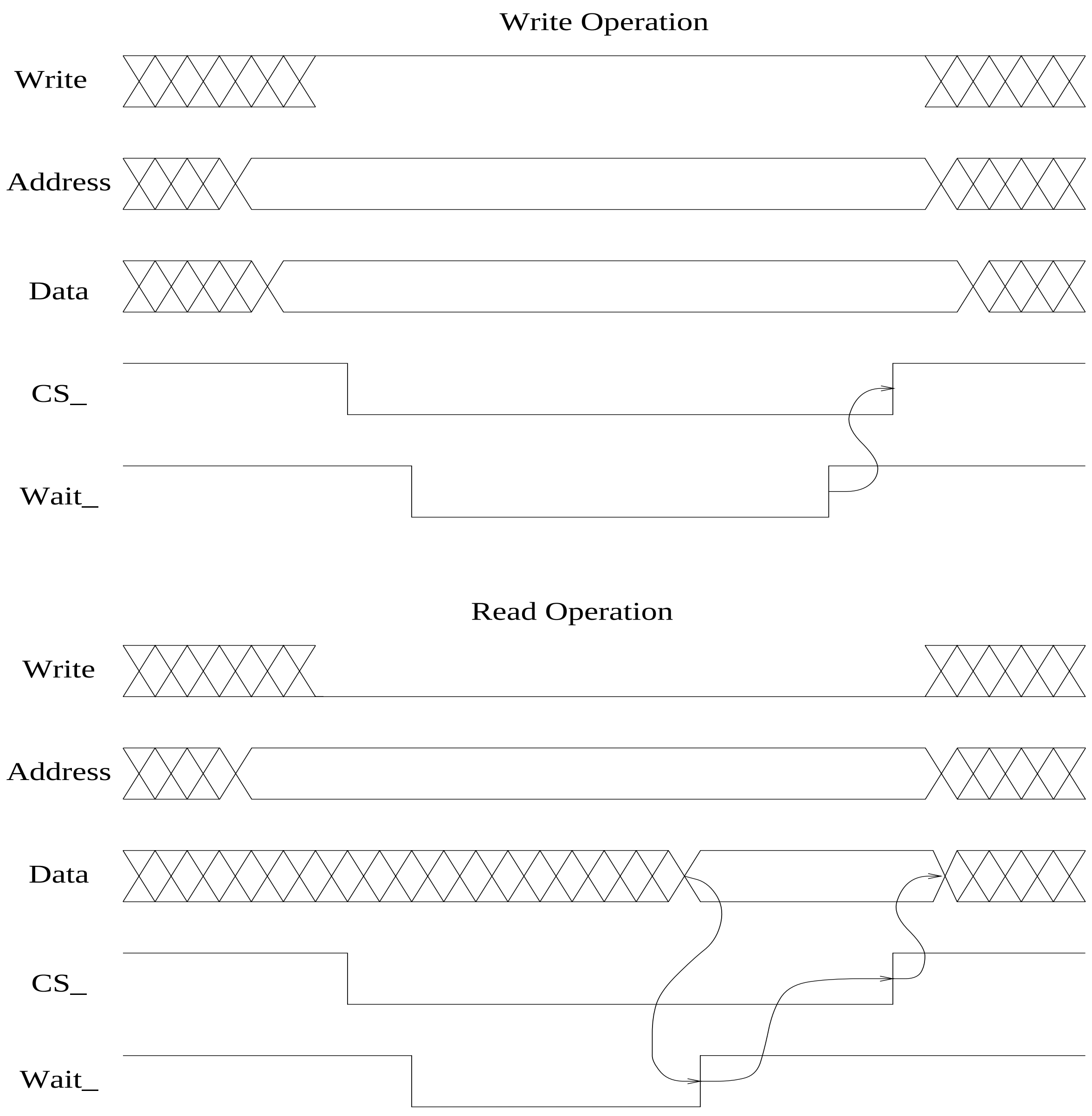

Figure 9: Timing for a General Read or Write Operation on the CPU Interface Bus.

According to our protocol a general read or write operation can be done as shown in Figure 9. When we have a write operation, the processor places the address and data on the corresponding buses and asserts the *Write* and the *CS_* signals (consider that *CS_* is an active low signal). Our internal control unit asserts the *Wait_* signal and when the data are “consumed” it deasserts it. This cause the deassertion of the *CS_* signal from the processor. When we say that the data are consumed we mean that either the actual write is done, or the address and the data are written to some temporary registers (see below for details). Of course in the case of the first operation, receiving a cell, the processor instead of writing a whole address in the address bus, just turns the 6th bit of the address into 1. In the case of reading cells, the processor does not provide us with any useful address. Thus in a read operation, the processor turns the 6th bit of the address into 1 (showing that we a have a cell operation), turns the *Write/Read_* signal to 0, tristates the data bus, and asserts the *CS_* signal. Our internal control unit asserts the *Wait_* signal and, when the necessary data are read, it drives the data bus and it deasserts the *Wait_* signal. If there are no data to be written, our

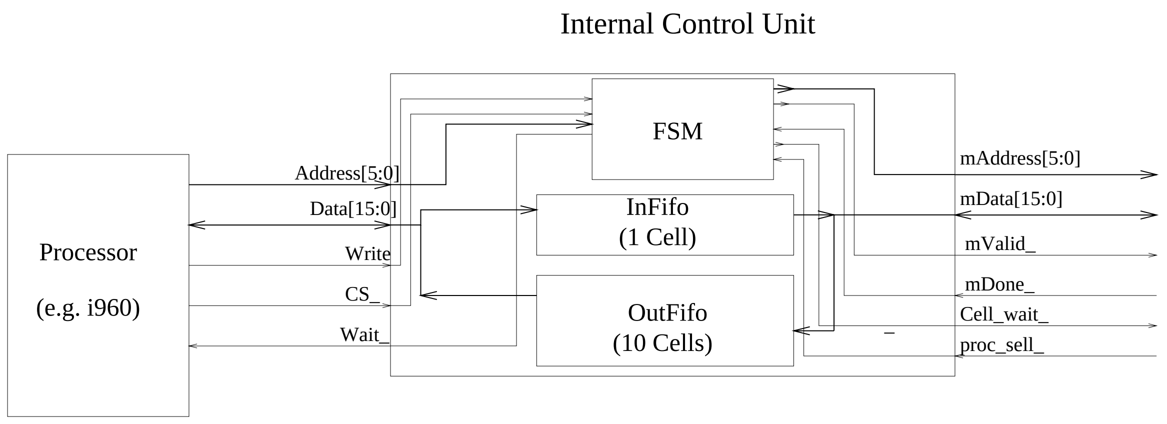

Figure 10: The Internal Bus Interface Control Unit.

control simply deasserts the *Wait_* signal immediately. The deassertion of the *Wait_* signal causes the deassertion of the *CS_* signal from the processor.

Internal Operations

Next we consider how we implement the bus operations internally. First of all there are the 4 control signals and the 2 buses shown in Figure 10. The width of the address bus is just 6 bits because we use the following trick when we want to write something in the lookup tables (this is an extension of the trick described above): we introduce one address and one data register for every one of our lookup tables, and we map one address to each of these pairs. As a result we have just 32 of these pairs, and so we need 5 bits to address them. Now let’s trace a writing operation to the actual table. We first write the corresponding address to the address register and the corresponding data to the data register and then we give a signal to an FSM to perform the actual write operation in the first idle DRAM cycle (see Figure 11). These writes are done by simply placing the address of the corresponding pairs in the address bus, turning the most significant bit of the address to 0 (indicating that we want to “communicate” with the VP/VC lookup tables) and asserting the *write* and the *mValid_* signal. After that, the FSM, which is next to the corresponding

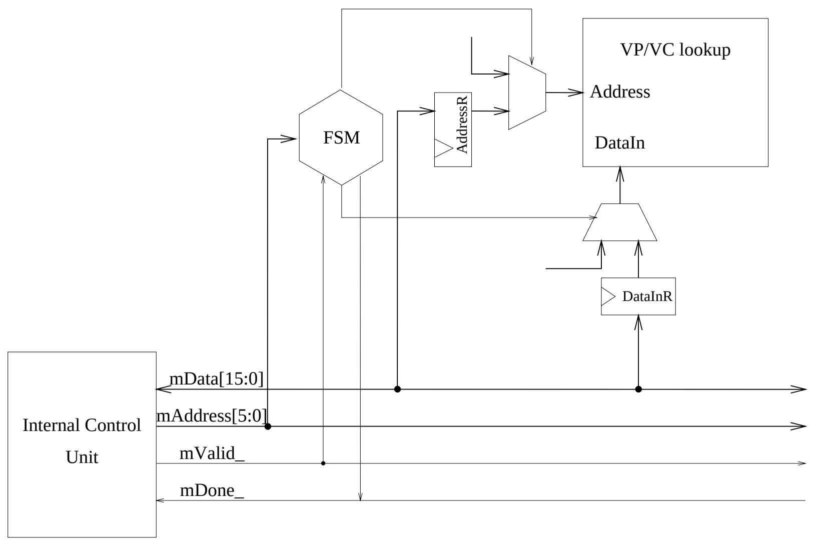

Figure 11: Connections between Internal Control Unit and VC/VP Lookup Tables.

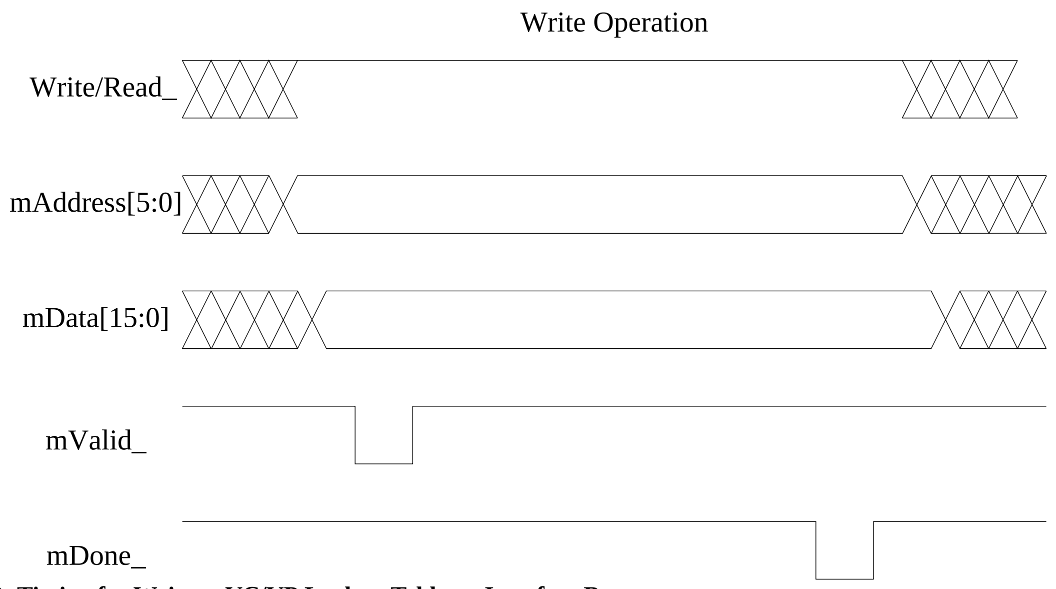

Figure 12: Timing for Write to VC/VP Lookup Table on Interface Bus.

table, decodes the address and sets the load signals of the address and data registers. When there is an idle cycle, this FSM sets the signal of the multiplexors shown in Figure 11 and it also asserts the *mDone_* signal. In Figure 12 we can see the actual timing of all the signals mentioned above.

When we want to write a whole cell to the switch so as to be transmitted in one of the following cycles, we do the following. First, the incoming cell is written into a temporary buffer which has a size of one cell. Then the internal control unit asserts the *cell_wait_* signal so as to inform the scheduler that there is a cell which is waiting. Then in the next idle cycle of the pipelined memory, we will start writing the cell into this memory. At that time we also deassert the *Wait_* signal (see above), which enables the processor to do another operation. Of course, after that the cell will be transmitted into the corresponding output link just as ordinary data cells are transmitted.

In order for a cell to be sent to the processor we do the following: we “dedicate” one VP/VC address for all the cells that must be sent to the processor. Let us assume that this address is 0xFFFFFF. When a cell arrives in the switch and at the time we do the VP/VC lookup, we also check this field to see if it is equal to 0xFFFFF. If this is true, we write the cell into the output buffer of the internal control unit instead of writing it into the pipelined memory. This actual writing begins when the *proc_cell_* signal is asserted. After that, the internal control unit waits for a read operation from the processor in order to send this cell to the processor via the “read” operation described above. The size of the output buffer is 10 cells. As a result if there are 10 cells waiting to be transmitted to the processor, and another cell arrives for the processor, that cell must be dropped. However, since we expect that there will be few of these cells, the probability of dropping one is small.

4 Simulation Results

In this section we present results from a cell-level switch simulation. These results illustrate the performance advantages of using a large DRAM memory buffer, one of the most significant and unusual features of our switch.

4.1 Motivation for Simulation

In our single-chip ATM switch, the total shared memory size for all ports is 85,000 cells. This amount of inexpensive on-chip and high-bandwidth memory is the most valuable advantage of our IRAM-based ATM switch design, and can be attained because we are using a DRAM process to build our ATM switch and thus can benefit from a large amount of memory with huge internal memory bandwidth at low cost. In contrast, due to the fact that high-speed memory (SRAM) is large and expensive, most conventional ATM switches which are made in logic processes use only a small amount of high-speed memory for their switch buffers. This either results in limited switching performance (if the memory is small) or high cost (if the memory is large).

The reason why larger memory buffers in an ATM switch produce better performance is that with a larger buffer, an ATM switch can hold more cells in the buffer when network congestion occurs and thus can release them into the network later (when network congestion lessens) without losing any cells. On the other hand, if an ATM switch has only a small amount of buffer space, when network congestion occurs it may have to drop some cells due to buffer overflow. This can have negative effects on the performance of higher-layer transport protocols such as TCP: when a cell is dropped, TCP may be forced to wait for a long time to timeout its transmissions, to do retransmission, and to reenter its inefficient slow-start phase. To make matters even worse, since an IP packet needs to be first chopped into many ATM cells before it can be sent onto the ATM network, dropping one ATM cell makes all other cells belonging to the same packet useless when they finally arrive at the destination host. Thus dropping a cell not only causes TCP to timeout and enter inefficient slow-start phase, it also wastes network bandwidth to forward useless cells across multiple hops until they reach their destination.

To examine the likelihood of encountering the performance penalties that result from dropped cells, we wrote a cell-level ATM switch simulator to examine the performance of our large memory buffer on some simple traffic workloads; the results illustrate how our large memory buffer can enhance network bandwidth utilization, thus showing the advantage of our IRAM-Based ATM switch design.

4.2 Simulation Setup

We use a Poisson process with some mean cell sending rate to represent a VC's traffic pattern. Although more recent research shows that network traffic has self-similar characteristics and can not be totally and accurately represented by Poisson process, in some time scales, Poisson process can still match real traffic pattern quite well [2], so we decide to use it as a VC's traffic pattern.

Because our ATM switch can support 32 OC-3 ports, in the experiment we create 31 VCs coming from 31 different input ports and going out to the same output port. The reason why we only create 31 rather than 32 VCs is that in normal usage, it is unusual to have a VC coming from and leaving out a switch via the same port. The result that we are interested in is the buffer overflow time as a function of the offered Poisson load. By offered load, we mean the aggregate load coming from all of these 31 VCs; it is represented as a number ranging from 1 to 31. In each simulation case, if the offered load is specified as *n*, we let each of these 31 VCs carry Poisson load with mean cell sending rate being *n*/31.

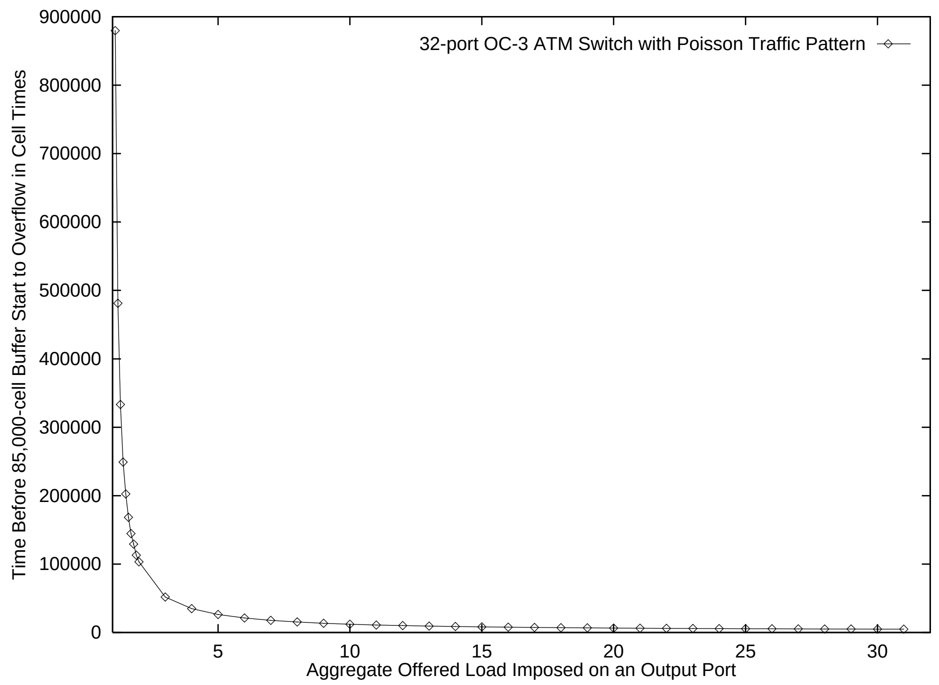

Figure 13: Time to Fill Memory Buffer as a Function of Aggregate Offered Load. This plot depicts the time for the 85,000 cell internal DRAM buffer to fill as a function of a simulated Poisson traffic load. The offered load is the aggregate load from 31 VCs all destined to one output port.

4.3 Simulation Results

The simulated results are graphed in Figure 13. The table below presents some of the more interesting results:

Offered Load	Time to Fill Buffer (OC-3 Cell Times)
1.0	∞
1.1	879,679 (2.4 s)
1.2	481,248 (1.3 s)
1.5	202,419 (552 ms)
1.8	129,149 (352 ms)
2.0	103,318 (281 ms)
7.0	17,679 (48 ms)
31.0	4,849 (13 ms)

As an OC-3 link can transfer roughly 366,796 cells per second, when the offered load is 1.1, our switch can sustain roughly 2.4 seconds before buffer overflows, which is pretty long in terms of network condition. When offered load gradually increase from 1 to 31, the time before buffer overflows gradually decreases, as predicted. Although at the first glance it seems that our switch cannot sustain high loads for long periods of time (e.g. 4849 cell times, or about 13 ms, when the offered load is 31), notice that this situation is not likely to happen or to persist for a long time: Simcoe et al. have calculated the probability that *n* input ports would all simultaneously deliver data to the same output port in one cell time, assuming a uniform distribution model (i.e., when a cell arrives at an input port, the probability of it going to any output port is the same) [3]. The following table summarizes their results for a 64-port switch:

Offered Load per Output Port	**Probability of this load**
0	36.49%
1	37.07%
2	18.55%
3	6.09%
4	1.47%
5	0.28%
6	0.04%
7	0.01%
8 and above	too small to measure

As we can see, under the uniform distribution model, the offered load persistently imposed on an output port is almost never more than 7, and most of the time it is in the range between 0 and 2, which can be handled extremely well by our 85,000-cell ATM switch given our above simulation results.

5 Cost and Market Analysis

In this section we consider the potential cost and marketability of our ATM switch design. While our ATM switching fabric provides the bulk of the functionality of an ATM switch, there are other components that are required to build a fully functional ATM switch. The cost of our IRAM chip itself has been estimated at approximately $100 in large quantities (large enough to offset the cost of the Mask Set). The implementation of the switching fabric in IRAM provides us with a low-cost, flexible component in a small package. We believe that this product will appeal to vendors looking to design a wide range of network switches—the complexity and cost of designing such switches can be greatly reduced by the availability of a low cost, high-bandwidth, integrated switching fabric.

The design of a switch based upon our product requires a separate processor to handle Connection Control. This function includes the transmitting and receiving of connection establishment and teardown signals, and the updating of entries in our VC/VP Lookup Table. This processor can also be used to implement ABR Flow Control—the chip transmits and receives Resource Management (RM) cells, and monitors the utilization of buffer space in our switching core. Such processing functionality could be provided by an integrated RISC processor such as the Intel i960 or a traditional processor such as the Intel i386/i486 and some small amount of DRAM (no more than, say, 4MB), to execute the switch firmware. Because the performance requirements are not demanding for these tasks, cheap processors such as these are believed to be more than adequate.

In addition to the switching core and the external switch processor, we will need to interface with our network medium. In the case of high-bandwidth trunk connections, we assume that fiber-optic cabling will be needed. For such connections, two external circuits are needed. The first type of chip that is needed simply performs the optical-electrical conversion, reading serial bits from the fiber channel. The cost of such a chip has been estimated at $50 per port. As the adoption of fiber for high-bandwidth networks becomes more common, we expect this price will fall over time as demand increases.

The second type of chip that will be needed to interface fiber-optic channels to our switch is a SONET-framing chip. This chip will basically decode the SONET frames and convert the serial bit stream provided by the optical-electrical chip into a parallel sequence of cell bits. At OC-12 speeds, we have priced the PMC-Sierra PM 5355 S/UNI-622 at $250 in a quantity of 1000, and $375 in single-unit quantities. At OC-3 speeds, we have priced the PMC-Sierra PM 5348 S/UNI-DUAL at $80 in a quantity of 1000, and $150 in single-unit quantities. The S/UNI-DUAL is essentially two OC-3 SONET framing chips in one, so only one S/UNI-DUAL chip is required for every two ports. These cost figures are based upon current market prices; it has been estimated by PMC-Sierra's distributors that the cost of their SONET framing chips can be expected to drop by 20% over the next year.

Finally, we can assume some additional miscellaneous costs for integrating all these components onto a single board, and the cost of providing some firmware control for the switch processor.

We can estimate the cost of producing an ATM switch based upon our switching component as follows. Our cost analysis is primarily based on the case where all switch connections are optical, as in LAN and data backbone switching. In applications such as cable systems, where most of the connections are with coaxial cable and simple framing protocols, the cost should be considerably less, as the $50 per port opto-electrical circuitry can be eliminated, and the expensive SONET decoders can be replaced with simpler shift registers plus some logic to implement a very basic framing protocol. We cannot easily estimate the cost of such logic, as such parts are not readily available in today's market (they would need to be custom-built, most likely).

	Configuration: 32 OC-3 Ports	
Quantity	**Part**	**Estimated Cost**
1	IRAM-based switching core	$100
16	PMC-Sierra PM5348 S/UNI-DUAL OC-3 SONET-framing chip	$80 x 16 = $1280
32	Opto-electrical interface	$50 x 32 = $1600
1	CPU	$20
1	4MB DRAM SIMM (for CPU)	$20
	Misc. integration costs	$150
	Total Actual Cost	**$3170**
	Incremental Per-port Actual Cost	**$90**

	Configuration: 8 OC-12 Ports	
Quantity	**Part**	**Estimated Cost**
1	IRAM-based switching core	$100
8	PMC-Sierra PM5355 S/UNI-622 OC-12 SONET-framing chip	$250 x 8= $2000
8	Opto-electrical interface	$50 x 8= $400
1	CPU	$20
1	4MB DRAM SIMM (for CPU)	$20
	Misc. integration costs	$150
	Total Actual Cost	**$2690**
	Incremental Per-port Actual Cost	**$300**

5.1 Comparison to Existing Products (“The Competition”)

To assess the market viability of our IRAM product, we believe it is important to examine its features and pricing against a commercial ATM switch. The following tables present a feature and price comparison against the Cisco LightStream 1010.

Feature Comparison	
Cisco LightStream 1010	**Our IRAM Switch**
Fully non-blocking switching fabric	Fully non-blocking switching fabric
5 Gb/s throughput	9.6 Gb/s throughput
Support for 32,000 active VCIs	Support for 256,000 active VCIs
Buffer capacity of 65,536 cells	Buffer capacity of 85,000 cells
16 OC-3 or 4 OC-12 ports	32 OC-3, 8 OC-12, or 4 OC-24 ports
Multicast-capable	Can be made multicast-capable
Support for PNNI connection protocol	Support for PNNI connection protocol (through off-chip processor & firmware)
Support for advanced management features	Support for advanced management features (through off-chip processor & firmware)
Support for ABR-based flow control	Support for ABR-based flow control (through off-chip processor & firmware)

The Cisco Lightstream switch is positioned as a “Campus and Workgroup ATM Switch,” meaning that it is targeted for use in network backbones and LAN/WAN switching. Our ATM switch is capable of fulfilling this role well. In fact, we believe that an ATM switch developed with our switching implementation compares quite favorably with the Cisco switch’s feature set.

If we compare the pricing of the Cisco switch to an ATM switch based upon our switching technology, we will see substantial advantage offered by our switch. The cost of the Cisco Lightstream 1010 has been estimated as follows:

	Configuration: 16 OC-3 Ports	
Quantity	**Part**	**Estimated Cost**
1	Chassis and memory unit	$20,000
4	4-port OC-3 port card	$5000 x 4= $20,000
	Total Retail Cost	**$40,000**
	Incremental Per-port Retail Cost	**$1,250**

	Configuration: 4 OC-12 Ports	
Quantity	**Part**	**Estimated Cost**
1	Chassis and memory unit	$20,000
4	OC-12 port card	$5000 x 4= $20,000
	Total Retail Cost	**$40,000**
	Incremental Per-port Retail Cost	**$5,000**

While we expect that introduction of a low-cost ATM switch will cause vendors such as Cisco to lower their prices, we believe that our level of integration will still allow us to offer substantially lower prices than conventional ATM switch implementations.

Market Viability

Our cost figures make the assumption that we will have sufficient sales of our product to offset our production costs and to achieve favorable pricing of external circuitry. The production of our IRAM product is only worthwhile if the market for this item can be expected to be sizable.

First, it is worth noting that our basic switching architecture is not necessarily limited to ATM. We believe that it will not be too difficult to design external circuitry to allow our switching fabric to work within a Fast Ethernet or Gigabit Ethernet switch. This could create a substantially larger market for our switching core, and allows a switch designer maximum flexibility.

Second, we believe that the market for ATM switches will increase considerably. With standards not yet developed for future high-bandwidth technologies such as Gigabit Ethernet, and with ATM's support for Broadband-ISDN, ATM will be a leading technology for efficient, high-bandwidth communication in coming years; the demand for high-bandwidth networking can be expected to only increase. Proposals for applications such as interactive or distance learning, telemedicine, telecommuting, video conferencing, and video-on-demand all rely on the assumption of a high-bandwidth network underlying these technologies. As network service providers prepare for high-bandwidth services, they can be expected to upgrade or replace their current networks of twisted-pair and coaxial cable with fiber backbones. This will substantially decrease the obstacles to widespread deployment of fast ATM networks.

Third, we believe ATM switches based upon our design would be appealing to cable television and telephone companies deploying high-bandwidth WANs, as described in Section 1.2. Such networks would conceivably be used to carry simultaneous voice, video and computer data traffic. Deployment on a large scale would require a substantial investment in large numbers of ATM switches. The use of mixed fiber-coaxial WANs would eliminate much of the associated costs of ATM deployment, while still providing an effective solution to the high-bandwidth and high-performance demands of such users.

6 Conclusions

We can implement a 1.2GByte/s ATM switching fabric at low cost through the use of Intelligent DRAM technology and our 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.

7 References

[1] 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*, August 1996.

[2] Mogul, J., "Network Behavior of a Busy Web Server and its Clients," DEC WRL RR 95.5.

[3] Simcoe, R., et al., "Perspectives on ATM Switch Architecture and the Influence of Traffic Pattern Assumption on Switch Design," *ACM/SIGCOMM Computer and Communication Review*, April 1995.