Appeared in proceedings of
IEEE INFOCOM 1995, pp. 179-187.

Design of the APIC: A High Performance ATM Host-Network Interface Chip†

Zubin D. Dittia zubin@dworkin.wustl.edu
Jerome R. Cox, Jr. jrc@hobbs.wustl.edu
Guru M. Parulkar guru@flora.wustl.edu

Department of Computer Science, Washington University in St. Louis
St. Louis, MO 63130, USA.

Abstract

We present the design of a high performance ATM host-network interface for multimedia workstations and servers. At Washington University, as part of an ARPA-sponsored gigabit local ATM testbed, we are building a prototype of this interface that can support a sustained aggregate bidirectional data rate of 2.4 Gbps. The centerpiece of our interface design is a custom chip called the APIC (ATM Port Interconnect Controller). Multiple such chips can be interconnected to yield a desk-area network (DAN) which would serve as a high speed I/O interconnect for the host computer. This paper details the internal design of the APIC chip, and outlines some of its key features. Noteworthy among these are: connection caching, transmit pacing, cell batching, remote control, and support for AAL-0, AAL-5, multipoint, and loopback connections. We have chosen to defer to a later paper the details pertaining to several other features which provide support for zero-copy, improved interrupt handling, direct control of the chip from user-space, and efficient buffering and demultiplexing.

1 Introduction

The rapid emergence of very high speed packet switched networks has opened up a plethora of possibilities for new and exciting distributed applications. But even as the first gigabit networks make their appearance, it is becoming increasingly evident that merely raising raw network bandwidth does not translate to improved performance for end-applications. The bottleneck is easily identified to lie within the end-hosts (i.e., the workstations or servers that run the user applications). Within an end-host, a number of problem areas can be identified that contribute to this bottleneck, not the least of which is a lack of integration between the hardware architecture (particularly that of the host-network interface), the operating system, and network communication protocols. It is a well known fact that operations such as data copying, software checksumming, servicing of interrupts, and context switches are often responsible for poor performance. By carefully designing the network interface, OS, and protocols in a coherent and integrated fashion, it is possible to minimize, or in some cases to completely annul, the handicaps resulting from the above operations.

Solving the aforementioned problems would not, however, overcome the inherent limitations imposed by the peak system memory and bus bandwidths, which are often lower than the peak network bandwidth in a gigabit network. By rethinking the host I/O subsystem, and moving processing power closer to network I/O, it is possible to bypass the main system memory and bus altogether. This approach is especially well suited for continuous media data (e.g. video, audio, etc.), where the processing is usually of a very specialized nature (e.g. signal processing, compression, encryption, etc.) and needs to be carried out in real time. However, for most other types of data, and in particular for discrete media data (e.g. graphics, text, simulation data, etc.), it is still important to provide for a high speed pathway to the main system memory.

In this paper, we describe the design of an ATM host-network interface architecture for high performance multimedia workstations and servers. A prototype of the interface, which will allow for a sustained data transfer rate of 1.2 Gbps simultaneously in both directions (into and out from the network), is currently in the process of being built. This effort, which is slated for completion in 1995, is part of a bigger project (funded by ARPA), which involves the setup of a gigabit local ATM testbed at Washington University and the demonstration of some example applications.

Our proposed host-network interface architecture is built around a single custom chip called the APIC (short for *ATM Port Interconnect Controller*). A desk-area network (DAN) is constructed by interconnecting a number of APIC chips; each APIC in the DAN interfaces to an I/O device or to the main system bus. The resulting interconnect, which can be likened to a high speed I/O bus, serves to provide a communication pathway that would allow data on some connections (such as those carrying continuous media traffic) to

† This work was supported in part by the Advanced Research Projects Agency (ARPA), the National Science Foundation (NSF), and an industrial consortium of Ascom Timeplex, Bell Communications Research, Bell Northern Research, Goldstar Information and Communication, NTT, Italtel SIT, NEC America, Southwestern Bell, Bay Networks, and Tektronix.

skirt the main system bus. This paper is dedicated to describing the interconnect architecture itself, a number of features of the APIC chip, and the internal design of the APIC.

Due to space limitations, we have chosen not to include the description of several features of the APIC design relating to the system bus interface. These include the novel buffering mechanisms employed by the APIC, which can be exploited by the host operating system to avoid data copying and software checksumming. Other features *not* described here include:

(a) support for user-space protocol implementations by allowing direct control of the APIC chip from user-space. Elimination of kernel intervention also makes this very attractive for applications requiring low latency;

(b) a highly flexible and efficient interrupt mechanism;

(c) a novel threaded connection strategy that allows for efficient handling of interrupts by the OS;

and several others. All of these features are detailed in the system architecture document for the APIC chip [7]; a future paper will focus only on these aspects of the APIC design.

It should be noted that although our design assumes an underlying ATM-like network, many of our ideas are applicable (perhaps with some modification) to other types of networks, including connectionless (datagram) networks.

The outline of the paper is as follows: Section 2 provides the motivation for a new host-network interface design, and Section 3 relates our work to that of others. Section 4 describes our proposed architecture. Section 5 lists various features of the APIC chip. Section 6 outlines the internal design of the APIC, and includes aspects relating to the implementation of some of the features that were presented in Section 5. Finally, Section 7 provides some concluding remarks and outlines our future plans.

2 Motivation

The research community has expended considerable effort toward the design of high speed networks and switching systems [15]. Experiences from the five gigabit testbeds have shown that gigabit networking is not only possible, but can be very useful in a wide variety of application areas. At Washington University, we have an installed and operational 155 Mbps packet switched network [4], and we are currently in the process of building a multirate ATM switching system [18] that will support data rates of 622 Mbps (SONET), 1.2 Gbps (G-link), and 2.4 Gbps (G-link) per port. Our primary objective in designing the host-network interface has been to support applications that can utilize these data rates, yet run on typical workstations and servers and deliver reasonably high performance.

The specific classes of applications we are targeting fall into two broad classes:

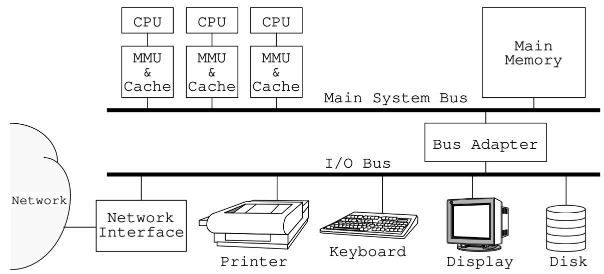

Figure 1: A Typical Workstation Architecture

- point-to-point as well as multipoint multimedia and virtual reality applications, which are characterized by high bandwidths and real-time requirements; and
- distributed computing applications that are characterized by their requirement for very low latency.

In order to understand the challenges posed by this target environment, it is instructive to examine the current state-of-the-art in workstation architectures. Referring to Figure 1, we see that the hardware in a typical workstation includes: support for multiple processors; a main system bus that interfaces to the processors and main memory, and provides support for snooping based cache coherency protocols; a main memory that is organized as a single DRAM bank; and an I/O bus to which most of the I/O devices, including the network interface, are connected. This type of architecture is not well suited to the classes of applications we are interested in—the primary limitations arise from the bottlenecks imposed by the two buses[1], and from the fact that the effective memory bandwidth is normally no more than 500 Mbps. To compound the problem, protocol stack implementations in existing operating systems require data to be copied multiple times (once from the network interface to kernel space, and then from kernel space to the application's address space). Each such copy step increases the number of times the data has to traverse the system bus, and also increases the number of times memory is accessed. Furthermore, an application may need to access the data, possibly more than once, in order to perform some computation on the data, or just to copy it on a disk. Add to this the fact that multiple processors working on multiple media streams have to share the same bus, and it becomes apparent why many of the interfaces designed to support high speed networks fail to deliver even a small fraction of the network bandwidth to end applications. Clearly, there is a need to revise the host communication architecture and I/O subsystem, if we are to be able to meet

[1] The effective bandwidth of the fastest buses in use today is no longer much higher than the network bandwidth (of emerging networks). If data belonging to the same stream traverses the bus multiple times, the available bandwidth usually drops to less than the network bandwidth. The effective bus bandwidth for typical high performance workstations ranges from 500 Mbps to 2 Gbps.

the demands imposed by our choice of target applications.

3 Related Work

Several research groups have attempted to design and implement high speed host-network interfaces over the past few years [12]. Noteworthy among these are: the network adaptor board (NAB) [13] from Stanford, the Nectar communication accelerator board (CAB) [2], the Bellcore OSIRIS ATM interface [6,9], the University of Pennsylvania ATM interface [17], the Afterburner [5] and Medusa [3] interfaces, and Washington University's Axon [16] network interface. Our interface design builds upon the success of these efforts by adopting a number of useful features and improving or improvising them for our target environment. In addition, it also incorporates a number of innovative and novel ideas that have not been tried before. There is one way in which our approach is radically different from earlier efforts. While all of the aforementioned interfaces connect to the host's I/O bus (and thus share the limitations mentioned earlier—see Section 2), our interface serves as a *replacement* for (or *complements*) the I/O bus. Also, we believe that our design is one of the first to effectively exploit the connection-oriented aspect of ATM to improve performance; most of the earlier designs were either targeted toward datagram networks, or retained some vestige of the datagram world in a connection-oriented environment.

The APIC interconnect described in this paper is an example of a desk-area network (DAN). A DAN is an architecture in which a packet switch serves as a workstation interconnect. Work on desk-area networks was pioneered at the University of Cambridge [10,14] by M.D.Hayter, I.M.Leslie, D.R.McAuley, and D.L.Tennenhouse, and is now being pursued independently at MIT by D.L.Tennenhouse [11]. Both of these efforts are aimed at using a general switch based interconnect within the host; our approach differs in that we use planar interconnections of APIC chips, with each chip separately interfacing to an I/O device or to CPUs and memory. While general switch based architectures have the potential for higher performance, they do come with some drawbacks, at least in the short term, and predominantly from an economic viewpoint. More on this can be found in some of our other publications [7,8].

4 Proposed Architecture

A high level schematic of our proposed DAN architecture is shown in Figure 2. At the heart of this architecture is a daisy-chained interconnect comprising of a number of *ATM Port Interconnect Controller* (APIC) chips. Individual chips interface either directly to the main system bus, or through a dual or triple ported memory (in this paper, we shall concentrate only on dual and triple ported memories that are implemented as VRAMs) to an I/O device (both the device and the APIC use the serial port(s) of the VRAM to allow for full data rate transfers). One of the APIC chips on the interconnect is connected to the external ATM network through a link interface. ATM cells arriving from the network pass from one APIC to another until they reach an APIC chip that is directly connected to the destination device (or to the system bus). Similarly, data originating from a source device is passed as an ATM cell stream from the APIC directly connected to that device, down the APIC interconnect towards the link interface, and finally out to the network. It is also possible for two local devices to communicate over the APIC interconnect, without cells ever having to leave the desk area. Thus, for example, a video cell stream originating at a camera could traverse the interconnect in order to reach the local display device. The APIC which directly interfaces to the system bus can be used for traffic originating from, or destined for, the system's main memory. All other traffic movement happens only over the APIC interconnect, thus sparing the main system bus of this burden.

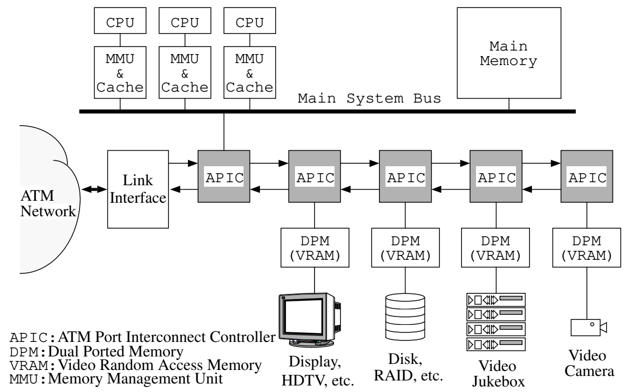

Figure 2: Host-Network Interface Architecture

It is easy to see that the APIC interconnect bears a strong resemblance to the traditional notion of an I/O bus. In fact, the APIC interconnect can be thought of as a replacement for the I/O bus. From an economic standpoint however, it would be expensive to have one APIC chip for every device in the system. In a real system, we expect the APIC interconnect to coexist with lower speed I/O buses—only the devices capable of sustaining high bandwidth transfers would be connected to an APIC on the interconnect, and all other devices would reside on I/O buses. A detailed description of why an APIC interconnect based architecture outperforms a bus based architecture appears in [7,8].

In our prototype implementation, each APIC interfaces to another APIC over two links (one is used for incoming traffic, and the other for outgoing traffic) which can operate at up to 1.2 Gbps each. ATM Forum's UTOPIA (*Universal Test*

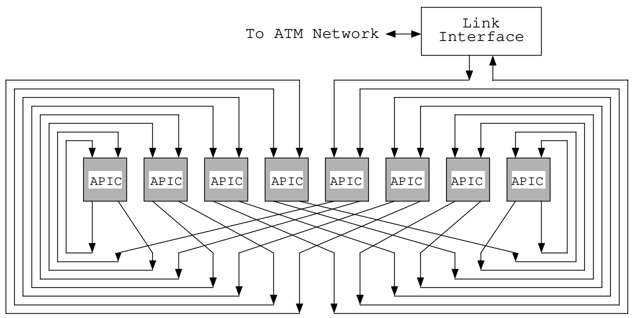

Figure 3: Perfect Shuffle Interconnection

& Operations PHY Interface for ATM) standard [1] is used on these links. This standard specifies the data path interface between the ATM layer (performing segmentation and reassembly) and the physical (PHY) layer components of an ATM subsystem. It can support a wide range of speeds, with data path widths of 8 or 16 bits. The prototype APIC uses the 16 bit version. The link to the ATM network switch port can also operate at up to 1.2 Gbps. Further, each individual APIC chip is capable of sourcing and sinking data at a peak rate of 4.8 Gbps over a 64-bit wide bus (of course, the actual data rate cannot exceed the system memory or bus bandwidth, or the VRAM serial port bandwidth). Clearly, the maximum sustained aggregate bidirectional data rate that can be utilized by all the devices sharing the APIC interconnect is 2.4 Gbps.

One of the major advantages of allowing full rate bidirectional traffic on the interconnect is that in the absence of local (constrained to within the desk-area) traffic, each device can access the network in much the same way as if the device had exclusive access to the network; the bandwidth reservation on the network can, in this case, be thought of as extending into the APIC interconnect, all the way to the device. This advantage would be lost if, for example, the interconnect had a unidirectional ring topology in place of a daisy chained topology.

Nor is the choice of topology constraining. For example, if delay is a concern, a perfect shuffle topology (see Figure 3) will guarantee that you can get from any device to the network or to any other device in $O(\lg n)$ hops (where n is the total number of APICs in the DAN). Note that the connections to the device VRAMs or to the system bus have not been shown in Figure 3, but exist all the same. Other topologies such as a unidirectional toroid (mesh) or a delta network can also be realized.

4.1 Media Processing

It would appear that our design does not allow for any type of processing to be done on continuous media streams that do not traverse the main system bus. This is not the case however; media processing on traffic restricted to the APIC interconnect can be done in two ways. The first approach involves embedding a signal processor on the data path between a continuous media device and VRAM. This method suffers from the drawback that the processor cannot be easily shared by multiple devices. However, in some cases this would be the approach of choice, such as, for example, when the media processor is a hardware compression/decompression engine. This ensures that uncompressed data would not have to traverse the interconnect, which is important because the data rate of the uncompressed stream might be a significant fraction of, or more than, the maximum bandwidth of the interconnect. A second approach is to have a special media processor residing as a separate "device" on the interconnect. This method would work especially well when the processing device is an encryption/decryption engine, for example. Data destined for the network could be made to "pass through" the encryption device on its way to the link interface. Similarly, data arriving from the network could be made to pass through this device so the data can be decrypted before being forwarded to the destination I/O device.

4.2 The APIC Chip

The APIC chip, which is central to our design, behaves like a 3×3 ATM switch, two of whose ports are connected to two other APIC chips, and the third port interfaces to external memory (possibly over a shared bus). One of the jobs of the APIC is to take specified data from the memory, create an ATM cell stream using that data, and forward the cell stream to a downstream APIC[2]. In the reverse direction, the APIC is responsible for doing a VPI/VCI virtual channel lookup on every arriving cell to determine if it is destined for the local device. In addition to this simple "switching" operation, the APIC implements a host of other features. We now go on to describe those features that were selected for inclusion in this paper.

5 APIC Features

In addition to mundane features such as segmentation and reassembly which can be found in almost any network interface designed for ATM, the APIC includes a number of unique and novel features which make it highly comprehensive and flexible. For the features described below, a detailed treatment of the implementation is beyond the scope of this paper. However, we do provide comments hinting at the implementation method being used—these

[2] We refer to *downstream* and *upstream* APICs several times in this paper; while no such distinction can be made out of context, we typically use these terms when discussing a specific APIC chip, to refer to its two neighboring APIC chips. In the context of a cell arriving at the APIC chip that is the focus of the discussion, we refer to the originating neighbor as the *upstream APIC*. On the other hand, when a cell is forwarded from the APIC in question to another APIC, then the latter will be referred to as the *downstream APIC*.

can be found scattered over both this section and the next.

5.1 Multipoint Connections and Loopback

If we think of the APIC as a switch with three input ports and three output ports, and if we allow an incoming cell to be forwarded to any combination of output ports, then we are in effect providing a multipoint capability. Such a feature is useful to have if, for example, we wanted a video stream from a camera to be viewed on the local display device, as well as sent out over the network to a remote host. Note that loopback within the APIC comes free with this extension.

5.2 AAL-5 Segmentation and Reassembly

Since the (ATM) educated reader will be familiar with AAL-5, and with segmentation and reassembly functions using AAL-5, we shall not dwell on these here. Suffice it to say that the APIC does not do store-and-forward of entire AAL-5 frames; instead, it periodically reads cells from appropriate locations in external memory (be it the system's main memory or a device VRAM) based on the current position in an AAL frame (for the connection in question), and sends them out into the network. On the receive side also, the APIC reconstructs the original frame in external memory without storing an entire frame in a local memory. Full AAL-5 functionality, with frames of up to 64 Kilobytes and a 32-bit hardware CRC, is implemented.

5.3 Raw (AAL-0) Connections

On a per-connection basis, it is possible to specify whether the APIC should use AAL-0 or AAL-5. AAL-0 is termed the *null* AAL—cells belonging to these connections are treated as separate entities and not as part of a larger frame. These "raw" connections are especially useful for sending or receiving signalling, OAM, and other control cells; this is because the host, rather than the APIC, is responsible for generating and stripping ATM cell headers. This same feature also facilitates software implementations of other AALs, such as AAL-3/4.

5.4 Batching

When interfacing to a continuous media device through a VRAM, the APIC reads/writes cell contents from/to the serial access memory (SAM) in the VRAM over its serial port. In order to achieve high data rates over the serial port without sacrificing bandwidth on the random access port of the VRAM, it is important to fill the SAM as much as possible. This ensures that there will be a minimal number of wasteful transfer cycles between the SAM and the random access memory. The SAM in a 64-bit wide organization of a VRAM can be as large as 512 bytes. With only 48 bytes of payload in an ATM cell, it is clearly inefficient to waste one transfer cycle on a per cell basis, with the SAM filled to only a small portion of its capacity. The APIC overcomes this problem through the use of *batching*: on transmit, the APIC reads batches of up to 8 cells in a single read operation. Similarly, on receive, the APIC lets cells accumulate in on-chip storage, and then writes batches of up to 8 cells (all from the same connection) into external memory. For the APIC connected to the system bus also, batching has the potential to reduce bus arbitration overhead and improve overall bus efficiency by moving more data in a single transaction.

5.5 Low Latency Bypass

The APIC includes the capability to specify (on a per-connection basis) whether a connection carries traffic that has a low latency requirement. Cells from such connections are always processed preferentially by the APIC (over cells not belonging to a low-latency connection). Wherever queueing occurs inside the APIC, there is always a bypass queue which is used for low-latency traffic. Transit traffic (i.e., traffic which is not sourced or sinked by the APIC) is always put into the low-latency category. Low latency connections can be used for some distributed computing applications, and for signalling and other control traffic.

5.6 Connection Caching

For each connection, the APIC has to maintain a considerable amount of state information which includes how far it has got reading or writing to the current user buffer, where it can find the next buffer, the partially computed CRC for the current AAL-5 frame, where it is in the current frame (start, middle, or end), when to generate interrupts, and a host of other information. All of this information is stored in a *Connection State Block* (CSB). The prototype APIC's CSB contains 32 bytes of information (per connection). To get an estimate of the total on-chip storage requirements for all connections, we need to have an estimate of the number of connections that will be supported. A reasonable maximum number of connections for a typical host, which also happens to be the maximum number of connections actually supported by a single prototype APIC, is 1024 transmit and 1024 receive connections (the maximum number of connections over the entire APIC interconnect is 1024 times the number of APICs, for both transmit and receive). Thus, with 32 bytes per CSB, the APIC would have to have as much as 64 Kilobytes of on-chip SRAM storage dedicated to connection state information alone. While not an impossible proposition, this would result in a significant portion of the available chip area being dedicated to CSBs. Further, this approach does not scale to larger numbers of connections.

Connection caching is a technique which enables us to support many more connections on a chip than we have area for. The concept of "active" connections is central to the idea of connection caching. We define "active" connections as those that are involved in sending or receiving data during a sample observation interval. Reasonable values of the observation interval that can be used to decide if a connection is active range from a few milliseconds to a few seconds, depending on the average connection lifetime. The idea for connection caching comes from the realization that not all of the 1024 transmit and 1024 receive connections would be simultaneously *active*; indeed, the total number of active connections is usually quite small. In our campus LAN environment, even a busy file server that serves 20 to 30 desktop workstations was observed to have no more than ten TCP connections simultaneously sending or receiving data (over a 1 second period of observation). With the advent of applications that utilize continuous media (which can be thought of as "permanently active connections"), this number will presumably increase. However, given the high bandwidth requirements and the potential rate of consumption of information associated with continuous media streams, it is unlikely that this increase will be much more than 10 or 20. Of course, large-scale media servers can have many more continuously active connections, and in such cases, it may be necessary to use multiple APICs.

In summary, it suffices if the APIC can support a few tens of *active* connections, but the total number of open connections that should be supported needs to be much higher. Only some critical information from the CSB need be on-chip for all supported connections; for the prototype APIC, this is less than 4 bytes per connection. The remaining CSB information can be stored off-chip for all but the currently active connections; the latter have their entire CSB in on-chip storage. The CSB for connections that have remained passive for a long time may eventually get "swapped out" into external DRAM or VRAM memory (which is relatively cheap), with a connection that has newly become active replacing it in the on-chip cache. This is illustrated in Figure 4. The prototype APIC design includes support for 32 cached CSBs for transmit and 32 for receive connections.

When a cell is received on an open but uncached connection, then the CSB for the connection needs to be fetched from external memory. A candidate has to be chosen for replacement; the most intuitive one, and one that we have verified through simulation as suitable, is the *Least Recently Active* (LRA) connection. This is defined as a cached connection on which a cell hasn't been received for the longest duration of time[3].

The implications of caching for transmit connections

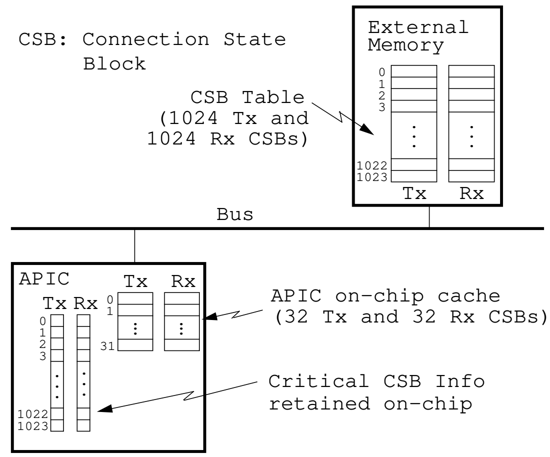

Figure 4: Connection Caching in the APIC

requires knowledge of the pacing strategy used in the APIC, so we have deferred its discussion to Section 5.7.

A note on the cost of connection caching is in order. On a cache miss, the APIC has to save the CSB (32 bytes) for the connection being replaced in external memory, and load the CSB (32 bytes) for the connection that just became active. Sending or receiving a cell involves a 48 byte read or write of the payload to external memory. If we assume the bus transaction overhead is the same for both cases (this is a valid assumption in the prototype APIC environment), and if the cache miss probability is p, then the throughput, as a percentage of the total external memory bandwidth, is $\left(1+\frac{4}{3}\cdot p\right)^{-1}$. This demonstrates that caching is not as expensive as it may seem; even if every arriving cell results in a cache miss (something that is almost impossible given that we are using a LRA replacement policy), the throughput would still be more than a third of the maximum.

5.7 Pacing

ATM networks usually do not provide any feedback to the sender of a cell regarding successful transmission. Given that switch cell buffers are usually restricted to a few hundreds of cells, it would be suicidal to send a big burst of cells back-to-back into the network. Such bursts can cause overflowing of switch buffers, network congestion, increased latency for small packets, and it can also result in swamping a slow receiver that is not prepared to handle the high rate of arrival of cells.

Second, the network's connection bandwidth reservation policy, which is used to provide quality of service (QoS) to applications, usually places an upper bound on the rate at which cells can enter the network. If this peak rate specification is not met, the network cannot guarantee QoS to applications.

To prevent back-to-back transmission of large bursts of

[3] If a cached CSB is referenced every time a cell is received on the corresponding connection, then the LRA connection can easily be identified by keeping track of the *Least Recently Used* (LRU) cache entry.

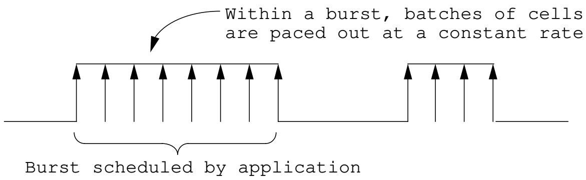

Figure 5: APIC Pacing

cells, and to prevent the transmission rate from exceeding the reserved peak rate of a connection, many network interfaces send cells belonging to the same connection spaced out over time, and allow cells from other connections to occupy interleaving cell slots. This operation is commonly referred to as *pacing*. The APIC does pacing based on a single parameter which specifies the connection's data rate within a burst (see Figure 5). The bursts themselves are scheduled for transmission by applications in a way commensurate with the bandwidth allocation scheme in use.

The APIC implements pacing using an event queue based on a *d*-heap data structure. For each connection, we maintain the time at which the next cell (or batch of cells) is scheduled for transmission. This "expiration time" is the key used to order the *d*-heap, so that the connection with the minimum expiration time can be extracted. We transmit a new cell (or batch of cells) whenever the current time equals or exceeds the expiration time; following this, the expiration time is updated to the time at which the next cell (or batch of cells) should be sent (the increment depends on the pacing bandwidth parameter for the connection). The pacing logic supports simultaneous pacing on all 1024 transmit connections—this is made possible by an innovative modification to the *d*-heap algorithm which makes it amenable to hardware implementation. Further details of the pacing strategy are not included here due to paucity of space, but we plan to present these in a future paper.

If a low-latency connection has any data queued (by the host) for transmission, the APIC dedicates all available transmission bandwidth to that connection.

The APIC also allows for *best-effort* (or *datagram*) connections. These connections have no associated pacing bandwidth; instead, they share (in a fair manner) all of the leftover bandwidth after accounting for all other (reserved bandwidth) connections.

So far, we have made no mention of the interaction between caching and pacing. It is possible to allow the host to decide which transmit connections need to be cached, and do pacing only for these cached connections. However, we felt that this was too constraining, since it limited the number of active connections to 32 or less (unless the host was continuously swapping connections in and out of cache). Further, it places an unnecessary burden on the host. So caching decisions are left to the APIC, and pacing over all 1024 transmit connections is implemented in the APIC. The cache replacement policy selects the connection with the maximum expiration time as the candidate for replacement (the expiration time for a connection that has nothing to send is assumed to be ∞). A simple modification to the *d*-heap implementation allows keeping track of both the minimum and the maximum expiration time.

5.8 Controlling the APIC

The APIC can be controlled by two methods: either using memory mapped I/O over the APIC bus interface, or through the use of control cells (perhaps sent by another APIC which is being controlled using memory-mapped I/O). Any operation that can be performed using memory mapped I/O can also be done using a control cell and vice versa. All control cell operations result in the return of a response cell. "At most once" semantics are guaranteed through the use of an alternating-bit protocol, so that loss of a control or response cell followed by a retransmission of the control cell by the controlling entity will not cause any problems.

While most devices including network interfaces are controlled using memory mapped I/O, the concept of remote control through the use of control cells is new. The major advantage of this approach is that a local control processor is not required at each of the APICs in an APIC interconnect; indeed, it does not seem worthwhile to dedicate control processors for devices such as cameras, although it may make sense to do so with devices such as disk arrays. Full remote configurability cannot be achieved however, unless the remote controlling process can also control the media device and read or write to/from the external memory or VRAM connected to the APIC and the device. To this end, the APIC provides for two special control cells that do not change the APIC'S internal state, but can be used to command the APIC to write or read data to or from external memory. The data to be written is specified in a control cell, along with the address; the data to be read is returned in a response cell. With this extension, a remote device can be controlled as follows (assume that memory mapped I/O is used to control the remote device): the APIC connected to the remote device can be made to read or write the device registers simply by sending to it read/write control cells containing the appropriate (memory mapped I/O) addresses.

6 Internal Design of the APIC

Figure 6 shows a high level block diagram of the internal design of the prototype APIC chip. The two input ATM ports are shown on the left, while the two output ATM ports are on the right. Note that each of these ports has a data path width of 18 bits: 16 data + 2 parity, as per the UTOPIA specification. The memory bus and VRAM access interface is at the bottom of the figure, and is 64 bits wide.

Figure 6 is best explained by identifying the various paths that can be taken by data passing through the APIC.

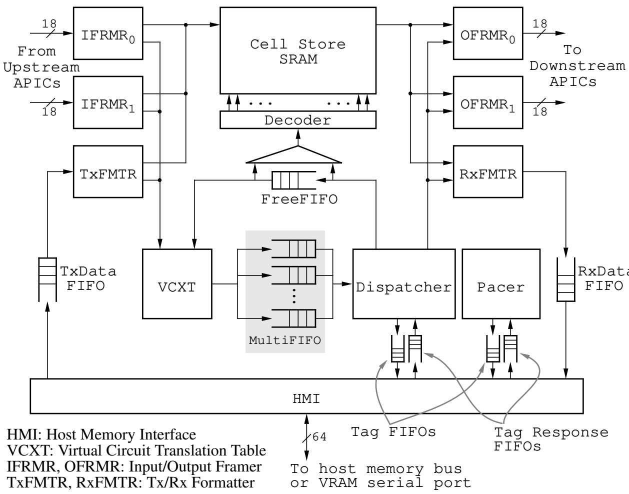

Figure 6: Internal Architecture of the APIC Chip

First, we consider the simplest case: a cell from an upstream APIC being forwarded to a downstream APIC. The cell is not destined for the local device, so it will not be written to external memory. The cell enters the APIC through one of the two **Input Framers** (IFRMR), whose job is to synchronize the incoming cell stream with the APIC's internal (core) clock (which is different from the link clock), and to verify the parity bits and the ATM cell header error check (HEC). The cell is then written into a cell slot in the central **Cell Store SRAM** (which can hold up to 256 cells); the specific slot to use is obtained from the **FreeFIFO**, which is a free list containing 8-bit addresses of empty slots in the Cell Store. A copy of the cell header is passed to a **Virtual Circuit Translation Table** (VCXT), which does a table lookup using the virtual path and channel identifiers from the cell header, and determines that the cell is not destined for the local device, and should therefore be forwarded to a downstream APIC. Following this, the VCXT places an entry for the cell in an appropriate FIFO contained in the **MultiFIFO**. The latter is a special structure which allows multiple FIFOs to share the same physical memory. It contains a set of FIFOs dedicated to each of the three output ports (two ATM ports and one receive port). For the two output ports, there is one FIFO each for low-latency traffic and one FIFO each for all other traffic. Transit cells are always assumed to be in the low-latency category, so the entry for the cell under consideration is placed in the low latency FIFO corresponding to the appropriate output port. Once every cell cycle, the **Dispatcher** reads an entry from one of the two FIFOs for each output port (the low-latency FIFO always gets priority), and schedules the cell contents to be read out from the Cell Store and passed to the appropriate **Output Framer** (OFRMR). When it does this, it also returns the address of the cell slot to the FreeFIFO. The OFRMR is responsible for generating the HEC byte and parity bits, and for transmitting the cell after synchronization with the link clock.

We now consider the case where a cell is received that is intended for the local device. To conserve space, we have chosen *not* to consider the case where the CSB for the connection to which the cell belongs is absent from the on-chip cache (a cache miss). Suffice it to say that a cache miss results in the CSB for the LRA connection getting written to external memory, following which the CSB for the connection that caused the miss replaces it in the on-chip cache. For a cache hit, the VCXT lookup for the cell yields a 5-bit cache line number. The MultiFIFO contains 32 different FIFOs, one per cached receive connection; the 5-bit cache line number can be used to select the appropriate FIFO from among these. The Dispatcher processes entries from these FIFOs in a round-robin fashion; however, it always gives priority to low-latency connections. By reading more than one entry from a FIFO before going on to the next one in a round-robin order, the Dispatcher can process batches of cells. In fact, the reason for having 32 different FIFOs instead of just one is to facilitate batching of cells belonging to the same connection. After reading an entry from the MultiFIFO, the Dispatcher uses information in the CSB for the connection to determine the location in external memory where the cell is to be written. It then encapsulates this address and other information such as the amount of the data to be written, into a *tag* which is placed in the Dispatcher's **TagFIFO**. The tag itself can be thought of as a command to the **Host Memory Interface** (HMI) instructing it to read or write data to external memory. Note that the Dispatcher issues only one tag for each batch of cells, if batching is in effect. The Dispatcher also arranges for the cell contents to be read out from the Cell Store and passed to the **Receive Formatter** (RXFMTR). Following this, the address of the cell slot containing the cell that was read out is returned to the FreeFIFO. The job of the RXFMTR is to format the data into 64-bit wide words, and verify the AAL-5 CRC (if applicable). The data from the cell is then placed in the **RxData FIFO** from where it can be read at a later point in time by the HMI. It is important to note that the HMI operates on a different clock than the core of the chip—it uses the clock from the bus on which the external memory resides. This allows the HMI to operate independent of, and asynchronously with, the rest of the chip; when it is granted access to the bus (perhaps by external arbitration logic), it reads tags from one of the TagFIFOs and data from the RxData FIFO, and writes the data to external memory at the address specified in the tag.

We now consider transmission of cells by the APIC. The **Pacer** is the block responsible for implementing the pacing algorithm. It generates a *pacing tag* for a connection whenever the connection's entry has the minimum expiration time (see Section 5.7), and this time is less than or equal to the current cell clock value. This pacing tag contains the address in external memory from where the cell (or batch) contents are to be read, the ATM cell header to use, the destination ATM port, and some other information. This tag is placed in the Pacer's TagFIFO, from where it can be read by the HMI when it has access to the external bus and it is not busy doing something else. The HMI reads data from external memory using the information in the tag, prepends the cell header to each payload, and places the cell in the **TxData FIFO**. The **Transmit Formatter** (TXFMTR) extracts cells from the TxData FIFO, computes the AAL-5 CRC (if necessary) and writes the cell into a free slot in the Cell Store. The VCXT does not do any table lookup for transmit cells. It merely writes an entry for the cell in the MultiFIFO (into a FIFO corresponding to the destination ATM port). From this point on, the Dispatcher treats the cell as if it were a transit cell, and forwards it to the correct output port.

The simple description above does not include a number of important aspects of the design. A few of the omitted details bear special mention:

- The MultiFIFO allows sharing of space for a maximum of 256 FIFO entries among a large number of FIFOs (about 40). It allows the same entry to be queued in multiple FIFOs, and this feature is used to support multipoint connections in the APIC.
- The Dispatcher and the Pacer issue tags to the HMI for storing CSBs or buffer descriptors to external memory, and for loading new CSBs or buffer descriptors from external memory.
- All memory mapped I/O commands as well as control cell commands are executed by the HMI.

A complete specification and detailed design can be found in the APIC's system architecture document [7].

7 Conclusions

We have estimated the complexity of various APIC modules. Based on these estimates, the prototype APIC will need about 200 square millimeters of silicon on a 0.5 micron process. The APIC's core clock runs at 80 MHz.

Our design allows the APIC to serve as a single chip ATM host-network interface. Using multiple chips allows us to go beyond traditional network interfacing, and provide true desk-area network capabilities. In addition to interfacing to a few target workstations and I/O devices such as cameras and high resolution displays, we plan to use APIC based desk-area networks to prototype large video servers, and possibly a video wall.

VHDL coding has already commenced for the APIC, and as per our current estimate, the first chips will be ready toward the end of 1995.

References

[1] "An ATM PHY Data Path Interface: Draft Version 0.5," ATM Forum Draft, May 1993.

[2] Arnould, Emmanuel *et al*; "The Design of Nectar: A Network Backplane for Heterogeneous Multicomputers," *ACM SIGOPS Operating Systems Review*, Vol. 23, April 1989, pp. 205-216.

[3] Banks, David; and Prudence, Michael; "A High-Performance Network Architecture for a PA-RISC Workstation," *IEEE JSAC*, Vol. 11 No. 2, February 1993, pp. 191-202.

[4] Cox Jr., Jerome R.; Gaddis, M.; and Turner, Jonathan S.; "Project Zeus: Design of a Broadband Network and its Application on a University Campus," *IEEE Network*, Vol. 7 No. 2, March 1993, pp. 20-30.

[5] Dalton, C.; Watson, G.; Banks, D.; Calamvokis, C.; Edwards, A.; and Lumley, J.; "Afterburner," *IEEE Network*, Vol. 7 No. 4, July 1993, pp. 36-43.

[6] Davie, Bruce S.; "The Architecture and Implementation of a High-Speed Host Interface," *IEEE JSAC*, Vol. 11 No. 2, February 1993, pp. 228-239.

[7] Dittia, Zubin D.; Cox, Jerome R.; and Parulkar, Guru M.; "A Gigabit ATM Desk Area Network: *System Architecture Document* for the ATM Port Interconnect Controller (APIC)," Draft Version 1.1, Washington University in St. Louis, December 1994.

[8] Dittia, Zubin D.; Cox, Jerome R.; and Parulkar, Guru M; "Using an ATM Interconnect asa High Performance I/O Backplane," *Proc. Hot Interconnects '94*.

[9] Druschel, Peter; Peterson, Larry L.; and Davie, Bruce S.; "Experiences with a High-Speed Network Adaptor: A Software Perspective," *Proc. ACM SIGCOMM '94*, Vol. 24, No. 4, 1994, pp. 2-13.

[10] Hayter, Mark D.; and McAuley, Derek R.; "The Desk Area Network," *Operating Systems Review*, Vol. 25 No. 4, October 1991.

[11] Houh, Henry H..; Adam, Joel F.; Ismert, Michael; Lindblad, Christopher J.; and Tennenhouse, David L.; "The VuNet Desk Area Network: Architecture, Implementation, and Experience," to appear in *IEEE JSAC*, First Quarter, 1995.

[12] "IEEE Journal on Selected Areas in Communications, Vol. 11 No. 2," February 1993.

[13] Kanakia, Hemant; and Cheriton, David R.; "The VMP Network Adapter Board (NAB): High Performance Network Communication for Multiprocessors," *Proc. ACM SIGCOMM '88*, Vol. 18 No. 4, 1988, pp. 175-187.

[14] Leslie, Ian M.; McAuley, Derek R.; and Tennenhouse, David L.; "ATM Everywhere?," *IEEE Network*, Vol. 7 No. 2, March 1993, pp. 40-46.

[15] Partridge, Craig; "Gigabit Networking," *Addison Wesley*, 1994.

[16] Sterbenz, James; and Parulkar, Guru; "Axon: Host-Network Interface Architecture for Gigabit Communication," *Protocols for High Speed Networking*, Elsevier (North Holland), 1991.

[17] Traw, C. Brendan S.; and Smith, Jonathan S.; "Hardware/Software Organization of a High-Performance ATM Host Interface," *IEEE JSAC*, Vol. 11 No. 2, February 1993, pp. 240-253.

[18] Turner, Jonathan S.; "An Optimal Non-Blocking Multicast Virtual Circuit Switch," *IEEE INFOCOM '94*, June 1994, pp. 298-305.