ATM Network Striping

by

Michael Ismert

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 1995

Author .....
Department of Electrical Engineering and Computer Science
February 6, 1995

Certified by .....
David L. Tennenhouse
Associate Professor of Computer Science and Engineering
Thesis Supervisor

Accepted by .....
Frederic R. Morgenthaler
Chairman, Departmental Committee on Graduate Students

ATM Network Striping

by

Michael Ismert

Submitted to the Department of Electrical Engineering and Computer Science
on February 6, 1995, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Science and Engineering

Abstract

There is currently a scaling mismatch between local area networks and the wide area facilities provided by the telephone carriers. Local area networks cost much less to upgrade and maintain than the wide area facilities. Telephone companies will only upgrade their facilities when the aggregate demand of their customers is large enough to recover their investment. Users of LANs who want a wide area connection find themselves in a position where they are unable to get the amount of bandwidth they need or want at a cost that is acceptable. This coupling of technology in the two domains is a barrier to innovation and migration to new equipment.

Network striping is a method which will decouple the progress of technology in the local area from progress in the wide area. This thesis proposes a reference model for striping in a network based on Asynchronous Transfer Mode (ATM). The model is used to analyze the degrees of freedom in implementing network striping, as well as the related issues and tradeoffs. An environment that interconnects three different ATM platforms has been used to explore and experiment with the functional space mapped out by the model. The results are used to validate and improve the predictions of the model.

Thesis Supervisor: David L. Tennenhouse
Title: Associate Professor of Computer Science and Engineering

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis examines the use of *network striping* as a means to increase network performance without requiring the entire network to be upgraded. Network striping, also referred to as inverse multiplexing, is the use of multiple low speed channels through a network in parallel to provide a high speed channel. Our examination of striping is done in two parts. First, a reference model will present the degrees of freedom characterizing a network striping implementation. The model will be used to identify several interesting and previously unexplored striping cases. These cases will be examined in greater detail to see how they could be implemented and to determine what other requirements are needed from the hosts and the networks in order to make these striping cases function properly.

Along with the theoretical work, a set of experimental apparatus has been developed to actually explore some of the striping cases which have been identified. The apparatus consists of an experimental ATM-based gigabit desk-area network connected to a high-speed ATM-based local-area network which is then connected to wide-area SONET facilities.

1.1 Motivations for Network Striping

The main factor limiting progress in the wide area is scale. Due to the large scale, it costs a great deal of money, time, and effort to install and upgrade equipment. Thus, the telephone companies can only afford to upgrade their equipment relatively infrequently. In order to recover their costs, the telcos only upgrade when they believe that there is sufficient demand for new or improved service. They also upgrade to a much higher level of service, which allows them to meet increasing demands for bandwidth for a long period of time. For

example, the digital hierarchy originally used by the telcos provides channels at 64 kbps (DS0), 1.544 Mbps (DS1), and 45 Mbps (DS3)[1]. As equipment is upgraded to SONET, 155 Mbps (OC-3) will be the next level of service provided, with 622 Mbps (OC-12) as the next probable upgrade. It takes a great deal of time for the entire network to be upgraded. Initially only the links in the center of the network which carry the heaviest traffic will be replaced; upgraded links will slowly diffuse out to the edges of the telephone network as demand grows and time and money permit them to be replaced.

The main factor limiting progress in the local area is technological progress. Because of the smaller scope, the cost of installing and maintaining a LAN is relatively low. As soon as a new network technology becomes well-developed and supported, early adopters will begin to upgrade their LANs. It is even feasible for research groups to design and deploy their own custom LANs. The wide range of LAN technologies means a wide range of LAN speeds, ranging from 10 Mbps Ethernet to 100 Mbps FDDI and Fast Ethernet to gigabit-per-second custom LANs.

The differences between wide and local area networking create several major problems when the two domains are interconnected. First, the bandwidth numbers don't match up very well, leading to inefficient use of either network or monetary resources. Bandwidth in the wide area is governed by a strict hierarchy, where each level is some fixed multiple of the previous level with the addition of some extra signalling and synchronization information. Local area bandwidth is much more flexible, where functionality or available technology is the driving factor. LAN users will frequently find themselves stuck between two levels of the hierarchy, with two options available to them. They may opt for the lower level and settle for less available bandwidth over the wide area. They may opt for the higher level, but they will then pay a higher price for more bandwidth than they require. High-end users with custom LANs will find themselves out of the hierarchy completely, above the highest level of service that the telcos currently provide, and will have to settle for insufficient bandwidth. In the past, LAN users could accept the lower wide area bandwidth, making the valid assumption that only a fraction of the traffic on the LAN would be bound for the wide area. However, inter-domain traffic is increasing as the World Wide Web grows and becomes more popular as a means of gathering information. Video distribution applications such as the MBone[2] and the WWW Media Gateway[3] not only contribute to the inter-domain traffic, but create a demand for high bandwidth connections which serve very few

users at a time.

In many cases, the links within the telephone network may have enough raw capacity to handle a high bandwidth connection; unfortunately, the links are partitioned into smaller logical channels which cannot provide the necessary bandwidth. The size of these channels is determined by the amount of bandwidth which the average user needs or wants. Higher bandwidth channels are simply not available. Even though the entire path between two geographically separate hosts may have the aggregate capacity to support their high bandwidth applications, each point along the path must also be upgraded to support a higher level of service for the average user.

There is yet another problem created by the local access to the telephone network. The connection between the LAN and the telephone network must be able to support the amount of bandwidth which the users need. This may not be something which the local provider is able or willing to do. For example, someone wanting OC-12 access to the telephone network may get it in one of three forms. They may be provided with a single connection which gives them access to a single 622 Mbps channel. This is the optimal situation from the user's standpoint but also the most difficult for the local provider to offer and support. The user may be provided with a single connection which gives them access to 622 Mbps in the form of four 155 Mbps OC-3 channels. These four channels may be co-located on one fiber or may be four separate fibers. This would be the case if the local provider's equipment was only able to handle channels at the OC-3 level or below. Finally, the user could be provided with a 622 Mbps channel onto their premises with an OC-12/OC-3 demultiplexor that provided them with access to four separate OC-3 channels. This would be the case if the local provider was only able to handle OC-3 channels and only allowed OC-12 links that were internal to their network. Either of the last two cases would allow a single application access to only 155 Mbps.

The differences also create a coupling effect which slows the migration to new technology in both domains. When a company or research group upgrades their LAN, they will immediately see increased performance locally. However, performance across the wide area will probably change very little, if at all. This is unfortunate, particularly if the company has several geographically separated sites which have upgraded their LANs; the individual sites have better network performance, but their interoperation is limited by the wide area facilities. The company will either need to lease higher speed lines from the telcos, which

will probably provide significantly more bandwidth than necessary, or wait for the telcos to upgrade their equipment so that the required bandwidth to connect the sites is available. These factors act as a deterrent for groups to upgrade their LANs until the wide area services they want are available and slows progress in the local area. The telcos will delay providing improved service until there is a sufficient demand for it. This slows progress in the wide area.

1.2 Network Striping

Any new wide/local area interoperation scheme should have two properties. First, it should have the flexibility to provide LAN users with whatever reasonable amount of bandwidth that they require. For example, 10 Mbps Ethernet users should have access to 10 Mbps of wide area bandwidth, not 1.544 Mbps or 45 Mbps. High-end users should have some way to get the bandwidth which they require as well. This will allow LAN and wide area facilities to be used more efficiently and will reduce costs for the telco customers.

The scheme should also have the ability to decouple LAN and wide area progress. This includes solving the problem of a path which has not been completely upgraded. Users of upgraded LANs should be able to see immediate benefits when interoperating with other upgraded LANs, even across the wide area facilities. This is currently the situation with modem technology; anyone purchasing an new modem can plug it in and instantly get higher performance with anyone else who has a fast modem. In addition, it would also be desirable for the scheme to be as independent of the physical network equipment as possible; this will allow migration to improved technology to be made very easily.

A scheme which can support these properties is network striping. Network striping uses multiple low speed channels through a network in parallel to provide a high speed channel. It provides the desired bandwidth flexibility by allowing the use of as many low speed channels, called stripes, as necessary, assuming that the channels are available. The ability to add and remove stripes as desired also provides the decoupling between the local and wide areas by allowing users with upgraded LANs to simply increase the number of stripes they are using tp increase the bandwidth available to them.

Of course, there are advantages and disadvantages to network striping. In addition to the two characteristics listed above, network striping allows users to control the amount of

bandwidth available not only when they upgrade their LANs, but also to change the number of stripes used dynamically as the amount of traffic increases or decreases. This would be particularly useful in the case where there are usage charges on each stripe. Network striping also provides for some protection against network failures. If the bandwidth through the wide area is provided by one channel, if this channel fails, then there is no connectivity until it is repaired or replaced. If one of the channels in a striped connection fails, the connection will still be intact, just at reduced bandwidth. On the negative side, while network striping will provide high bandwidth to those users and applications which require it, it incurs a cost in complexity; it requires more work to send data over several channels in parallel than to just send data over one channel.

1.2.1 ATM and Network Striping

A large portion of this thesis addresses network striping as it may exist in an Asynchronous Transfer Mode[4] environment. One might argue that ATM will, in theory, be able to provide solutions to the same problems which network striping is attempting to solve. Users should be able to request virtual circuits with the amount of bandwidth they require, providing both the flexibility and the decoupling desired. However, there are still cases motivating the use of network striping with ATM. Virtual circuits provided by ATM will be limited to the amount of bandwidth which the physical channels can carry. Any user who wishes more bandwidth than that will need to use multiple physical channels, which is striping.

1.3 Contents of this Thesis

This introduction has provided justification for examining network striping in more detail. Chapter 2 will describe a framework with which all striping designs can be categorized and analyzed. In addition to the framework, Chapter 3 contains some previous work related to network striping and some of the tools available to solve the problems associated with it. Chapter 4 uses the framework to present all the possible striping cases; from these we have selected a few which we analyze in greater detail.

Chapters 5 and 6 will present the details of the existing experimental environment and the custom hardware which was developed to allow a complete interconnection. This experimental setup consists of three ATM-based networks (Figure 1-1). The VuNet is an

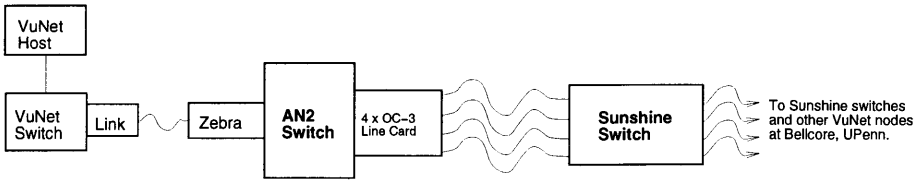


Figure 1-1: Network Striping Experimental Apparatus

experimental desk-area network developed at MIT. It is connected to AN2, a local-area network developed by Digital, through a piece of custom hardware called the Zebra. AN2 is connected to the Aurora[5] testbed's wide-area facilities through a 4 x OC-3c line card on the AN2 which is connected to an experimental OC-3c to OC-12 multiplexor developed by Bellcore. The OC-12 multiplexor can connect either directly to the facilities or to another OC-12 multiplexor attached to Bellcore's Sunshine switch[6].

Following the description of the experimental apparatus, the analysis of the remaining striping cases, along with some experimental results, is presented in Chapters 7 and 8. We will show that the ATM layer is an appropriate layer at which to implement striping. Lower layer striping is very constrained by the network equipment, while higher layer striping provides benefits only to those using that protocol. Finally, we state our conclusions and present some directions for future work.

Chapter 2

Striping Framework

Our goal is to identify and examine a subset of the possible striping cases which meet the criteria in the previous chapter. The first task, then, is to characterize the important aspects of any striping implementation so that all the cases can be easily identified. This chapter will present a general framework which we will use as a means to consistently describe possible striping implementations. As part of this framework, we will identify several degrees of freedom which capture all the important aspects of a particular striping implementation.

To be certain that all the striping cases are covered by the framework, consider how an arbitrary striping implementation might function. The connection between the source and destination hosts must be split into stripes somewhere along the route. The possible places in the network where this split may occur will affect how the striping implementation functions. We will refer to this as the *striping topology* and it will be the first degree of freedom.

Since it is the hosts at the edges of the network that are going to benefit from striping, the next thing to consider is the role that the hosts play in the striping implementation. This will be affected by the striping topology; in some cases the topology may dictate that the hosts may be required to be completely responsible for implementing the striping while in others they can be completely unaware of it.

Once the striping topology is known and the hosts' places in it are determined, data could potentially be striped. In order for this to happen, the data which is being transmitted by the hosts needs to be broken up into pieces which are then transmitted along the stripes in parallel. We will refer to these pieces of data as *striping units*. The choice of striping

unit is actually a subset of issues associated with the network layer at which the striping is implemented. The *striping layer* is affected by both of the previous degrees of freedom. For example, if the hosts are performing the striping, then the network to which they are connected affects the possible striping layer. If the striping is completely internal to the network then only the portion of the network over which the striping occurs affects the striping layer.

The three elements above allow the description of the most basic striping implementations, under ideal conditions. However, network conditions are rarely ideal, and so we will consider what difficulties must be dealt with in order to insure that a striping implementation will function correctly. Many difficulties associated with striping are the result of *skew*. Skew is a variation in the time it takes a striping unit to travel between the transmitting and receiving hosts on different stripes. The combination of skew and data loss will cause data to be delivered out of order and make it necessary to implement some sort of *synchronization* across the stripes to determine the correct order at the receiver.

The remainder of the chapter will examine each of the degrees of freedom in more detail, determining the various possibilities for each and examining how they affect each other.

2.1 Striping Topology

The first degree of freedom we will consider is that of striping topology. This is the description of where the path between two hosts is split into stripes. There are two basic reference topologies. In the simplest case, the connection is completely split into stripes from one end to the other; this is the *end-to-end splitting* case. In the second case, the connection is split between two nodes in the network; this is the *internal splitting* case.

2.1.1 End-to-End Splitting

The end-to-end splitting topology is shown in Figure 2-1. This topology is the simplest to analyze because the stripes are discrete physical channels for the entire length of the connection; there is no opportunity for units on different stripes to mingle or suffer from routing confusion. However, it is very rare that two end-systems are fully connected by several stripes; almost all LAN-based hosts are connected to a network through just one physical interface.

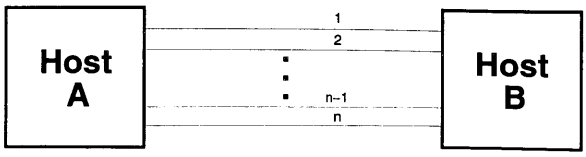


Figure 2-1: End-to-end Topology

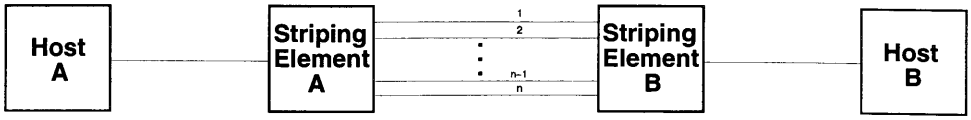


Figure 2-2: Fully Internal Topology

2.1.2 Internal Splitting

Figure 2-2 shows the internal splitting topology. This case is far more likely to exist; each host has only one connection to the network which then provides multiple possible paths between the source and destination. Unlike the end-to-end case, this topology adds the complexity of considering the effects of routing decisions at the point where the network is split to the analysis.

As network links are slowly upgraded, we expect that sections of paths between hosts will be collapsed into single channels. This will lead to striping topologies with cascades of internal splitting.

2.1.3 Combinations

The end-to-end and internal cases are the two simplest possible topologies; there are, of course, a large variety of topologies which may actually exist. For example, the hybrid topology (Figure 2-3) is half of the internal and half of the end-to-end topologies. This case could be applied to the rare case of an end-system with a normal network connection communicating with a host with multiple network connections.

Even more complex topologies are possible; as networks grow and paths added to provide redundant routes, a potential web of connections between any two geographically separate

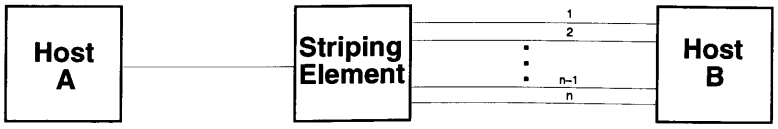


Figure 2-3: Hybrid Topology

hosts will exist. However, we will concern ourselves mainly with the two basic cases above, addressing more complex topology issues only when necessary.

2.2 Participation

An important aspect of a striping implementation is the role which the end-systems and networks play. The striping may occur without the knowledge of the end-systems; we will refer to this as the *passive* case. The other possibility is that the end-systems must perform some of the functions of the striping implementation. This will be referred to as the *active* case.

2.2.1 Passive End-Systems

In the passive case, the hosts are not aware that any striping occurs. This has the advantage that the hosts need not do anything special to transmit to any host, whether or not the two are connected through stripes. This allows the use of current host software, as well as reducing the overhead required in order to implement the striping. However, passive striping requires that the network handle the entire striping implementation, requiring more complex network equipment and/or modifications to the existing equipment.

Obviously, this case is limited to the internal and hybrid splitting network topologies. Thus, any time we refer to the passive case, we will assume that the striping topology is internal.

2.2.2 Active End-Systems

In the active case, the hosts are responsible for some part of the striping. This responsibility for the transmitting host can vary from handling the entire striping implementation to merely providing information to allow the network to properly carry out striping, depending on the other degrees of freedom. In almost all cases, it will be the responsibility of the receiving host to properly reassemble the striped data from the transmitter. The tradeoffs here are essentially the opposite of those in the passive case; the hosts require new software but the portion of the network containing stripes can use existing equipment.

Any time the topology is an end-to-end split, the hosts must handle all of the striping implementation.

2.2.3 Network Participation

The networks also play a role in any striping implementation. They implement the parts of the striping implementation not performed by the hosts. In the passive case, the network must handle the entire implementation, while in the active case, the responsibility may be shared in some way. In addition, since the network is providing the paths over which the striping is performed, an important part of a striping implementation is the guarantees which the hosts have from the network.

2.3 Striping Layer

As striping is a network function, there must be some way to associate it with the layers of the OSI network protocol stack. The networks over which the striping occurs will determine the possible layers at which striping can be performed. The lowest layer which is homogeneous across the portion of the network involved in the striping implementation is the lowest possible striping layer. In the passive striping case, only the network between the two points which handle the striping needs to be homogeneous at the striping layer; in the active case, the entire network between the source and destination must be homogeneous at the striping layer. An interesting thing to notice is that only the striping layer is required to be homogeneous; the layers above and below can be virtually anything. For example, striping at the ATM layer will support any combination of network layers and will operate using any combination of physical layers; striping at the network layer will support any combination of technologies which provides that layer.

2.3.1 Striping Unit

After determining the possible striping layers, we can make a list of the available striping elements. Figure 2-4 shows the striping units associated with the most likely layers in the ATM protocol stack.

Note that while the figure only shows striping units for the transport layer and below, the same idea holds all the way up to the application layer.

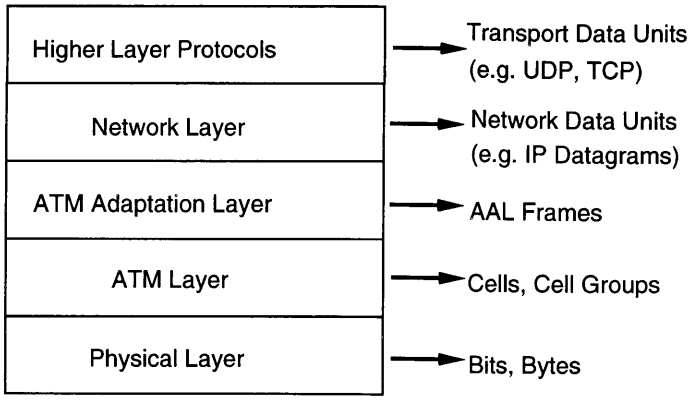


Figure 2-4: Identifying Striping Units by Protocol Layer

2.4 Implementation Issues

The previous degrees of freedom can be viewed as situational and architectural; they are concerned with aspects of striping which are probably not completely under the control of someone designing a striping implementation. Next, we will look at some implementation issues; things with which, given the situation, the designer has to be concerned.

2.4.1 Striping Unit Effects

One characteristic of the striping units at different levels is that they differ in bit length. This will have some profound effects on the network and on performance, which will be explored below.

Throughput and Packetization Delay

As the bit length of the striping unit varies, it will affect the throughput of amounts of data which do not require the use of all the available stripes. The aggregate throughput seen by bursts of data many striping units in length will be the sum of the throughputs of all the stripes. However, single striping units which are not transmitted in bursts will only see the throughput provided by the stripe on which they are transmitted. As the size of the striping unit increases, the hosts must be able to generate larger bursts of data in order to see the benefits provided by striping.

Similarly, as the size of the striping unit varies, the latency due to packetization delay increases. This effect occurs even with the use of a single channel, but it is aggravated by the fact that although the stripes provide an aggregate bandwidth equal to the sum of

the bandwidths, the packetization delay is determined by the bandwidth of an individual stripe. For example, the packetization delay of a 1000 bit striping unit on a single channel providing 100 Mbps is 10 μsec. However, if four 25 Mbps stripes are providing the same amount of aggregate bandwidth, the packetization delay will be the packetization delay of the individual stripes, which is 40 μsec.

Unbalanced Loads

The two previous effects were due to an overall change in length due to varying the striping unit. Some striping units may have the characteristic that their length may vary; for example, naively striping variable length IP packets at the network layer will result in each stripe carrying packets of differing lengths. Under some conditions, this could cause some stripes to be overwhelmed with traffic while others go grossly underutilized; as a simple example, take the case where small and large packets arrive in an alternating pattern to be striped over an even number of stripes. This problem is due to the simple round robin strategy used to deliver the packets to the stripes; we can potentially modify the round robin process to balance the load on each stripe[1].

One such modification, Deficit Round Robin[7], can be shown to provide the load balancing required. In this case, each stripe has a balance of bytes which it may take, some of which may be left over from previous rounds. At the start of a new round, a fixed quantity of bytes is added to the balance for each stripe. The standard round robin algorithm then proceeds with the following modifications. The size of an element placed on the current stripe is subtracted from its balance. If the new balance is larger than the size of the next element, then the next element is also transmitted on that stripe. This process repeats until the balance for the stripe is lower than the size of the next element. At this point, the same procedure begins with the next stripe.

Effect of the Striping Unit on Synchronization

The striping units at each layer have widely different formats. Aside from the bit/byte layer, all of them have some sort of header or trailer with various fields used to manage that layer. In addition, each layer is handled in a different fashion both by the nodes in

[1]Modifying the round robin process to perform load balancing will at least restrict the synchronization options available at the receiver, and possibly make a more sophisticated scheme necessary.

the network as well as by the hosts. The layer overhead is one place where synchronization information may be placed, and the layer specifies how the receiver will handle the incoming striping unit. This will cause the details of striping implementations at each layer to vary quite a bit as well, which will be seen in greater detail below.

2.4.2 Skew and the Need for Synchronization

As stated earlier, skew is a difference between the travel time on two stripes. Suppose that sources submit striping units to the stripes in a round robin order. The receiver will expect them to arrive in the same order[2]. Differences in the paths traversed by each stripe can introduce skew between the stripes and cause the striping units to arrive at the destination misordered.

There are three main causes of skew. These are the physical paths of each of the stripes, the multiplexing equipment which carry the physical channels, and switching elements through which each stripe passes. Different physical paths introduce skew by having different propagation delays due to traveling different distances. For example, if a striped connection between Boston and Seattle has two stripes which are routed directly between the two cities and two stripes which must pass through Dallas, then the transmission delay of the second pair of stripes will be longer simply because the path that they have to travel is longer.

Even if the stripes are constrained to follow the same physical path between the source and destination, skew may still be introduced by the multiplexing equipment along the route of the stripes. This is the case in the Aurora testbed, where the manipulation of the separate OC-3c channels in the OC-12 connections by the transmission and multiplexing equipment was enough to introduce skew between the SONET payloads[3]. Finally, switching elements may cause an even greater amount of skew if they introduce different queueing delays on each stripe.

Skew Assessment

When we begin to discuss some striping cases in more detail, we will want some idea of the worst case skew which will be introduced by the network. We can measure skew in two

[2]We will refer to one cycle by the transmitter or receiver through all the stripes as a *round*.
[3]The details of the Aurora facilities will be presented in Chapter 5.

different forms; for an absolute sort of measure we can consider skew as an amount of time in seconds. However, when considering how to design a synchronization method to protect against skew, we will want to know the amount of skew in terms of striping unit times; that is, how many striping elements can be received by the destination in the amount of skew time. We will consider the absolute skew introduced by the path difference, multiplexing equipment, and queueing delay separately, and then convert these times into striping unit times.

The amount of skew introduced by different physical path lengths depends upon the transmission media. We will assume the media to be optical fiber, which has a propagation delay of 7.5 μs per mile. If the difference in path length is M miles, then the skew introduced by this length is $7.5 \times M \mu$s. For example, if we consider the worst case path length difference in the U.S. to be about 1000 miles, then the skew this introduces will be 7.5 ms. On a world-wide scale, we might expect the worst case path difference to be as large as three or four thousand miles, or 30 ms. This amount of skew is fixed as long as the paths remain the same.

The amount of skew introduced due to the multiplexing equipment depends on the number of multiplexors in the path. We expect this skew to be relatively small based on our experimental results at the ATM layer, which we will present in chapter 7. Two co-routed OC-3c channels passing through 8 OC-3c/OC-48 multiplexors, a distance of approximately 500 miles, suffer a skew of only one cell time, which is approximately $2.2\mu s$. Since we do not expect significantly more multiplexors to be present on a channel of longer length, we will assume the worst case skew introduced by the multiplexing equipment to be less than an order of magnitude greater, around $11\mu s$. We expect that this skew will not vary much, remaining constant for long periods of time.

If the striping is performed at or about the network switching layer, then there will also be skew due to variable queue lengths on each path. The skew due to queueing delay will tend to vary quite a bit. If we assume two striping units arrive at two different switch ports which are each capable of buffering b striping units, then the worst difference in queue length which these two units will see is b. We can calculate the approximate skew introduced by this queue difference by calculating the time it takes for one striping unit to be removed from the queue. This will be approximately the size of the striping unit divided by the

outgoing line rate[4]. Multiplying the maximum queue difference times this time results in the maximum skew introduced due to queueing delay.

In order to convert the measure of skew in seconds to a measure of striping unit times, we calculate the length of a single striping unit time by dividing the length of a striping unit by the line rate into the destination host. By dividing the amount of skew time by this length, we get the amount of skew measured in striping unit times. For example, the length of an ATM cell time on an OC-3c link is 424 bits $\div$ 155 Mbps, which is 2.73μs.

The fixed skew components should be relatively simple to protect against; once they have been determined, the parameters of the synchronization method can adjusted to compensate. However, the variable components may vary too widely for the synchronization method to efficiently protect against them. The best approach in some cases may be to set some maximum amount of skew for which the synchronization method will compensate and to treat any data skewed by more than that amount as loss. For example, consider two paths which have an average skew of 7.5 ms, but whose maximum skew may be as great as a second. The receiving host may set some limit on the skew that it will attempt to compensate for. This value may depend on timeouts in the network software; if an entire packet hasn't arrived in some amount of time, the software may consider it lost anyway, so there is no need to worry about skew larger than that amount of time.

Skew vs. Data Loss

The need for synchronization arises due to the combination of two network effects: skew and striping unit loss. Due to skew, units on some stripes may arrive much later than expected. This would not be a problem if it were guaranteed that they would arrive eventually; the receiver could simply wait for striping units on stripes with longer delays to arrive before proceeding with the reassembly of the incoming data. However, the possibility that striping units may be lost complicates matters. In figure 2-5, striping unit 2 has been lost in the middle of transmission. Instead of the receiver correctly placing the units in sequential order, the order will be 1, 6, 3, 4, 5, etc.

If one of the striping units at the end of the data burst is lost, the receiver may be stuck

[4]This calculation becomes more complex as different ports on the switch may have different line rates; the striping units we are examining will have to wait not only for traffic following the same path, but also for cross-traffic.

Round 3	Round 2	Round 1	
Unit 9	Unit 5	Unit 1	Stripe 1
Unit 14	Unit 10	Unit 6	Stripe 2
Unit 11	Unit 7	Unit 3	Stripe 3
Unit 12	Unit 8	Unit 4	Stripe 4

Figure 2-5: Improper Reassembly Due to a Lost Striping Unit

waiting for data which will never arrive. In either of these cases, we have to rely on a higher layer protocol to either catch the erroneously reconstructed data units or to time out and signal the striping recovery algorithm that something is broken.

The algorithm just described compensates for skew but, as we have seen, offers no protection against lost striping units. The other obvious simple choice is to design the algorithm so that it detects lost striping units and ignores the possibility of skew. Assume again that the receiver is expecting striping units to arrive in a round robin order. When a striping unit arrives on a stripe, the receiver then goes to the next stripe and waits for a striping unit to arrive on that stripe. Since the receiver is not aware of the possibility of skew, if a striping unit arrives on the next stripe in the round robin order before one arrives on the current stripe in the order, the receiver believes that it has lost a cell.

This is illustrated in figure 2-6. The second stripe has enough additional delay such that striping units arrive one round later than they would if there were no skew. In this case, the receiver takes unit 1 from the first stripe and then moves to the second stripe. Since unit 3 arrives before unit 2, the receiver assumes that unit 2 was lost because it believes that the only order in which the units can arrive if they are not lost is 1, 2, 3, etc. However, since unit 2 arrives during the next round, it will be mistaken for the striping unit which really belongs in that round. Thus, the receiver will place the striping units in the order 1, 3, 4, 5, 2, 7, etc, which is clearly incorrect.

Obviously, in either case, something extra needs to be done to compensate for skew and

Round 3	Round 2	Round 1	
Unit 9	Unit 5	Unit 1	Stripe 1
Unit 6	Unit 2		Stripe 2
Unit 11	Unit 7	Unit 3	Stripe 3
Unit 12	Unit 8	Unit 4	Stripe 4

Figure 2-6: Improper Reassembly Due to Skew

data loss to allow the data arriving at the destination to be reconstructed in the proper order. In the next chapter, we will examine some well-known techniques for maintaining synchronization on a single channel and show how these can be applied to maintaining synchronization across a number of stripes.

2.5 Summary

Striping topology, host participation, and striping layer define the situation and architecture, and synchronization are the primary degrees of freedom associated with the implementation. The remaining tools which we are missing are those concerned with the various options for implementing synchronization.

The largest problem associated with striping is that data loss and skew cause data to arrive at the destination misordered, something which does not occur when using a single channel. In the next chapter, we will examine some well-known techniques for dealing with synchronization on a single channel and show how these apply to striped channels.

The next chapter will also present some previous work related to network striping. These examples will demonstrate some of the validity of our reference model. In addition, they will present some examples of synchronization across multiple channels. Finally, they represent the exploration of the striping space which has already been performed; when determining the striping cases which we wish to investigate we will be able to set these aside.

Chapter 3

Previous Work

This chapter presents several flavors of previous work related to the striping problem. Recall from the previous chapter that the fundamental problem with striping data across multiple channels is maintaining or recovering the proper data order; the solution to this problem is to use some form of synchronization across the stripes. The first portion of previous work will examine the known techniques for synchronization on a single channel; we will find that similar approaches will apply to the multiple channel synchronization problem as well.

We will also examine several existing striping implementations, related to both networks and to disks. The discussion will be divided into ATM-related striping, ISDN striping, HiPPi striping, and disk striping. This work will present a good test of the framework developed in the previous chapter, as well as providing some examples of specific synchronization schemes.

3.1 Single Channel Synchronization

When transmitting data through a digital channel, there are several requirements beyond the physical equipment which provides the channel. Simply transmitting bits of data down the channel does not provide any way for the receiver to know when and where to start expecting data. Some means is necessary to allow the receiver to find the boundaries of bursts of data from the random bit stream it is receiving. To accomplish this, the transmitter encapsulates any data which it transmits in some form of frame; the receiver knows how to find the frames, which allows it to extract the higher layer data from the channel. In addition to framing, the receiver requires some way of determining if data has been lost

during transmission.

3.1.1 Framing

The process of framing involves surrounding the data with some recognizable pattern at the transmitter. This pattern is then used by the receiver to correctly recover the framed data. There are two basic ways to place the framing pattern into the data stream; patterns can either be placed at periodic or aperiodic intervals. Some combination of these two methods can be used as well.

Periodic Framing Patterns

This technique places an easily recognized pattern at periodic locations in the data stream at the transmitter. Initially, the receiver searches the incoming data for this pattern. Once it has found it, the receiver considers itself synchronized and then continuously verifies that the pattern reappears at the proper times. If the pattern does not appear, the receiver has lost synchronization and must return to the state where it is hunting for the pattern. A good example of an existing system which uses a periodic framing pattern is SONET[8]. The beginning of a basic SONET frame is marked by two bytes which are used by the transmission equipment to align to the start of a frame.

Aperiodic Framing Patterns

This technique is virtually identical to the one above, except that the pattern for which the receiver is searching can appear at random points in the data stream. This means that the receiver must always be searching for it. Many asynchronous data link control layers, such as HDLC[9], use an aperiodic framing pattern in the form of a flag which marks the end of a link idle period.

Spoofing

One problem with inserting a pattern which needs to be recognized by the receiver into the data which needs to be processed and forwarded by the receiver is the possibility that the real data will contain the framing pattern. This phenomena is known as *spoofing*. There are several techniques which are used to avoid spoofing. One method, used by SONET, is scrambling; the actual data is subjected to a transformation before being placed into the

SONET frames which reduces the probability that the data bytes will contain the start of frame byte. Another method, used by HDLC, is bit-stuffing. In this case, the flag which marks the beginning of a frame contains a large number of consecutive ones; zeros are inserted into the data to prevent the same number of consecutive ones from appearing anywhere but the flag.

Yet another scheme for avoiding spoofing is coding. These schemes use a code to map a fixed number of bits of real data into a word with a larger number of bits which is then transmitted. The receiver performs the inverse mapping. Since the words which are transmitted contain more bits than the data words which are mapped into them, there will be some words left over which are reserved for the link protocol. These reserved words can be used to fill the transmission channel when the link is idle and to frame the real data so that it can be properly recovered[1]. The Hewlett-Packard G-Link chipset is an example of a transmission system which uses a coding scheme; it allows either 16B or 17B/20B or 20B or 21B/24B[10].

3.1.2 Loss Detection

Another important requirement for transmission on a single channel is the ability to detect the loss of data elements. Again, there are several well-known schemes to accomplish this. One possible way is to attach some sort of a tag to each data element. Using the tags, the receiver can detect missing data. Sequence numbering is one simple example of a tagging scheme. Another method is to calculate the length of the frames being transmitted to the receiver and then to convey the length to the receiver. This is the method used to detect missing cells in AAL5 frames.

Note that a desirable aspect of a framing scheme is to convert framing errors into data losses. This will prevent incorrectly constructed frames from being passed up to the next layer as valid data.

3.1.3 Synchronization Methods Applied to Striping

We stated earlier that the primary difficulty with implementing striping is the possibility that the combination of skew and data loss will cause data to arrive at the destination

[1] In addition, the mapping into code words is arranged so that approximately the same number of ones and zeros are transmitted over the link, allowing the receiver to obtain the DC balance.

Round 6	Round 5	Round 4	Round 3	Round 2	Round 1	
Unit 5	Unit 1	Unit 13	Unit 9	Unit 5	Unit 1	Stripe 1
Unit 14	Unit 10	Unit 6	Unit 2			Stripe 2
Unit 3	Unit 15	Unit 11	Unit 7	Unit 3		Stripe 3
Unit 8	Unit 4	Unit 16	Unit 12	Unit 8	Unit 4	Stripe 4

Figure 3-1: Tagging Striping Units Across the Stripes

misordered. In order to prevent or compensate for this, some method of synchronization and data loss protection across the stripes are necessary. The standard tools for single channel synchronization can be applied here in order to accomplish this.

The first general approach that can be used is the tagging method applied to striping units across all the stripes (Figure 3-1). The first striping unit on the first stripe would get the first tag, the first unit on the second stripe would get the second, etc. This provides the destination with an absolute ordering scheme which it can use to determine not only the proper order of the incoming striping units but whether any units are lost as well. The largest drawback is that the cost in bits per striping unit increases rapidly as both the number of stripes and the amount of skew which must be protected against increases.

Another approach is to use the tagging method on a per stripe basis; the striping units on each stripe can then be tagged with the same set of tags (Figure 3-2). Data loss on each stripe is detected, and if there are n stripes, this scheme requires either 1/nth of the number of tags of the previous scheme or provides protection against n times more skew. The problem then becomes aligning the stripes so that the receiver performs its round robin rotation on units from the same transmitted round on each stripe. This has a simple solution, however, if we ensure that units from the same round at the transmitter get the same tag. The tag then becomes not only a sequence number on each stripe, it also becomes an indicator of the round number. The receiver can align each stripe on the same round number and be assured that, as long as the number of tags is long enough to protect against the skew introduced by the network, it will correctly reconstruct the original data. In a sense, the round numbers make up a type of frame on each stripe; the start of frame pattern is the first tag, and the length is the number of tags on each stripe.

Finally, there is the framing approach which is completely divorced from per-element

Round 6	Round 5	Round 4	Round 3	Round 2	Round 1	
Unit 2	Unit 1	Unit 4	Unit 3	Unit 2	Unit 1	Stripe 1
Unit 4	Unit 3	Unit 2	Unit 1			Stripe 2
Unit 1	Unit 4	Unit 3	Unit 2	Unit 1		Stripe 3
Unit 2	Unit 1	Unit 4	Unit 3	Unit 2	Unit 1	Stripe 4

Figure 3-2: Tagging Striping Units on Individual Stripes

Round 6	Round 5	Round 4	Round 3	Round 2	Round 1	
Data Unit	Framing Unit	Data Unit	Data Unit	Data Unit	Framing Unit	Stripe 1
Data Unit	Data Unit	Data Unit	Framing Unit			Stripe 2
Framing Unit	Data Unit	Data Unit	Data Unit	Framing Unit		Stripe 3
Data Unit	Framing Unit	Data Unit	Data Unit	Data Unit	Framing Unit	Stripe 4

Figure 3-3: Using Striping Elements with Metaframing Patterns

tagging. We will refer to this as metaframing. The transmitter places a well-known pattern in a striping element on each stripe at either periodic or random intervals (Figure 3-3). The receiver looks for this pattern and uses it to align the stripes so that it receives striping units from the same transmitter round during each receiving round. Failure to detect the pattern on one of the stripes means that the receiver has lost synchronization. This method provides synchronization and data loss protection reasonably cheaply in terms of the number of bits used. Protection against larger amounts of skew can be achieved by increasing the length of time between synchronization patterns. However, there is a cost in the lack of granularity in loss detection; the receiver can know that it has lost data in a frame but not necessarily which striping element in a frame was lost. There is also a cost in wasted time and network resources. A longer frame length will require more time before loss of synchronization can be detected; all of the striping units in a frame that has lost a striping unit may need to be dropped.

Of course, the metaframing and tagging approaches can be combined; the per stripe tagging scheme can be viewed as metaframing with a metaframe size of one element, and a tag which indicates the location of the metaframe in a larger framing entity. To provide more protection against skew, either the length of the tags or the length of the metaframes

could be increased, depending on what is required.

When we analyze possible striping cases in later chapters, we will only address the general tagging and framing synchronization possibilities, with the awareness that the combinations can be generated in a straight-forward manner.

3.2 ATM Network Striping

We will examine two pieces of work which are concerned with striping related to ATM networks. The first is the Unison testbed ramp. The Unison testbed was one of the first efforts to build and study a network based on ATM[11]. Built prior to the standardization of the ATM cell, the Unison ATM cells consisted of 32 bytes of payload and 6 bytes of header and trailer combined. The testbed itself consisted of Cambridge Fast Rings (CFR) located at four sites; the sites were linked by European primary rate ISDN[2]. The Unison ramps were developed to connect the CFR at each site to the ISDN network. Each CFR operated at 50 Mbps, so the ramps striped data over the ISDN network in order to provide communication links of reasonable bandwidth, referred to as U-channels, between the sites.

The second piece of work related to ATM striping the the Osiris host interface. Osiris is an ATM-based host interface for the DEC Turbochannel[12]. It was developed at Bellcore as part of the Aurora gigabit testbed[13]. It is connected to the Aurora SONET facilities at the OC-12 rate, and accomplishes this by generating four STS-3c signals which are placed into SONET frames and multiplexed into an OC-12 by a separate multiplexing board.

3.2.1 Unison

Each site in the Unison testbed had a number of local client networks. A single CFR at each site was used to interconnect these networks using ATM. In addition, each CFR was connected to a Unison ramp, which was in turn connected to the ISDN network. In order for hosts to communicate with hosts on other local networks, a local network management facility would maintain ATM connections between the two networks through the CFR, and a higher layer protocol would use these ATM connections as part of the path between the hosts. In order to communicate with hosts at other sites, a local ATM connection would

[2]ISDN in Europe provides 30 64 kbps B-channels for data and 2 64 kbps D-channels for signalling, as opposed to ISDN in the US which provides 23 64 kbps B-channels and one 64 kbps D-channel.

be set up between the local networks and the ramps on each CFR. The ramps would also create U-channels between each site for the transmission of cells; as cells passed from one ring to another they would be mapped from the local address space of the first ring into the address space of the second.

The splitting in the Unison testbed is internal to the network; data between two hosts travels over a single physical channel until it reaches the ramps. At this point, it still travels over a single logical U-channel. However, the U-channel is made up of several physical B-channels over which data is striped. Since the creation and maintenance of the U-channels from the B-channels is purely the responsibility of the ramps, this implementation is clearly a passive striping case. To determine the striping layer and synchronization method, let us consider how the striping is implemented[14].

Unison Striping Implementation

Initially, the transmitting ramp fills the ISDN frame with a particular byte indicating that the transmitter is idle[3]. When there is data to be transmitted, it determines how much bandwidth is requested for the data and then requests the appropriate number of B-channels from the ISDN network over the signalling channel. The network informs the transmitting ramp of the slot numbers allocated for the transmission. These numbers are stored in a slot map and correspond to a U-channel. The transmitting ramp then generates an appropriate synchronization pattern for each U-channel. The synchronization pattern consists of four consecutive bytes. The first and third bytes are two particular bytes to mark the synchronization pattern, the second byte is that slot's position in the U-channel, and the fourth byte is the number of slots which currently make up that U-channel.

When transmitting data, the ramps place the synchronization pattern into the slots of a new U-channel in the first four frames. It then sends the cells from the CFR which have been queued up for transmission over that U-channel by placing the bytes of the first cell into the appropriate slots in round robin order, then the second cell, etc. The order in which the slots are used is the same as the order given by the second byte in the synchronization pattern on each B-channel. After some fixed number of ISDN frames, the transmitter places the synchronization pattern into the slots, inserting it into the stream of data bytes which it is transmitting. At this point, the transmitting ramp has the ability to add or drop

[3]The frame which refer to here is made up of 32 one byte slots, one slot per 64 kbps channel.

B-channels from the U-channel by changing the parameters in the synchronization pattern.

Skew affects the striping implementation in the following manner. In ISDN primary or basic rate service, bytes from a given B-channel will always arrive in the same slot in frames at the receiver. There is no guarantee that the position in which the data is received will be the same slot in which it was originally placed. The individual channels are switched separately through the telephone network, so each channel may experience a different propagation and switching delay. Thus, data placed in adjacent slots at the transmitter may appear at the receiver in non-adjacent slots; if the delay is large enough, data may even appear in a different frame from its original neighbors. This is the source of skew, and in order to make ISDN striping possible, there must be some way to compensate for or eliminate skew and allow the data to be properly reassembled.

The receiving ramp is informed of the slots associated with a particular U-channel when the transmitting ramp requests the slots. The receiver buffers a number of ISDN frames and searches the slots of each new U-channel for the synchronization pattern. Using the information in the pattern, it is able to determine which slot number in the frame is associated with the slot position in the U-channel. It is also able to determine whether it has moved to a different frame, and how many frames it is offset from the first frame in which it received data for that U-channel. This information goes into a slot offset map, which the receiver uses to reconstruct the incoming data for each U-channel. If the synchronization pattern fails to arrive after the designated number of ISDN frames, then the receiver assumes that it has lost synchronization and goes about reconstructing the offset map for that U-channel just as before.

Since the ramps are submitting bytes rather than entire cells to the B-channels during each round, the striping implementation is clearly operating at the byte layer. The synchronization method uses aperiodic framing to find the beginning of a burst on each stripe and then used periodic framing until the burst ends. The receiver compensates for skew by aligning the framing patterns on each stripe, and so the overall synchronization method uses framing as well.

3.2.2 Osiris Host Interface

Even though Osiris is attached to the SONET facilities at the OC-12 rate, it really has four individual STS-3c connections. To operate at OC-12 rates it needs to stripe data across

these four channels. This is the rare case of a host having multiple connections to the network; any striping implemented with the Osiris board will be an end-to-end splitting topology. As such, it must also be an active striping case.

Osiris transmits and and receives data on the four channels in the form of ATM cells. Originally, it was hoped that by requiring the OC-3c channels to be in the same optical fiber throughout transmission, skew would be prevented and the striping would remain simple. Unfortunately, when Osiris first transmit cells over the SONET facilities, it revealed that there was still skew introduced between the OC-3c payloads. This made establishing the original cell order much more difficult.

Two mechanisms were proposed to solve this problem end-to-end. The first was to put a sequence number in each cell; this would allow the receiver to know the correct order in spite of the skew. On the Osiris board, the sequence number was used to determine the host memory address at which to store the arriving cells. The possibility that the first cell received would not be the first cell of the higher layer frame made the reassembly code reasonably complex[15]. Another problem was the possibility that the skew introduced by queueing delays would be unbounded, making it difficult to guarantee a large enough series of sequence numbers.

The second method was to break any outgoing packets into four AAL5 frames and stripe these instead; the cells within an AAL5 frame would stay ordered, and the receiver would be able to concatenate the contents of the frames on the four channels to reassemble the original packet. The one problematical case was the case of a higher layer frame which is less than four cells long. This was handled by using another framing bit in the ATM header which indicates the end of the higher layer data unit which is being striped as AAL5 frames. This approach would require a change to the existing standards to support the extra framing bit.

The first scheme is clearly a case of striping at the ATM layer using a simple tagging scheme. We will consider the second case to be a case of adaptation layer striping, even though one adaptation layer frame only carries a portion of a higher layer data unit rather than the entire thing. The synchronization scheme is an aperiodic framing pattern in the form of the end-of-frame bit for an AAL5 frame.

3.3 ISDN Striping

This section will examine striping implementations over ISDN networks. Since ISDN service provides the customer with a set of 64 kbps channels, a natural idea is to use them in parallel in order to increase the available bandwidth. We have already seen one example of ISDN striping in the Unison testbed; in this discussion we also described how skew is introduced into ISDN networks. The implementation we will examine in this section is the BONDING standard[16]. BONDING was developed by a consortium of companies building ISDN inverse multiplexors so that their products will be able to interoperate.

3.3.1 BONDING

The BONDING standard defines a frame structure used to stripe data across multiple B-channels. The frame structure is used to encapsulate the bytes on each of the B-channels separately. The receiver uses the framing information on each channel to align them correctly.

A BONDING frame is defined to be 256 bytes long; 64 frames make up a multiframe (Figure 3-4). Byte 64 in a frame is the frame alignment word (FAW), byte 128 is the information channel (IC), byte 192 is the frame count (FC), and byte 256 is the CRC. The frame alignment word is just a specific byte which is used by the receiver to find frame boundaries. The information channel allows for in-band communication between the transmitter and receiver which will be discussed later. The frame count is a six bit counter which is the current frame's position in a multiframe; it is used to determine the amount of skew between channels at the receiver. Thus, we can see that the framing scheme uses a combination of periodic framing and tags. The frame defines a structure which the transmitter and receiver can use to protect against skew and exchange data in a coherent manner. By tagging each frame, protection against larger amounts of skew is available.

The proper order of the channels at the receiver is determined using the information channel. Data in the information channels is contained in 16 byte frames; these frames are transmitted using the IC byte in sixteen consecutive BONDING frames. The start of an information channel frame is designated by another specific byte. The next byte contains the channel ID, which is an individual B-channel's place in the round robin order used to deliver data to the B-channels. From the channel ID in each channel, the receiver can find

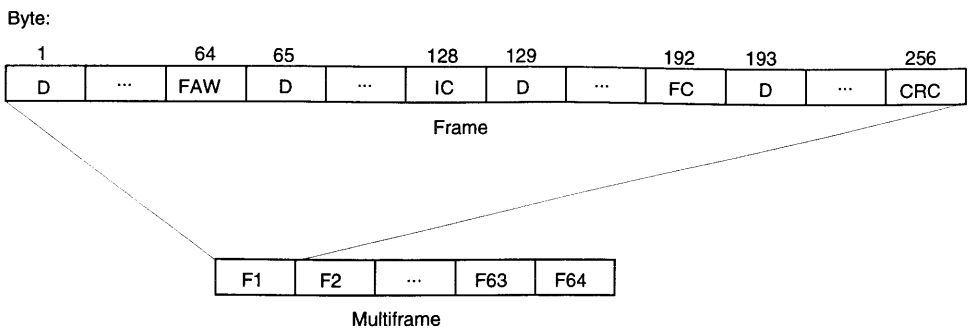


Figure 3-4: BONDING Frame/Multiframe Structure

the correct order of the incoming bytes. The information channel frame also contains a group ID, so that multiple logical connections can be maintained between two BONDING endpoints simultaneously. There are also fields which allow for the addition and deletion of channels during the course of a connection.

Just like the Unison ramps, the BONDING hardware performs striping at the byte layer and is clearly an internal splitting case. The partitioning of host responsibility depends on the operating mode used. The BONDING standard supports four modes of operation; they are:

1. Mode 0: This mode simply stripes data without adding any of the framing overhead to compensate for skew. It is intended for transmissions where the data synchronization is accomplished in some other fashion. This is an active striping case.

2. Mode 1: This mode provides minimum striping functionality. It adds the overhead to the channels in order to synchronize them, and then removes them to transmit data. Some external means is expected to detect a loss of synchronization. We consider this to be an active case as well, since it requires the hosts to participate in some part of the striping implementation.

3. Mode 2: This mode is the same as mode 1 except that the synchronization overhead remains in each channel through the course of the connection so that errors and loss of synchronization may be detected and corrected by the BONDING equipment. This is a passive striping case.

4. Mode 3: This mode is virtually identical mode 2 except that the BONDING equipment adds another B-channel, if possible, to compensate for the amount of bandwidth used

by the synchronization overhead. This is also a passive striping case.

3.4 HiPPi Striping

Within the gigabit testbeds[13] there has been some work done attempting to stripe HiPPi packets across the wide area facilities. This is part of an effort to link geographically separated supercomputers of different types in order to get an advantage in processing speed. Although this thesis focuses on the ATM protocol stack, we will briefly examine two HiPPi-based striping implementations to demonstrate the flexibility of the framework we have developed. HiPPi replaces the ATM and adaptation layers in the reference model; we will view it as a link layer protocol.

The HiPPi/ATM/SONET network interface (HAS) developed in the Nectar testbed connects two HiPPi-based LANs through a high-speed SONET/ATM network[17]. The HiPPi/SONET gateway, part of the Casa testbed, connects two HiPPi-based LANs through high-speed SONET channels.

3.4.1 HiPPi/ATM/SONET Network Interface (Nectar)

The HAS multiplexes 8 OC-3c channels into an OC-24 connection. Individual HiPPi connections may use some subset of the 8 connections, depending on how much bandwidth is required, and may add or drop channels as necessary. There are two ways in which HiPPi data can be placed into AAL frames and transmitted across the SONET facilities: a Data Only format and a Complete HiPPi format. In the first format, the data is extracted from a HiPPi packet reaching the HAS; this data is then placed into an AAL frame and segmented into ATM cells for transmission on one of the OC-3c channels. At the far end, the payload of the AAL frame is recovered and placed into a HiPPi packet for transmission to the destination. In the second format, part of the HiPPi overhead is included with the data which is placed into the AAL frame. This data is used for routing decisions at the receiver.

This is another internal splitting topology. In this situation, however, the HAS is only providing access to the multiple OC-3c stripes; it implements no special striping scheme. Insuring that the striped packets can be placed in the proper order becomes the job of the hosts, so this is an active striping case.

Depending on the format of the data placed into AAL frames, the striping layer can

be either the network layer in the Data Only case or the link layer in the Complete HiPPi case. One may ask why the adaptation layer was not chosen in the Data Only case, since the actual entities which are transmitted across the stripes are AAL frames. To justify this decision, we turn to the requirement of a homogeneous striping layer. The only data which survives the transition from the HiPPi realm to the ATM realm and back again is the network layer data encapsulated in the HiPPi packets. This would not be important if this were a passive striping case, but we have already determined it to be an active case. Thus, any sort of synchronization needs to be done between the hosts and must use the network layer because synchronization information placed in the HiPPi, ATM, or AAL overhead will be destroyed at the boundaries between HiPPi and ATM. However, in the Complete HiPPi case, some of the HiPPi overhead is preserved when moving from the HiPPi to ATM realm and back, and so the link layer becomes the lowest homogeneous layer.

3.4.2 HiPPi/SONET Gateway (Casa)

The HiPPi/SONET Gateway also multiplexes 8 OC-3c channels into a single OC-24 connection. The gateway uses up to six of these stripes to send actual HiPPi data. The other two stripes are used for forward error correction and as hot spares. These are additional benefits from striping which we have not addressed in this thesis; we are concerned mainly with increasing network performance.

Yet again, this case is one of internal splitting. The Gateway provides a path between a single HiPPi channel and a number of SONET OC-3c channels, but performs none of the striping functions besides submitting incoming packets to the stripes and retrieving them from the stripes at the far end without verifying the order. This last task is left for the hosts to manage, making this another active striping case.

Similar to the HAS, the gateway stripes entire HiPPi packets across the available connections. In this case, though, there is no conversion between HiPPi and ATM; the HiPPi frames are maintained intact throughout the entire path; the striping layer is the HiPPi layer.

3.5 Disk Striping

Disk striping is one proposed solution for closing the gap between the speed of a high-performance computer's CPU and it's I/O devices. In this scheme, several disks are read/written simultaneously, providing an aggregate I/O bandwidth which is approximately equal to the sum of the individual disks. Some method may be used to insure that data is available even in the event of individual disk crashes. Katz, Gibson, and Patterson, in [18], present three degrees of freedom associated with the organization of this disk array.

1. Degree of Interleaving: Logical disk blocks can be fragmented into a number of identical pieces; these pieces are then written to individual disks in the array. The range of the size of these pieces is the degree of interleaving. Data can be interleaved by bit, byte, block, track, or cylinder.

2. Arm Independence: The arms of the individual disks can either seek as a unit or move independently.

3. Rotation Independence: The spindles of each disk may rotate together, so that the same sector is moving beneath the head of each disk at once, or the spindles may rotate independently.

A number of different disk array organizations are then examined based on various choices for the degrees of freedom. Some of these organizations are roughly analogous to some of the possible network striping cases:

- Synchronous Rotation and Unit Arms: Similar to the ideal case of network striping at the physical layer in the absence of skew, the disk array appears as a single disk with more capacity and transfer bandwidth.

- Asynchronous Rotation and Unit Arms: The difference from the first organization is that the disks rotate asynchronously. This is similar to network striping at the physical layer with skew present.

- Asynchronous Rotation, Independent Arms, and Block Interleaved: In this case, rather than spreading a logical disk block across the individual disks, the logical blocks themselves are the element which is interleaved. With fixed sized blocks, this is similar to network striping at the cell layer.

There are two other organizations which are not analogous to new striping cases, but which differ from those above in that they provide some level of redundancy to protect against the failure of a disk.

- Asynchronous Rotation, Independent Arms, Redundant, and Non-interleaved: An image of the logical disk is maintained simultaneously on multiple real disks; a failed disk will not affect the availability of data from the array. Read performance increases by scheduling multiple reads properly, but write performance is the same or worse.

- Asynchronous Rotation, Independent Arms, Redundant, and Block Interleaved: Data availability is assured by keeping multiple copies of the logical disk blocks on different disks. Large sequential reads are supported by block interleaving the copies of data across multiple disks and scheduling independent transfers for each block.

The disk striping schemes presented above are all schemes which deal with striping across the disks after the data has been partitioned into logical blocks by the filesystem; they operate at a fairly low layer. This is also somewhat analogous to passive network striping; the filesystem generates disk blocks for the aggregate disk it believes exists, and some mechanism below it transparently handles the striping scheme. In [19], Vaitzblit discusses the implementation of disk striping at higher layers; in these cases, the filesystem or even the application understands that there are a number of disks available and organizes the data it is generating so that it can direct portions of the data to each disk. This is roughly like the active network striping case.

3.6 Conclusion

At the beginning of this chapter, we covered some general, well-known tools for the synchronization of a single digital channel. These tools were then applied to the case of synchronizing several independent channels. We listed two basic schemes to accomplish this: tagging and framing; a combination of these two is also possible.

Next, we presented some of the previous network striping efforts in terms of the framework which we have developed. The results are summarized in table 3.1. This represents a portion of the striping space within our framework which has already been explored to some extent. In the next chapter, we will present the entire striping space, from which we

Striping Effort	Striping Layer	Topology	Host Participation	Synchronization Scheme
Unison	Physical	Internal	Passive	Per. Framing
Osiris	ATM	End-to-End	Active	Tagging
	Adaptation	End-to-End	Active	Aper. Framing
BONDING (Modes 0 & 1)	Physical	Internal[4]	Active	Per. Framing w/ Tagging
(Modes 2 & 3)	Physical	Internal	Passive	Per. Framing w/ Tagging
HAS	Link	Internal	Active	?
	Network	Internal	Active	?
H/S Gateway	Link	Internal	Active	?

Table 3.1: Striping Cases Explored by Previous Efforts

will identify a few cases which we believe to be the most interesting for further analysis.

Chapter 4

Exploring the Striping Space

The two previous chapters have laid out the framework and tools needed to analyze any given striping case. In this chapter we will examine the striping space created by the framework. We will begin by presenting all the striping possibilities defined by the striping layer, topology, and host participation. Some of these cases will not make any sense to implement or will be uninteresting for a variety of reasons, others will have already been explored by the previous work, and several more will be shown to be equivalent to other cases. After we set these aside, we will be left with a set of new and interesting cases which we will analyze in more detail.

4.1 The Striping Space

Since the striping layer has the largest effect on the specifics of the implementation by affecting both the striping unit and the synchronization options, we will first divide the striping space into smaller subspaces according to layer. Each layer subspace will then be divided by network topology and host participation. We choose these two degrees of freedom, rather than synchronization, because they are what may be viewed as situational or architectural issues while synchronization is an implementation issue.

The resulting possibilities are shown in Table 4.1. From this table we can immediately identify several of the cases which make no sense to implement. For example, the case of passive striping at the transport layer is not sensible. The transport layer is not visible to nodes within the network and so the only way transport layer striping can be performed is by the hosts; this requires transport layer striping to have active host participation. For a

Striping Layer	Topology	Host Participation	
Physical	Internal	Passive	2
		Active	1
	End-to-End		3
ATM	Internal	Passive	3
		Active	
	End-to-End		2
Adaptation	Internal	Passive	3
		Active	
	End-to-End		2
Network	Internal	Passive	1
		Active	IP
	End-to-End		3
Transport	Internal	Passive	1
		Active	TP(4)
	End-to-End		3

1 Cases Which Make No Sense to Implement
2 Cases Covered by Previous Work
3 Cases Deemed Similar to Other Cases

Table 4.1: Initial Striping Space

detailed justification for the elimination of the cases of types 1 and 3, see appendix A.

4.1.1 Relevance of the Previous Work

Several cases have already been explored by the previous work. Although these are valid striping possibilities, we will provide some justification for setting them aside rather than spending any further time investigating them.

Physical Layer Striping - Unison and BONDING

The Unison ramps and BONDING both operate at the physical layer, and both are passive striping cases. We do not wish to consider these cases because they do not meet our criteria for what a striping implementation should accomplish. Both of these works are intimately tied to the type of network which provides the stripes. Striping at the physical layer, as these two cases do, is the most restrictive form of striping and does not support the slow migration of the network to new technology. In order to stripe at a certain layer, that layer must be homogeneous throughout the parts of the network which support the

striping. To make the physical layer homogeneous, these portions must be constructed from equipment which supports the same physical layer. This eliminates the ability to upgrade just a portion of the network, for when network equipment is upgraded, very rarely does the physical layer stay the same. In fact, it is usually the physical layer which imposes the bandwidth restrictions in the first place.

ATM and Adaptation Layer Striping - Osiris

There are two striping cases which are covered by the Osiris work. Both are active end-to-end cases, one at the cell layer and the other at the adaptation layer. The fundamental problem with end-to-end cases such as this is that they don't model the real world very well. Almost every host has only one physical connection to the network, so to design striping schemes which rely on multiple connections will only provide a solution for a very small group of people. So, in our search for solutions which are applicable to more realistic networks, we will not consider this case.

Network and Transport Layer Striping - IP and TP(4)

If we consider specific network and transport layer protocols, then we can find striping somewhat covered IP and TP(4). IP provides the ability to handle misordered IP datagrams, but no real work has been done to make striping at this layer efficient. TP(4) allows the use of multiple paths throught the network to be associated with a single transport layer connection. We will mention these two cases in slightly more detail in chapter 8 when we talk about striping at layers above the ATM layer.

4.1.2 Determining Equivalences

The final step in isolating the cases to be considered is to eliminate cases which are either equivalent to or a subset of some other case or cases. Most of the end-to-end and some of the internal and passive cases will fall into this category. For example, even if the Osiris work had not covered end-to-end cell striping, we would eliminate this case as a subset of the active, internal cell striping case. This is justified by considering that the only problem to solve in the end-to-end case is the synchronization of data across the stripes. Any of the synchronization methods developed for the active, internal case should be applicable to the end-to-end case as well.

Striping Layer	Topology	Host Participation
ATM	Internal	Active
Adaptation	Internal	Active
Network (e.g. IP)	Internal	Active
Transport (e.g. OSI TP(4))	Internal	Active

Table 4.2: Interesting Cases

4.2 Interesting Cases

Table 4.2 shows the striping cases discussed in more detail. Notice that they are all internal and active striping cases; for the next several chapters we will refer to these cases only by the layer at which they are operating. These cases are interesting to consider because they represent the most general striping possibilities, so portions of any discussion about them will be relevant to other striping possibilities at the same layer.

These striping cases can all provide the characteristics which we would like from a striping implementation. First, they will increase the amount of available bandwidth. The fact that they are internal cases means that they will model a far more realistic situation. By operating above the physical layer, they allow the networks they use to migrate slowly to new technology without requiring major changes in the striping implementation. Finally, since they are active cases, they will allow more flexibility in the striping implementation and further reduce the dependence on specific network equipment.

4.3 Conclusion

This chapter has presented the entire striping space of our framework, from which we have determined four interesting striping cases which we wish to examine in more detail. This examination will explore the actual implementation possibilities. For example, the various synchronization schemes will be presented, as well as the requirements from the hosts and networks for initiating and manipulating a striped connection.

There are a couple issues to keep in mind while doing the implementation analysis. First, there is the target for a striping implementation. We can either be retrofitting an existing network with a striping implementation or planning a new network which will have striping

capabilities built into it. In the first case, we will probably have to implement a scheme which stays within existing standards; in the second case we may have more flexibility.

The second thing we must keep in mind is how the responsibility for the various striping implementation functions must be partitioned. We have made this binary distinction between passive and active host participation; in reality we must remember that a case where half of the striping implementation is handled by the network elements and half is handled by the hosts.

Before proceeding to the analysis of these cases, however, we will present the experimental apparatus which will be used to actually implement some forms of these cases. This will be done in two parts; first, the network apparatus which existed previous to this work will be examined, followed by the description of a board designed to provide the interconnection between the desk and local-area networks.

Chapter 5

Experimental Apparatus

This chapter will describe the existing experimental apparatus which will be used to implement some experimental versions of the four interesting network striping cases. This apparatus consists of three independent ATM platforms: the VuNet, AN2, and Aurora. The integration of these three platforms provide sufficient flexibility and network capacity for a wide range of possible striping experiments.

5.1 VuNet

The VuNet is the underlying network infrastructure for the ViewStation project within the Telemedia, Networks, and Systems Group at the MIT Lab for Computer Science[20]. A sample VuNet topology is shown in figure 5-1.

The VuNet is based on a modification of the ATM standard which uses 56, rather than 53, byte cells[1]; this allows the switch ports to be quad-word aligned, which is convenient for communication with workstation I/O buses. There are three primary components to the VuNet: switches, links, and clients.

The VuNet's role in striping experiments will be to serve as a high-speed ATM network whose clients will generate the traffic to be striped. The VuNet hardware also has a number of characteristics which will make it particularly good for handling striping implementations.

[1]The extra three bytes are appended to the cell header and are unused.

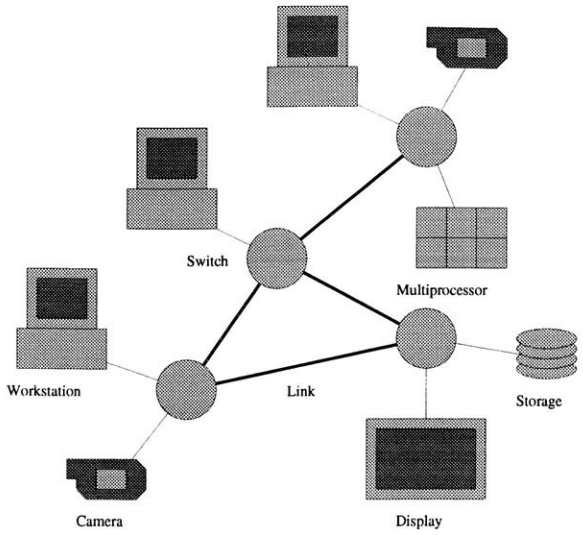


Figure 5-1: Sample VuNet Topology

5.1.1 VuNet Switches

The VuNet switches consist of a crossbar matrix with small FIFO buffers on the inputs to the crossbar (transmit buffers) and larger FIFO buffers at the outputs of the crossbar (receive buffers). The switches have either four or six bi-directional ports.

The switch is completely controlled by the clients; it has no ability to independently route cells. Clients on one port direct cells to another port by writing the destination port address and cell data into the transmit buffers. The cell at the head of each port's transmit buffer is written to the receive buffer of the appropriate port. Multiple requests for an output port during a cycle are resolved through round-robin arbitration. The current switch operates at a speed of 700 Mbps, and has been tested at speed of up to 1.5 Gbps.

5.1.2 VuNet Links

The VuNet links are high-speed optical links, operating at 500 Mbps, which interconnect the switches. Links plug directly into switch ports and use the same interface as any other client. Physically, the links consist of two separate boards. The main board contains the lookup tables, the interface to the VuNet switch, and all control circuitry. The second board is a daughterboard containing the electrical/optical conversion circuitry, the optics, and the HP G-Link chipset which provides the physical layer protocol.

The links contain an on-board VCI lookup table used to route cells through the network; the VCI of an incoming cell from the daughterboard is used as the index into the table,

which outputs the new VCI for the cell as well as the destination port address. The link remaps the old VCI to the new VCI provided by the table and writes the cell data, along with the port address, into the switch transmit buffers.

The VCI lookup tables are manipulated through the use of control cells which are transmitted on particular VCIs. There is a separate VCI to write and to read a table entry. When the link receives a cell on one of these VCIs, it either alters the lookup table or outputs a cell containing table entries, depending on the VCI of the control cell. This allows clients to manipulate their own connections by prepending control cells onto their data streams. As we shall see in the next chapter, the ability of the hosts to determine the routing of the VCIs it is using will be quite useful.

5.1.3 VuNet Clients

There are currently two operational VuNet clients: the vidboard and the vudboard. The vidboard is a video capture and processing device which connects directly to a switch port. It can generate video streams with many different characteristics, such as picture size and frame rate. Hosts can program the vidboard through the VuNet with special command cells. These cells specify what form of video stream to grab, which switch port to send the video to, and what VCI to send it on. Hosts then use another command cell to request video frames from the vidboard.

The vudboard is a simple DMA host interface for the DEC Turbochannel. It performs programmable length DMA bursts of cells between the network buffers and the host's main memory; higher layer functions, like packet reassembly, are left for the host software to perform. Outbound cells in the host's memory contain the routing information necessary for the switch to route the cell; the host interface extracts this information and presents it to the switch.

The simplicity of the vudboard allows more flexibility as far as what striping options the hosts can support. Host interfaces which perform on-board packet segmentation and reassembly would not be able to handle something like a single packet arriving on multiple VCIs unless they had a great deal of programmability. Since the vudboard simply serves as a pathway for cells between the host memory and the network, the necessary programmability can be, and is, in the host processor.

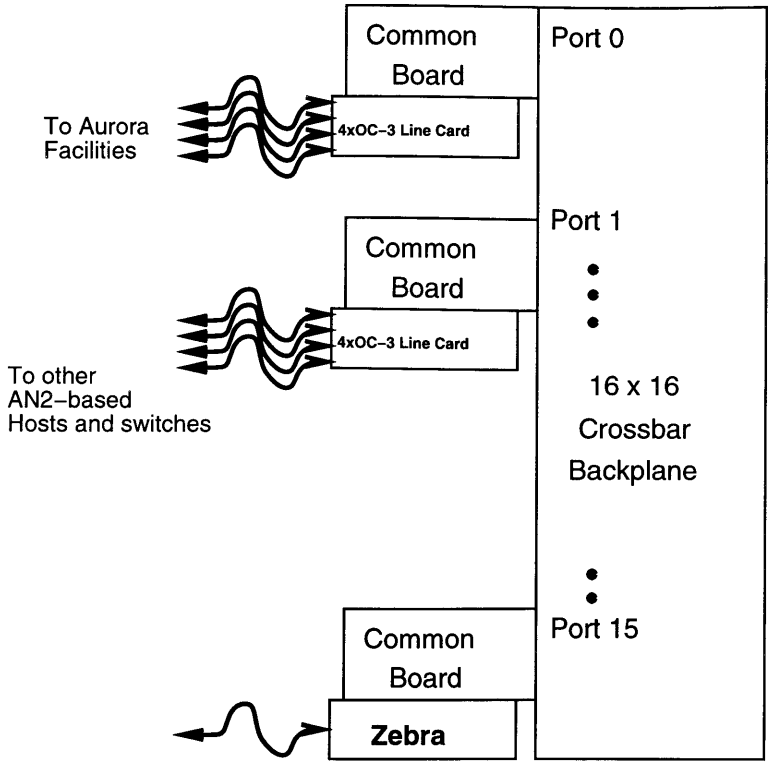


Figure 5-2: Example of an AN2 Node

5.2 AN2

AN2, short for AutoNet2, is Digital's ATM-based successor to the original AutoNet[21]. AN2 uses a per-virtual circuit credit-based flow-control method between switches to avoid dropping cells due to congestion. The three components of an AN2 node are a 16 port crossbar, common boards which plug into the crossbar ports, and line cards which attach to a daughtercard connector on the common boards (Figure 5-2).

The AN2 common board implements all the functions of transmitting and receiving cells which are not physical layer-dependent[2]. This includes the arbitration and scheduling of cells through the crossbar to other common boards. The times during which cells are forwarded through the crossbar are slots. As cells are forwarded, the common board will perform VCI mapping as well. The common board also holds a large set of input buffers to hold cells while they wait for the crossbar. These buffers are accessed randomly by

[2]The cells handled by the common board and the crossbar are 52 bytes long; these are standard ATM cells with the HEC omitted. This allows the common board to handle cells which are quad word aligned. The line cards are expected to generate the HEC for outgoing cells and to verify and then discard the HEC for incoming cells (if necessary).

VCI, as opposed to FIFO buffers, which allows greater flexibility in scheduling cells for transmission through the crossbar. The common boards can create a fixed schedule of slots for certain virtual circuits, supporting constant bit-rate traffic. The remainder of the slots are scheduled through a process called statistical matching[22], which allows the switch throughput to approach 100% of the communications link bandwidth.

The common board also manages the AN2 flow control mechanisms. The inbound direction processes incoming cells for credit acknowledgments piggy-backed on the header which it passes the the outbound credit counters. The inbound direction also informs the outbound section when it frees a buffer for a particular VCI so that a credit may be sent back to the source upstream.

To control the various functions of the common board and line card, the common board has an R3000 processor (known as the Line Control Processor, or LCP) on board. The LCP has the ability to transmit and receive cells as well; it is only able to send and receive with the switch port, however, and not the line card. The ability to program the LCP to control the basic functions of the line cards will be quite useful in implementing striping.

There are several flavors of line cards providing various physical layers which attach to the common board daughtercard connector. First, there is a quad line card, which provides four OC-3c connections. These were originally intended to be connected to OC-3c equipped host interfaces, but can be connected to any OC-3c SONET gear. An OC-12/OC-12c line card is being developed to study the issues associated with ATM LAN/WAN migration, but is not yet available[23]. Finally, the Zebra connects the AN2 to the VuNet. The design of this line card will be described in the next chapter.

5.3 Aurora

Aurora is one of six testbeds examining issues associated with gigabit networking[13]. The sites which are part of Aurora are MIT, UPenn, Bellcore, and IBM (Figure 5-3). Two different network technologies were studied as part of the Aurora testbed: Asynchronous Transfer Mode (ATM) and Packet Transfer Mode (PTM). The two technologies share a SONET infrastructure which connects the four sites.

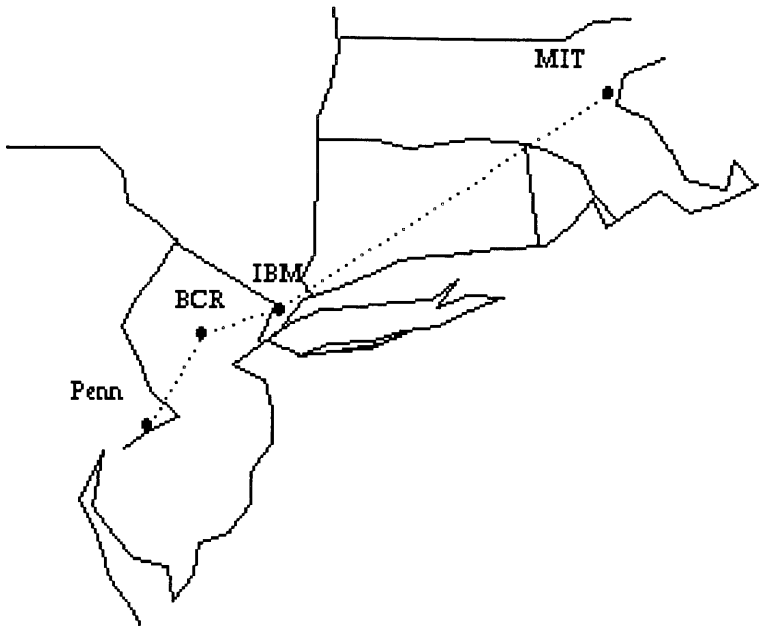


Figure 5-3: Aurora Testbed Participants

5.3.1 SONET Infrastructure

The four Aurora sites are connected with SONET OC-48 lines. Equipment at each site connects to the SONET facilities at the OC-12 rate; the ATM equipment uses SONET multiplexors developed by Bellcore to accomplish this. There are two types of these OC-12 multiplexors. One type provides four STS-3c interfaces which are combined into an OC-12 stream. The other provides four OC-3c interfaces which are combined into an OC-12 stream.

The important thing to realize about the SONET infrastructure is that the highest concatenated SONET level is OC-3[3]. In addition, the SONET equipment handles the manipulation of the higher level connections at the STS-3c level. For example, at White Plains, one of the OC-12 lines is removed from the incoming OC-48 line from MIT and delivered to IBM. At the same location, an OC-12 line from IBM is placed in the outgoing OC-48 line towards Bellcore. However, the add/drop muxes performing these operations must break all the higher layer OC-12 units down into their component OC-3c lines, do the necessary routing, and then rebuild the higher level OC-12s. Consequently, the SONET facilities should

[3]Higher level SONET frames are generally constructed by byte-interleaving some number of lower level frames. Concatenation is a process which keeps the payload envelope of each lower level SONET frame from shifting relative to the others, allowing the higher level frame to be treated as a single entity.

be considered as a network of OC-3c connections even though the connections between the sites run at higher rates.

The result of the operations performed on higher level SONET lines by the facilities equipment is that data striped across the four OC-3c channels in an OC-12 will arrive at the destination skewed relative to each other. Amazingly enough, this still happens when the four OC-3c channels are constrained to follow the same path through the network; this was the effect discovered by the Osiris board when it was first connected to the facilities. This is representative of the problems associated with the local provider not being able to supply a single high bandwidth channel; until the local access can be upgraded to a single OC-12c channel, some form of striping implementation is necessary to provide the lead users with the bandwidth they desire.

5.3.2 Sunshine Switch

The other major portion of the Aurora ATM infrastructure is Bellcore's Sunshine switch. Sunshine is an experimental ATM switch being developed by Bellcore[6]. The switch fabric is a Batcher-Banyan network. It has 32 ports, each of which operates at 155 Mbps and connects to an STS-3c. Four ports are multiplexed together into an OC-12 using the STS-3c to OC-12 multiplexor described above. Originally, these multiplexed ports were supposed to act as one logical port running at 622 Mbps. However, the OC-3c payload skew has made it much more attractive to leave the switch ports running at 155 Mbps and solve the skew problem end-to-end[24].

5.4 The Whole Picture

The configuration resulting from the interconnection of the three networks is shown in Figure 5-4. Due to the fact that the OC-12 line cards were unavailable, the connection between the AN2 switch and the Aurora SONET facilities was made using one of the quad line cards in conjunction with the Bellcore OC-3c/OC-12 multiplexor. The use of the quad line card also allows the VuNet/AN2 combination alone to simulate a wider variety of skew problems by looping the virtual circuits on each of the OC-3c lines some number of times.

There is one missing component which prevents the interconnection of the three networks from being complete. This is the piece of hardware connecting the AN2 to the VuNet. This

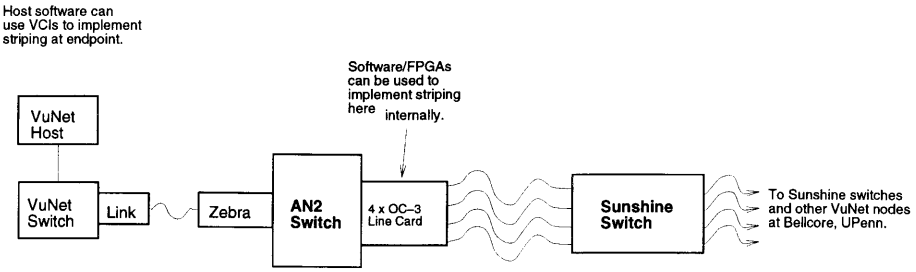


Figure 5-4: VuNet/AN2/Sunshine Topology

piece was designed for the purposes of this thesis and its design will be described in detail in the next chapter.

Chapter 6

The Zebra

This chapter presents the Zebra, the custom hardware used to interconnect the VuNet and AN2. The Zebra performs the ATM bridging functions necessary to allow cells to pass between the two networks. The initial issues which directed the design of the Zebra will be discussed first, followed by the presentation of the final design.

6.1 Initial Design Issues

The Zebra design addresses a number of issues:

- Placement: This describes how the physical points of attachment to both networks.
- ATM Bridging: A number of incompatibilities between the two networks must be bridged through the Zebra[1].
- AN2 Flow Control: In order to function properly as a part of the AN2 network, the Zebra must terminate the AN2 flow control.
- Choice of Technology: There were several options as to the actual technology used to implement the Zebra design.

6.1.1 Zebra Placement

In order to provide a path for cells between the VuNet and AN2, the Zebra needs to be physically attached to each network. The points of attachment should balance simplicity of

[1] For a detailed examination of the issues associated with ATM bridging, see [25].

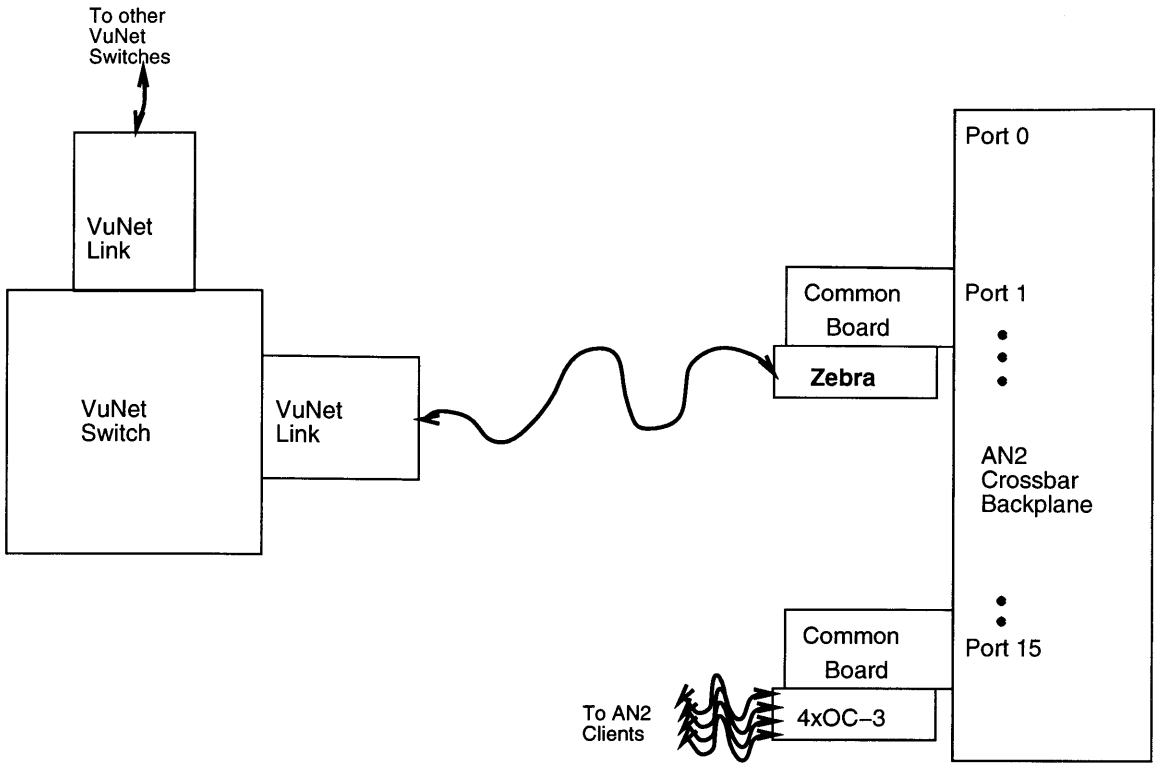


Figure 6-1: Final Path between the VuNet and AN2 through the Zebra

design with flexibility and interoperation. These locations also affect the implementation of ATM bridging between the networks. The path of connectivity between the VuNet and AN2 through the Zebra is shown in figure 6-1.

AN2 Points of Attachment

There are two potential points of attachment for the Zebra to the AN2 switch. Recall the configuration of an AN2 node from figure 5-2. The first option is to connect directly to a backplane, just as the common boards do. In order to interoperate with the existing common boards, the Zebra would need to duplicate their crossbar arbitration and scheduling scheme. The other option, which was chosen, is to make the Zebra a type of line card for the AN2 switches which would connect to the AN2 daughtercard interface on the common board, just like the 4xOC-3 and OC-12 line cards. This eliminated the need to redesign logic which was already present on the common boards and also allows the Zebra to take advantage of the processing power of the LCP and the large input buffers present on the common board.

VuNet Points of Attachment

There are two potential points at which to attach the Zebra to the VuNet. These are the VuNet switch port and the fiber optic interface to the VuNet link. Making the decision to build to the AN2 daughtercard interface already required that one new board be designed. Designing to use the VuNet link eliminated the need to build a second board which would plug into the VuNet switch. More importantly, it allows AN2 switches to appear as though they are 16 port VuNet switches.

6.1.2 Flow Control Considerations

In addition to connecting an AN2 switch and a VuNet switch, two Zebras may be connected back-to-back in order to connect two AN2 switches. When connected to the VuNet, there is no need to support the AN2 flow control algorithms. However, in the second configuration, flow control is useful. The Zebra can operate in either of these two modes.

6.1.3 Design Technology

The hardware previously designed for the VuNet used mostly discrete TTL components and simple programmable logic devices (22V10 PALs). More recently, some higher density programmable logic, such as the AMD MACH series parts, have been used as well. There was a choice of staying with known technology to speed the design process or to use Xilinx parts. Since the other line cards for the AN2 are Xilinx-based, there is software support on the LCP to load any number of various size Xilinx parts across the daughterboard interface. The ability to change the Xilinx configuration while the Zebra is still attached to the common board, along with the extra flexibility provided by the Xilinx FPGAs over the MACH parts, was enough of an inducement to implement the Zebra using Xilinx parts. The ability to load new designs into the Xilinx parts offers an easy way to handle the two different flow control modes of the Zebra; each mode uses a different Xilinx configuration.

6.2 Zebra Design Description

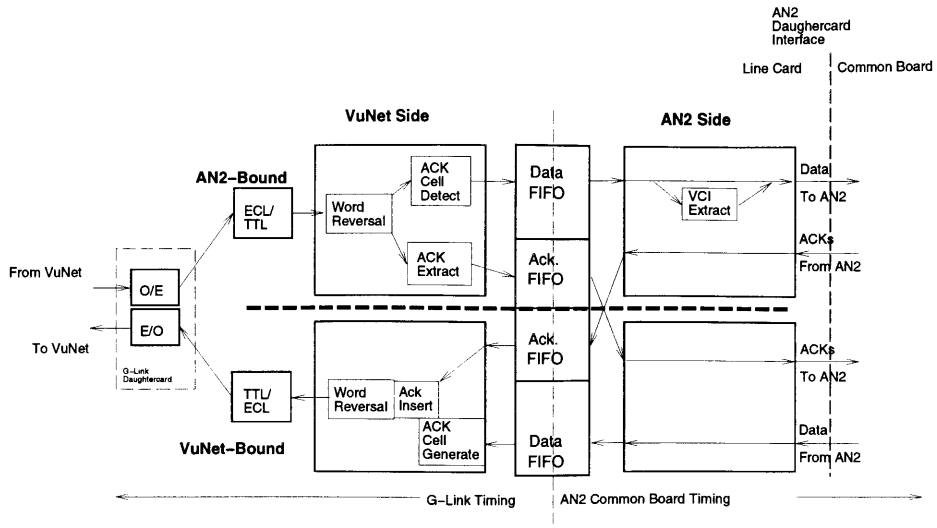


Figure 6-2: Top-Level Zebra Block Diagram

A block diagram of the Zebra is shown in figure 6-2[2]. It consists of everything between the G-link daughtercard and the AN2 daughtercard interface. The board has two blocks implementing the AN2-bound and VuNet-bound data paths[3].

Notice that each block has two separate timing domains; the components which transmit and receive cells to the VuNet operate using the G-Link clock, while the components which transmit and receive cells across the AN2 daughtercard interface operate using the AN2 common board clock[4]. These two timing domains are completely decoupled by banks of FIFOs; the details of the decoupling will be discussed later.

6.2.1 AN2-bound Direction

The logic in the AN2-bound direction performs the basic functions of receiving cells from the VuNet link, formatting them properly for the AN2 common board, and transmitting them across the AN2 daughtercard interface. In addition, the two timing domains must be decoupled and flow control must be handled in the AN2-to-AN2 mode. We will describe the details of the VuNet and AN2 sides separately, and then discuss how the two timing domains are made to operate together properly.

[2]This description focuses on the AN2 to VuNet configuration of the board because this mode has been debugged and is operational. However, the datapath and details of the AN2-to-AN2 configuration are described as well.

[3]VuNet-bound denotes the portion of the board attached to the G-Link daughtercard, even if the Zebra is being used in the AN2-to-AN2 configuration.

[4]These two domains will be referred to as the VuNet side and the AN2 side respectively.

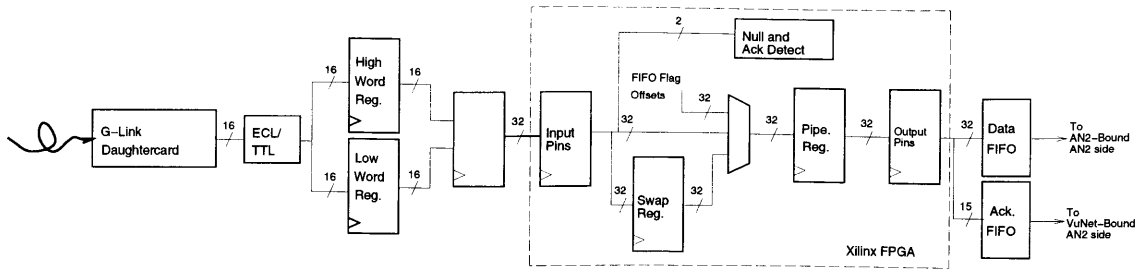


Figure 6-3: AN2-Bound/VuNet Side Block Diagram

VuNet Side

A more detailed data path block diagram of this portion of the Zebra is shown in figure 6-3. All of the logic before the Xilinx FPGA runs on the 16 bit word clock recovered by the G-Link daughtercard. Cell data arrives from the G-link daughtercard 16 bits at a time. The data passes through ECL-TTL conversion and gets alternately clocked into either the high word or low word register. The outputs of these two registers feed a 32 bit wide register which is clocked at the recovered clock rate but which is only enabled every other cycle. The effect of these registers is to multiplex the 16 bits words arriving at the recovered clock rate into 32 bits words which arrive at half the recovered clock rate. This 32 bit word clock is slow enough to allow the timing constraints within the Xilinx FPGA to be met. The remainder of this portion of the board runs on the 32 bit word clock.

The 32 bit register feeds the inputs to the Xilinx, which are registered. The inputs feed two elements: a multiplexor input and a swap register. The swap register feeds a second multiplexor input[5]. The swap register and the multiplexor are used to swap every pair of 32 bit words. This is necessary because the VuNet link transmits the 32 bit halves of a 64 bit word in reversed order.

The output of the multiplexor feeds a pipeline register which reduces the propagation delays in the Xilinx sufficiently so that it will operate properly using the 32 bit word clock. The output of the pipeline register is connected to the registered data output pins of the Xilinx. The output pins are connected to the data FIFO which decouples the timing of the VuNet side of the board from the AN2 side.

The data path is controlled by a finite state machine which is located in the Xilinx FPGA. The FSM receives three signals from the G-Link daughtercard:

[5]The multiplexor has a third input which is selected right after a power-up or reset. This input is connected to a set of hardwired bits which are the offsets for the programmable FIFO flags.

RXCAV and RXDAV: These are two control signals, provided by the G-link chipset, whose state is controlled by the VuNet link transmitter to indicate when cells are arriving at the receiver.

RXLINKRDY: This signal indicates whether the the incoming communication link is in a state where it will deliver valid data.

RXCAV and RXDAV are used by the FSM to perform cell delineation. On the clock cycle before cell data enters the Xilinx FPGA, RXCAV will be asserted. On the next cycle, RXCAV is deasserted, RXDAV is asserted, and cell data is clocked into the first register in the Xilinx. RXDAV will continue to be asserted as long as cell data continues to arrive; 14 clock cycles with RXDAV asserted make up one cell.

Using this signalling pattern to determine the beginning of a cell burst, the FSM knows which 32 bit words make up each individual cell; it uses this information to ensure that the proper pairs of 32 bit words are reversed. The signalling pattern also allows the FSM to determine when to begin writing data into the FIFOs.

The final function which the FSM performs is to reduce the size of the cells from 56 to 52 bytes. It does this by inhibiting the write enable for the data FIFOs during the cycle in which the HEC and three unused header bytes would be written to them.

AN2 Side

A more detailed block diagram of the AN2 side of the AN2-bound direction is shown in figure 6-4. All of the logic on this portion of the Zebra operates using the clock provided by the AN2 common board. The data FIFOs contain 52 byte cells which have been placed there by the VuNet side. The AN2 common board provides the line card with slots thirteen clock cycles in length[6] . During these slots, the line card may transfer cell data across the daughtercard interface. As it does this, there is a signal CELLVALID which the line card must assert in order to inform the common board that a valid cell is being transferred across the interface. In addition, the common board requires that, during the same slot, the VCI of each incoming cell be placed on a separate bus which extends across the daughtercard interface. Data transferred during these slots are written into local VRAM cell buffers

[6] 1024 of these slots make up a scheduling frame which the common board uses to provide fixed bandwidth for CBR traffic.

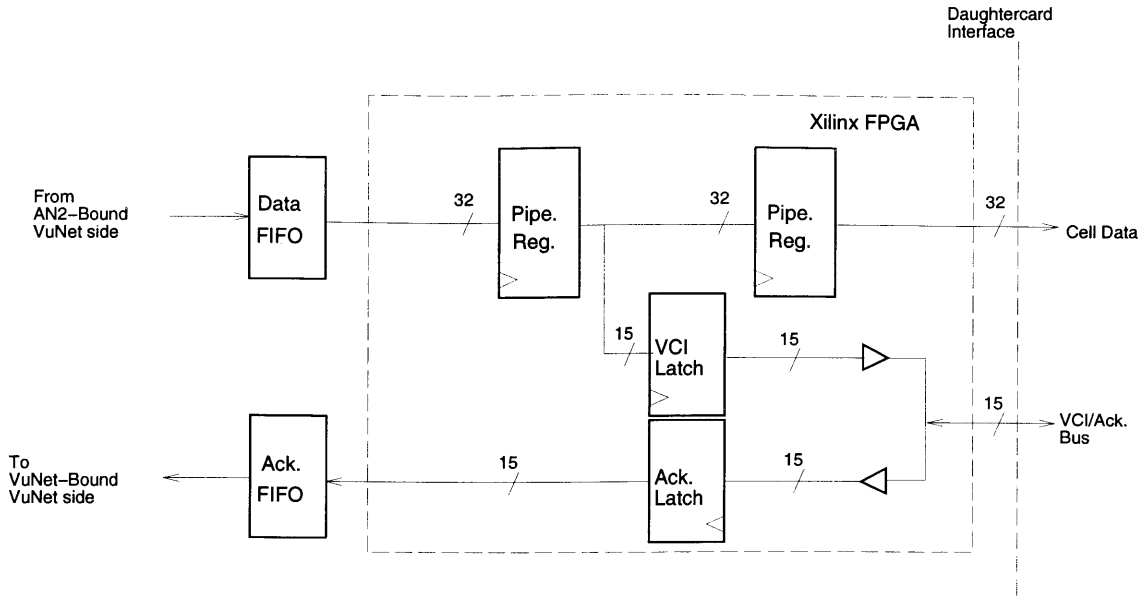


Figure 6-4: AN2-Bound/AN2 Side Block Diagram

on the common board. The common board will then schedule these cells for transmission through the crossbar.

The output of the data FIFOs feeds directly into the Xilinx FPGA. After passing through two pipeline registers, the Xilinx outputs feed the data bus which crosses the AN2 daughtercard interface. In addition to the cell data path, there is a path through which the VCI of each cell is delivered to the VCI bus. After the first pipeline register is a 15 bit register connected to the data bits which carry the VCI[7]. This register is then connected to a tri-state buffer which drives the VCI onto the VCI bus.

The finite state machine in this Xilinx is controlled by two signals:

PAE: This is the programmable almost empty flag from the data FIFOs. It will be asserted from the time the data FIFOs are empty until exactly one cell has been written into the FIFOs.

SYNC: This signal is provided by the AN2 common board through the daughtercard interface and signals the beginning of a slot.

The FSM is a free-running loop consisting of thirteen states. When the Zebra comes on line after a reset or a power-up, the FSM uses SYNC to start the loop at the beginning of

[7]The VCIs used by AN2 are only 15 bits long.

a cell slot. There is a state in the loop during which the FSM samples PAE to see if there is a cell in the data FIFOs which is ready to be transmitted. This state is four cycles prior to the beginning of a slot; one cycle is required to begin asserting the FIFO read enables, another cycle is required for the FIFOs to begin to output valid cell data, and two cycles are required for the data to pass through the two pipeline registers. The FSM will continue to enable the FIFOs for thirteen clock cycles to read the entire cell from the FIFOs. It will also latch the VCI when the header is stored in the first pipeline register and drive the VCI onto the VCI bus at the proper time during the slot.

Timing Issues

The two sides of the AN2-bound direction run on two different clocks. We need to ensure that the FIFOs between the two sides will not overflow, causing cells to be dropped. The VuNet side runs on a 16 bit word clock of 33.3 MHz and a corresponding 32 bit word clock of 16.7 MHz. The AN2 side, on the other hand, runs on the AN2 common board 32 bit word clock of 25 MHz. The VuNet side writes cells into the FIFOs at the rate of 32 bits $\times$16.7 MHz $\div$448 bits/cell, which is 1.2 Mcells/second. The AN2 side can read cells from the FIFO at 32 bits $\times$25 MHz $\div$416 bits/cell, which is 1.9 Mcells/second. Thus, we see that there is no danger of the VuNet being able to overrun the FIFOs.

Since both sides are operating asynchronously, there is also an issue of allowing communication between them while preventing problems such as metastable behavior. The FIFOs accomplish this task for the data bits by allowing each side to read or write at its own clock rate. The FIFO flags are used to decouple the signalling between the two sides. The almost empty flag is programmed by the VuNet side to signal the AN2 side when there is exactly one cell in the data FIFOs. This ensures that cells will not be trapped in the FIFO and also ensures that the AN2 side will not attempt to begin reading a cell before all of the words are available.

The FIFOs only change the state of the almost empty flag synchronous to the read clock provided by the AN2 side, and the flag is registered as it enters the Xilinx FPGA just as an extra precaution. Thus, the flag ensures that there is no asynchronous communication between the two sides.

6.2.2 VuNet-Bound Direction

The logic in the VuNet-bound direction receives ATM cells from the AN2 common board across the AN2 daughtercard interface, formats them properly for the VuNet, and forwards them to the VuNet using the G-Link daughtercard. This half of the board must also ensure that the two timing domains are decoupled and handle the flow control in the AN2-to-AN2 mode. Just as with the AN2-bound direction, we will discuss each side separately and then discuss how they are made to operate properly together.

AN2 Side

A more detailed data path block diagram of this portion of the Zebra is shown in figure 6-5. All of the logic runs off of the clock supplied by the AN2 common board. The common board provides the same slots for cells that it is transmitting as for cells that it is receiving. Cell data which arrives across the AN2 daughtercard interface during a slot passes right into a multiplexor in the Xilinx FPGA[8]. The multiplexor is followed by two pipeline registers. Data moves out of the registers, out of the Xilinx, and into the data FIFOs.

The finite state machine which controls the data path is controlled by SYNC and two other signals:

XVALID: This signal comes across the daughtercard interface from the common board and indicates that the common board is forwarding a valid cell from the crossbar during this slot.

OUTLINE: This signal comes across the daughtercard interface from the common board and indicates that the cell coming across the interface is meant for the line card. This is necessary because there is only one data path from the crossbar to the LCP and daughtercard interface; XVALID is asserted in both cases as well.

The finite state machine which controls the data path idles until the common board asserts the XVALID signal. It then begins to receive data from the common board. On the next cycle, it examines the OUTLINE signal. If this signal is asserted, then it continues to allow data to pass through the pipeline registers and it also begins to assert the FIFO write

[8]The other inputs to the multiplexor are used to generate the offsets for the programmable almost empty and almost full flags.

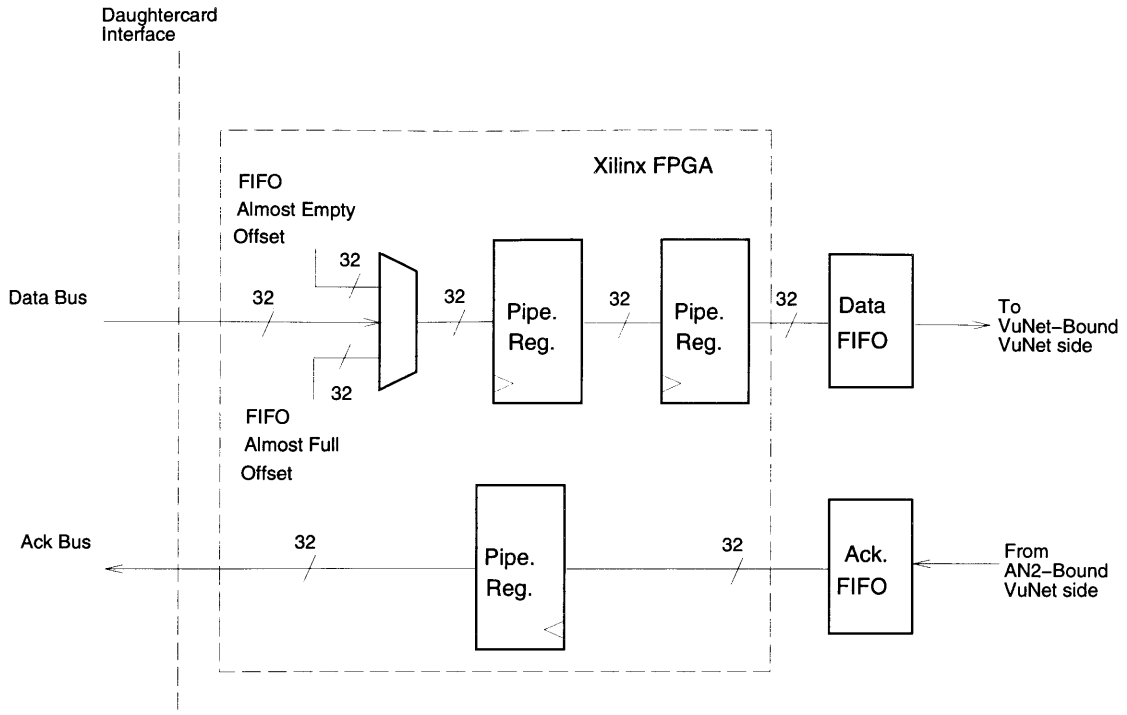


Figure 6-5: VuNet-Bound/AN2 Side Block Diagram

enables. Otherwise, it starts to idle again and waits for the next assertion of XVALID. After thirteen clock cycles, a whole cell has been written into the FIFOs, and the FSM examines XVALID again to see if there is another incoming cell.

VuNet Side

A more detailed block diagram of this portion of the board is shown in figure 6-6. A local oscillator provides the 16 bit word clock which is then divided in half to provide the 32 bit work clock which is used to run this entire portion[9].

The AN2 side of this half of the board has written 52 byte cells into the data FIFOs. Before the Zebra can deliver these cells to the VuNet, it must increase the size to 56 bytes. Data from the FIFOs enters one input of a multiplexor in the Xilinx. A second input is used to generate 32 bits of zeros which will be appended to the header to increase the cell to the proper size.

The output of the multiplexor feeds a swap configuration identical to the one in the AN2-bound/VuNet side Xilinx. This is to reverse the 32 bit halves of 64 bit words so that

[9]In theory, this local clock can be a different frequency than the link clock which is recovered on the VuNet side of the AN2-bound direction. In practice, however, we use the same frequency.

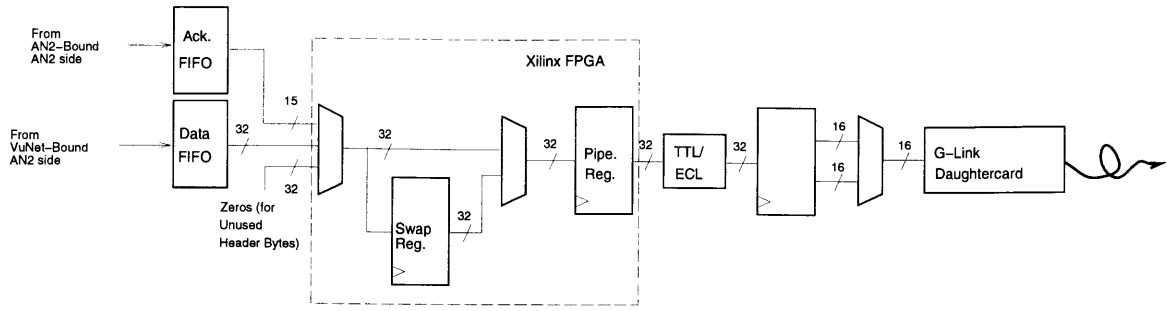


Figure 6-6: VuNet-Bound/VuNet Side Block Diagram

the VuNet link will receive the data in the order in which it expects it. This allows us to use unmodified links.

The output of this swap configuration passes through a pipeline register and then goes out of the Xilinx into a TTL to ECL conversion block. After being converted into ECL, the data is latched and then multiplexed down to 16 bits for transmission by the G-Link daughtercard to the VuNet link downstream.

The finite state machine which controls this data path examines the programmable almost empty flag from the FIFOs. This flag is set to indicate when there is at least one cell in the FIFOs. When the FSM detects that there is a whole cell in the FIFOs, it asserts the read enable for a cycle to read the four header bytes into the Xilinx. It then deasserts the read enable for the next cycle and enables the multiplexor input which generates the zeros which pad the header. It then asserts the read enable for twelve more cycles to read the rest of the cell out of the FIFO.

After insuring that the 32 bit words are swapped properly, the FSM allows the data to pass into the ECL portion of the logic. In order to allow the VuNet link to perform cell delineation on the data, it uses TXCAV and TXDAV. These are the control signals at the transmitter which will generate RXCAV and RXDAV at the receiver. The FSM asserts TXCAV for a cycle just prior to the cell (or cells), and then asserts TXDAV for every consecutive cycle thereafter during which it delivers cell data.

Timing Issues

Just as the AN2-bound direction, the two sides of the VuNet-bound direction run on two different clocks. Since the clock frequencies are the same for this half of the Zebra as the other, we can use the same numbers. The AN2 side can deliver cells to the FIFO at the

rate of 1.9 Mcells/second, which the VuNet side can only drain them out at the rate of 1.2 Mcells/second. In order to keep the FIFOs from overflowing, the Zebra relies on two things. First, there is the assumption that there will never be extended periods where the AN2 side will be operating at 100% capacity. Second, the AN2 flow control which is active within the switch will not allow cells to be forwarded to line cards with output buffers which are almost full. The AN2 is based around large input buffers on the common boards and small output buffers on the line cards. Anytime the output buffers on a line card are almost full, requests to transmit to that port will be denied, and cells for that port will collect in the input buffers on the common board. Since these buffers are so large, there is little danger that they will overflow and cause cells to be dropped. Thus, we can be almost certain that, except under extreme traffic conditions, no cells will be dropped due to timing differences.

The other timing issue is that of communication between two asynchronously timed domains. The VuNet-bound half uses exactly the same method to avoid this problem as the AN2-bound half. The FIFOs decouple the data timing, and the programmable almost empty flag is used to ensure that cells are not left trapped in the FIFO or that the VuNet side does not attempt to transmit a cell before all the cell data is guaranteed to be available. In addition, the almost empty flag is synchronous to the 32 bit word clock on the VuNet side, preventing any other metastability problems.

6.2.3 Modifications to Support Flow Control

Although we have included the elements for AN2-to-AN2 flow control in the block diagrams, thus far we have only described the operation of the Zebra with no flow control support. This section will describe the modifications to the Xilinx configuration which must be made to support the AN2 flow control.

The AN2 flow control method operates in the following manner. The outbound portion of the line card/common board combination maintains a count of the number of input buffers on the downstream common board which each VCI may fill with cells[10]. When a cell is transmitted, the balance for that VCI is decremented by one.

Buffers allocated to a particular VCI in the downstream node will be emptied as cells

[10] We will use the term downstream to refer to the direction in which cells which will be occupying input buffers are traveling; the term upstream will refer to the direction in which acknowledgments are flowing. In reality, each downstream cell may carry an upstream ack.

are forwarded through the crossbar. When this happens, the downstream node will send an acknowledgment to the upstream node, indicating that there is a free buffer and so the balance for that VCI should be incremented. Acknowledgments on the 4xOC-3 line card and the Zebra take the form of the VCI which is being credited; this VCI is piggybacked onto a cell traveling upstream[11]. In the event that there is no cell traveling upstream, a null cell is generated to carry the acknowledgment. These null cells are detected and discarded at the upstream node after the acknowledgment has been extracted.

AN2-Bound/VuNet Side

This portion of the board must strip credit acknowledgments from incoming cell headers. The acknowledged VCIs are placed in the three unused bytes of the header, along with a bit to indicate whether the cell carries an acknowledgment and a bit to indicate whether the cell is a null cell which exists solely to carry an acknowledgment. In the latter case, the write enables for the data FIFOs are not asserted during that cell; null cells pass through the Xilinx chip just as any other cell but are not written into the data FIFOs. When the unused bytes with the acknowledgment are on the output pins of the Xilinx, they are written into a separate acknowledgment FIFO. This FIFO feeds the AN2 side of the VuNet-bound direction, an odd arrangement which will be explained later.

AN2-Bound/AN2 Side

The bus upon which the VCI of the current cell is placed also supplies the VCIs of cells being forwarded through the crossbar. To indicate whether a credit should be generated for this VCI, the common board provides two signals. The first indicates that it is forwarding a cell though the crossbar and the second indicates whether acknowledgments are disabled for the VCI it is providing. If the Xilinx detects that the common board is forwarding a cell it latched the VCI of that cell from the VCI bus into a register. If an acknowledgment needs to be generated, the Xilinx writes the VCI into another acknowledgment FIFO which feeds part of the VuNet side of the VuNet-bound direction.

[11]The OC-12 line card is experimenting with a scheme which batches mane acknowledgments together into special cells

VuNet-Bound/AN2 Side

Remember that the stripped acknowledgments from the VuNet side of the AN2-bound half of the Zebra are placed into an ack FIFO which feeds this section. This was done out of convenience; this Xilinx had enough unused I/O pins to allow this function to be located here. The Xilinx examines the empty flag of the ack FIFO to determine when acknowledgments are available. When there is an acknowledgment, it removes it from the FIFO and places it on an ack bus for transfer across the daughtercard interface at the proper time in a cell slot. It also asserts another signal indicating that it is delivering a credit.

6.3 Summary

The Zebra has been fabricated and assembled. The VuNet-to-AN2 mode has been extensively tested and will support the transfer of billions of cell at nearly 100% utilization with a negligible error rate. The final piece of equipment necessary to interconnect the three networks is in place. The transfer of video from a vidboard on the VuNet through the AN2 to the SONET facilities and back to a host on the VuNet has been performed as well. We may now perform striping experiments using the three interconnected networks.

The following chapters will describe striping at the cell, adaptation, network, and transport layers in greater detail. Our purpose will be to gain some understanding about the tradeoffs associated with the various options, the usefulness of some methods under various circumstances, and some understanding about how striping can be done with networks not designed for it. We will also be considering what future networks should do in order to make striping less difficult and more useful.

Chapter 7

ATM Layer Striping

This chapter provides a detailed examination of ATM layer striping. We begin by compiling a list of synchronization alternatives. A basic description of each alternative is given, followed by an example which will be used to discuss some of the issues and tradeoffs associated with each method. In particular, we are interested in examining the overhead incurred by each synchronization method and how this overhead scales as the amount of skew increases. We would also like to examine how each method will work with existing ATM networks and what future networks should provide in order to support striping. Two methods will emerge as viable alternatives, one using virtual paths and the other taking advantage of the ATM OAM (operations and maintenance) facilities.

In addition, we shall examine an experimental implementation of one of the proposed striping methods using the experimental apparatus. This will provide some practical considerations about how to implement a striping implementation, the characteristics our networks have which support striping, and talk a bit about performance.

7.1 Synchronization Options

Synchronization is particularly important to consider when striping at the ATM layer as in-order cell delivery is guaranteed to the adaptation layer[1]. In order to identify the possible synchronization options, there are two axes to consider. The first is the synchronization tools at our disposal, and the second is where in an ATM cell the synchronization information

[1]The adaptation layer is not required to verify and correct the cell order, and so we must ensure that the cell order is correct before the cells reach the reassembly portion of the adaptation layer.

Header Tagging	Payload Tagging
Header Metaframing	Payload Metaframing

Table 7.1: Cell Synchronization Option Space

may be placed. Recall from the discussion of synchronization in chapter 3 that there are two basic tools available for synchronizing multiple channels: tagging and metaframing.

In an ATM cell, there are two places to put synchronization information: the header and the payload. Each of the header fields could potentially carry information, but using some of them, such as the GFC and PTI fields, may conflict with current or developing standards. However, one possible way to use the PTI field which will not conflict with the standards is to use end-to-end OAM F5 flow cells to carry synchronization information[26, 27]. We will present this idea in more detail in later sections.

The VCI and VPI fields are also good candidates to carry synchronization information. The standards define these fields as routing information but do not specify that they carry certain values (with a few exceptions). The potential cost, however, is that using portions of the VCI/VPI space for synchronization reduces the total number of virtual circuits available. In addition, designers of ATM equipment have already started to encroach on the available VCI space. AN2 only uses 15 rather than 16 bits for the VCI, and many commercial host interfaces are only capable of supporting a fraction of the VCI space. This may make it less attractive to use a synchronization method which uses multiple VPI/VCIs.

The only requirement of the payload at the ATM layer is that it carry the bytes of higher layer data units. It is not partitioned into fields with defined functions, so we could potentially place whatever information we chose into the payload. The effect of this, however, may be to create a custom adaptation layer which may or may not be compatible with existing AALs.

Both of the synchronization tools can be used on both of the locations in the cell, creating a possible option space which looks like Table 7.1.

7.2 Header Tagging

This scheme allows synchronization by tagging N consecutively transmitted cells with a modulo N identifier, like a sequence number. After N cells have been tagged, the next N cells are tagged with the same set of numbers[2]. The receiver can put cells in the proper order using the tags. In order to place the tags in the header, we need a header field with a substantial number of bits; the VPI/VCI is the most attractive candidate.

In this case, the transmitter will request N virtual circuits from the network; it assumes that the network will choose whatever routes are necessary to provide the bandwidth it requests for each circuit. It will then send cells on the different circuits in a round robin order. The receiver will know the order in which the transmitter is using the circuits, and so it will be able to rearrange them if they arrive misordered. We can think of this as using a cluster of real virtual circuits as a single logical virtual circuit. The low-level software in the hosts can provide applications which need striped connections with a single interface, such as a socket, and then handle the details of ensuring that data moves between the socket and the multiple virtual circuits properly.

7.2.1 Effects of Skew

If there were no skew in the network, then if the transmitter in our example sent cells on all eight VCIs in round robin order, they would arrive at the receiver in the same order. If there is skew introduced by the network, then the receiver must be able to re-order; the modulus of the tags must be long enough so that the receiver can distinguish between rounds.

One approach to accomplish this is to associate multiple VCIs with each virtual channel; this can be thought of as a VCI cluster. The VCIs would be used as the tags, and the number of them on each channel must be sufficient to deal with the worst case skew.

7.2.2 Network Considerations

In order to make this synchronization method work we need the ability to request these clusters from the network; there is no support for their transmission and signalling the the standards. Another network concern is related to the type of virtual circuits used in the striping implementation. Let us say that, in the example above, the hosts want a connection

[2]We will refer to a set of N cells as a cycle.

which provides a peak bandwidth of 800 Mbps. The amount of bandwidth requested for each virtual circuit is 100 Mbps. If the virtual circuits have been set up to be CBR, then we are assured of having an aggregate bandwidth of 800 Mbps no matter how the virtual circuits are routed. However, if they are set up as ABR virtual circuits, there may be a problem. Since the network has no awareness of striping, it may assume that even though 8 VCIs with 100 Mbps have been requested, they will not all be in use at the same time. Thus, there may be less than 800 Mbps of bandwidth available; the network is relying on the VCIs being independent from each other in order to offer more bandwidth than is really available. This may be alleviated somewhat by the use of flow control, but may require some form of load balancing across the virtual circuits; we will discuss this point in greater detail later in the chapter.

7.2.3 Header Tagging Summary

Header tagging uses clusters of VCIs associated with individual virtual circuits to provide groups of cells which contain unique identifiers. The number of tags determines the amount of skew against which the method protects. The number also determines the size of the VCI clusters needed to implement this synchronization scheme. Given that the number of VCIs required increases linearly with the amount of skew in cell times, this scheme is not attractive for cases with a large amount of skew.

Header tagging also requires a fair amount from the network. Support for the VCI clusters requires changes to existing standards and equipment. There may also be problems related to the type of virtual circuits used. If used with CBR virtual circuits, the desired aggregate bandwidth can be guaranteed. In addition, the maximum delay on each virtual circuit can be bounded, which will ease the choice of cluster size and perhaps eliminate the positive feedback effect. For ABR there is problems with wildly varying skew. There may also be less bandwidth than required since the network will expect to get some statistical multiplexing advantages from the individual virtual circuits which make up a cluster; this will not be the case since we intend to use the full bandwidth on all the virtual circuits simultaneously.

7.3 Payload Tagging

This synchronization method operates in a similar manner as header tagging, except part of the payload is used as the tag rather than a portion of the VPI/VCI space. This is one of the methods explored in the Osiris work to deal with skew in an end-to-end striping topology. In effect, we are creating a custom adaptation layer by taking part of every cell payload for our striping implementation[3]

Another major difference between header tagging and payload tagging is the use of virtual circuits. With header tagging, VCIs were the tags; we hoped to be able to simple request them as needed from the network and add them into the round robin rotation. With payload tagging, we are simply using virtual circuits as a way to access stripes[4]. Thus, we may set up the minimum number of virtual circuits to provide the aggregate bandwidth we require and then use payload tagging to synchronize those circuits. As our example of payload tagging, we will consider a host which has set up four virtual circuits and is using twelve tags (Figure 7-1).

7.3.1 Effects of Skew

The size of the tag depends on the amount of skew, in rounds, against which we need to protect[5]. It is necessary to have enough tags to distinguish between the number of rounds of skew plus one. Since the number of tags to distinguish one round of cells is equal to the number of stripes, we have a need for at least (Rounds of skew + 1) $\times$ (Number of stripes) tags. As with header tagging, this many tags requires log_2(tags) bits. In the example, the maximum skew between two stripes is two rounds, so twelve tags is just enough to allow the method to function properly.

Notice that header tagging using clusters of VCIs within a virtual circuit is very similar to payload tagging. What we have essentially done in that case is to take some of the bits normally used for routing and caused them to no longer have that function; instead they are just free bits which we can use as a tag. The only difference between these two schemes is the location of the bits.

[3] We will refer to the portion of the payload which we are using as the tag field, or simply the tag.

[4] Whether or not the virtual circuits we set up actually traverse different physical stripes is not really relevant as long as they provide the bandwidth we desire. Therefore, we shall treat them as though they are routed over different physical stripes with the awareness that this is the worst case.

[5] Recall that we refer to one round robin rotation through the stripes as a round.

Round 8	Round 7	Round 6	Round 5	Round 4	Round 3	Round 2	Round 1	
VCI 1 Tag 4	VCI 1 Tag 0	VCI 1 Tag 8	VCI 1 Tag 4	VCI 1 Tag 0	VCI 1 Tag 8	VCI 1 Tag 4	VCI 1 Tag 0	Stripe 1
VCI 2 Tag 9	VCI 2 Tag 5	VCI 2 Tag 1	VCI 2 Tag 9	VCI 2 Tag 5	VCI 2 Tag 1			Stripe 2
VCI 3 Tag 2	VCI 3 Tag 10	VCI 3 Tag 6	VCI 3 Tag 2	VCI 3 Tag 10	VCI 3 Tag 6	VCI 3 Tag 2		Stripe 3
VCI 4 Tag 7	VCI 4 Tag 3	VCI 4 Tag 11	VCI 4 Tag 7	VCI 4 Tag 3	VCI 4 Tag 11	VCI 4 Tag 7	VCI 4 Tag 3	Stripe 4

Figure 7-1: Payload Tagging

An interesting question to consider is how large the tag may get in a case where the skew is very large. Using our example and the skew assessment from chapter 2, we will assume a path difference of 3000 miles, a multiplexing delay of 20μs, and a line rate into the receiver of 600 Mbps. 3000 miles becomes 22.5 ms of skew, which results in approximately 32000 cell times of skew. A cell time is equal to a round, and there are four stripes, which requires approximately 128,000 tags. This requires a tag which is 17 bits long. Not including other adaptation layer overhead, the tags alone use up 4.2% of the available bandwidth in the cell payloads for striping overhead.

7.3.2 Network Considerations

Since we are requesting multiple virtual circuits from the network, we again have the issue that the network may expect to get some statistical multiplexing gain, which may reduce the amount of bandwidth which is actually available. The use of CBR virtual circuits will guarantee that the desired amount of aggregate bandwidth is available and will allow reasonable bounds to be placed on the maximum skew. ABR virtual circuits may still not provide the desired bandwidth, but the use of fewer VCIs makes it more likely that the scheme will work correctly. In addition, there is no need to deal with the network when increasing the number of tags, so it will be easier to implement an efficient algorithm which dynamically adjusts the length of the tag when the skew changes.

7.3.3 Payload Tagging Summary

Payload tagging is a method similar to header tagging. Just as with header tagging, the number of tags determines the amount of skew against which the method protects. The number of tags scales linearly with the amount of skew and with the number of stripes.

Payload tagging has several advantages over header tagging. It requires less from the network since it requires fewer virtual circuits; these virtual circuits only need to be set up once. No negotiation with the network is required to set up new virtual circuits as tags; instead, the host software can add or remove tags as it wishes. This allows payload tagging to dynamically adjust to changing skew. In addition, the tag is not restricted to be some subset of the 16 VCI bits, so larger amounts of skew can be protected against.

The major disadvantage of payload tagging is that striping is no longer transparent to the adaptation layer. This may be acceptable with some lead users who can use their own adaptation layer. However, any equipment which has been designed to use a specific adaptation layer will not be able to operate if payload tagging is used. In addition to the lack of transparency, a portion of the payload which was originally used for higher layer data in the header tagging scheme must now be used as the tag; as the tag gets larger, the overhead used for synchronization increases.

The only requirement of the network that payload tagging has is that the VCIs which the source requests must provide the desired amount of aggregate bandwidth, either in the form of CBR virtual circuits or in some type ABR circuits which understand the bandwidth requirements of striping.

7.4 VPI Header Tagging

In the section discussing header tagging we came to the conclusion that the need for a large tag space made header tagging an unattractive option. Using the idea of attaching a cluster of VCIs to a single virtual circuit allowed header tagging to work effectively. However, these clusters are more or less just virtual paths with less functionality. Furthermore, there is no existing standard for cluster functionality or signalling. If we consider using virtual paths in addition to virtual circuits, we arrive at a much more viable version of header tagging which is almost like payload tagging in some ways.

The arbitrary misordering problem was due to the fact that we had no guarantees from the network about how the virtual circuits would be routed and whether the network would maintain the order of cells on different virtual circuits. However, if we set up a virtual path

Round 8	Round 7	Round 6	Round 5	Round 4	Round 3	Round 2	Round 1		
VCI 4	VCI 0	VCI 8	VCI 4	VCI 0	VCI 8	VCI 4	VCI 0	Stripe 1	(VPI 1)
VCI 9	VCI 5	VCI 1	VCI 9	VCI 5	VCI 1			Stripe 2	(VPI 2)
VCI 2	VCI 10	VCI 6	VCI 2	VCI 10	VCI 6	VCI 2		Stripe 3	(VPI 3)
VCI 7	VCI 3	VCI 11	VCI 7	VCI 3	VCI 11	VCI 7	VCI 3	Stripe 4	(VPI 4)

Figure 7-2: VPI Header Tagging

for each stripe we would like to have, and then the entire VCI space is available for tags[6][7].

For our example, we assume that the transmitter has set up four virtual paths to use as stripes. The transmitter is now free to use the VCIs on each virtual path however it would like, and the receiver is guaranteed that the cells on a virtual path will arrive in the same order in which they were transmitted[8]. So, our transmitter will use the VCIs as a modulo 12 counter across the stripes (Figure 7-2).

This method is affected by skew in the exact same manner as the payload tagging method. Assuming that we can use all of the virtual circuits within a virtual path, we have a tag size of 16 bits plus the number of VPI bits in use; this is just enough to compensate for the large skew case presented above; any larger amount would be impossible to handle with this method.

Finally, our requirement from the network is that the virtual paths and virtual circuits provide us with the aggregate bandwidth we desire. CBR paths and circuits will definitely accomplish this; ABR paths and circuits probably will not.

7.5 Header Metaframing

To perform header metaframing, two types of cells are defined. There is a standard cell which is used to transmit data and a metaframing cell which is transmitted at either periodic or aperiodic intervals on each stripe. The receiver can find the start of a metaframe on each stripe and align the stripes so that cells which appear during the same round at the receiver are cells which were transmitted during the same round at the transmitter.

[6] This is due to the fact that the network must maintain cell order within a virtual path[28].

[7] We are left with a synchronization method which is very much like the payload tagging method.

[8] Just as the tags encapsulated in cells on a virtual circuit are guaranteed to arrive in order, so are VCIs encapsulated within a virtual path.

Round 8	Round 7	Round 6	Round 5	Round 4	Round 3	Round 2	Round 1	
Data VCI	Data VCI	Data VCI	Meta. VCI	Data VCI	Data VCI	Data VCI	Meta. VCI	Stripe 1
Data VCI	Meta. VCI	Data VCI	Data VCI	Data VCI	Meta. VCI			Stripe 2
Data VCI	Data VCI	Meta. VCI	Data VCI	Data VCI	Data VCI	Meta. VCI		Stripe 3
Data VCI	Data VCI	Data VCI	Meta. VCI	Data VCI	Data VCI	Data VCI	Meta. VCI	Stripe 4

Figure 7-3: Header Metaframing using Pairs of VCIs

The two different types of cells are distinguished by their VCIs. As an example, consider figure reffig:headmeta. A metaframing and data VCI pair have been requested for all four stripes. As with payload tagging, we use different VCIs as a way of accessing stripes at the hosts; we also expect that by requesting enough bandwidth for each data VCI we will ensure that they are routed over different physical stripes.

7.5.1 Effects of Skew

The length of the metaframe, including the metaframing cell, determines the amount of skew which can be protected against. Any skew which is shorter than the length of the metaframe will be compensated for. However, there are two drawbacks to using a longer frame; it takes longer for the receiver to detect a loss of synchronization and a loss of synchronization causes a larger number of cells to be discarded.

Recall that the large skew case which we presented for the payload tagging method resulted in almost 32,000 rounds of skew at the receiver. In order to compensate for that amount of skew with header metaframing, we need a metaframe length of at least 32,000 cells, which is roughly 1.5 Mbytes of data on each stripe, for a total of 6 Mbytes. This is equivalent to sending several very large files or many smaller ones before verifying that the receiver is still synchronized. The loss of just one cell on one of the stripes can result in the retransmission of all 6 Mbytes of data.

7.5.2 Network Considerations

This scheme runs into several difficulties related to the network. In the simple internal striping case, the VCIs need to be co-routed in pairs. This might naturally happen if the data VCIs requested most of the bandwidth on each stripe first, followed by the metaframing

VCIs. However, once the network topology became more complex than the basic internal case, the probability of the network naturally co-routing pairs of VCIs is reduced dramatically.

Another problem created by the network is that switches are not required to maintain the order of cells on different VCIs. The receiver is relying on the fact that the order will be maintained. So, if a switch reverses the order of a metaframing VCI and a data VCI on a stripe, the receiver will lose its lock on the metaframing cells and will need to resynchronize. In order to prevent both of these problems the network needs some way to be informed about the pairs of VCIs so that it will co-route them and maintain cell order.

Finally, there is once again the problem that the virtual circuits may not actually be able to provide the desired bandwidth if they are not CBR.

7.5.3 Header Metaframing Summary

Header metaframing uses a pair of virtual circuits on each stripe; one of the VCIs is used to define the beginning of a metaframe which the receiver uses to synchronize the stripes. In order to compensate for a varying amount of skew, the length of a metaframe can be changed. As we have shown, however, large amounts of skew require metaframes so large that they may not detect a loss of synchronization until many megabytes of data have been transmitted; the loss of synchronization will require that all that data be retransmitted.

This method also has network problems. Pairs of VCIs must be co-routed in order to be used for metaframing; this is not something that the network guarantees. In addition, metaframing requires that the switches maintain the order of cells on different VCIs; exchanging a metaframing cell with a normal data cell will cause the receiver to lose synchronization even though no cells have been dropped. The idea of VCI clustering could be used in this case as well; if we requested clusters of two VCIs, they would be guaranteed to have the same path through the network and to not suffer reordering at the switches.

The only other requirement which this scheme has from the network is that the VCIs on each stripe which it requests provide the desired aggregate bandwidth, either as CBR circuits or some special form of ABR circuits.

7.6 Payload Metaframing

This method is similar to header metaframing except that the payload of the cell rather than the header is used to mark the start of a metaframe. We make the same assumption here as in the payload tagging discussion about virtual circuits; the source sets up one virtual circuit per stripe at the start of the connection. A special metaframing payload is placed at periodic or aperiodic intervals on each stripe. The transmitter must insert this payload into the stream of data which it is transmitting and the receiver must search for it and filter it out.

Metaframe size and the amount of skew are related in the same way as with header metaframing. Larger metaframes protect against greater amounts of skew, but they provide less frequent detection of synchronization loss due to lost cells. In addition, the use of the payload to carry the metaframing pattern results in some bandwidth being used for overhead rather than for data. This amount of overhead is just the reciprocal of the metaframe size. Thus, smaller metaframes use more of the available bandwidth for overhead.

Dynamic reconfiguration due to changing skew may be performed with this method; the receiver can be searching for an aperiodic framing pattern or the metaframing cells can inform the receiver when the size of the frame changes. This method also has the requirement that the requested VCIs provide the desired amount of bandwidth.

7.7 Hybrid Metaframing

This section describes a synchronization method which uses a combination of header metaframing and payload tagging. We will refer to this as hybrid metaframing. It overcomes many of the shortcomings of the other four synchronization methods:

- Header Tagging: This method can only compensate for a limited amount of skew; the number of tags are limited by the small number of VCI bits.
- Payload Tagging: The number of tags is no longer limited, but this method causes striping to no longer be transparent to the adaptation layer.
- Header and Payload Metaframing: The use of metaframes allows compensation for large amounts of skew, but at the cost of the loss of large amounts of data when

Round 8	Round 7	Round 6	Round 5	Round 4	Round 3	Round 2	Round 1	
Data Cell	Data Cell	Data Cell	OAM Cell 2	Data Cell	Data Cell	Data Cell	OAM Cell 1	Stripe 1
Data Cell	OAM Cell 2	Data Cell	Data Cell	Data Cell	OAM Cell 1			Stripe 2
Data Cell	Data Cell	OAM Cell 2	Data Cell	Data Cell	Data Cell	OAM Cell 1		Stripe 3
Data Cell	Data Cell	Data Cell	OAM Cell 2	Data Cell	Data Cell	Data Cell	OAM Cell 1	Stripe 4

Figure 7-4: Hybrid Metaframing

synchronization is lost. In addition, payload metaframing requires the ability to insert and delete the synchronization payloads.

Operation and maintenance (OAM) cells provide an attractive way to implement a scheme which overcomes all these shortcomings. OAM cells may be sent end-to-end on existing virtual circuits. They are distinguished from data cells by one of the payload type bits. Since they are on the same virtual circuit, this eliminates the co-routing difficulties present with header metaframing. In addition, the OAM facilities handles inserting and deleting them, rather than requiring the host software to handle it.

The standards already support a performance management function using OAM cells; a request to send an OAM cell is initiated every N cells[9] and the OAM cell is inserted at the first free cell slot after the request[27]. The transmitter could make an OAM request on each virtual circuit and then halt transmission for one cell slot so that the OAM cells could be transmitted immediately; this would ensure that OAM cells are sent at the same time on all of the virtual circuits. The OAM cells would be able to function in the same way as header metaframing VCI cells, but would not incur the same network difficulties.

7.7.1 Effects of Skew

In the pure metaframing methods, the metaframes were used to compensate for skew and to maintain synchronization; in order to improve the ability to do one of these functions, the other one suffered. The advantage of hybrid metaframing is that it uses header metaframing only to maintain synchronization; within the OAM cell payload is a tag which allows the construction of superframes which are many metaframes long (Figure 7-4).

The length of the metaframes can be such that synchronization is verified often enough

[9]N may be 128, 256, 512, and 1024

to avoid having to retransmit a huge amount of data. The length of the superframe is used to compensate for skew. The amount of skew which can be protected against is the length of a metaframe $\times$ the number of tags[10]. If we assume that we are using the smallest metaframe size provided by the OAM standards, which is 128 cells, then we require 250 tags to compensate for the worst case skew we presented in the payload tagging section. This only requires an 8 bit tag; there 45 function specific bytes available in an end-to-end OAM cell[11].

7.7.2 Network Considerations

The standards already provide for almost all of the functionality we have described here. The one thing required is the addition of the striping maintenance type to the OAM standards. This addition should provide for the ability to create and synchronize to metaframes using OAM cells; it should also provide the ability to create superframes. The hosts should be able to provide some of the parameters to the maintenance facilities, such as the length of the metaframes and superframes; however this should not be required. In addition, the frequency with which OAM cells can be transmitted should be made more flexible so that the synchronization algorithm can be dynamically adjusted as the skew changes; it should be possible to increase the metaframe size slightly if the skew increases slightly. Under the current standards, the metaframe size could only be doubled or halved.

The other requirement this method has of the network is that the virtual circuits provide all the bandwidth which they guarantee.

7.7.3 Hybrid Metaframing Summary

Hybrid metaframing appears to be an optimal choice of synchronization method for ATM layer striping. It overcomes the shortcomings of all the other methods. The ability to combine metaframes into superframes allows large amounts of skew to be compensated for, which the header tagging method lacked, and still allows for frequent verification of synchronization, which the header and payload metaframing methods lacked. In addition, it does not require that large amounts of each cell payload be used for overhead. In addition,

[10] Notice that, in our example, we have tagged the metaframes per stripe. This allows the use of the same set of tags on each stripe simultaneously which increases the amount of skew which can be protected against.

[11] Using all 45 bits, the maximum skew we could protect against with 128 cell metaframes becomes 4500 trillion rounds.

all the synchronization information is carried by OAM cells, so hybrid metaframing is transparent to the adaptation layer, unlike payload tagging. In addition, it only requires one VCI to be set up per stripe, which eliminates much of the network trouble caused by the two header-based methods.

In some sense, we can view the use of OAM cells as a "code-based" synchronization method. The network provides the hosts with access to a virtual circuit but not to the OAM cells; instead, these cells are reserved by the network for various maintenance functions. We can interpret the synchronization of a striped connection as a maintenance function and leave it up to the facilities responsible for those functions.

7.8 Dynamically Reconfiguring the Number of Stripes

One issue which we have not yet addressed is the ability of each of these schemes to dynamically add or drop stripes as the bandwidth requirements increase or decrease. In order to perform this function, two elements are required. First, the source must have some way to inform the destination of the increase or decrease in the number of stripes, along with any other relevant information, such as the VCI associated with that stripe, etc. We have described methods for in-band communication for each of the four synchronization possibilities; this in-band communications channel can be used for this purpose.

In addition, the source must be able to actually add or drop a stripe. In both of the striping cases which use the payload to carry the synchronization information this just requires setting up one more VCI which travels over the new stripe or tearing down one of the existing VCIs. The same requirement exists for the header metaframing case which uses OAM cells as metaframing cells; only one VCI needs to be set up or torn down. The header tagging method requires the same number of VCIs be set up on the new stripe as the number which currently exists on each of the other stripes.

7.9 Load Balancing

In all of the descriptions of the synchronization alternatives we have mentioned the possibility that ABR virtual circuits may not be able to guarantee the aggregate bandwidth which we desire. This is due in part to the fact that the network expects to be able to get some advantage from statistical multiplexing. This means that, although several different

virtual circuits may request a peak amount of bandwidth, it is unlikely that they will all be used at the peak rate simultaneously because they are independent from one another. This is clearly not the case with the virtual circuits used for striping; they will all be used at the peak rate simultaneously. Thus, the bandwidth which the network has "promised" to the transmitting host may not be available.

When this situation occurs to the virtual circuits of a striped connection, the transmitting host may want to consider implementing some sort of load balancing on the stripes. This would allow the load to be shifted to the stripes which have more available bandwidth. In order to perform this function, however, the transmitting host requires some sort of feedback from the network concerning which virtual circuits are congested and which are not. This feedback generally takes the form of flow control, of which there are two general types in the ATM domain.

The first type, rate-based flow control, attempts to control the rate at which a virtual circuit is allowed to transmit cells into the network. In this case, the transmitter may not need to perform any active form of load balancing. If we assume that the rate-based flow control algorithm attempts to be fair, it will try to equalize the rates of all the virtual circuits associated with the striped connection. The transmitter merely needs to adhere to these rates in order to perform load balancing.

The second type of flow control is credit-based. This is the sort of flow control performed by AN2. The basic idea with credit-based flow control is that each point in the network which is transmitting cells has a count of the number of buffers it may fill in the receiver; these are called credits. When cells are removed from the buffers, the transmitters receive credits; when they transmit cells they debit their buffer count. As downstream buffers become full, backpressure will keep the upstream nodes from transmitting, which will cause their buffers to fill, etc. This pressure will extend all the way back to the hosts, who will the cease to generate traffic until the congestion is cleared. We can view the number of credits which each virtual circuit has at the transmitter as the necessary feedback. The transmitter can send more cells on VCIs which have more credits, and send more slowly on VCIs which have fewer credits. However, this will require a more complex synchronization strategy, since cells may no longer be transmitted in a strict round robin rotation.

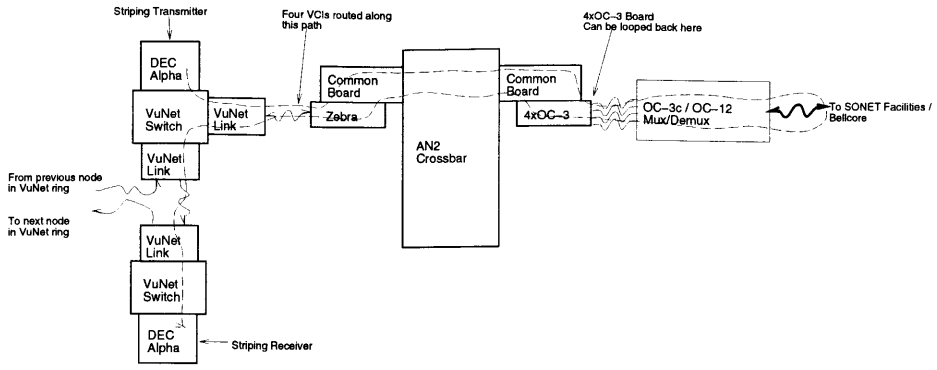


Figure 7-5: Experimental Topology

7.10 Experimental Implementation and Results

We implemented two striping schemes at the ATM layer. The experimental topology used for both schemes is shown in figure 7-5. The first consists of two VuNet hosts performing a very basic striping transmission and reception algorithm with no synchronization. This scheme was used to detect and show the effects of skew when no attempt is made to compensate for it. The second uses a header metaframing scheme to compensate for skew introduced in the network either by the SONET facilities or the multiple loopback of virtual circuits on the AN2 quad OC-3c line card (QLC).

7.10.1 Skew Detection and Measurement

This striping scheme implements the striping unit loss detection method described in chapter 2: the receiver does nothing to protect against skew and assumes that striping units which do not arrive on time have been lost. This scheme serves mostly as a test tool for verifying that the network is operational. It is also used to verify that skew which should be present in the network is actually there and to determine how much skew is introduced.

Striping Transmitter

The transmitter takes the following arguments: a destination, a base VCI, the number of cells to transmit, and a node in the network through which to route the VCIs. Four stripes are used by default since there are four OC-3c channels which can be used as stripes.

The transmitter uses the connection setup software developed for the VuSystem in order to set up four virtual circuits between the source and destination that pass through the

specified node. This software required modification in two ways. First, it needed to be modified in order to be able to program the AN2 common board VCI mapping tables. It also needed to be modified in order to be able to route the virtual circuits through the specified node[12]. In addition, software was needed for the Line Card Processor (LCP) in order to allow it to receive cells and modify its VCI tables based on the contents.

When routing VCIs through the QLC on the AN2, the software running on the LCP can extract the output line which that VCI will be transmitted on. The transmitting host sets up four virtual circuits, and each of them is routed using a different output line on the QLC; this is the internal splitting point in the experimental apparatus.

The four virtual circuits are on four contiguous VCIs starting with the base VCI given to the transmitter. After they have been set up, the striping transmitter begins to send cells on them in round robin order, starting with the base VCI. In order to monitor the effects of skew, the cell payloads contain a sequence number indicating the order in which the cells were transmitted.

Striping Receiver

The striping receiver runs on a different host on the VuNet and takes as arguments the source of cells and the base VCI which the transmitter is using. It sets up four sockets corresponding to the VCI for each stripe. The remainder of the receiver consists of two loops. The first one just handles the reception of cells without any processing; it examines the status of the four sockets to see if any new cells have arrived. If there are new cells, they are appended to a list of cells corresponding to the appropriate stripe.

The other loop implements the algorithm which the receiver uses to reassemble incoming data. For the testing scheme, this algorithm checks to see if any new cells have been added to any stripe's list. Initially, if there are new cells, this loop begins looking at the stripe corresponding to the base VCI. It prints the contents of the cell payload and removed it from the list, and then moves on to the next stripe in the rotation. It continues to do this until it reaches a stripe which has no cells waiting for processing, at which point it exits. The next time the loop runs, it begins looking at this same stripe. If there are still no cells

[12]This was necessary in order to set up connections between two hosts on the VuNet which traveled through the AN2 switch and the SONET facilities in loopback instead of setting up the connections directly between the two hosts.

waiting on the list for that stripe, it assumes that the cell has been lost and moves onto the next stripe in the rotation, prints the contents of that cell, and continues until it reaches another empty list.

If there is enough skew to delay cells on a stripe by a round, then the list for that stripe will be empty not only the first time the processing loop examines it but also the next time the host receives cells and enters the processing loop. Thus, cells on this stripe will be displayed in the output as being out of order. This is easily determined by using the sequence numbers in each cell.

Testing Scheme Results

There are two ways to introduce skew into the experimental apparatus. The first way is to create a chain of virtual circuits on each of the OC-3c lines on the QLC. The VCI which is routed through each output line is directed back to that line on another VCI. The length of the chain depends on the number of times this process occurs. The final VCI is routed back to the Zebra on the original VCI. By choosing a different chain length for each output line, a variable amount of skew can be introduced between all of the stripes. The testing scheme allowed this code to be debugged and also allowed us to easily determine the chain length required to skew a stripe by a given number of rounds.

The other interesting thing the testing scheme allowed us to do was to characterize the skew introduced by the SONET facilities. The SONET facilities were looped back at a distant location and the testing scheme was activated. We found that the skew introduced by the SONET facilities was fixed; the first and fourth stripes were delayed by a single round. This skew has not changed the entire time we have been running experiments over the SONET facilities.

7.10.2 Header Framing Striping Implementation

This scheme actually attempts to compensate for the skew introduced by the network. Again, there are a striping transmitter and a striping receiver which are running on separate hosts. The striping transmitter is largely the same as the testing scheme's transmitter. It takes an additional argument which is the number of sequence numbers to use. It then sets up a large number of virtual circuits; using the base VC it determines a base VC for each stripe. The base VC for each stripe is then used to set up a set of contiguous VCIs, one

for each sequence number. So, with four stripes and a sequence number size of eight, the transmitter sets up 32 VCIs. These VCIs are paired with sockets in such a way that the transmitter can send cells on sockets 0 through 31 sequentially and the cells will be sent in round robin order on the stripes with the VCI corresponding to the current sequence number for that round.

Similarly, the striping receiver is largely the same as the testing scheme's receiver. The only major changes were to the cell processing loop; the cell reception loop stayed the same. The reception loop insures that cells are placed on the queue for the proper stripe based on the two low order bits of the VCI. The processing loop uses the rest of the bits of the VCI to determine the sequence number of each cell. Initially, the receiver starts looking for the first sequence number on the first stripe, and keeps track of the sequence number which the current round of cells should contain and the stripe on which it is expecting the next cell. If the cell arrives on the current stripe with the proper sequence number, it is considered to be in the proper order, removed from the queue for that stripe, and displayed. If the cell has a greater sequence number, then the receiver knows that some number of cells have been lost on that stripe, displays this fact, and leaves the cell on the stripe until its proper place in the order comes around.

This striping method was implemented and used to successfully reorder cells which had been skewed after traveling through the SONET facilities. In addition, VCI chains on the QLC were used to introduce a skew greater than that introduced by the facilities, which the method still protected against. The sequence number length does not protect against skew so much as it protects against the number of consecutive cells which can be lost on one stripe.

The striping code which was written for this thesis operates in user space. This was done for simplicity, so that some results could be achieved without a massive investment of time and effort. If the ATM layer striping were to be made a fully functional part of the VuSystem software, then the striping implementation would be hidden in the hosts' kernel. Since both the ATM and adaptation layers are handled in the kernel, it would be a relatively straight-forward task to implement the striping implementation between the two layers. Cells coming in on the virtual circuits associated with a striped connection would be handed off the the striping code. The striping code would place the cells in the proper order and then hand the cell payloads off to the adaptation layer portion of the kernel.

The inverse operation would be performed during transmission. Packets would be placed into AAL5 frames, which would be segmented into ATM payloads. These payloads would be handed to the striping code for placement into cells with the proper VCIs. These cells would then be transmitted. To any user application, the striped connection will appear to be one logical path; the striping code in the kernel provides the support for using multiple virtual circuits.

7.11 Conclusion

In this chapter we have presented a detailed look at implementing striping at the ATM layer. In particular, we have examined several possible ways to perform the necessary synchronization and some of the tradeoffs related to each method. We also examined some of the functions which we require from the network in order to implement some of these methods properly. From this discussion we have identified two methods which would be acceptable ways to implement striping at this layer: VPI header tagging and hybrid metaframing using OAM cells.

The VPI header tagging method eliminated most of the misordering problems inherent in simple header tagging; it also avoided the creation of a custom adaptation layer, thus remaining transparent to all the layers above the ATM layer. Finally, it does not place any new demands upon the current standards, such as requiring the addition of VCI clusters, so it could be retrofitted to function on existing networks. The major drawback is that the VPI space is limited at the UNI; there are only 256 VPIs available as stripes.

The hybrid metaframing method overcomes all of the same shortcomings as the VPI header tagging method. However, by tagging metaframes rather than individual cells, a much greater amount of skew can be protected against. By using OAM cells as the metaframe boundaries, the co-routing problems associated with header metaframing are avoided, but the synchronization method does not touch the payload of data cells and so is transparent to higher layers. It does not infringe on any current standards. However, it requires that some consideration be given in the future for the use of OAM cells with striping.

The next chapter will present our discussion of striping at the adaptation, network, and transport layers. A great deal of the discussion in this chapter will be relevant to our

examination of striping at other layers, particularly adaptation layer striping.

Chapter 8

Striping at Upper Layers

The previous chapter examined striping at the ATM layer in detail. We will now consider adaptation layer striping, network layer striping, and finally, transport layer striping. The process in each case will be much the same as the cell striping examination; the possible locations for synchronization information will be identified and ways to use the available tools will be described. Finally, we will examine some of the other issues and requirements which go into a successful striping implementation.

We will show that, although striping can be implemented at each of these layers, their use is strictly limited; to achieve the same scope of application as striping at the ATM layer would require much more time and effort.

8.1 Adaptation Layer Striping

There are several adaptation layers which are defined to carry various types of data through an ATM network. In this discussion, we will only examine AAL5. This is because this is the adaptation layer used by the VuNet clients. Study of the other adaptation layers is left for future work.

8.1.1 Synchronization Options

In general, we will see that the same techniques used for synchronization at the ATM layer can be used at the adaptation layer as well; all that we need is some place to put the synchronization information. However, there are a few ways in which AAL5 frames differ from ATM cells which will affect which of the various synchronization options we wish to

use. AAL5 frames will be many cells long; this will make them more resistant to skew than ATM cells, in the same sort of way that metaframes were more resistant to skew than individual cells. It requires a much larger amount of skew to make the entire frame appear as though it is lost or out of order. There is also an issue about how higher layer packets are placed into AAL5 frames. A higher layer packet may be placed entirely in one AAL5 frame. In this case, synchronization could either be done at the adaptation layer or at the higher layer. However, if higher layer packets were segmented and sent in several AAL5 frames, then the synchronization would have to be done at the adaptation layer.

Location of Synchronization Information

There are two basic places in an AAL5 frame where synchronization information may be placed: the payload and the trailer. The trailer has a couple of fields which could be used; there is a byte for the transfer of user-to-user information and the CPI byte which aligns the trailer on a 64 bit boundary[29]. As with the ATM cell, anything could be done with the payload. Unlike the ATM cell, however, we have a little more freedom to modify the payload of the AAL5 frame without violating the standards. Since the standards define a service-specific convergence sublayer in the adaptation layer, we can consider any information and processing necessary for striping as part of a striping service.

Use of Synchronization Tools

Again, we can apply our standard tools to striping at this layer basically in the same fashion as with cell layer striping. The tagging scheme can be implemented by placing sequence numbers in either the trailer or in some new field in the payload. Similarly, the metaframing scheme can be implemented by using either the trailer or part of the payload to create recognizable boundaries on each stripe which the receiver can align.

Effects of Skew

Since it is possible for the AAL5 frames to vary in length, it is difficult to get an exact measure for how the number of tags or the length of a metaframe should scale as the skew increases. We can address the extreme cases, however.

The largest AAL5 frame size is 64 Kbytes, which is 1362 cells, results in a 3.7 ms frame time on an OC-3 line. Our worst case scenario from the payload tagging method in the

previous chapter resulted in a skew of 22.5 ms, which is about 7 frame times. Thus, if every frame were maximum size, we would need 8 × (number of stripes) tags or metaframes which were 8 frames long.

The opposite extreme is an AAL frame size of one cell. This case becomes equivalent to either of the payload-based methods presented in the previous chapter.

8.1.2 Requirements and Limitations

Since the adaptation layer uses the virtual circuits provided by the ATM layer, we will need to have a virtual circuit for each stripe, just as we did for the various payload-based striping methods at the ATM layer. This requires some way for the source to ensure that the network routes the virtual circuits properly. In addition, we must also be concerned about the type of virtual circuit which has been set up; CBR circuits guarantee us the bandwidth we desire, ABR circuits provide no such guarantee.

In addition, the presence of varying length frames may require some form of load balancing. We could consider implementing the deficit round robin scheme in order to accomplish this. This complicates the synchronization scheme, however, due to that fact that the receiver no longer knows how many frames will be sent on each stripe during a round. Therefore, a framing scheme becomes more difficult to implement and using sequence numbers may be more appropriate.

In this discussion we have only addressed AAL5; there are currently three other standard adaptation layers. Separate consideration is necessary for each of them, not to mention any possible future AALs. The techniques used to stripe using one AAL will not apply to another, creating a great deal of redundancy in software; different code will be necessary for each AAL. Since all adaptation layers operate over ATM, there seems to be little advantage gained by striping at the adaptation layer rather than the ATM layer. That way, one method would support all types of traffic from higher layers.

8.2 Network Layer Striping

The discussion on striping at the network layer will focus primarily on IP. It will vary quite a bit from the considerations of ATM and adaptation layer striping because there is already support for out-of-order delivery of IP packets. In addition, there is some flexibility for the

hosts to determine the routes and addresses used to transmit the data. In order to make striping at the IP layer operate with reasonable performance, both of these characteristics must be optimized for striping.

Note that there are two reasons why IP packets segment higher layer data. The first is that higher layer units may be too large for a single IP packet. In this case, the higher layer protocol passes blocks of data to the IP layer which are then placed into individual IP packets. The reordering and assembly of these packets is managed by the higher layer protocol. In the other case, passing through a channel or router from one network type to another may require that an individual IP packet be fragmented into several smaller IP packets. The assembly of these fragments is handled by the IP layer at the receiver.

There is no support for the efficient striping of IP traffic between a given source and destination host pair. Instead, the out-of-order arrival case is treated as a possibility that can result from a routing change in the network. The two elements of IP which need to be changed to support striping at this layer are the routing choices made by a router with access to multiple stripes and the reassembly algorithms in the destination host.

8.2.1 Router Changes for Striping

The component at this layer which would do striping would be an IP router. The router will have several possible routes to a single destination address available and must decide how to route each incoming IP packet on that address. This is the approach we must pursue since we cannot use multiple addresses, each with a different route. There are a number of permitted ways which the router may choose the route[30]. It may pick one route based on some type of cost function. It may randomly pick a route. Finally, it may perform load-splitting. Load-splitting is not explicitly defined, but most routers do it on a per-destination rather than a per-packet basis in order to minimize packet re-ordering. IP fragments are treated the same as IP packets at the routers, so we need not worry about dealing with them in a special manner.

There are two things necessary for a router to be able to stripe properly. The first is some way to indicate that traffic between a specific source and destination should be striped. Otherwise the router will stripe all the traffic which passes through it, which may not be desirable. The second change is to the routing code so that the router does load-splitting by striping on a per-packet basis; this code would almost certainly contain some sort of load

balancing algorithm as well.

8.2.2 Host Changes for Striping

Since an IP address uniquely determines one network interface on one hosts, we must look elsewhere to implement the striping capability. The primary change which should be made to the hosts is the optimization of the software which handles the reassembly of incoming IP packets. Currently, the assumption is that packets will arrive largely in order, so handling misordered packets is something which only needs to be dealt with infrequently. As the routers begin striping, however, misordering will become more common, and so the software must be able to deal with it without taking a major performance hit.

Another feature which may be implemented in the hosts but which is not necessary is the ability to fragment IP packets even if the MTU of the network to which the source is attached does not make it necessary. This will allow large datagrams to be broken into smaller pieces which will derive more of an advantage from striping.

8.2.3 Limitations

Striping at the network layer suffers from a similar limitation as striping at the adaptation layer; each different network protocol needs to be considered separately. This can lead to either a different striping implementation for each network protocol, or for striping support for only a few select protocols. Assuming that ATM will be the technology used in a new generation of wide area networks, implementing striping at the ATM layer allows one striping implementation to provide an increase in performance to all network protocols.

8.3 Transport Layer Striping

The OSI transport protocol TP(4)[31] actively supports the use of multiple network connections to carry traffic associated with one transport layer connection. Similar to TCP, it compensates for the possibility that packets may arrive in the wrong order by using a sequence number to determine what the proper order is. Unlike TCP, however, it also provides the transport layer to specify more than one address to be used for that transport layer connection. TP(4) can then stripe the data units which it generates across the multiple network addresses. Source routing for each network address can be used in order to use

the physical stripes in the network properly.

Transport layer striping has the same limitations as network and adaptation layer striping; a separate implementation is required for every different transport layer. Either there will be a large amount of redundant software or some transport layers will simply not have striping capability.

8.4 Application Layer Striping

A concept that meshes well with striping is that of Application Level Framing, or ALF[32]. ALF defines an aggregate of application data called an Application Data Unit (ADU). The defining characteristic of ADUs is that they can be processed out of order by the application. This characteristic would be particularly useful because it would eliminate the need for any lower layer synchronization by the receiver. The transmitting application could simply stripe ADUs and the receiving application would know how to deal with receiving them out of order.

Once again, application layer striping is limited to applications which have clearly defined ADUs and know how to stripe them; all other applications will remain limited.

8.5 Conclusion

This chapter has taken a look at the issues associated with striping at the adaptation, network, and transport layers, with a mention of application layer striping as well. The detailed striping discussion about implementing ATM layer striping serves as a great aid in discussing adaptation layer striping, since the synchronization tools and requirements are almost identical; the only significant differences being that the adaptation layer provides more flexibility to fit within existing standards at the cost of some extra complexity due to the variability in adaptation layer frame size.

The examination of network and transport layer striping was somewhat different since tools are already in place to overcome the largest problem related to striping: that of synchronization. Instead, we looked at what changes to the existing networks are required to make striping function well.

We can see that, while it is possible to describe a striping implementation for a specific protocol at each of these layers, in order allow a wide range of protocols to benefit from

striping, we must either stripe at a layer below all of these protocols, or we must design a separate striping implementation for each one.

Chapter 9

Conclusion

In this thesis, we have proposed striping as a means of providing increased network performance which alleviates the problems of innovation and technology coupling between the local and wide area networking domains. A framework was presented upon which an arbitrary network striping implementation could be laid out and analyzed. We then took that framework and identified a few interesting and previously unexplored striping cases which we felt solved the problems we presented earlier. These cases were then analyzed using the framework and several well-known synchronization tools; the implementation of each was discussed and other requirements beyond those related to striping were revealed. The following sections will summarize our results and propose some directions for future work.

9.1 Striping Framework

By considering a general striping implementation, four key degrees of freedom which completely specify a striping case were isolated. Three of these degrees of freedom specify the situation and architecture in which the striping occurs; striping topology describes where it happens, host participation describes who does the work, and the striping layer describes what gets striped and what network protocols need to be dealt with. The final degree of freedom, synchronization options, addresses the implementation of the striping.

The striping topology describes how the path between two hosts is split into stripes. We considered two basic cases: end-to-end splitting and internal splitting; the possibility of more complex topologies based on these two simple cases was presented as well. After examining the various striping implementations, particularly at the ATM layer, the problem

with more complex topologies is assuring that striping units are routed through the network in a way that takes advantage of stripes. Rather than simply splitting into stripes once, real world topologies will split and converge and split again; it is important to either guarantee that the stripes will be used properly or to choose a method which does not depend on how the stripes are used. In most of the ATM layer cases we handled this issue by assuming that if hosts requested some number of virtual circuits with some amount of bandwidth, and if these requests were granted, then the network would provide us with the aggregate bandwidth we desired and it didn't matter how the VCIs were routed.

The participation of the hosts or endsystems was the next degree of freedom. Either the source and destination can be completely unaware of any striping in the network or they can be responsible for some part of the striping functionality. Whatever portions of the striping implementation are not handled by the hosts are handled by the network. The topology affects the options here; end-to-end splitting requires active hosts, while passive hosts require that the splitting be internal. We only considered cases where the hosts were active; this serves as a way to decouple most of the details of the striping implementation from the actual network equipment. This is desirable because it allows portions of the network to be upgraded and still use the same striping implementation. However, the network still has a role in the striping implementation, because without certain functions and guarantees which it provides, striping becomes impossible.

The striping layer, along with the issues associated with it, is the next degree of freedom. The possible striping layers are determined not only by the location of the splitting, but also by whether or not the hosts are participating in the striping. The striping layer affects the size and format of the striping units which are transmitted in parallel on the stripes. In addition, the striping layer determines what layer in the network we require to be homogeneous; this makes some layers more attractive than others.

We stated earlier that the primary problem with using several channels in parallel rather than one channel is that skew can arise between the channels; skew, coupled with concern for the loss of striping units, can lead to data arriving at the destination in the wrong order. Some form of synchronization across the stripes is necessary in order for the striping implementation to function properly. The synchronization options are affected most by the striping layer. The striping units, determined by the striping layer, define where the information needed by a synchronization algorithm can be placed. From the well-known

techniques for synchronization on one channel we derived two basic types of synchronization methods: tagging and framing. So, the options for implementing synchronization come from using one of these two methods in each of the places in the striping unit which can hold the necessary information.

Having created the framework, we then laid out the entire striping space made possible within it. After eliminating striping cases which were nonsensical, covered by previous work, or subsets of other cases, we were left with a set of four striping cases to consider. These were the active, internal striping cases at the cell, adaptation, network, and transport layers.

9.2 The Case for ATM Layer Striping

We argue that the ATM layer is the most appropriate layer at which to implement network striping. Striping at the physical layer below the ATM layer creates an implementation which is highly dependent upon the specific network equipment; as equipment is upgraded the striping implementation will cease to function. This violates one of our primary reasons for striping in the first place.

To be able to use different physical layers transparently, we must implement striping at a higher layer. However, at each layer, there are a number of different technologies and protocols which each need their own striping implementation. Striping at the network layer requires a different implementation for IP and IPX; striping at the transport layer requires implementations for TCP and TP(4), etc. The ATM layer is a good compromise between supporting a wide variety of higher layers and requiring that separate striping implementations be developed for other protocols at the same layer. The telephone companies are moving towards using ATM over their wide area facilities; since we are interested in using striping to alleviate problems associated with communicating over the wide area facilities, it makes sense to use ATM as well. In addition, the ATM layer supports all AALs, network protocols, transport protocols, and applications. As we begin striping at higher layers we begin to exclude some of them.

9.3 Future Work

There are several directions in which future work should be performed, both on the implementation and the model. As far as the implementation, there is work to be done to incorporate the striping implementations into the VuNet host software so that actual traffic could be striped, rather than fixed patterns generated by some simple user programs. This work includes adding ATM and adaptation layer striping implementations into the kernel, and modifying the IP software to deal with misordered packets in a more efficient manner.

As far as modelling work, we have only considered striping cases which are active and internal cases as part of this thesis. This was due to our desire to have the striping implementation be as independent from the network equipment as possible. By implementing striping in the network using equipment such as the Unison ramps or BONDING equipment, the striping implementation becomes tied to the network. Future work needs to be done on architectures which support internal striping but which are not dependent on specific network equipment. This would support a slow migration to new technology in the network while allowing hosts to continue to use their current network software.

Along with the consideration of new architectures to support striping should come consideration of new standards to support striping as well. The BONDING standard is one current effort in this direction. As we've seen from the consideration of ATM and adaptation layer striping, the lack of attention to striping in current standards can lead to great difficulties in implementing striping. Without this consideration, however, users of wide-area facilities will continue to be limited to the wide area's slow progress.

Appendix A

Justification for Unconsidered Striping Cases

There are three striping cases eliminated from the original striping space because they made no sense to implement. These are:

1. Bit/Byte Layer Striping, Internal Splitting, Active Hosts
2. Network Layer Striping, Internal Splitting, Passive Hosts
3. Transport Layer Striping, Internal Splitting, Passive Hosts

We consider the first case to be uninteresting for the following reason. The only way for the end-systems to participate in the striping is with some sort of tunneling procedure. It must create frames which the network can use to properly get the contents from the source to the destination using the proper stripe. The contents of the frames must be the interleaved pattern of cells which would be transported down each stripe in the end-to-end or transparent cases. We see the creation of this framing structure as pointless, since perfectly good framing structures for this purpose, such as ATM cells, already exist. So now the striping unit is essentially these frames rather than bytes, with the format of the payload just making extra work for the receiver.

The second case is uninteresting because while a network node has the ability to split incoming network units and choose different routes for them, they generally don't support the ability to reassemble and reorder network units. This function is handled by the transport layer in the hosts, which makes this an active rather than a passive striping case.

The third case was the example given in Chapter 4. Since the transport layer is only defined at the hosts, the hosts must be involved in striping at this layer and so this is an active and not a passive case.

There are five cases which have been determined to be either equivalent or subsets of other cases. These are:

1. Bit/Byte Layer Striping, End-to-End Splitting, Active Hosts
2. Cell Layer Striping, Internal Splitting, Passive Hosts
3. Adaptation Layer Striping, Internal Splitting, Passive Hosts
4. Network Layer Striping, End-to-End Splitting, Active Hosts
5. Transport Layer Striping, End-to-End Splitting, Active Hosts

The first case is equivalent to the internal, passive case which was covered by the Unison and BONDING work. To justify this statement, let us consider how both cases function. The striping components in the internal, passive case see an arriving stream of packets from the next higher layer in the protocol stack. They take the bytes of these packets, stripe them across the available channels, and frame them with synchronization information. The receiver reconstucts the incoming packets and then forwards them onto the network to which it is attached. The arrival of the packets to the first node and the routing at the end of the second is determined by information in the higher layer packet and is independent of the striping. So, we can envision replacing the networks with hosts and using the same implementation. Thus the two cases are functionally equivalent.

The second case is a subset of the end-to-end, active cell layer striping case. We can use an argument similar to the one above in order to demonstrate this. In the end-to-end case, the hosts segment higher layer packets into a sequence of ATM cells. These cells are then presented to some entity which handles the striping functions. Some sort of framing technique is used to allow the receiving host to determine the proper cell order and reassemble the higher layer packet. The internal striping can be viewed in the same way; the first internal node receives a sequence of cells which it can frame and stripe; the second internal node can reorder the cells properly and send them on their way. The reason why the internal case is a subset of the end-to-end case is that the hosts have the option of using

an AAL which supports cell layer striping; this is something which isn't available to the nodes just handling cells.

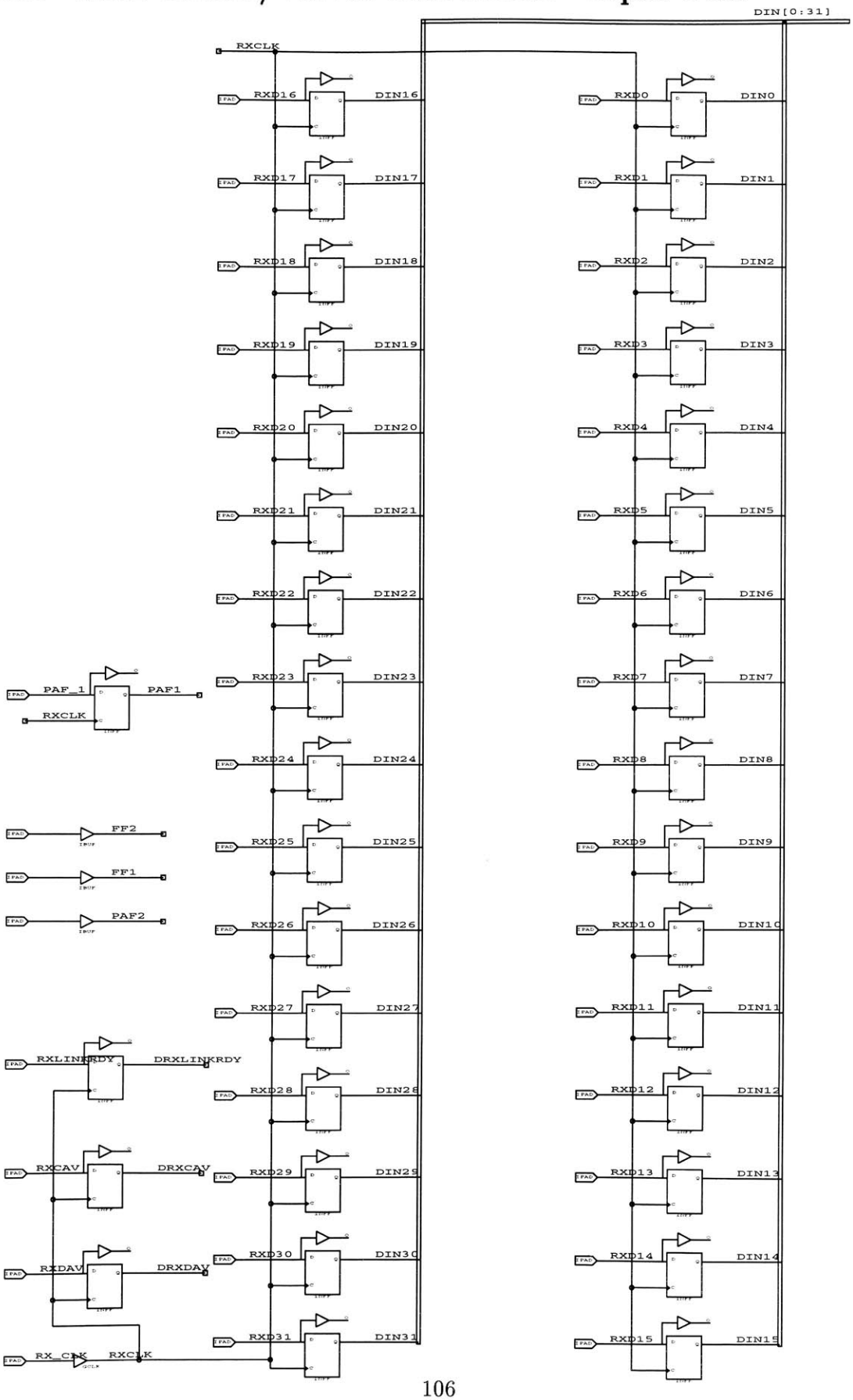
The third case is equivalent to either the end-to-end, active case of adaptation layer striping or some form of network layer striping. The adaptation layer is only defined at the edges of an ATM network, so any network nodes sending frames must be using them to transmit network layer units. This means they have at least the functionality of something like an IP router, for example. There are then two possible reasons that the stripes exist. First, they may be intended to be used as a single path at the network layer between two nodes. In this case, any striping implementation at the adaptation layer will be equivalent to an end-to-end and active implementation because the adaptation layer gets terminated at the two nodes. Next, the stripes may actually be different routes which the first node can choose from at the network layer. This is equivalent to striping at the network layer and just using the adaptation layer as a transport mechanism.

Appendix B

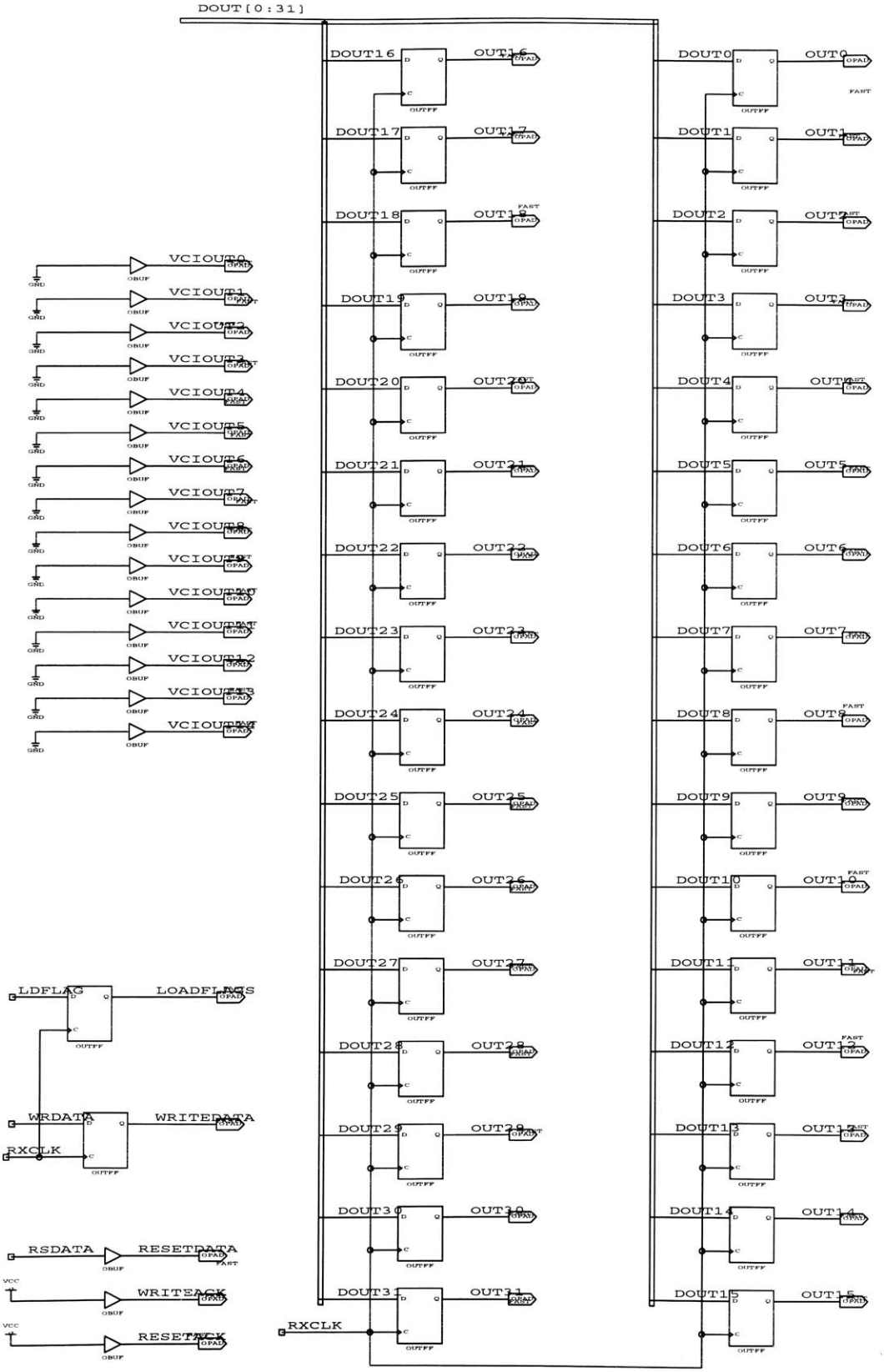
Xilinx Schematics

This appendix contains the schematics which are used to configure the Xilinx FPGAs on the Zebra. These schematics were generated using the ViewLogic PRO suite of CAD tools.

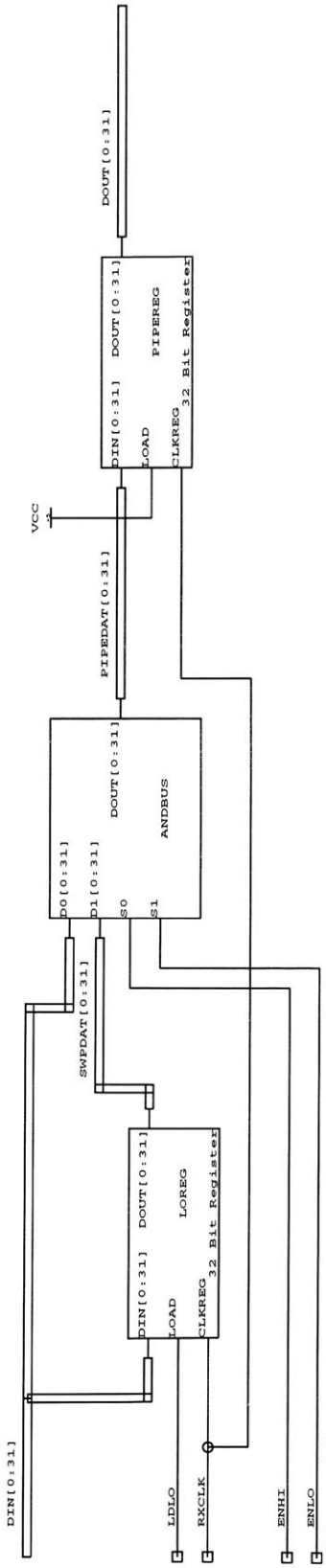
B.1 AN2-Bound/VuNet Side Xilinx - Input Pins



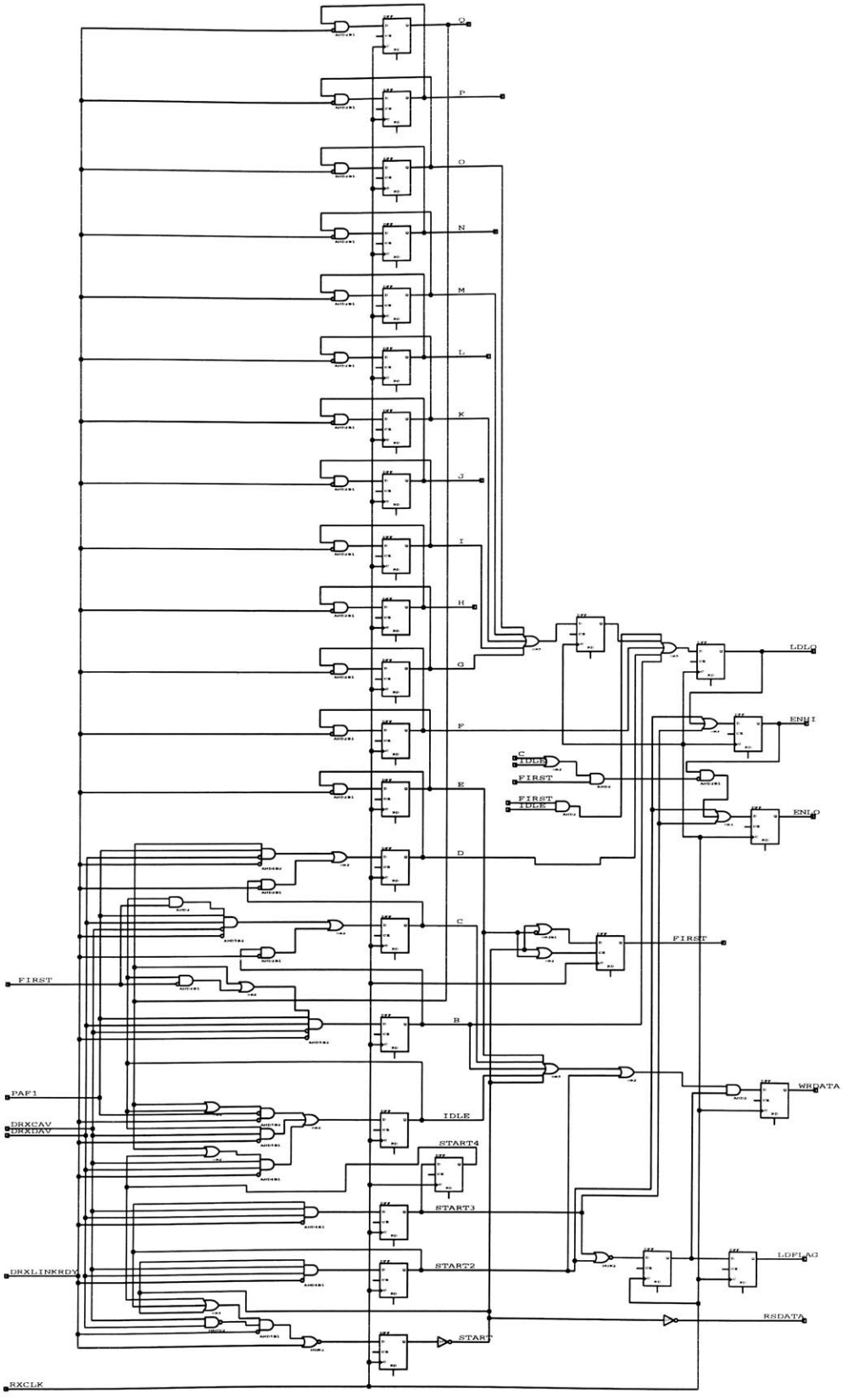
B.2 AN2-Bound/VuNet Side Xilinx - Output Pins



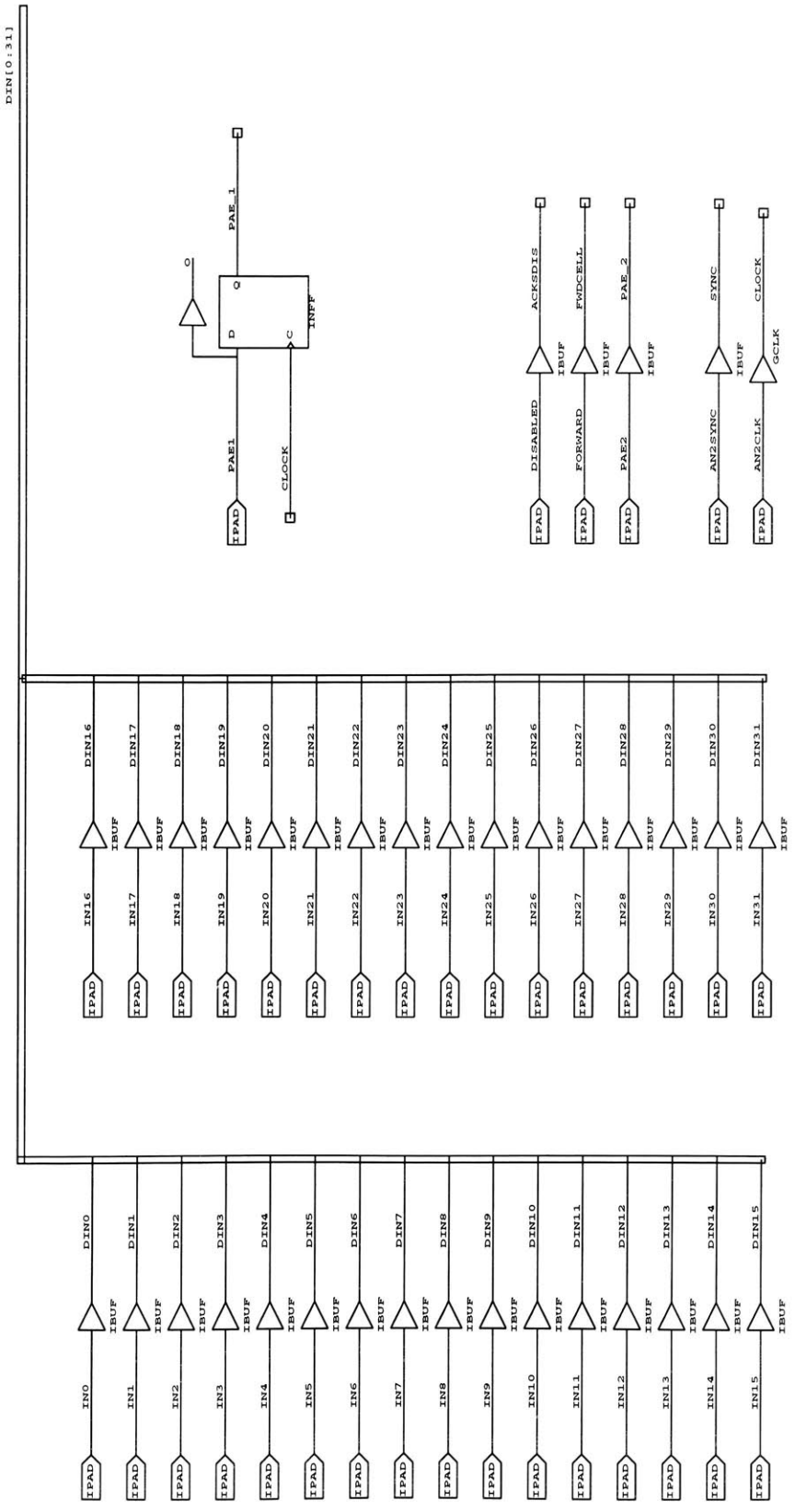
B.3 AN2-Bound/VuNet Side Xilinx - Data Path



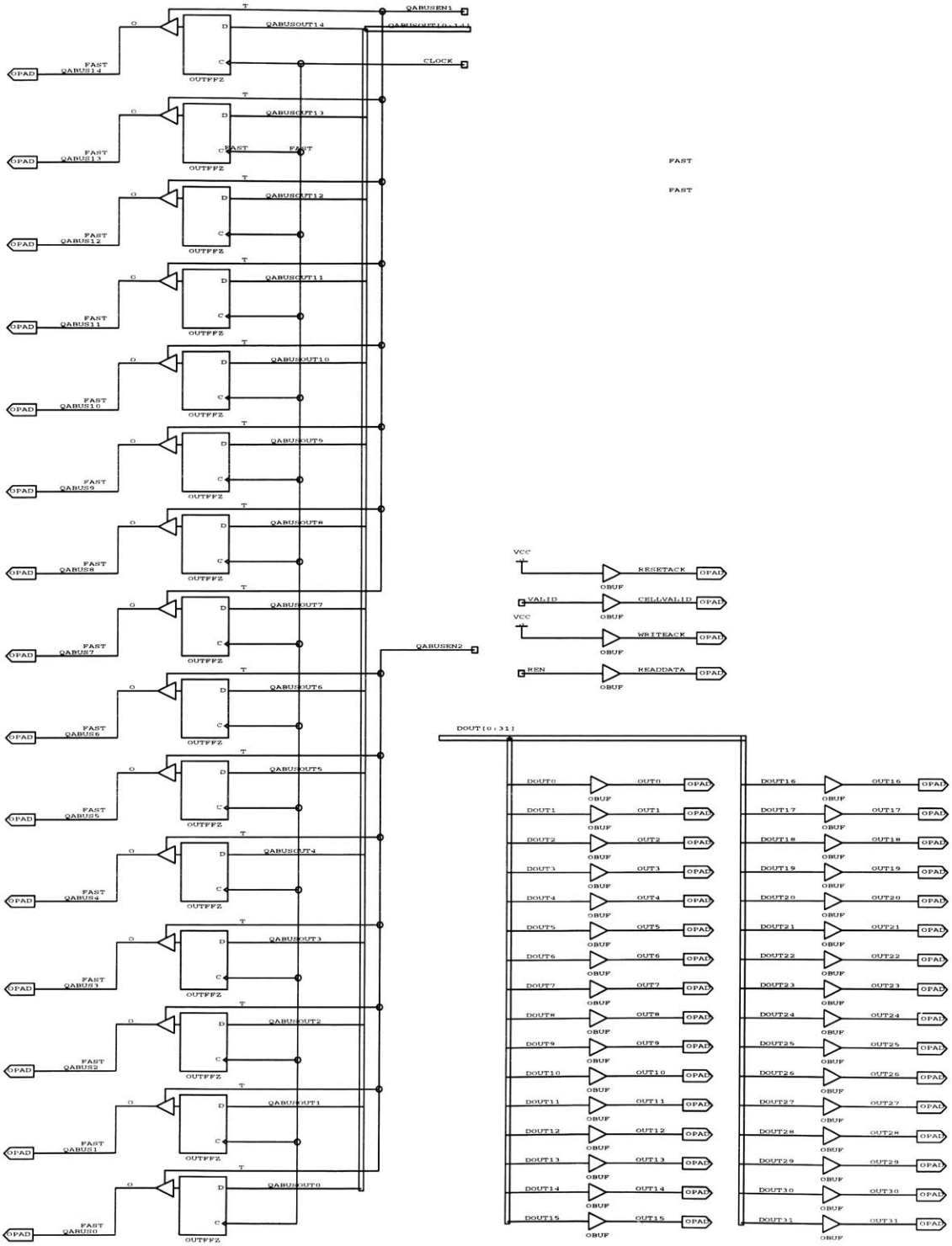
B.4 AN2-Bound/VuNet Side Xilinx - FSM



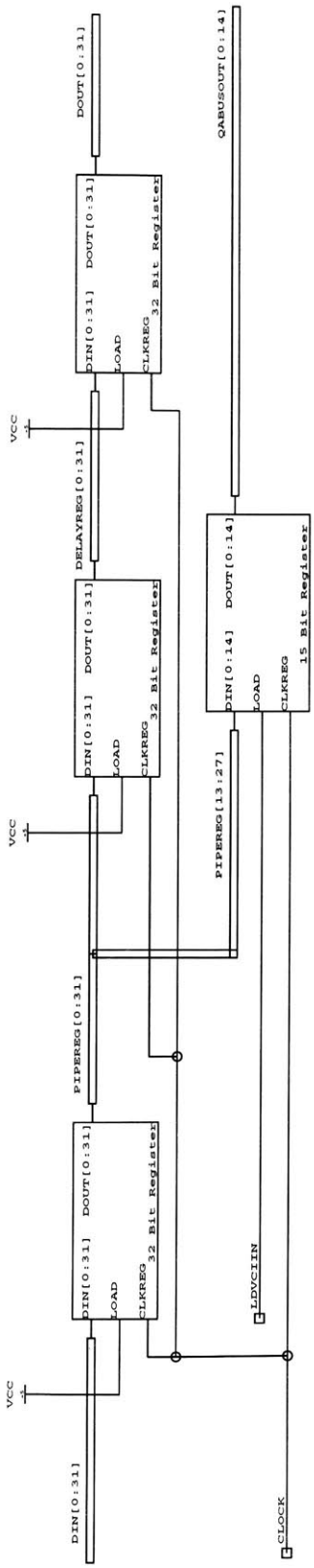
B.5 AN2-Bound/AN2 Side Xilinx - Input Pins



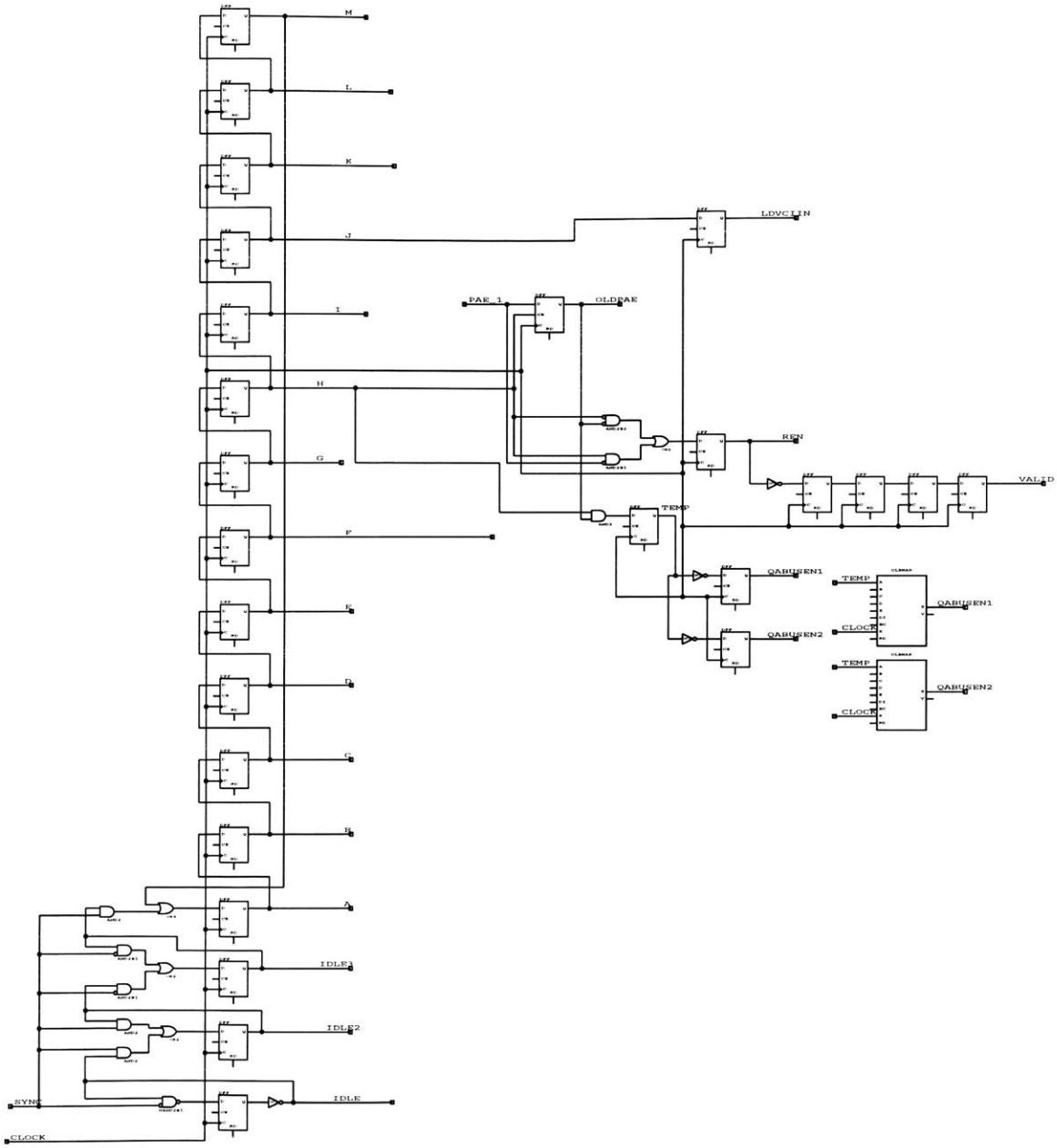
B.6 AN2-Bound/AN2 Side Xilinx - Output Pins



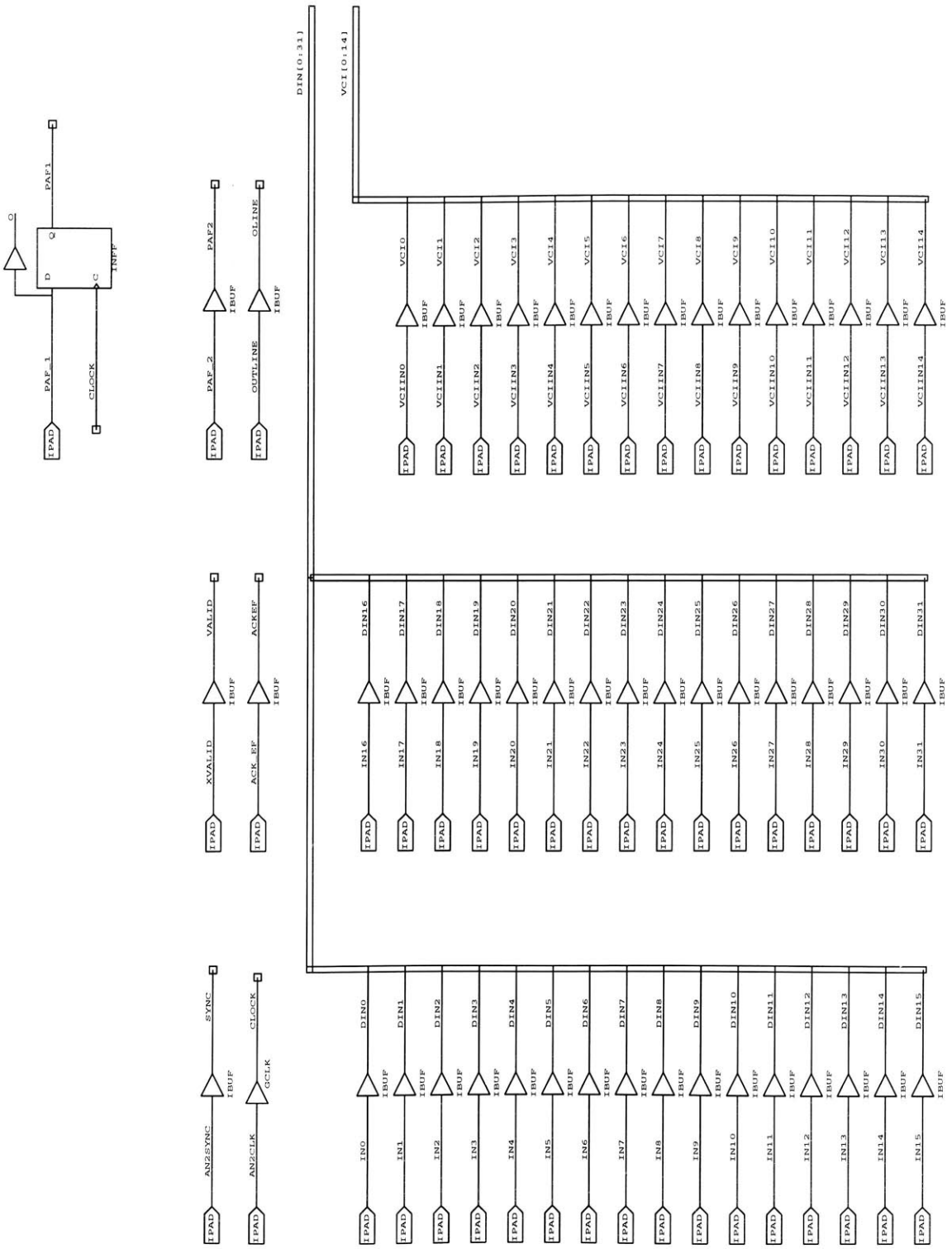
B.7 AN2-Bound/AN2 Side Xilinx - Data Path



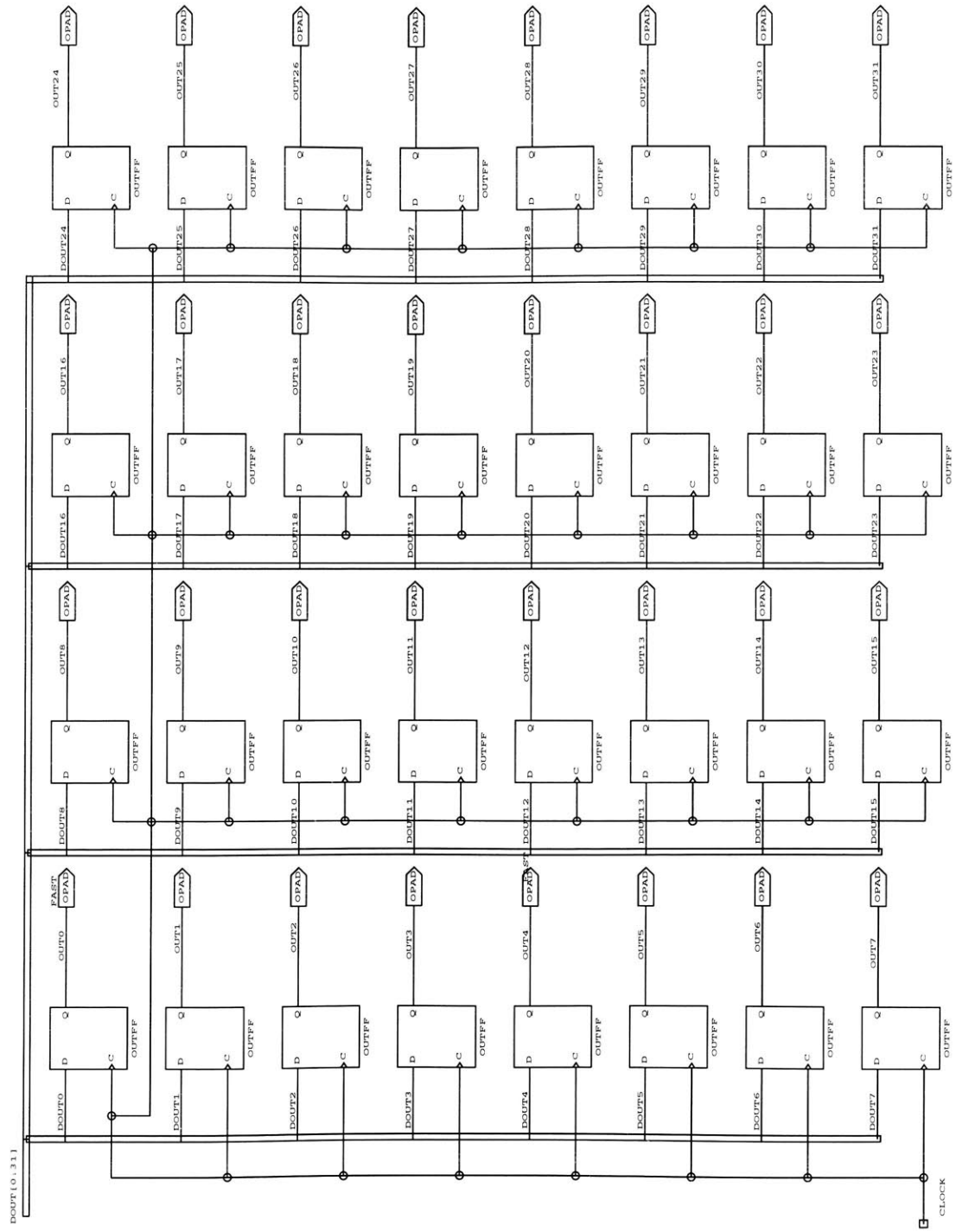
B.8 AN2-Bound/AN2 Side Xilinx - FSM



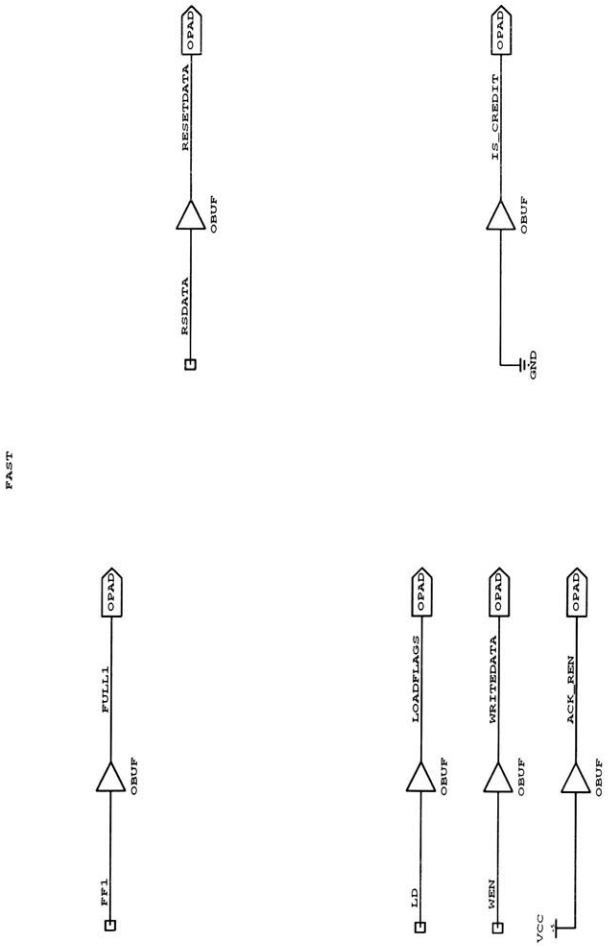
B.9 VuNet-Bound/AN2 Side Xilinx - Input Pins



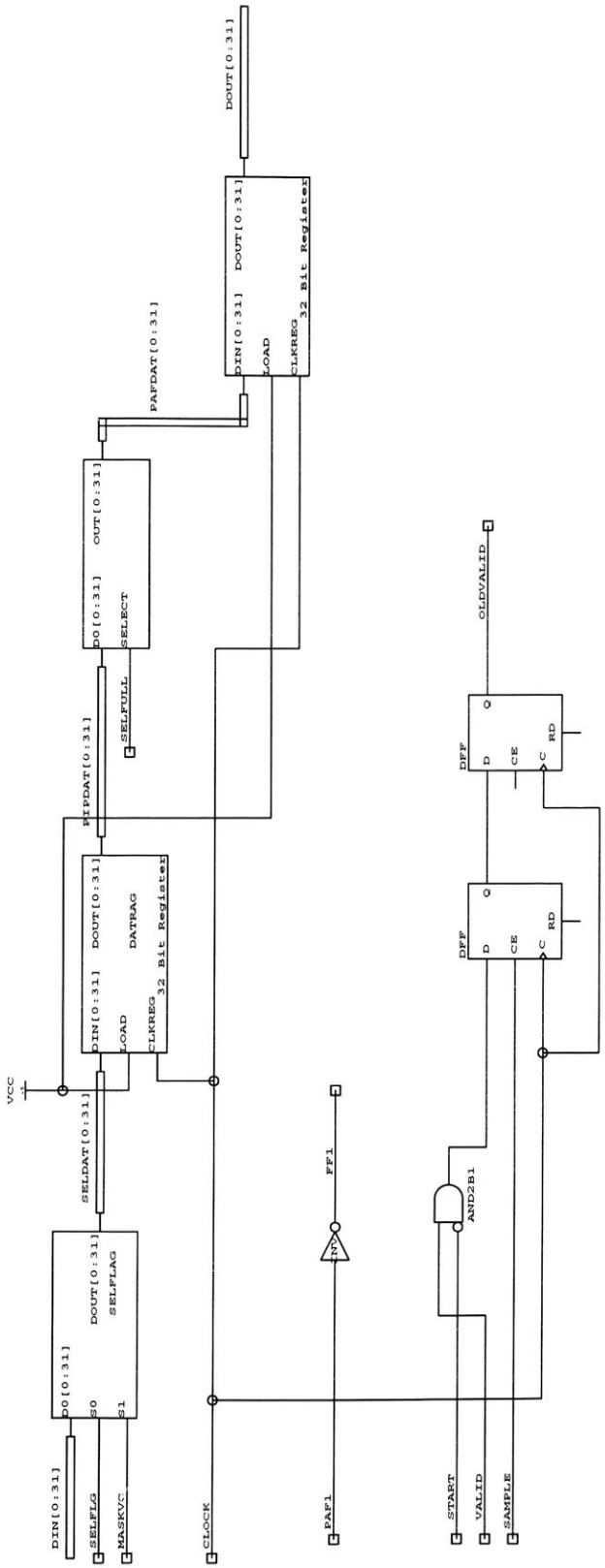
B.10 VuNet-Bound/AN2 Side Xilinx - Output Data Pins



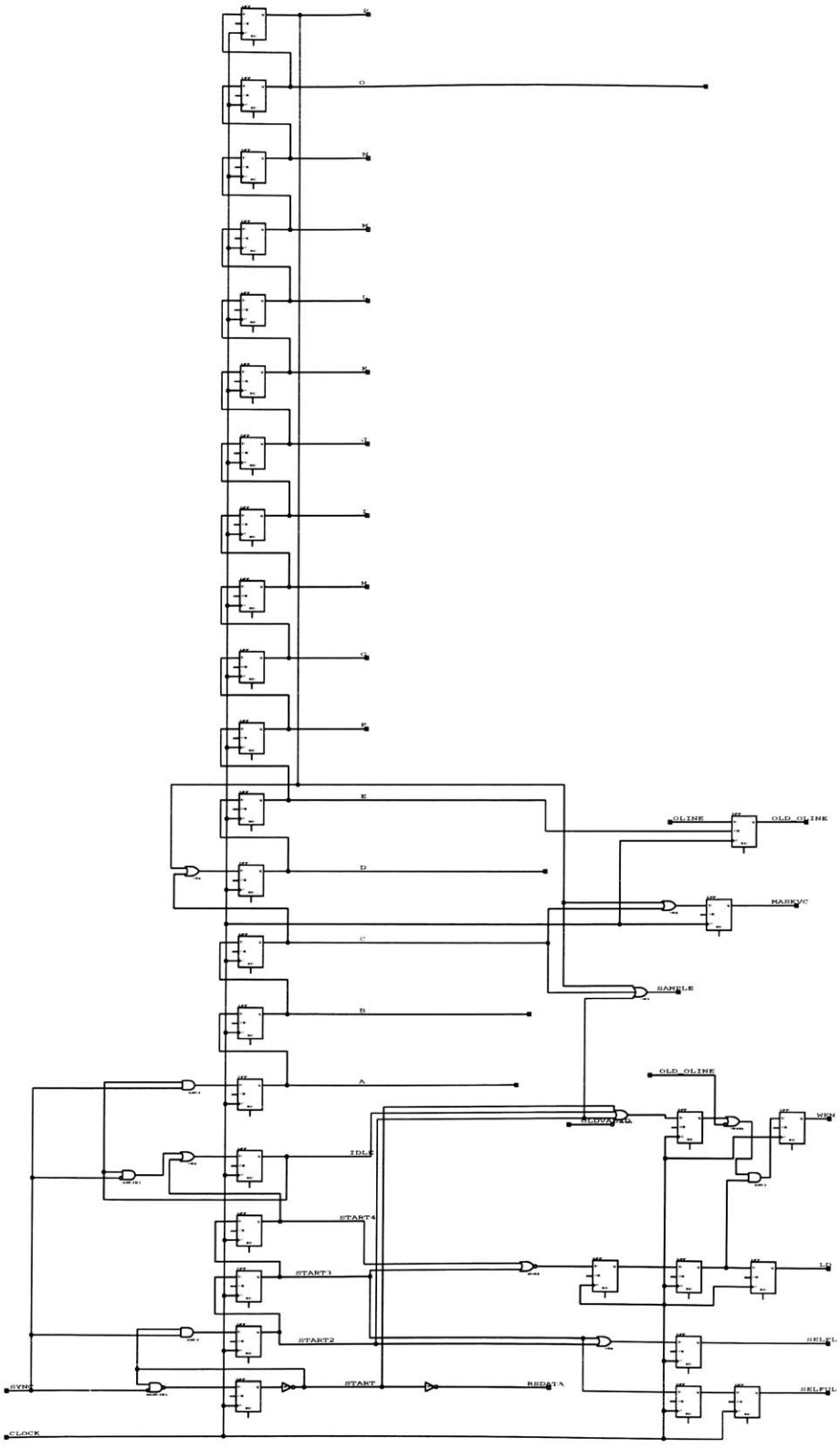
B.11 VuNet-Bound/AN2 Side Xilinx - Output Control Pins



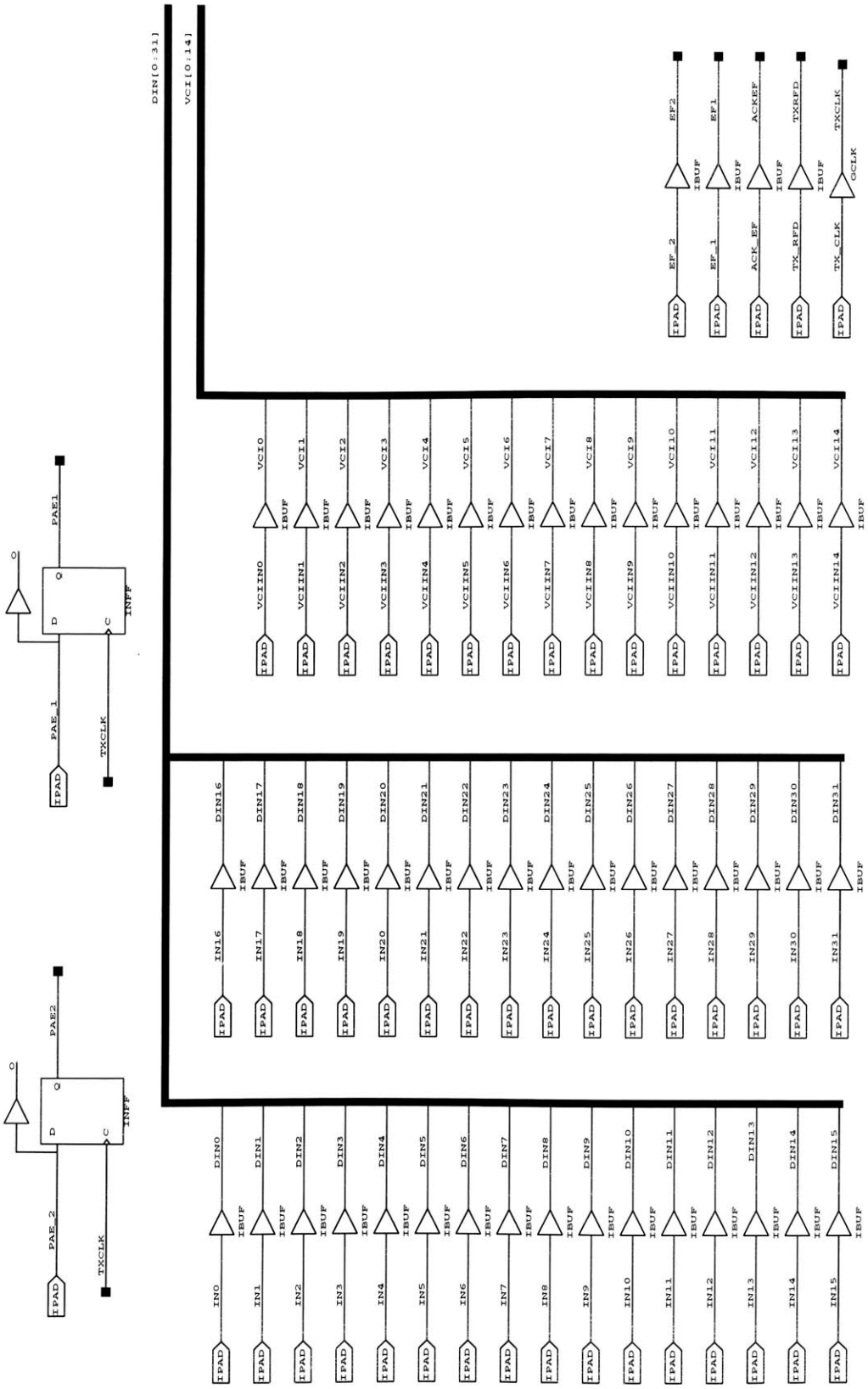
B.12 VuNet-Bound/AN2 Side Xilinx - Data Path



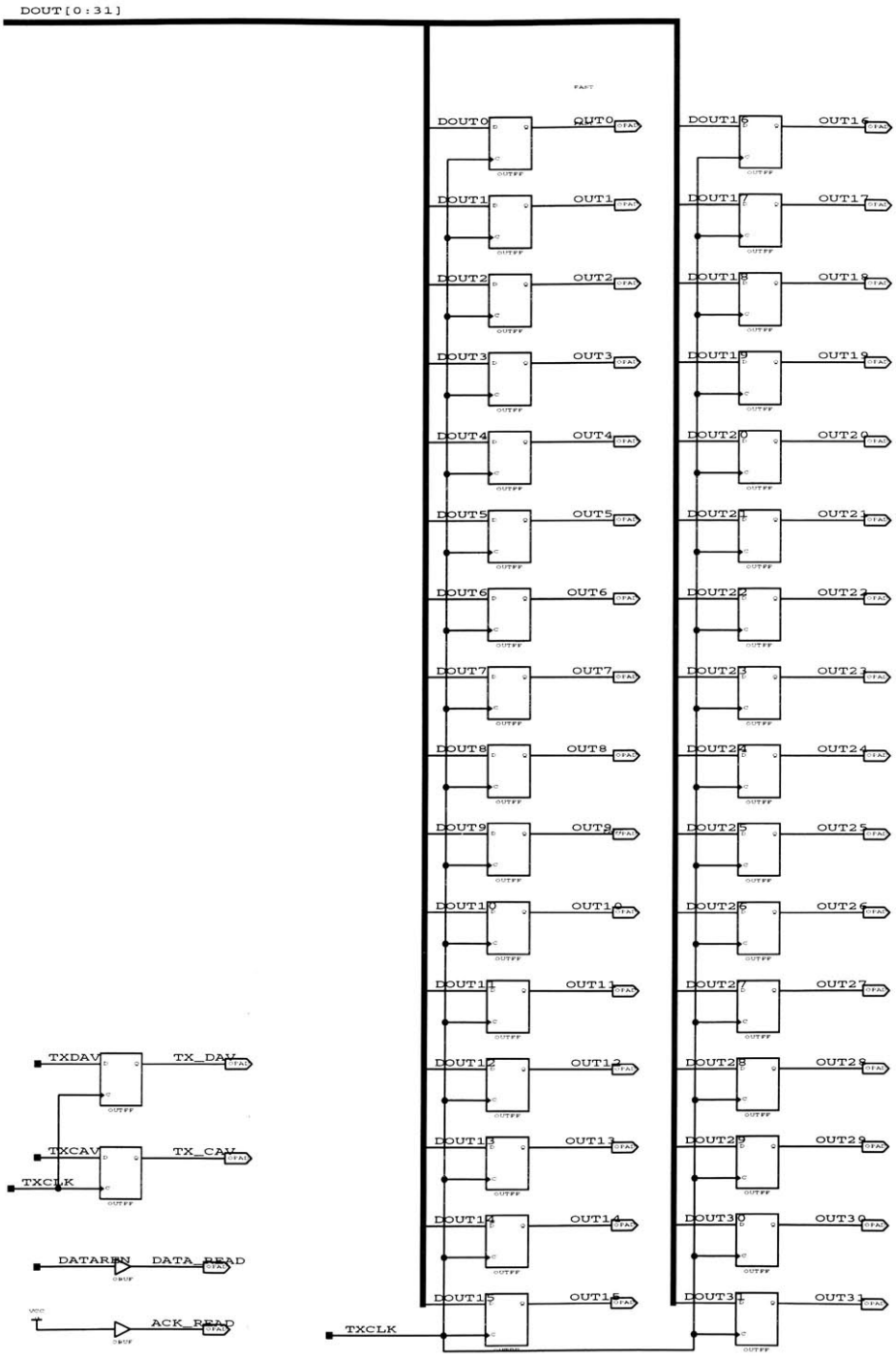
B.13 VuNet-Bound/AN2 Side Xilinx - FSM



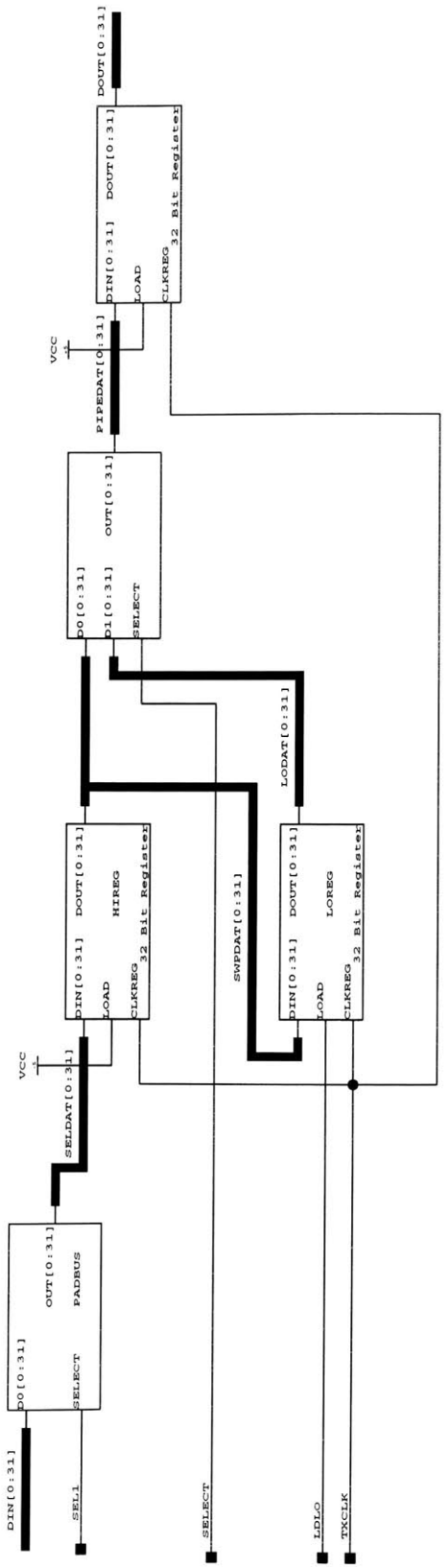
B.14 VuNet-Bound/VuNet Side Xilinx - Input Pins



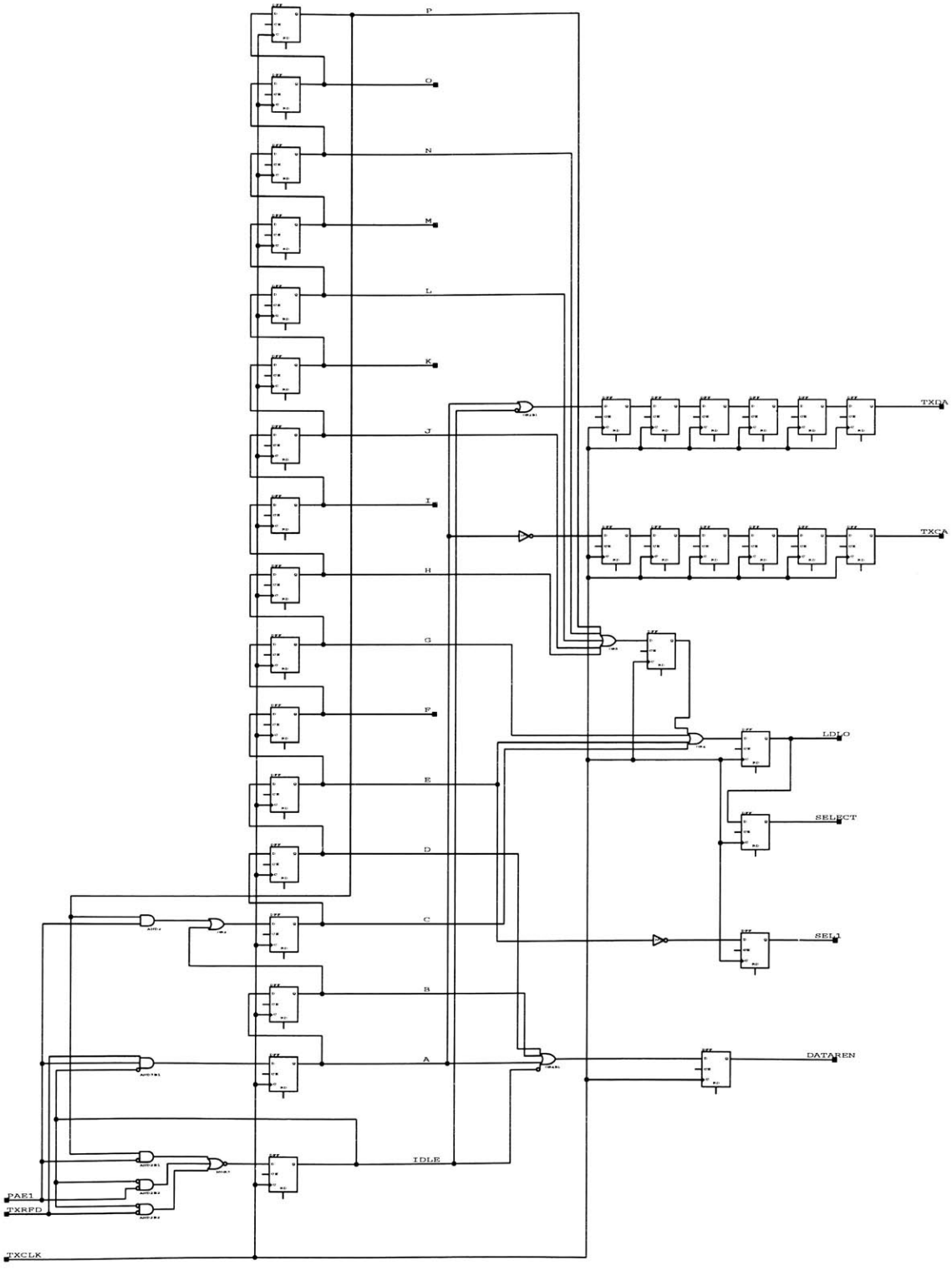
B.15 VuNet-Bound/VuNet Side Xilinx - Output Pins



B.16 VuNet-Bound/VuNet Side Xilinx - Data Path



B.17 VuNet-Bound/VuNet Side Xilinx - FSM

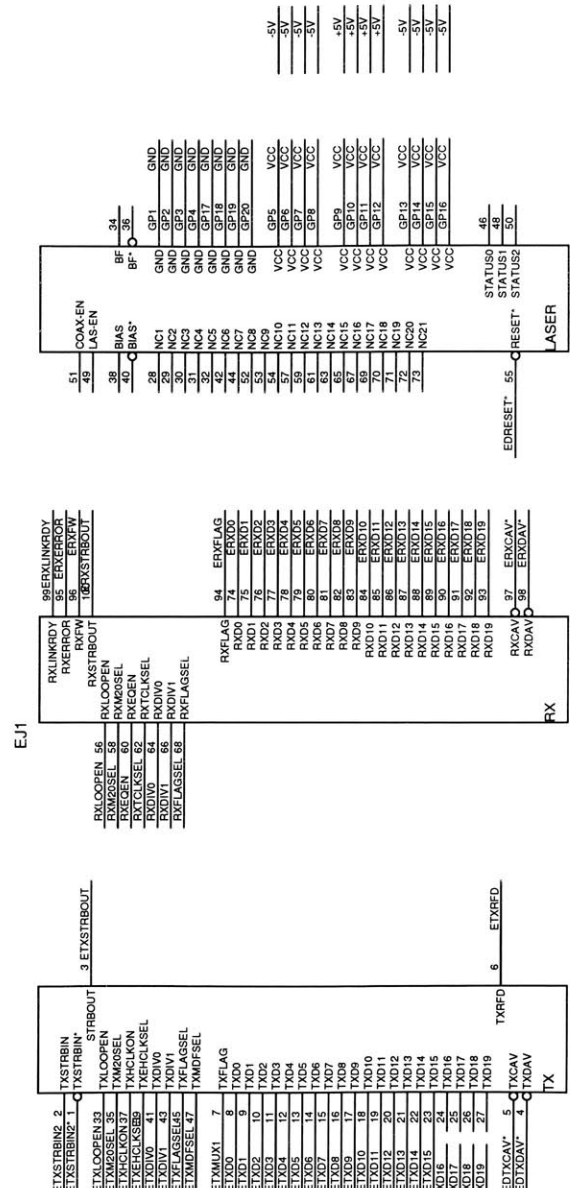


Appendix C

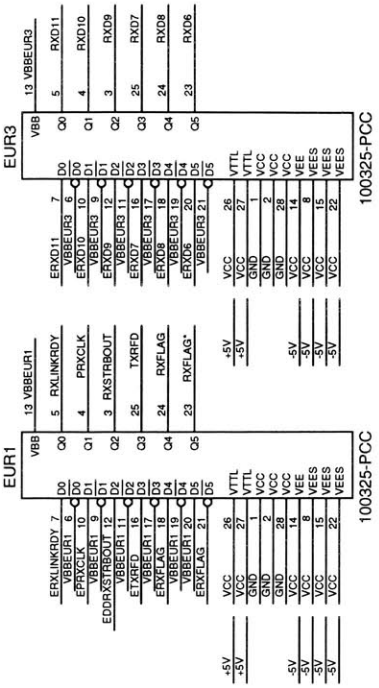
Zebra Schematics

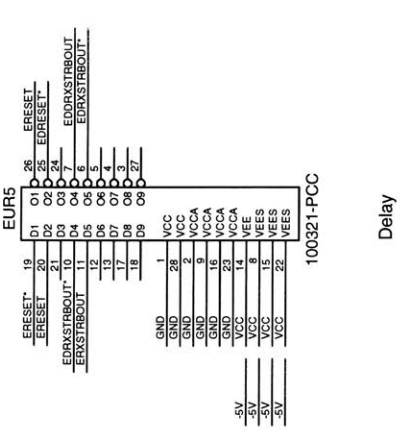
The following pages show the board level schematics for the Zebra. These schematics were generated using the Tango suite of CAD tools.

C.1 ECL - G-Link Interface

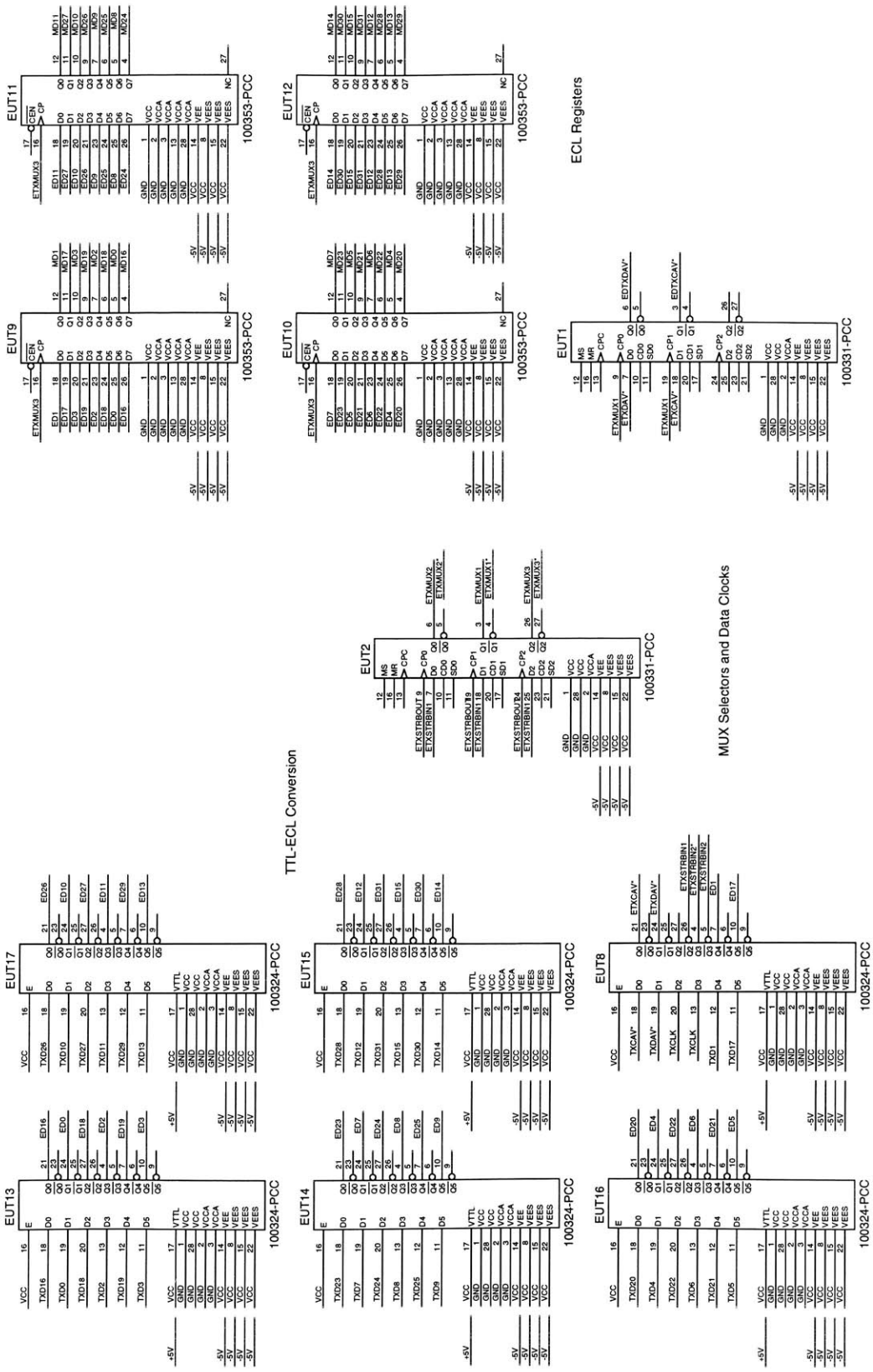


C.2 ECL - AN2-Bound Direction

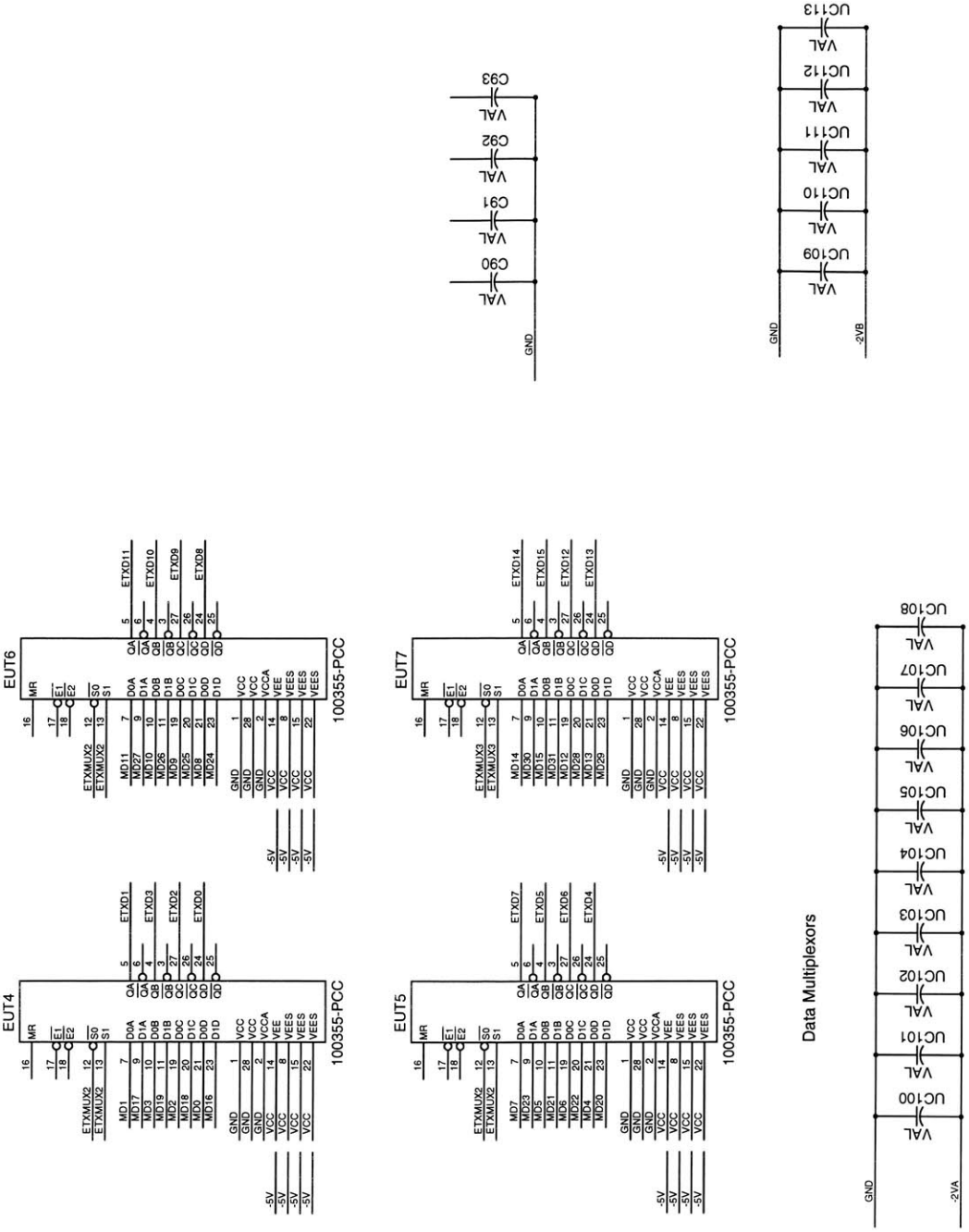




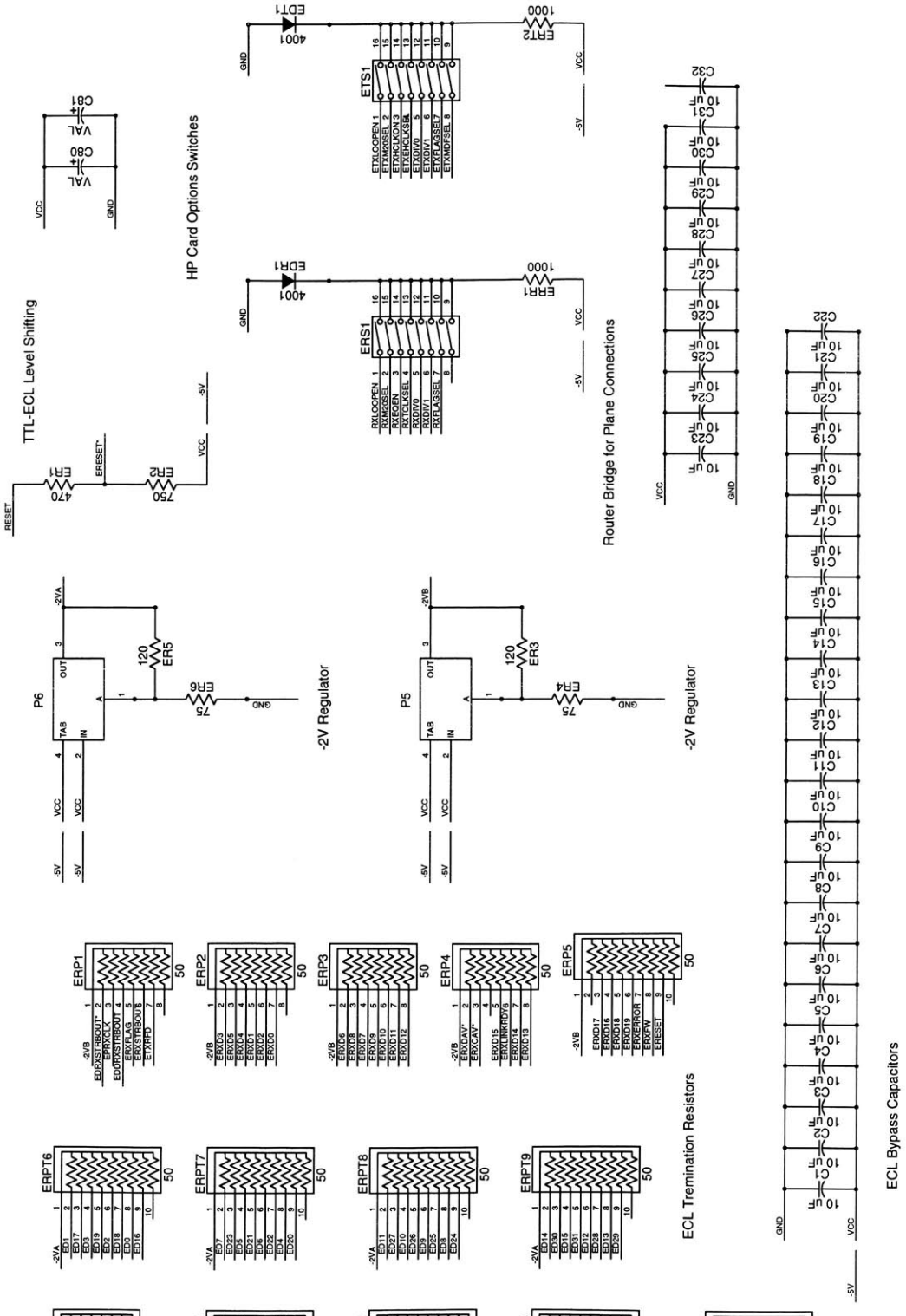
C.3 ECL - VuNet-Bound Registers



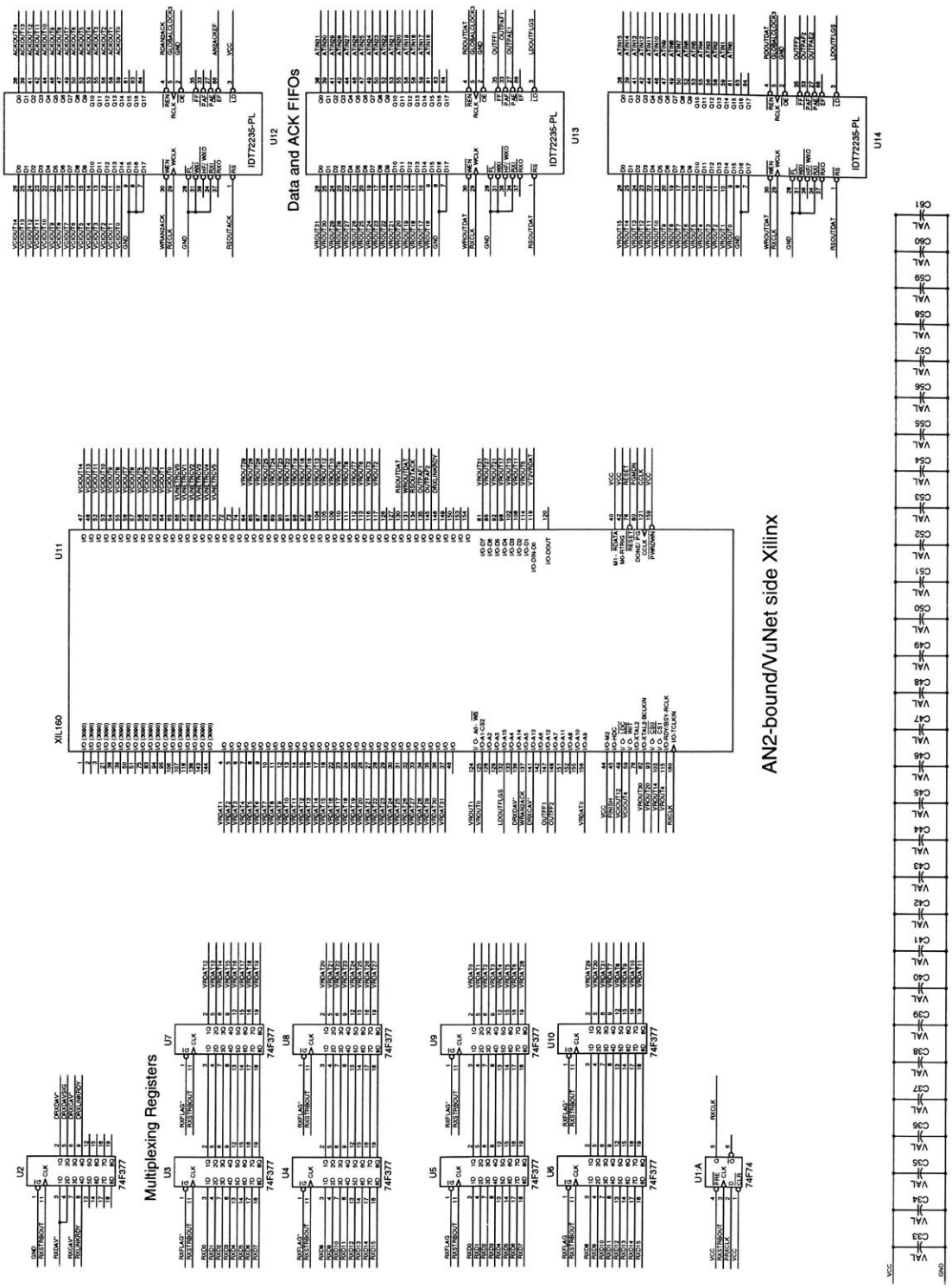
C.4 ECL - VuNet-Bound Multiplexors



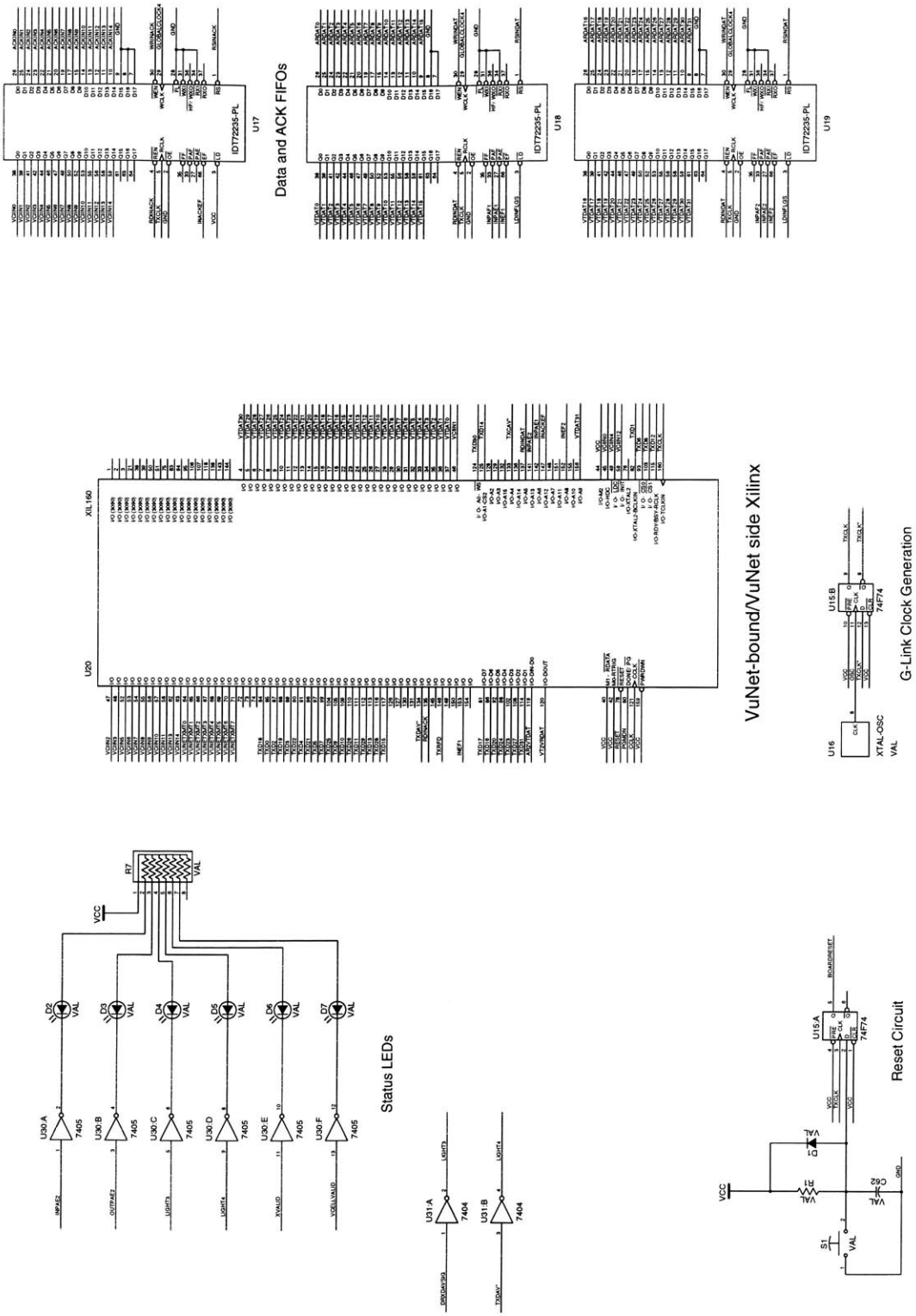
C.5 ECL - Miscellaneous Components



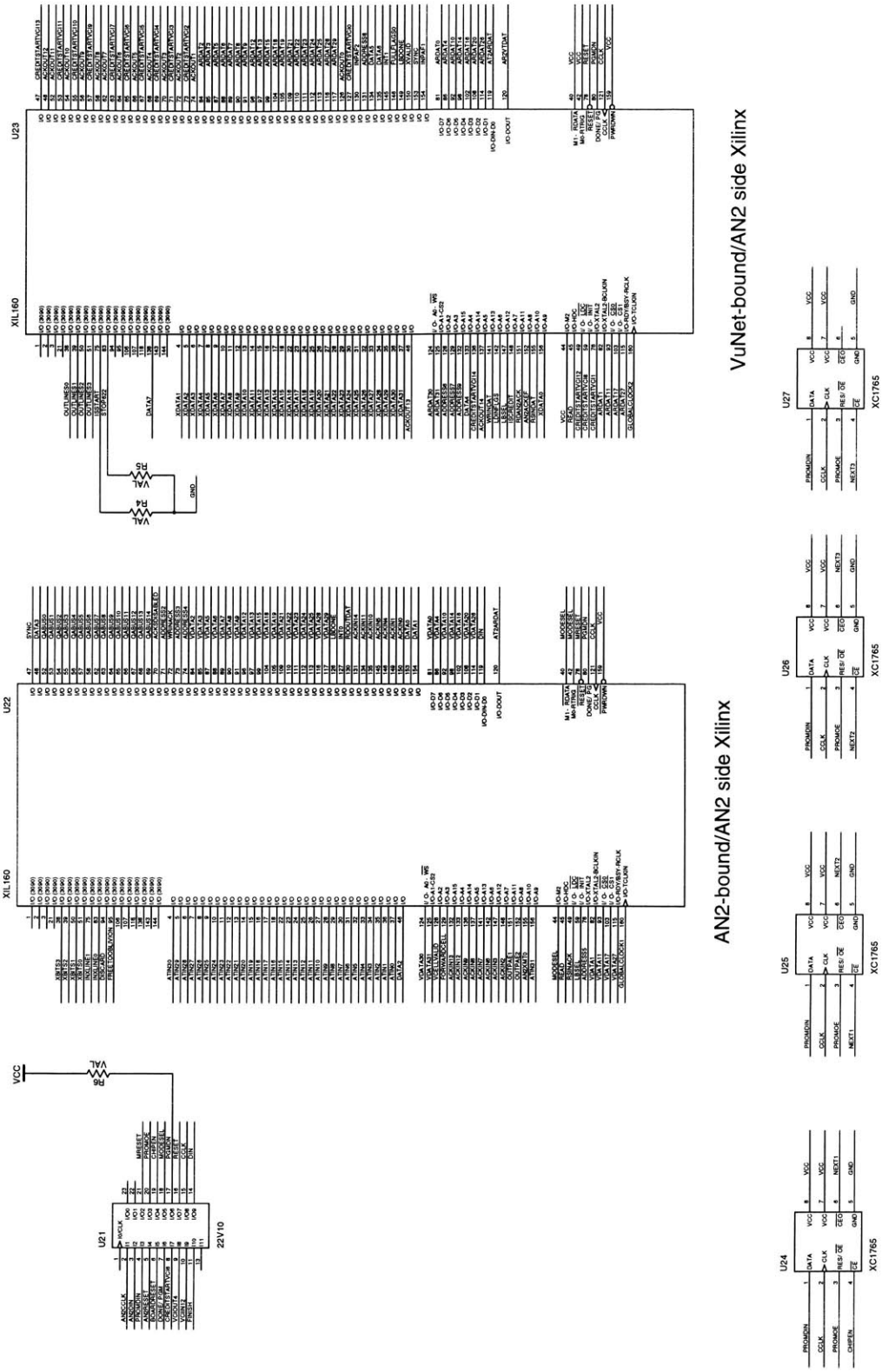
C.6 TTL - AN2-Bound/VuNet Side



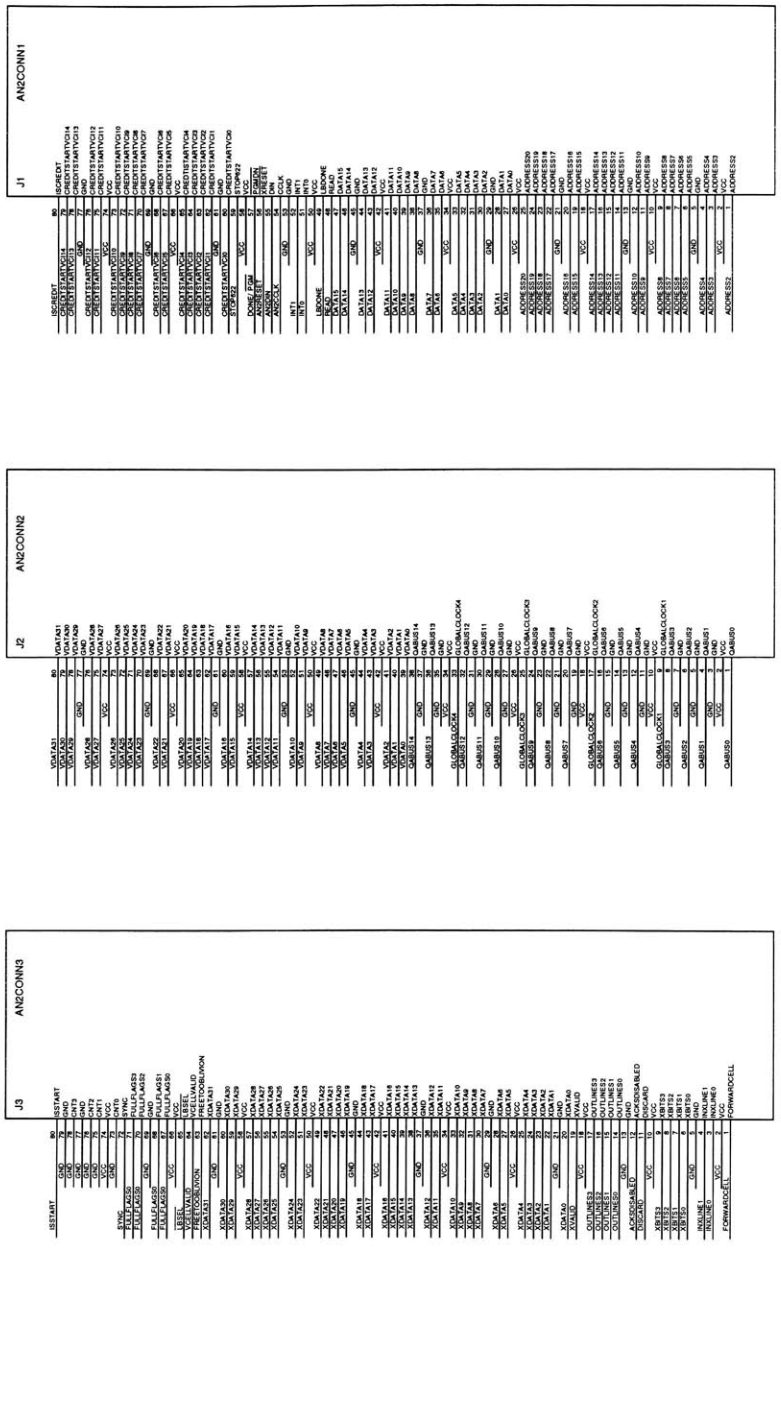
C.7 TTL - VuNet-Bound/VuNet Side



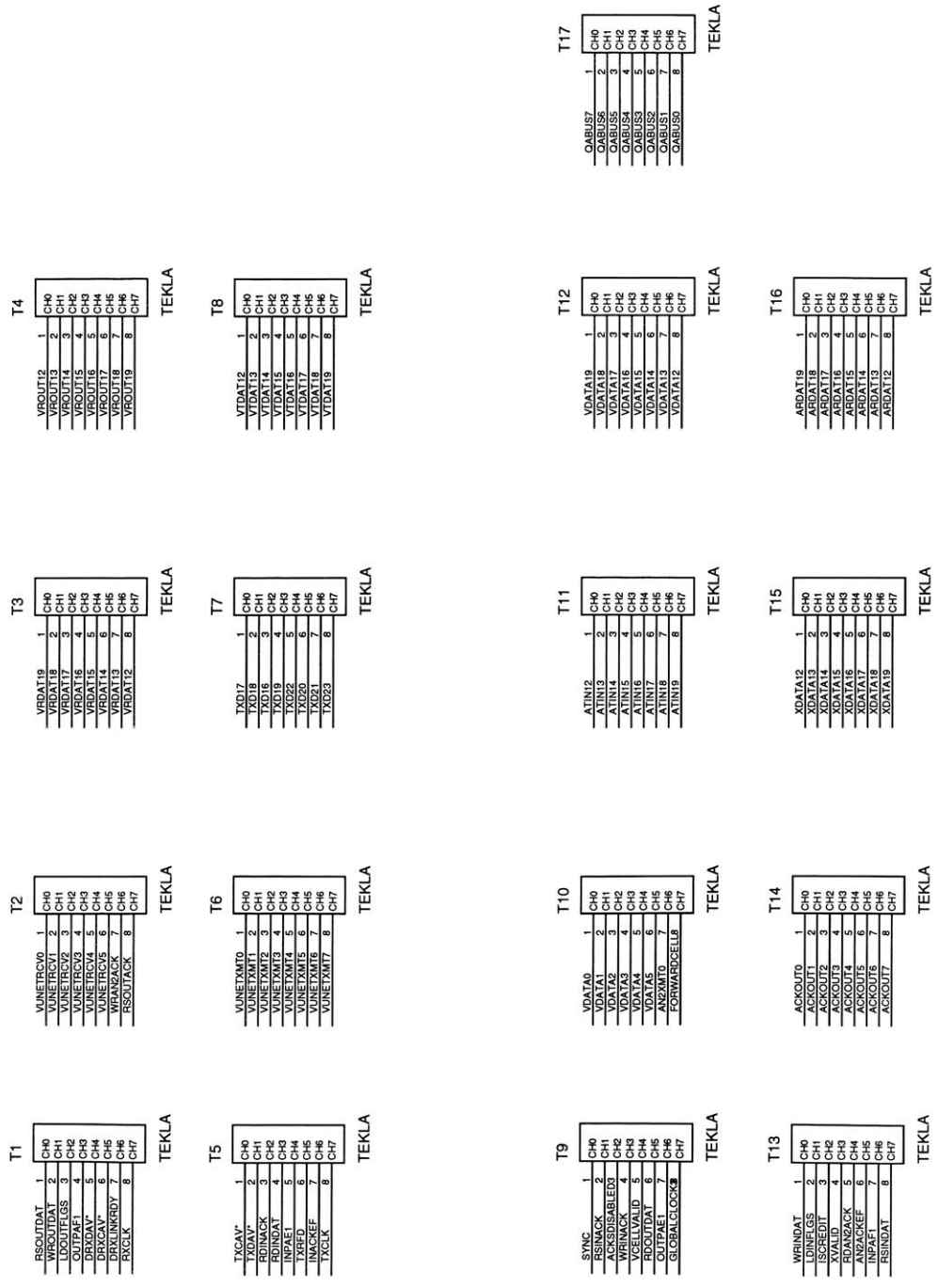
C.8 TTL - AN2-Bound and VuNet-Bound/AN2 Side



C.9 TTL - AN2 Daughtercard Interface



C.10 TTL - Probe Points



Bibliography

[1] A.G. Fraser and P.S. Henry. Transmission Facilities for Computer Communications. *Computer Communication Review*, 22(5), 1992.

[2] Hans Eriksson. MBone – the multicast backbone. In *Proceedings of the International Networking Conference (INET)*, San Francisco, California, August 1993. Internet Society.

[3] H. H. Houh, C. J. Lindblad, J. Soo, and D. Wetherall. The Media Gateway: Live Video on the World Wide Web, May 1994. Workshop at the 1994 World Wide Web Conference, Geneva, Switzerland.

[4] Martin de Prycker. *Asynchronous Transfer Mode Solution for Broadband ISDN.* Ellis Horwood, West Sussex, England, 1991.

[5] The AURORA testbed annual report. April 1991 - March 1992.

[6] J.N. Giacopelli, J.J. Hickey, W.S. Marcus, and W.D. Sincoskie. Sunshine: a high performance self-routing broadband packet switch architecture. *IEEE Journal on Selected Areas in Communications*, 9(8), October 1991.

[7] M. Shreedhar and G. Varghese. Efficient Fair Queueing using Deficit Round Robin, 1995. Submitted to SIGMETRICS.

[8] R. Ballart and Y. Ching. SONET: Now it's the standard optical network. *IEEE Communications Magazine*, March 1989.

[9] Bertsekas and Gallager. *Data Networks.* Prentice Hall Inc., Englewood Cliffs, NJ, 1993.

[10] Lasertron, Inc., Burlington, MA. *Calliope BFE20.R Technical Bulletin and Application Guide.*

[11] David L. Tennenhouse and Ian M. Leslie. A Testbed for Wide Area ATM Research. In *SIGCOMM '89 Symposium*, September 1989.

[12] Bruce S. Davie. The Architecture and Implementation of a High-Speed Host Interface. *IEEE Journal on Selected Areas in Communications*, 11(2), February 1993.

[13] Gigabit network testbeds. *IEEE Computer Magazine*, March 1990.

[14] J. Burren, C. Adams, H. Pitura, D. Tennenhouse, and I. Leslie. Variable Data Rate Channel for Digital Network, 1986. UK Patent Application No. 8618424, London. (Also subject of further UK, US, pand European patents.).

[15] Bruce S. Davie, Peter Druschel, and Larry L. Peterson. Experiences with a High-Speen Network Adaptor: A Software Perspective. In *Proceedings of SIGCOMM '94*, September 1994.

[16] Bandwidth ON Demand INteroperability Group. Interoperability Requirements for Nx56/64 kbit/s Calls, September 1992.

[17] K.C. Young and C.A. Johnston et al. A SONET/ATM Terminal Adapter for Connecting High-Speed LANs. In *Proceedings of the Gigabit Testbed Workshop*, June 1993.

[18] Randy H. Katz, Garth A. Gibson, and David A. Patterson. Disk System Architectures for High Performance Computing. In *Proceedings of the IEEE*, December 1989.

[19] Lev Vaitzblit. The Design and Implementation of a High-Bandwidth File Service for Continuous Media, September 1991. Masters Thesis.

[20] Joel F. Adam, Henry H. Houh, Michael Ismert, Christopher J. Lindblad, and David L. Tennenhouse. The VuNet Desk Area Network: Architecture, Implementation, and Experience. *IEEE Journal on Selected Areas in Communications*, 1st Quarter 1993.

[21] Michael D. Schroeder, Andrew D. Birrell, Michael Burrows, Hal Murray, Roger M. Needham, Thomas L. Rodeheffer, Edwin H. Satterwaite, and Charles P. Thacker. Autonet: A high-speed, self-configuring local area network using point-to-point links. *IEEE Journal on Selected Areas in Communications*, 9(8), October 1991.

[22] Thomas E. Anderson, Susan S. Owicki, James B. Saxe, and Charles P. Thacker. High Speed Switch Scheduling for Local Area Networks, April 1993. Digital Equipment Corporation SRC Research Report 99.

[23] G.J. Minden, J.B. Evans, V.S. Frost, and D.W. Petr. Implementation of an OC-12c and 4xOC-3c ATM/SONET Gateway based on Xilinx FPGAs. In *Proceedings of the Gigabit Testbed Minijam*, January 1994.

[24] Bruce S. Davie. Experiences with an ATM Host Interface. In *Proceedings of the Gigabit Testbed Workshop*, June 1993.

[25] Michael Ismert. The AVlink: An ATM Bridge between the VuNet and Sunshine, May 1993. Bachelors Thesis.

[26] ITU-TS (CCITT) Rec. I.361. B-ISDN ATM Layer Specification, March 1993.

[27] ITU-TS (CCITT) Rec. I.610. B-ISDN Operation and Maintenance Principles and Functions, March 1993.

[28] ITU-TS (CCITT) Rec. I.150. B-ISDN Asynchronous Transfer Mode Functional Characteristics, March 1993.

[29] ITU-TS (CCITT) Rec. I.363. B-ISDN ATM Adaptation Layer (AAL) Specification, March 1993.

[30] P. Almquist and F. Kastenholz. Towards Requirements for IP Routers, Request for Comments 1716, November 1994.

[31] William Stallings. *Handbook of Computer Communications Standards: The Open Systems Interconnection Model and OSI-Related Standards.* Howard W. Sams and Co., Indianapolis, Indiana, 1987.

[32] David D. Clark and David L. Tennenhouse. Architectural Considerations for a New Generation of Protocols. In *Proceedings of SIGCOMM '90*, September 1990.