A Workstation Architecture to Support Multimedia

Mark David Hayter

St John's College
University of Cambridge

A dissertation submitted for the degree of
Doctor of Philosophy

September 1993

Abstract

The advent of high speed networks in the wide and local area enables multimedia traffic to be easily carried between workstation class machines. This dissertation considers an architecture for a workstation to support such traffic effectively. In addition to presenting the information to a human user the architecture allows processing to be done on continuous media streams.

The proposed workstation architecture, known as the Desk Area Network (DAN), extends ideas from Asynchronous Transfer Mode (ATM) networks into the end system. All processors and devices are connected to an ATM interconnect. This is shown to be capable of supporting both multimedia data streams and more traditional CPU cache line traffic. The advocated extension of the CPU cache which allows caching of multimedia data streams is shown to provide a natural programming abstraction and a mechanism for synchronising the processor with the stream.

A prototype DAN workstation has been built. Experiments have been done to demonstrate the features of the architecture. In particular the use of the DAN as a processor-to-memory interconnect is closely studied to show the practicality of using ATM for cache line traffic in a real machine. Simple demonstrations of the stream cache ideas are used to show its utility in future applications.

Preface

Except where otherwise stated in the text, this dissertation is the result of my own work and is not the outcome of work done in collaboration.

This dissertation is not substantially the same as any I have submitted for a degree or diploma or any other qualification at any other university.

No part of this dissertation has already been, or is being currently submitted for any such degree, diploma or other qualification.

Trademarks

Acknowledgements

I would like to thank my supervisor, Derek McAuley, for his encouragement and advice during my time at the Computer Laboratory. I would also like to thank Roger Needham, Head of the Laboratory, for his support and for encouraging me to spend one summer getting a new perspective on research by working as an intern at DEC Systems Research Center in Palo Alto.

I am grateful for the help and friendship of the members of the Systems Research Group, who have always proved ready to engage in useful discussions. Joe Dixon, Richard Black and Timothy Roscoe deserve a special mention. Thanks are also due to Ian Leslie for providing insights into computer networking and David Greaves for many discussions about networking hardware.

I am indebted to Richard Black, Simon Crosby, Joe Dixon, David Evers, Derek McAuley and Cosmos Nicolaou who read and commented on earlier drafts of this dissertation.

This work was supported by a studentship from the Science and Engineering Research Council.

Contents

List of Figures

List of Tables

Glossary

AAL	ATM adaptation layer
AC	Address Consultant (in Atomic network)
AN2	Autonet 2 (an ATM network).
ANSA	Advanced Networked Systems Architecture
ARM	Advanced RISC Machine
ATDM	Asynchronous Time Division Multiplexing
ATM	Asynchronous Transfer Mode
ATUM	Address Tracing Using Microcode
AXP	DEC Alpha project logo
B-ISDN	Broadband Integrated Services Digital Network
CBN	Cambridge Backbone Network
CCD	Charge Coupled Detector
CCITT	Comitée Consultatif International Télégraphique et Téléphonique (now the ITU-TS)
CDCS	Cambridge Distributed Computer System
CFR	Cambridge Fast Ring
CLB	Configurable Logic Block (on a Xilinx FPGA)
CLP	Cell Loss Priority (a bit in a B-ISDN cell header)
CODEC	Coder/Decoder - especially transforming audio to/from digital form.
CRC	Cyclical Redundancy Check
DAN	Desk Area Network
DAQ	Data Acquisition Device
DEC	Digital Equipment Corporation
DMA	Direct Memory Access
DSP	Digital Signal Processor
ECL	Emitter Coupled Logic

EISA	Enhanced Industry Standard Architecture (a bus)
FDDI	Fiber Distributed Data Interface
FIFO	First In First Out
FIQ	Fast Interrupt Request (on ARM processors)
FPGA	Field Programmable Gate Array
GFC	Generic Flow Control (a 4 bit field in a B-ISDN cell header)
GNU	GNU's Not Unix!
HEC	Header Error Check (an 8 bit CRC in a B-ISDN cell)
IOB	Input Output Block (on a Xilinx FPGA)
IP	Internet Protocol
IRQ	Interrupt Request
ISDN	Integrated Services Digital Network
ISI	Information Sciences Institute
ISO	International Standards Organisation
ITU	International Telecommunication Union
JPEG	Joint Photographic Experts Group (a video compression standard)
LAN	Local Area Network
LANC	Local Area Network on a Chip
LCA	Logic Cell Array
LRU	Least Recently Used
MAN	Metropolitan Area Network
MMU	Memory Management Unit
MSNL	MultiService Network Layer
NACK	Negative Acknowledgement
NPC	Null Port Controller
ORL	Olivetti Research Limited (Cambridge)
OS	Operating System
OSI	Open Systems Interconnection
PAL	Programmable Array Logic
PCB	Printed Circuit Board
PCI	Personal Computer Interconnect
PLPL	Programmable Logic Programming Language
PLT	Payload Type (a 3 bit field in a B-ISDN cell header)

QOS	Quality of Service
RDRAM	Rambus Dynamic Random Access Memory
RGB	Red, Green and Blue
RISC	Reduced Instruction Set Computer
RPC	Remote Procedure Call
SAR	Segmentation and Reassembly
SAS	Single Address Space
SBUS	Sun Bus
SCI	Scalable Coherent Interconnect
SCSI	Small Computer Systems Interconnect
STM	Synchronous Transfer Mode
TAXI	Transparent Asynchronous Transmitter/Receiver Interface
TBUF	Tristate Buffer (on a Xilinx FPGA)
TDM	Time Division Multiplexing
TLB	Translation Lookaside Buffer
UDL	Unison DataLink
VC	Virtual Channel or Virtual Circuit
VCI	Virtual Circuit Identifier (also Virtual Channel Identifier)
VM	Virtual Memory
VME	Versa Module Europe
VP	Virtual Path
VPI	Virtual Path Identifier
VRAM	Video RAM
WAN	Wide Area Network
WDFS	Wanda Dumb File System
XUNET	Experimental University Network

Chapter 1

Introduction

This dissertation presents a novel architecture for a workstation to support the full integration of multimedia into a distributed environment. The use in such a machine of an interconnect based on the Asynchronous Transfer Mode (ATM) is advocated. It is argued that this will remove bottlenecks that are becoming apparent in existing systems and allow quality of service guarantees to be extended all the way to the end-points of a communication. An ATM interconnect is shown to be suitable for both multimedia and more traditional memory traffic. To enable processing of multimedia data it is proposed that the CPU cache act directly as a destination or source for data streams. A prototype implementation has allowed experiments to be carried out to validate the architecture.

Media such as audio and video have a real-time nature and are therefore often described as continuous or isochronous. This dissertation follows the common usage of the term "multimedia" to refer to the use of one or more of these media in conjunction with traditional computer data such as text and graphics. It is now becoming common for workstations to have "multimedia peripherals" attached, i.e. devices which source and sink continuous media. When these devices are used in a distributed system, a large amount of jitter-sensitive data flows between the machine and the network. These flows stress both the network interface, which is already a bottleneck in many workstations, and the internal interconnect of the machine. Current multimedia applications only present the data for a human user, but it is expected that future applications will also want to process the data. This will require support from both the hardware and system software.

Recent networking research has concentrated on supporting multi-service traffic. In particular, these networks aim to carry both computer data and real-time media such as voice and video. This mix of traffic types can be supported by ATM networks, which can offer clients a range of "Quality of Service" (QOS) parameters. For example, audio would request a QOS with tightly constrained jitter and a fairly constant bandwidth, but a file transfer may be tagged for bursty best-effort service. Unfortunately, while the network offers this guaranteed service the interconnect and operating system in existing workstations do not.

Asynchronous Transfer Mode is a compromise between the fully synchronous communications traditionally used by the telecommunication industry and the packet based transfers favoured in computer systems. ATM, or Asynchronous Time Division Multiplexing as it was previously known, has been used for many years. However, it has only become popular in recent years with its adoption by the CCITT (now the ITU-TS) as the mechanism to use in the Broadband Integrated Services Digital Network (B-ISDN). This is a high bandwidth digital network intended to become as ubiquitous as the telephone network is today. ATM has been suggested as a networking technology for both the wide and local areas.

The new architecture defined in this dissertation will support both types of multimedia application and integrate the workstation fully with the network. The name "Desk Area Network" (DAN) has been given to the architecture, reflecting the fact that it has taken many ideas from networking and is intended as a solution for the desktop workstation class of machines.

Implementation of a prototype DAN machine enabled refinement of the architecture. Following studies of a CPU node, the use of a cache as a high speed memory closely coupled to the CPU is extended to allow multimedia data to flow into the cache. Known as the "stream cache", this is shown to be a mechanism which provides an easy way of manipulating stream data, while giving high performance and simplifying the synchronisation of the processor with the incoming data.

1.1 The Research Environment

Previous research work has been done at Cambridge into both ATM networking and multimedia computing. There have been two main projects recently: the Pandora multimedia system (in collaboration with Olivetti Research Ltd, Cambridge), and the Fairisle network (in collaboration with Hewlett-Packard Research Labs, Bristol). This work has been followed up by both the Desk Area Network project, described in this dissertation, and the Pegasus project, outlined below.

The Pandora project investigated the management of time-critical data streams and first generation multimedia applications. "The Pandora Box" was produced to add audio and video capabilities to an existing workstation. An ATM network, the Cambridge Fast Ring, was used to interconnect the boxes. Communication is possible between the Pandora workstations and to an assortment of servers, including a storage service, a video mail system and a TV source. The Cambridge Backbone Network links three sites between which multimedia communication is regular.

Switch based local area ATM was investigated in the Fairisle project. While mainly concerned with the management of an ATM network, one of the project aims was to produce and deploy a network as a test-bed for the management mechanisms. This network supports multiservice traffic and forms the basis for future work.

Use of multimedia applications in the Pandora project demonstrated the problems outlined above. A need was seen for changes to both existing hardware and systems software. The Desk Area Network was proposed as a solution to the hardware architecture, and the more recent Pegasus project is investigating the software concerns. In its first year the Pegasus project has defined the software architecture and is in the process of an implementation.

1.2 Outline

The two projects central to this dissertation are Fairisle and DAN. These involved a number of people. The hardware development of the Fairisle Port Controller, based on pre-existing ideas and experimental requirements, is entirely my own work. My main task has been the development of the Desk Area Network architecture. This has been assisted by the implementation of hardware by others as noted in the text.

Background ideas and related work are introduced in the next chapter. The three main areas of networking, multimedia and cache memory are discussed and the terminology used in this dissertation is defined. The chapter concludes with a description of some of the hardware and software components used in building the experimental systems.

Chapter 3 describes the Fairisle network, and in particular the components of the network which are reused in the DAN demonstrator.

The Desk Area Network is described in the next two chapters. Both start with a discussion of the design concepts and reasoning and go on to describe in detail an implementation of these. Chapter 4 concentrates on the definition of the DAN architecture and its use for multimedia. Chapter 5 expands on the role of the CPU cache and the memory–I/O hierarchy for supporting multimedia processing.

The behaviour of the DAN demonstrator, particularly with regard to the CPU cache, is studied in a series of experiments reported on in Chapter 6. Many of the results are summarised graphically in this chapter. Appendix A contains tables of the numerical results. The experiments bring out some of the limitations of the hardware used.

Further work is suggested in Chapter 7. This falls principally into two areas: operating system support for the DAN, and the hardware requirements of a system with higher performance than the initial demonstrator.

Chapter 8 summarises the main points of the dissertation and makes some concluding remarks.

Chapter 2

Background

This chapter discusses the three areas that form the background to the work reported on in later chapters. These are networking, multimedia, and caching. Some related projects, which will be used for comparison, are introduced. Finally the hardware and software components used in the production of the demonstration system are presented, and limitations that affect the performance of the demonstrator are described.

2.1 Interconnection Networks

An interconnection network is used to connect processors, memory and devices within a single machine. In many machines a simple bus provides this function, but higher performance may be obtained from more general network organisations. Many interconnection networks have been designed for a specific system, and they have been highly tuned for the applications intended for that system. This has resulted in a wide range of often highly specific designs. However, there is now an interest in more general interconnection networks, and some standards are emerging. In this section, four interconnection networks will be considered, two of which will be used in Chapter 4 for comparison with the DAN.

2.1.1 Scalable Coherent Interconnect

The Scalable Coherent Interconnect (SCI) [**IEEE.P1596**] is based on high speed unidirectional links used to form a small ring, or ringlet, through the devices in the system. The connection is made either using 16 data lines, clocked at up to 250MHz with data transfer on both clock edges, or in a serial form, clocked at up to 800MHz. The official scope of the SCI project is for use over short distances within machines, but it has also been suggested for use in larger networks.

The SCI standard specifies both the physical behaviour and the protocols for use on the interconnect. There are a small number of defined transactions which all consist of a request subaction followed by a reply. To facilitate the use of wider paths clocked at a lower speed, all SCI packets are multiples of 8 bytes. The data component of the packet is either one of three fixed sizes or a variable size less than 16 bytes. The packet starts with a header containing destination address, source node and a command. 64 bit addresses are used, subdivided into a 16 bit node identifier and a 48 bit local address. Depending on the command there may be additional header information. The data portion of 0, 16 (used for the variable length packets), 64 or 256 bytes forms the body of the packet which is terminated with a CRC. A particular emphasis is placed on providing cache coherence protocols based on a distributed directory, though if the interconnect is only being used for I/O these are not required.

`Target Node id (16 bits)`
`Command`
`Source Node id (16 bits)`
`Sequence Number / Time to Live`
`Address / Status (48 bits)`
`Optional Extended header`
`0, 16, 64 or 256 bytes Data`
`CRC`

Figure 2.1: Simplified SCI Packet format

Ringlets are good for small SCI installations, but a single ring does not scale well. The initial solution suggested is a mesh of bridged ringlets, but space division switch based solutions are also being investigated [**RD24Status93**].

Dolphin SCI Technology have produced a GaAs NodeChip which provides a node connection to a 1GByte/s SCI ringlet using 16 ECL signal pairs clocked at 500MHz. A CMOS version of this device being produced by LSI logic is functionally equivalent, but uses less power and is slower, driving a 100 – 150MByte/s ringlet. CERN demonstrated the GaAs chip running at half speed in April 1993 [**Gustavson93**], and Dolphin have demonstrated a four node multiprocessor system [**Alnes93**]. A bus interface, the C-Bus [**Dol92**], is provided by the interface chips. This is a general purpose interface and is being used to provide SCI links to machines based on the VME bus, FutureBus, TURBOchannel and Apple Macintosh Quadra Internal bus.

While the SCI charter covers only short links (< 10 meters), several groups are proposing to use it in a LAN style. In particular CERN propose to link Data

Acquisition Systems (DAQs) and computing resources using SCI carried over fiber optic links [**RD24Status93**]. The advantage of a single standard for use both within the high performance machines used to process the data and the physically distributed DAQs is one of the main reasons behind this suggestion.

2.1.2 Rambus

Rambus [**Rambus92**] is primarily intended for connecting memory to other devices, but is also suggested for connection of video devices. In contrast to SCI it uses a bus based system with a single master and multiple slaves. The data-path is 9 bits wide and consists of a terminated PCB track of limited length. Low voltage swings of 600mV are used. A 250MHz clock transfers data on both edges. As shown in figure 2.2, the clock is sourced at the far end of the bus and tracked along with the data back to the master, from where it is sent out on a second track. Whenever data transfers are done they are timed with respect to the clock that is travelling in the same direction as the data. This avoids skew between the data and clock.

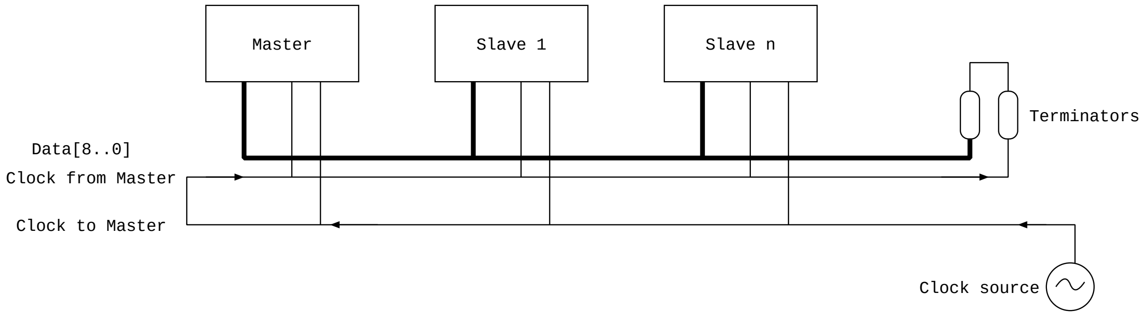

Figure 2.2: Rambus Clock and Data Organisation

Specialised memory chips, called Rambus DRAMs (RDRAMs), have been produced. These contain two standard DRAM arrays and a Rambus interface, thus they can directly act as a Rambus slave. The sense amplifiers of each of the DRAM banks are configured to act as a cache of the last row accessed in that bank. These caches provide high performance for localised accesses. It is intended that the master device be an ASIC. Macrocells exist to assist in driving the bus and multiplexing/demultiplexing from the 2ns data cycle to a wider 16ns cycle on the ASIC.

4.5Mbit RDRAMs (512k x 9 bits) have recently been announced by Toshiba, and have been demonstrated in a "Multimedia" system [**Ray93b**]. The system used has three live (NTSC) video inputs which feed an ASIC. In addition a 1280x1024, 18 bit-per-pixel display and some RDRAMs are attached. The display is divided into 4 quadrants each of which can be fed from either one of the live inputs or a buffer in the RDRAM. Using 8 RDRAMs each display quadrant may be updated from the memory at the same time as the three input video streams are being written to the memory (the bandwidth would allow for four incoming streams) [**Ray93a**].

The protocol used is restricted to request–acknowledge–data. Only the master may send a request. If the addressed slave is in a position to send (or accept) the data it will reply with an acknowledge message, otherwise it issues a negative acknowledgement (NACK) to indicate that the master should retry the request after a short delay. Following the acknowledge, between 1 and 256 bytes of data are transferred between the addressed slave and the master.

Rambus clearly provides a high speed method for connecting memory to other devices. In both the multimedia system described above and the high performance graphics system suggested in [**Rambus92**] the RDRAMs provide buffering. The single master mechanism becomes a limitation in a larger multimedia system where the network is a high bandwidth source of traffic. The master node, which will have to be a specially designed ASIC, will need to share the bus between the many devices that want to access the memory. Either the ASIC just connects the Rambus to some other interconnect, in which case a better solution would be to connect the memory directly to that interconnect; or the ASIC must provide a port for each device, which limits the extensibility of the system.

2.1.3 QuickRing

The QuickRing interconnect has been developed by National Semiconductor and Apple Computer[1]. Like SCI it proposes the use of a small (maximum 16 node) ring made from point to point links between the devices in the system. An interface chip provides both the ring drivers and the client interface [**Nat93**]. The links are made from seven differential signal pairs, six for data and one for a 50MHz clock. The six data lines are run at 7 times the clock speed, so 42 bits are transferred in each clock cycle. These contain 32 bits of data along with control and error detection/correction information. The client has two 200MByte/s connections to the ring, one for transmission and one for reception. The interface chips may be connected back to back in “bridge mode” to interconnect QuickRings.

Data connections across the ring are simplex, with a per packet handshake to allow flow control. A source sends a *voucher* to request to send, to which the sink will reply with a *ticket* when it is ready or an *abort* if it is not willing to communicate. Once it receives the *ticket* the source may send a variable length packet, which is multiplexed onto the ring by the interface.

[1] Apple have recently withdrawn from the project in favour of SCI.

2.1.4 The Mosaic System and Atomic Network

The Caltech Mosaic project [**Seitz92**] has produced a 128x128-processor multiprocessor machine. The Mosaic chip, shown in figure 2.3, contains the processor, 64kbytes of DRAM, a bootstrap ROM, and a two dimensional interconnect router. The router controls the four external links and an internal link which feeds packets to the processor. A multiprocessor machine is built from a regular mesh of these chips. Communication between the processors uses variable length packets, to which a source route is prepended. The route describes the relative co-ordinates (ΔX, ΔY) of the destination from the source. All X routeing is performed before any Y routeing, which prevents deadlock for the topologies used. At each node on the route the prefix is decremented until it is (0,0) when the packet is passed to the processor. This is an example of an interconnect that was designed specifically for processor–processor traffic.

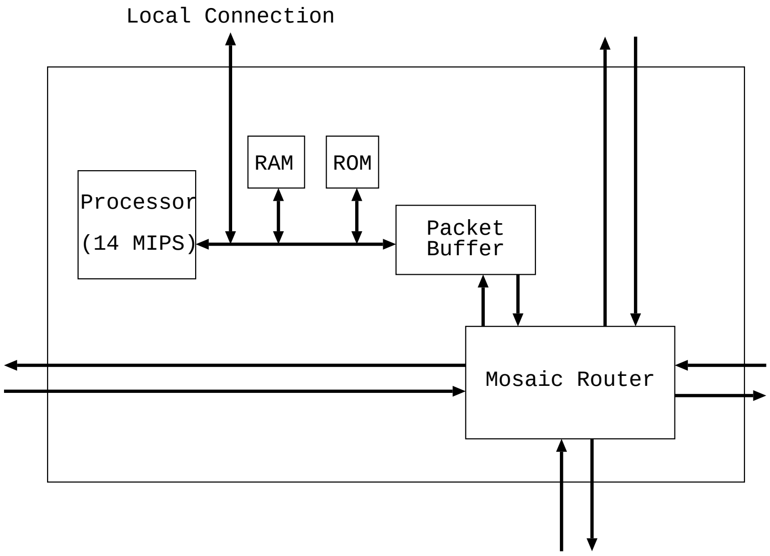

Figure 2.3: The Mosaic Processor

The Atomic project at ISI has used the Mosaic chip as a switch element to build a high speed LAN [**Cohen92**]. The regularity of the interconnect is not present in a LAN so the Mosaic source routeing has been extended to allow packets to be prefixed with a sequence of (ΔX, ΔY) pairs. Standard Mosaic routeing is used for hops described by each pair. The processor at which a pair terminates acts to form a conventional store-and-forward switch. The mechanism also allows multiple possible routes between any two network nodes. Host interfaces are provided through Caltech's VME and SBUS 'HI' boards, which contain four (two on the SBUS card) memoryless versions of the Mosaic chip.

Routes are provided by an "Address Consultant" (AC) service which runs on some of the network nodes. This tracks the network topology and provides hosts with the source route to use to reach some destination. If the AC is used when every connection is opened then it may be passed details of the type and data rate of the connection and it can allocate routes to share the network bandwidth between streams. The Atomic network is intended to be based on interconnected clusters of hosts. Within a cluster there will be a rich interconnection topology so routes can be allocated to avoid congestion.

Since the Mosaic chip was intended for building an internal interconnect, some work has had to be done on the connecting links to enable a LAN to be built. An interface chip is used to convert the Mosaic byte-by-byte acknowledgement flow control into a window based protocol. A line driver/receiver pair is used to interface to twisted pair cable at rates of up to 316Mbit/s [**Cohen93**]. The use of optical fiber will be considered when its cost drops.

Much stress in the Atomic project is put on the use of the Mosaic chips to provide flexibility at all points in the network. The processing power of these devices is present in every routeing engine, be it part of a switch or a host interface. Indeed a network can be built entirely of host interfaces, since each naturally provides the routeing resource. The presence of the processor also enables smart host interfaces to perform some protocol processing.

A network running the IP protocol suite has been demonstrated using Sun 3 and Sun 4 workstations as the hosts. These hosts are unable to generate large data rates, but traffic generation code has transferred data at over 400Mbit/s.

The use of the Atomic network for both intra- and inter- machine communications, as suggested by [**Finn91**], is discussed in Chapter 4.

2.2 Asynchronous Transfer Mode

The use of Asynchronous Transfer Mode (ATM), also known as Asynchronous Time Division Multiplexing (ATDM), has become popular in the past few years. ATDM has a long history of use in the local area. Bell Labs have been working in the area since 1969 producing three experimental ATDM LANs: Spider, Datakit and Incon [**Fraser93**]. In the Computer Laboratory the Cambridge Ring [**Hopper78**] and its successors are based on ATDM techniques. ATM is currently being standardised for use in the wide area Broadband ISDN [**CCITT90**]. This has caused local area ATM to become more popular, with the "ATM Forum" working on standards [**NCLATM92**, **ATMUNI93**]. The advantages and disadvantages of ATM, along with a description of the multiservice protocols used on the experimental networks in Cambridge may be found in [**McAuley90**].

There are two key principles underlying ATM: small fixed-size data transfer units known as cells (or minipackets in the original Cambridge networks), and lightweight virtual circuits.

The idea of circuits is familiar from circuit switched networks such as the original telephone system. In this a dedicated link is made between the end-points for the duration of the communication. The virtualisation of this idea is to multiplex many communication channels onto a single physical line. The abstraction provided to an end-point is a standard circuit. Each of the communications gets a share of the total line bandwidth. In Synchronous Transfer Mode (STM) the multiplex of the line is fixed, each connection having a pre-allocated time slot, relative to a fixed-size frame. For example, 32 64kbit/s voice calls may be carried on a 2Mbit/s trunk. The position in the frame is fixed for each link, so a small buffer is required at each switch. In Asynchronous TDM the multiplex schedule is not fixed, so each time slot must begin with a tag to identify the circuit, but a frame structure is not required. This tag is only meaningful on a single link and is replaced by the tag for next link at each switch (this operation is referred to as "remapping"). Both systems require a connection to be set up before data can flow, to allocate either the position in the frame or the identifying tag. A lightweight VC [**Leslie83**] is a virtual circuit where there is no hop-by-hop error control and which may be torn down by intermediate nodes at any time. The end system software needs to check for errors and possibly re-establish the link.

The cells are small and of fixed size both to allow ease of handling in hardware and to keep the multiplexing granularity small. The STM network provides a guaranteed fixed amount of bandwidth to the connection and the delay through the system is fixed. However, the multiplex is fixed, so if the sources are bursty (as many data sources are) the reserved bandwidth is wasted. The unused allocation of a source which is not transmitting cannot be used by sources which are. The asynchronous nature of ATM allows for better accommodation of bursty sources at the cost of increasing the jitter. Offsetting this, the small size of the cells allows traffic scheduling to be done on a fine enough granularity that the network is able to offer a range of Quality of Service parameters. These allow it to accommodate the whole range of data types from fixed-bandwidth jitter-sensitive sources such as voice through to high peak-use jitter-insensitive data such as file transfer.

An ATM cell consists of two parts: the header and the payload. Each is of a fixed size. The header contains at least an identifier of the virtual circuit to which the cell belongs – known as the VCI. This is only meaningful for a single hop in a multi-hop connection, and must be "remapped" at each stage. In addition, there can usefully be a single bit of end-to-end information allowing certain cells to be marked. Because of the importance of the header it is often protected by its own CRC, the Header

Error Check (HEC). The payload size that has become standard is 48 bytes. The ring based networks in use at Cambridge, which predate the standard, use 32 bytes.

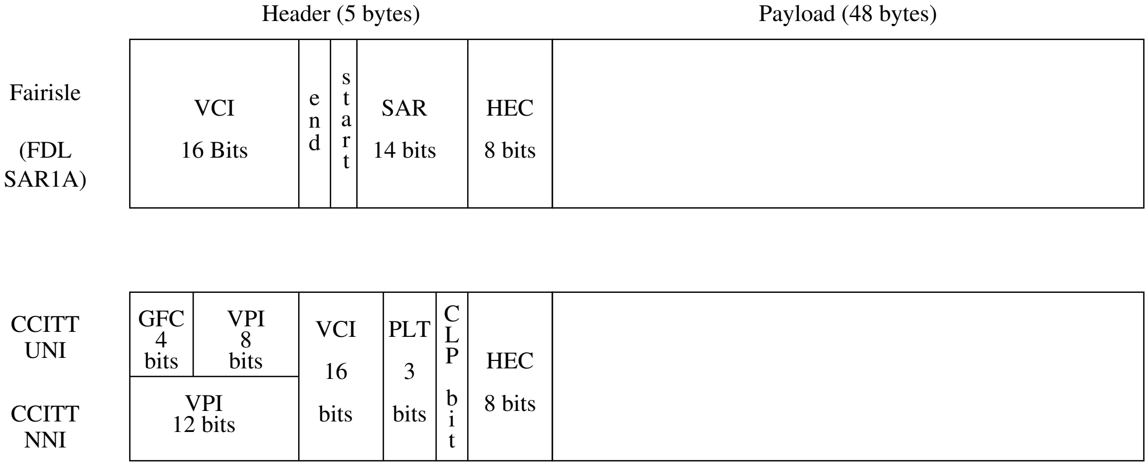

Figure 2.4: The CCITT and Fairisle Cell Formats

The ATM cell format suggested by the CCITT and ATM forum is shown in figure 2.4. Both the User-Network Interface (UNI) and the Network-Network Interface (NNI) form are shown. The virtual circuit identifier is divided into two parts: the Virtual Path and the Virtual Channel. Multiple Virtual Channels may be multiplexed onto a single Virtual Path between two points in the network. Switches between the VP end-points need only use and remap the VP field in routeing all cells on the path. The payload type (PLT) field is used to distinguish in-band management cells from user cells, for user cells it also contains the single bit of end-to-end information and may indicate whether congestion was met in the network. There are two additional fields: a single bit "Cell Loss Priority" field (CLP), which may be set to indicate cells which should be preferentially dropped if the network becomes congested; and 4 bits for Generic Flow Control (GFC) which are used at the User-Network interface for flow control. The exact behaviour of the CLP bit is still ill defined, and the standard currently requires that the GFC bits are all set to zero.

One of the issues involved with sending small fixed-size cells is that of transferring the large variable length data blocks (or frames) generated by most user programs. There is therefore a need to segment blocks into cells at the source and to reassemble them at the destination. The standards define a series of ATM Adaptation Layers (AALs) which provide this function. AAL 1 and 2[2] provide for transport of constant and variable bit rate isochronous traffic, and AAL 3/4[3] provides for transport of data traffic. Due to complexity in AAL 3/4 an additional version called AAL 5, which is both simpler and more efficient, has been introduced for data traffic. AAL 5, and

[2] At the time of writing AAL 2 has not been completed or approved.

[3] Originally adaptation layers 3 and 4 were distinct, but they have now been combined.

the similar protocol AAL X used in the Bell Labs XUNET project [**Berenbaum93**], use the end-to-end bit in the cell header to mark cells as being the end of a SAR block. These cells contain length and CRC fields for verifying correct reception and reassembly of a block.

The ATM cell in use at Cambridge during the experimental period of this thesis is also shown in figure 2.4. It is derived from the Unison DataLink format for cells on the CFR [**Tennenhouse86**]. This has passed through a series of modifications, with the intention being to move to something more like the standard cell. The Fairisle DataLink Cell format [**Black93a**] was used with the SAR1A segmentation and reassembly protocol [**Black93b**] as the adaptation layer. The cell header allocates two octets to the VCI; two bits to indicate frame start and frame end (the frame end bit fulfills the role of the end-to-end bit); 8 bits reassembly count, which gives the sequence number of the cell in the block; and 6 bits to the block number. The service provided is closest to that of AAL 5. The intention is to migrate to using AAL 5 so new designs are avoiding the use of any end-to-end header information other than a single bit.

2.3 Multimedia Computing

The addition of continuous media, such as audio and video, to the more traditional computer data forms has enabled "multimedia" applications to flourish in the last few years. The earliest systems, which have been available for at least a decade[4], worked in the analogue domain, but these only provided software control to enable or disable audio and video inputs and outputs. Where these early systems were networked a specialised analogue switched network was used. The systems that are considered in this dissertation are distinguished by using digital representations of the data, and which therefore have the ability to be fully integrated with computers and digital networks.

Several groups have worked on providing multimedia workstations using digital video and audio, and transferring these over standard computer networks. The applications that have been developed by these groups have generally included a videophone or conference tool, and, where storage is available, a multimedia mail system. In addition some sites have connected TV or radio tuners as input sources to provide access as a network service. The Olivetti Research Pandora system has coupled this with the storage facility to record and provide the latest news and weather from the TV [**Hopper90**]. Some systems also allow editing of multimedia "documents" or "presentations".

[4] I worked with such a machine based on a Phillips Laser Disc and a BBC Microcomputer in 1983/4.

This research has showed that providing video is an important addition to a system. A study by Sun Microsystems of a physically distributed group of collaborating workers reported increased productivity and greater cooperation using a system which provided video than when they used the identical system without the video link (but still with audio) [**Tang92**, **Redell92**]. A large improvement in the perceived quality occurs at about 12 frame per second (since the data is being displayed from a framestore this is not a screen refresh rate observation), with lower frame rates many of the gestures and visual cues are lost.

However, it is notable that continuous media are not yet first class data objects. There are three main causes for this. Most of the systems in currently in use, such as Pandora and the Lancaster MNI [**Blair93**], use multimedia peripherals that work beside the main machine rather than as part of it. Secondly, even in systems like the J-Video project [**Berc93**] that do take the video through the processor there is no operating system assistance, so the main concern of the application is the real-time aspects of the data. Finally the movement of the continuous data tends to cause the network subsystem to become the system bottleneck.

Despite their limitations, these early systems have shown the power of having multimedia (especially video) available in a workstation. The ability of a single digital network to carry multimedia data in addition to more conventional types has also been demonstrated. The current types of application, where the real-time data is delivered to the human user, is only the first stage. As machines become more powerful the processing of the new data types will become common; such systems have been termed "second generation" by the multimedia group at MIT [**Tennenhouse92**].

Operating system support for multimedia is being studied in the Pegasus project. Here the idea of quality of service is being used as the processor scheduling paradigm. At the lowest level processes are guaranteed a minimum percentage of the CPU. The project has produced the architectural description of the Nemesis kernel which forms the basis for the system [**Fairbairns93**]. This scheduling paradigm has been shown to support the reception of video that has been coded in a layered manner. Even simple JPEG compressed sources can be supported. JPEG represents the image as 8x8 blocks of video encoded as a discrete frequency spectrum. Thus a block is represented as an average level and a series of modifying components of progressively higher (spatial) frequency. [**Hyden93**] demonstrates an application which requests enough CPU resources to decode the average coefficients with some time over. This extra requested time, along with any extra that the system is prepared to give to the process to keep the CPU utilised, is used to add the higher frequency information – improving the picture quality. Since the average term alone gives a coarse image, the program would probably reserve time to decode at least the first frequency coefficients, increasing the minimum picture standard.

2.4 Cache Memory and Terminology

Caching refers to keeping a local copy of recently used data to allow for fast future access. One of the areas explored in this dissertation is the behaviour of a memory system using a CPU cache. A review of this subject is provided by [**Smith82**].

Several parameters are commonly used to characterise caches and the various parts of the hardware. Unfortunately different terms have been used by various authors. The terms described below will be used to maintain consistency when referring to the work of others.

Each cache entry is known as a cache **line**, also referred to as a cache block. The line, typically 8 to 64 bytes, is the smallest size unit of data in the cache: if any byte within a line is resident in the cache then all the other bytes in the line must also be present. The address of a particular byte may be given as the line number and an **offset** within the line. When a write is made to a cached line it may either be written to memory at the same time, giving a **write-through** cache, or retained in the cache and only written to memory when the cache space needs to be reused, or the operating system explicitly **flushes** the data. In the latter case the cache is known as **copy-back** (or **write-back**), and while the data in the cache is more up to date than that in main memory the line is described as **dirty**.

Many main memory addresses may be cached in a given line. There is therefore a **tag** associated with every line, storing details of the memory address of the data currently cached in the line. In addition there is usually a bit to indicate that the data is **valid**, this bit may be cleared by DMA hardware if the main memory copy of the data is updated or by the operating system if the memory is unmapped.

The lines of a cache may be divided in to a number of **sets**. The degree of **associativity** of a cache, normally equal to the number of sets, gives the number of places in a cache that a given address from main memory can reside. The cache is referred to as n **way set associative**. The case where the main memory address may reside in any cache line is called **fully associative**. In a **direct mapped** cache there is only one place in the cache where a given main memory address may be found. If the degree of associativity is greater than one then a replacement algorithm is required when a new line comes into the cache. Typically, Least Recently Used (**LRU**) is used for small associativity caches, and **random** replacement used in larger ones. *Simulation* studies, which know the future behaviour, allow investigation of **optimal** replacement, in which the line replaced is the one which will not be used for longest.

The total size of the cache is thus given by $linesize \times lines\_per\_set \times associativity$. When comparing cache results between different architectures, it is usual to compare results for caches of the same size, although sometimes results are given for the same size sets using different associativity.

The use of CPU caches has been the subject of much study. The main metric that was used for a long time for comparison between caches is the **miss ratio**, which is the number of accesses where the data is not in the cache divided by the total number of accesses. However, this only tells part of the story. In addition the time taken to retrieve data from the cache should be considered. [**Hill88**] presents "A Case for Direct-Mapped Caches", being that for large caches the change in miss ratio achieved by moving from direct mapped to set associative caches is cancelled out by the increase in cycle time required for the more complex cache design to work. More recently this problem has been addressed through novel cache architectures, for example [**Agarwal93**] presents a design for a cache with the miss performance of a 2 way associative cache, but the hardware simplicity of a direct mapped cache. Also, as [**Przybylski88**] points out, the effects of the system in which the cache is used must also be considered. It is usual to use the CPU cycle time as the base time, and quote cache and main memory access times as multiples.

It is now becoming common to have a two level CPU cache hierarchy. The **primary** cache is actually on the processor chip, and is typically a fairly small (4–16kbytes) direct mapped write-through cache. The **secondary** cache is a much larger external cache and is frequently copy-back. Recent CPUs, such as the MIPS R4000 [**MIPS91**] and the DEC Alpha AXP 21064 [**DEC92c**], provide on chip support for the secondary cache, improving the possibilities for pipelining tag checking. Studies have reported that the primary cache has a miss ratio of 10–20%, with the secondary cache missing 1–2% of the time.

2.5 Implementation Technology

Much of the work described in subsequent chapters of this dissertation is practical. Ideas were developed by building demonstration systems and exploring their behaviour. Hardware designs, which were extensively simulated before production, were based on reprogrammable components to allow for frequent modifications. Software tools were written on standard unix workstations, making debugging easy. In contrast, software for the demonstration systems was developed over a small microkernel with all protection disabled to allow "user-space" processes direct access to the hardware. While this lack of protection made debugging harder it was far faster to develop these small programs than integrate rapidly changing code with the kernel.

2.5.1 Xilinx FPGA

The Xilinx Field Programmable Gate Array (FPGA)[5] provides a useful tool for prototype and experimental hardware. The FPGAs are RAM based components which are programmed to take on a particular hardware configuration of an equivalent size to a gate array of several thousand gates. The device may read its configuration out of a PROM[6] or may be written to by a processor. Translation of a design from a hardware description language into configuration data may take as little as a few minutes for small changes to an existing design – allowing fast turnaround during testing, though generation of the configuration from scratch often takes many hours.

The major advantage of these devices for experimental hardware is the ease of modifying fairly complex designs. More traditional programmable devices such as E^2PALs are usually limited in size and require removal from the circuit for reprogramming. In contrast the FPGAs may be reprogrammed in circuit, a process which typically takes less than 100ms. Indeed an expansion card has been produced which is programmed at kernel startup time with whatever design has been selected in the boot parameters from a set of designs and drivers compiled into the kernel. The disadvantage of the FPGAs is a lower performance than the latest PALs. Thus in evaluating the performance of FPGA based hardware it is useful to consider some of the limitations of the devices. The Xilinx XC3000 series devices were used in all designs described in this dissertation.

The XC3000 series FPGAs [**Xilinx92**] consist of four main components: Configurable Logic Blocks (CLBs), Input/Output Buffers (IOBs), Tristate Buffers (TBUFs) and routeing resources. The routeing resources include local and long distance lines, and two low-skew clock distribution networks. A design is entered in a gate based hardware description language, which allows simulation using unit delay gates. This is then packed automatically into logic blocks which are placed using simulated annealing, and routed using a delay driven rip-up-and-retry algorithm. Manual partitioning of logic into blocks, and placement of the blocks on the chip are possible. The designs reported on in later chapters achieve utilisations of over 90% of the logic blocks, and use most of the package pins, though this is only possible through a large amount of manual placement. A tool allows extraction of the worst case delays between clocked points in a placed and routed design, giving the maximum clock speed.

Figure 2.5 shows a CLB, of which there are between 64 and 320 on the device. They have five inputs and two outputs and use a lookup table to implement combinatorial functions. There are two edge-triggered D flip–flops in each block, whose outputs

[5] Also referred to as a Logic Cell Array (LCA).

[6] A PROM emulator designed by Q. Stafford-Fraser is frequently used in the Computer Lab.

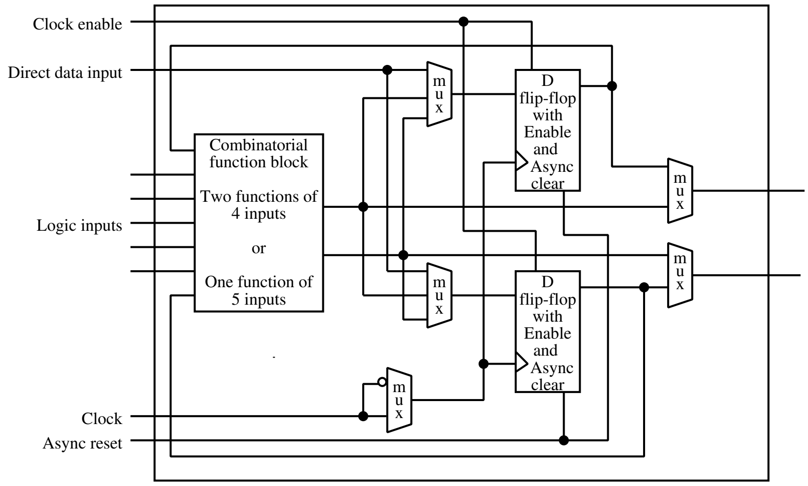

Figure 2.5: Xilinx 3000 Series Configurable Logic Block

may internally be fed back to the lookup table. The block outputs may come from the combinatorial or latch outputs. Subject to some additional rules, any two functions of up to four inputs, or a single function of five inputs may be put in each CLB. The configuration of the lookup table, the selection of possible inputs to it, and the state of the multiplexor selection blocks (labelled mux) is held in RAM which is written at device configuration time.

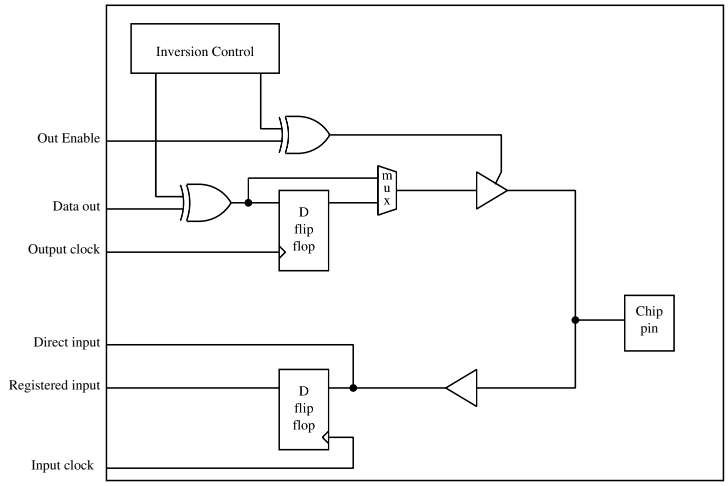

Figure 2.6: Xilinx 3000 Series I/O Block

Input and output to package pins is provided by the IOBs (figure 2.6). There are between 64 and 144 general I/O pins on the device, though some of these have special functions during device programming. The IOBs contain both an input and an output section each of which have optional latches and inverters. Unfortunately, the output inverter is on the wrong side of the latch for the correct behaviour on reset in some cases. The output may be made high impedance, allowing IOBs to be linked to bidirectional buses, and by linking the output and tri-state enable lines an “open collector” effect may be achieved. Both the direct and latched inputs may be taken into the internal logic, though experience has led to designs where the majority of inputs are latched onto the chip. The worst-case timing characteristics of these blocks are far from ideal: inputs require 17ns setup time and the output delay is 13ns. This contrasts with 7ns setup and 5 ns output delay in “10ns” PALs.

The combination of routeing delays and in particular the IOB characteristics limit the practical clock speed of systems using 3000 series Xilinx to 20–25MHz. The input setup requirements and long output delays frequently force the use of an extra external latch complicating pipelines. In the CPU node described in Chapter 5 fast (10ns input to output delay) PALs are used for high speed operations, controlled by a Xilinx - this gives most of the reprogrammability advantages and high performance.

2.5.2 Wanda

The Wanda Microkernel has been designed for building high performance dedicated systems. It provides the main platform for systems research at Cambridge, and is used in both the Fairisle switches and the Desk Area Network prototype. The design philosophy for the Wanda system is described in [**Dixon92**]. Wanda has been ported to many platforms and processors including Vax, Motorola 68000 series, ARM and R3000. There are two main Wanda variants, the standard or “target” version which uses an MMU to implement memory protection and management and the “single address space”(SAS) version which runs without an MMU.

Wanda provides a multi-threaded environment, both within the kernel and in user-space, using a simple priority-based run-to-completion scheduler. The simplicity of the standard scheduler reflects the design philosophy to provide a simple but complete system which allows more complex options to be constructed later. For example a real-time scheduler has been produced. The kernel provides a virtual memory system giving the user variable-sized segments that are mapped by the system onto fixed-size pages, on SAS systems a one-to-one virtual-physical address mapping is used. Inter-process and inter-machine communication is provided by kernel support for the MultiService Network Architecture [**McAuley90**], which is designed to support ATM networking. Other protocols, such as the IP suite, are provided through user-space server processes.

Most of the uses of Wanda reported in this dissertation were with the version for the ARM series of processors running in SAS mode. All the ARM based machines that Wanda runs on have an MMU, but it is configured to provide a direct mapping from logical to physical addresses. The ARM processor has two interrupt mechanisms: the standard interrupt request (IRQ), and the fast interrupt request (FIQ). The Wanda kernel only makes use of the IRQ and it never disables FIQs, with the exception of when it reports fatal status information or while the machine is being halted. This allows the FIQ code to run in parallel with the kernel at a higher priority; the existence of a separate FIQ bank of CPU registers makes for easy switching between the two. In this case the FIQ can present itself as a virtual device to the kernel, raising IRQ interrupts and using shared memory to communicate.

Chapter 3

The Fairisle ATM LAN

The Fairisle project [**Leslie91**] has developed a switch-based ATM network. All of the parts used to build the switches were designed to be highly programmable, originally to allow for experiments within the Fairisle project. This has allowed them to be reused in the Desk Area Network experiments. This chapter describes the Fairisle network and its components.

3.1 Introduction

The Fairisle project was interested in looking at the performance and management of a general topology switch based ATM LAN for carrying multiservice traffic.

The development of ATM networks at Cambridge has previously been based on ring networks. In particular the systems research environment has been supported by the Fast Ring [**Temple84**, **Hopper88**] and the Backbone Ring [**Greaves90**]. In these networks the hardware based media-access control provides the sharing policy. Using a switch based network allows policy to be implemented in each switch. In addition extra management issues, such as what happens when a machine moves, need to be faced on a non-shared-media LAN.

The Fairisle network is intended as a test-bed for signalling and management experiments and therefore needs both to be highly flexible and to be able to carry real data. As with most network switches the organisation of the switch is to have a central switching fabric surrounded by port controllers[1] as illustrated in figure 3.1. For the Fairisle system a single port controller card includes both the input and output ports. The actual design of the switches themselves is not a primary area of interest to the project, but the management of the switches and queues associated

[1] The device known as a port controller in this dissertation is also referred to as a line card.

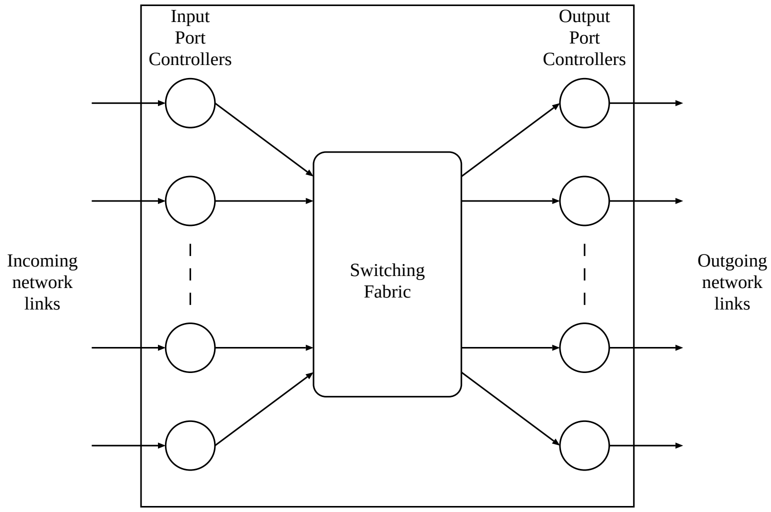

Figure 3.1: Typical Network Switch Organisation

with them is. This is reflected in the relative sizes of the single chip switching fabric and the port controllers which take up a double height Eurocard. The fabric and two types of port controller are described below.

3.2 The Switch Fabric

The Fairisle switch fabric [**Leslie91**, **Gordon93a**] forms the basis for both the ATM LAN switches and the interconnect used in the prototype DAN described in the next chapter. A switch fabric, adapted from that designed by Peter Newman [**Newman89**], is based on 4×4 self-routeing crossbar elements. Three sizes of fabric are currently in use: 4×4, 8×8 and 16×16. The 16×16 is made using a standard "Delta" formation of the 4×4 elements; the 8×8 is half of a 16×16 with no use made of the potential extra paths.

The crossbar element is implemented on a single Xilinx XC3064[2], and the main signals are shown in figure 3.2. The datapath is eight bits wide, and there is a single bit acknowledgement return path. Data is clocked through the crossbar at a byte rate of up to 20MHz, giving a bit rate of 160Mbits/s. The switch is of the synchronous self-routeing kind. All inputs with data to send will present their cells at the same time, referenced to a common frame pulse. The first byte presented contains a tag indicating which output is requested.

There may be contention for crossbar outputs. When the elements are combined to make a larger switch this results in internal blocking. Two priority levels are provided to resolve this contention, when cells of the same priority contend inputs

[2] A XC3042 was used for the first version.

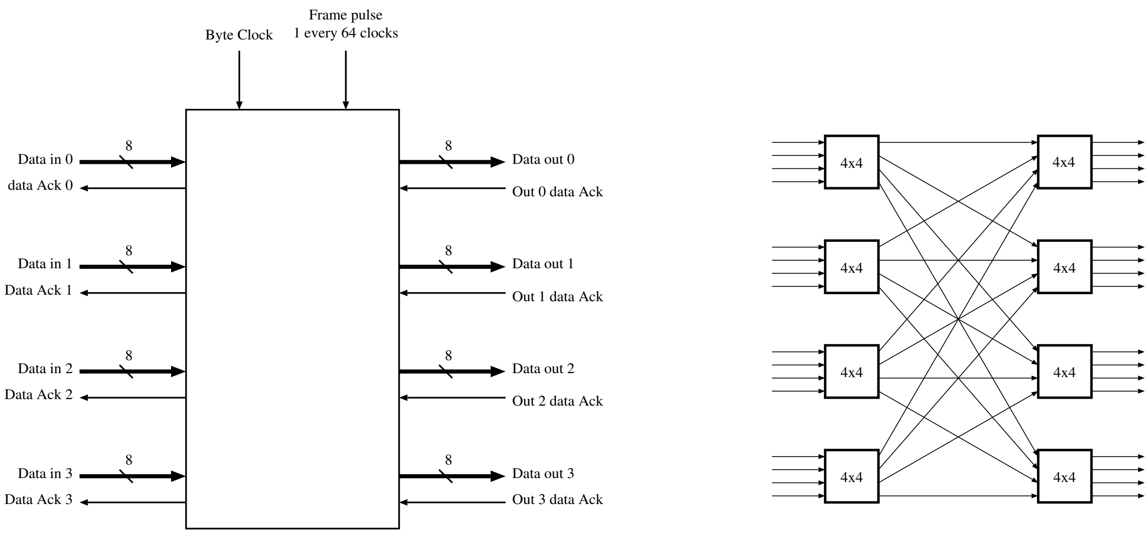

Figure 3.2: Fabric Signals and the Delta Formation

are serviced in a round robin manner. For every output the last input which was granted service is remembered. When there is contention for the output the recorded value is used as the base, and the next highest requesting input (modulo the number of inputs) is selected. Inputs whose cells are rejected have the acknowledge signal cleared, otherwise its value is copied from the output (which is either the second stage fabric or the output port controller). This allows a cell to be rejected by the output port controller, for example because flow control on the network link has stopped transmission. Thus when a cell is presented to the fabric the source is informed if it was rejected and may retry it.

3.3 The Port Controller

The Fairisle switches are input buffered on the Port Controllers [**Hayter93b**]. The Port Controllers have six main functions:

- Receive cells from the transmission system.
- Buffer incoming cells and queue them for insertion into the fabric.
- Insert cells into the fabric, prepending the routeing tag and remapping the VCI for the next hop.
- Manage contention in the fabric by sensing the acknowledgement line.
- Reception from the fabric and transmission on the outgoing line.

- Provide for experimentation with queueing, signalling and network management

Figure 3.3 shows an overview of the Port Controller hardware. This divides into two main parts: the network related section and a general microprocessor section.

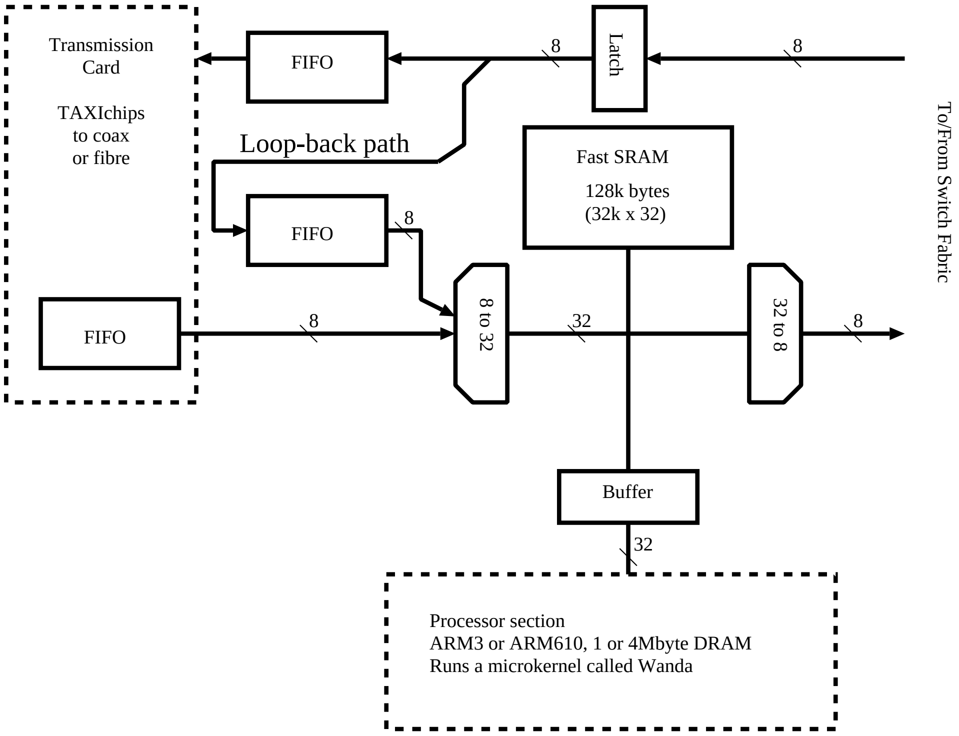

Figure 3.3: Port Controller

The network section is controlled by a Xilinx FPGA (not shown in the diagram). The SRAM contains 2048 cell buffers and the Xilinx contains subsystems to insert cells into the fabric, to pass cells from the fabric to the line, and to receive either from the line or via the loop-back path from the fabric. When a cell is received it is put into a cell buffer and the processor interrupted. The processor is then able to access the whole cell, but normally only the VCI is read. The processor manages the queueing of cells, and issues requests to the hardware to insert them into the fabric. This allows software to be fully involved with the queue management, which is an area of research, while the time critical operations are performed in hardware.

All the buffering is performed on the input to the switch. Cells coming out of the fabric are moved entirely by hardware. There are two possible paths for these cells. In the usual case they are sent to the transmission system via a speed matching FIFO, but it is also possible for them to be looped back into the input side of the port controller. The loop-back path is mainly used for switch and network management traffic, but it also allows a Port Controller to be used as a host. The use of the loop-back path to allow cells to be replicated both to the transmission line and back into the switch is being considered as a mechanism for implementing multicast.

The Port Controller processor section is based around the ARM3 or ARM600 microprocessor. The section is powerful enough to be used as a general machine for systems research when not being used as part of a switch. The standard ARM "podule" expansion bus is included, and is mainly used to allow connection between the switch and an Ethernet or CFR. The processor runs a version of the Wanda microkernel, along with a hand crafted fast interrupt routine (FIQ). The FIQ is responsible for queueing and other per-cell operations; the kernel provides the environment for overall management and signalling. As described in section 2.5.2 the FIQ appears to Wanda as a hardware device would – communicating via queues and standard interrupts. The FIQ is thus able to run even if the kernel has "interrupts disabled".

3.4 The Null Port Controller

The Null Port Controller is a replacement for the standard Port Controller which can be used on low bandwidth links [**Gordon93b**]. The card performs reception and transmission to the line and fabric, and performs remapping of VCIs, but it has no buffering functions. This design is much simpler, and is therefore cheaper. This design has been adapted for use in the Hewlett-Packard Sapphire switch [**Prudence93**].

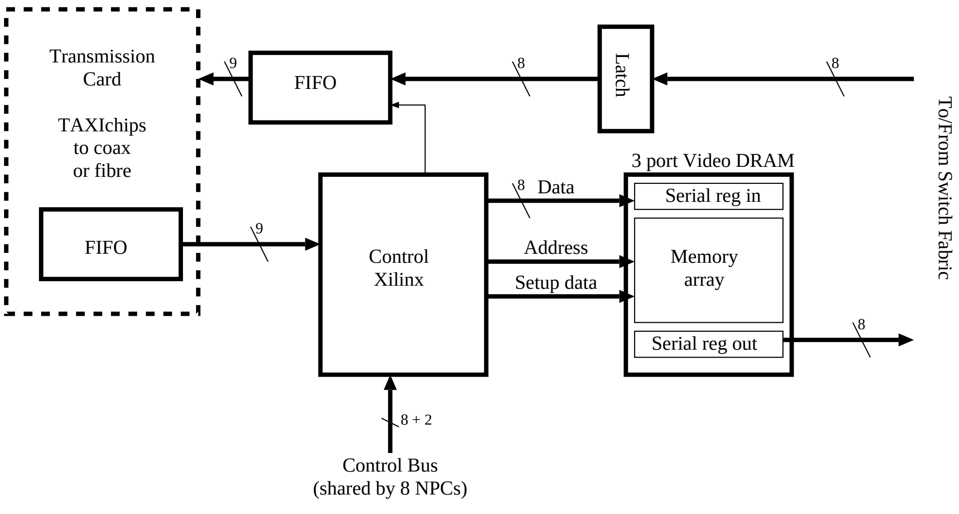

Figure 3.4: Null Port Controller

Figure 3.4 shows the components of the Null Port Controller. Triple ported VRAMs are used to provide the VCI remap memory. These devices have the normal DRAM memory access port and two serial registers. A single access cycle to the device is used to move data between the serial registers and the memory array. During this transfer a mask specifies which bits are transferred within the row. A cell arriving from the transmission system is strobed into one of the serial registers along with a mask which masks off the VCI. The incoming VCI is used to form the address in the memory array to which the serial register is written. The data, including the new VCI and routeing information, is then read out of the memory array into the second serial register for injection into the fabric. The control bus is used by an external processor to set up the VCI and routeing information using the standard memory access port.

3.5 Host Interfaces

A network is of little use if it is not possible to connect host machines to it. For a large number of experiments the Port Controllers were used as stand alone machines running the Wanda kernel. To permit the connection of other hosts a VME interface was built, but this was not used to any great extent [**Beeler92**].

The main host interfaces used for the Fairisle network are provided by boards designed at Olivetti Research, with whom the transmission interface was jointly devised [**Hayter92a**]. The “Yes” board provides a TURBOchannel interface for DEC workstations, and with a Mezzanine board gives a VME interface [**Greaves93**]. The “Maybe” board provides an Acorn Podule bus connection [**Gordon92**]. In addition an EISA interface is being designed at the Computer Laboratory for Hewlett-Packard; this will allow “PC”s and H-P Snakes to use the Fairisle network [**Leslie93**].

3.6 Port Controller Developments

One of the reasons for using the Xilinx as the basis for the Port Controller hardware was to permit modifications as the behaviour of the switch and network became better understood. The FPGA configuration is loaded by the kernel during its initialisation, and it is usual for the design used to bootstrap the Port Controller to be replaced once it has been used to load in the new kernel. This has proved very effective, both when debugging designs and for testing large changes of functionality.

The initial Xilinx design on the Port Controller was relatively simple. Software provided up to 8 buffers for storing received cells and up to 8 buffers could be queued for insertion into the fabric. If the reception list became empty then arriving cells would be thrown away, and in the event of a cell being blocked in the fabric the transmission system would stop until it was restarted by software. In addition the

Xilinx arbitrated between the three uses of memory – reception, fabric insertion and processor access – on a simple priority basis. An early modification allowed VCI remapping to be done by the hardware.

It soon became clear that the FIQ code would be small enough to fit in the CPU on-chip cache, so its execution time and therefore the cell throughput of the Port Controller were dominated by any other memory references that were made. The ARM processor uses a separate core and bus clock, so all external accesses entail a delay as the processor resynchronises with the bus clock. In addition, any access to the Xilinx status registers or cell buffer SRAM requires an asynchronous bus protocol to be used. It was also noticed that a large portion of the code in the FIQ was spent manipulating the queue of free buffers. While the intention behind the use of software cell management in the Port Controllers was to allow queueing experiments, the free buffer queue is not an area for experimentation. A free queue will be required irrespective of the design, so it is an obvious candidate for hardware implementation.

The control Xilinx was completely rewritten to support hardware management of the free queue, and with more attention paid to allowing the processor to read both status and queue information in a single access. In addition, the path for cells being forwarded through the Port Controller was optimised at the expense of cells moving between the network and the Wanda kernel. This design, known as "Xi3"[**Hayter93c**], became the standard and is the one used in all experiments described in this dissertation.

The third revision of the Port Controller PCB uses a larger Xilinx chip. The "Xi3" design was extended into "Xi5"[3] to take advantage of this device [**Hayter93a**]. Extra counters and timestamping mechanisms were added to assist with measurements and flow control experiments. In addition there was some re-ordering of control bits to assist in the move to the ATM Forum cell header format.

A series of performance measurements were made on the Port Controller and Fairisle switch [**Pratt93c**]. These showed that on a Port Controller using an ARM3 processor the cell forwarding rate was limited by software. Using an ARM600 processor (which decouples the processor from the system by using a write buffer) the system was limited by the Xi3 hardware to just less than the maximum line rate. This, especially the effect of using the write buffer, validated the decisions made in the Xilinx redesign. The hardware performance limitation in Xi3 was removed in the Xi5 design so this should be capable of the full line speed.

[3] For the curious reader it should be remarked that the missing "Xi4" design was an experimental version used to develop the later designs, Xi4 never escaped into general use.

Chapter 4

The Desk Area Network

This dissertation is concerned with the behaviour of the novel machine architecture known as the Desk Area Network (DAN). In this chapter the motivation for the development of this architecture is given, along with a description of the prototype implementation. The architecture was originally proposed in [**Hayter91**]. The target environment for the DAN is a multimedia workstation which uses a high speed ATM LAN to become part of a distributed system.

4.1 Introduction

The network forms a vital component of a distributed system. This has been true throughout the history of such systems – in the Cambridge Distributed Computer System (CDCS), for example, the network was the only I/O connection for many machines [**Needham82**]. The use of powerful workstations supported by many servers is becoming more common, and many sites are finding that the load on existing networks is becoming unacceptably high. One of the responses to this is to increase the performance of the network. Local area networks have always been described as "high speed". This has usually been in comparison to wide and metropolitan area networks, rather than a comparison with the interconnects found within a machine. However, in the latest generation of networks there has become a blurring in distinction in both speed and protocols over the local, metropolitan and wide areas, for example the 45Mbit/s T3 Internet backbone links many sites using 10Mbit/s Ethernets and 100Mbit/s FDDI rings. Perhaps of greater interest is the convergence with the internal machine interconnect. Several recent papers have observed this and expect the internal and external speeds to remain similar for some time [**Druschel93**].

Many workstations are now shipped equipped with multimedia peripherals. The

Pandora system [**Hopper90**] illustrates some of the many possibilities that will become available when networked multimedia systems become more common. In these machines there are a large number of data flows between the machine and the network, so the performance of the network interface becomes crucial. Research done on the Cambridge Fast Ring has demonstrated this. [**McAuley90**] observes that the existence of only a single cell buffer limits the performance of the ARM and VME interfaces. [**Dixon92**] describes some of the effects this has on network device driver code. The Pandora box also provides an illustration of this problem – the network interface card had to be upgraded to include a second Transputer to give satisfactory network performance.

It seems likely that the use of multimedia will grow. The applications produced thus far, termed "first generation" by [**Adam93a**], deliver the multimedia data to the user, and therefore are active on a human time-scale. As processor speeds increase it will become possible for the machine to manipulate the data in addition to simply displaying it. The new applications will work on a machine time-scale and can only increase the bandwidth requirements within the machine, though jitter requirements may be relaxed.

The area of internal machine interconnects has also been of high prominence in recent years, particularly as processor speed is increasing faster than the speed of the other components in typical systems. As was discussed in Chapter 2 the special purpose interconnects that were intended for use only in multiprocessor systems are giving way to more general interconnects which may also be used in workstation class machines.

4.2 The Network Interface Bottleneck

The increase in LAN speed and the use of multimedia have led to machines with connections to higher speed networks and peripherals which can consume a sizable proportion of the available bandwidth. These devices consume bandwidth on both the internal interconnect and through the network interface. [**Pasquale92**] uses the term "intensive I/O" to describe this behaviour, in a paper which provides a useful summary of the problem. The paper suggests refinement of existing buses and DMA techniques to alleviate the problems, but this will only work as a short term measure. It also highlights the idea of separating data and control, which is used to good effect both in the Fairisle Port Controllers described in the previous chapter and the workstation devices described later in this chapter.

In addition to raw data throughput there are two other areas for consideration. The first is the multiplexing granularity. The interface and system interconnect are being shared amongst all parts of the machine, and incoming data destined for various parts of the end-system will be arriving from a number of sources. In existing workstations data is often transferred using some form of DMA technique, and each transfer must complete before the next starts, potentially blocking other users of the interconnect. In the DEC 3000 AXP systems the problem of DMA blocking CPU access to memory is relieved by the "system crossbar" buffering DMA traffic and permitting interleaved CPU cycles [**Dutton92**].

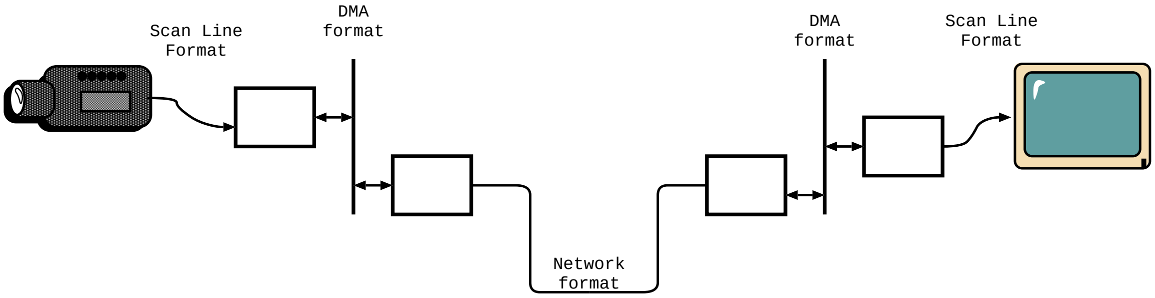

Figure 4.1: Data Format Transformations

The second problem is one of data format transformation. Frequently the unit of data movement within the machine (the DMA block size for example) is suited neither to the format of the data on the network, nor to its end-point format in the machine. This is illustrated in figure 4.1 showing one direction of a video connection. The natural unit for the end-point is a scan line[1]. These have to be converted into DMA blocks for transfer to the network interface, where they are again translated into the network form. At the other end the reverse double translation is seen. These extra format translation steps may be performed by hardware or software in the interface, and are likely to require an additional stage of buffering of the data. The end-to-end delay that the data experiences is therefore larger, and it is likely that the jitter will also increase.

The problems described above have consequences for the design of multimedia workstations. The network interface and interconnect need to be able to provide high bandwidth and be shared on a fine granularity. In addition, data buffering and transformation must be avoided. It is the thesis of this dissertation that an interconnect based on the Asynchronous Transfer Mode will meet these requirements.

4.3 Proposed Solutions

The importance of the machine–network interface has led to new designs being proposed for both the interface and its connection to the machine. To efficiently support

[1]The use of pixel tiles is also under investigation, but the same argument applies.

first and second generation multimedia applications requires that high bandwidth continuous media data can both be moved through the system without processor intervention and also be accessible to the processor. This rules out the solutions used in the Pandora [**Hopper90**] and Lancaster MNI [**Blair93**] systems, which totally exclude the high bandwidth traffic from the main machine. Surveying the research being done in this area reveals four classes of solution that are outlined below.

4.3.1 Efficient Network Interfaces

The most obvious approach is to design an efficient interface between the LAN and the internal interconnect. This is typified by the SCI-ATM interface [**Støa93**]. In these systems the hardware interprets the network protocol and translates data into the internal format. While the internal interconnect and the use of buffering may solve the multiplexing problem, the double translation is usually seen. In some cases the interface may include a protocol engine, but this still requires additional or new protocols to be moved from the network form via an intermediate generic format before final interpretation at the end-point. This may be seen in the SCI-ATM design, where the external format of the ATM cell containing 5+48 bytes is either treated as part of an AAL 5 block and changed into SCI 16 or 64 byte transfer units by repacking the cell payloads, or the entire cell is passed on as 53 bytes in a 64 byte transfer.

A variant on this approach has been proposed by the Multimedia group at Lancaster. They propose an internal interconnect which consists of a ring based single chip switch that transfers 32 bit mini-cells within the machine [**Lunn93**]. This LANC ("Local Area Network on a Chip") interconnect connects to an ATM interface via an optimised AAL 5 protocol engine. The multiplexing issue is faced through the use of very small data units, and the use of such small units reduces the data transformation to its simplest form.

4.3.2 A Special Multimedia Interconnect

Another approach is to use a special multimedia interconnect that is closely associated with the network interface. In this way the main interconnect is relieved from carrying the high speed traffic and the devices that need it may use a guaranteed interconnect. This allows both the double translation and multiplexing problems to be addressed. For example the SYMPHONY architecture uses two schedulable ATM buses in addition to the main system bus [**Bovopoulos93**]. On a more limited scale [**Sakata93**] uses a B-ISDN aware bus to connect the multimedia peripherals; this bus acts as a "peripheral" to the main machine.

This approach works well for first generation applications, where it behaves in a similar manner to the total exclusion method. Second generation applications require the multimedia data to be first class citizens; this forces all parts of the system to have a connection to the multimedia bus. The machine therefore ends up with two global interconnects, increasing both its complexity and cost.

4.3.3 Extend the Internal Interconnect into the LAN

A method of avoiding data format translation is to build a local area network based on the same techniques used in an internal interconnect. This is being done in the extensions to SCI for use with data acquisition systems, which are proposed by CERN (see section 2.1.1). The Atomic project at ISI is also following this path (see section 2.1.4), starting from the Caltech Mosaic interconnect. As a follow-on project [**Finn91**] suggests constructing multimedia systems using Mosaic and Atomic links; this is similar to the proposals in this dissertation and is discussed in more detail in the related work section (4.7).

4.3.4 Extend an ATM LAN into the Machine

A similar idea is to extend local area technology into a machine. This is the approach taken in this dissertation. The use of ATM is extended into the end system. The use of small fixed-size cells allows ease of multiplexing permitting the extension of the network guarantees to the real end-points of the communication.

4.4 The DAN Architecture

The Desk Area Network takes local area ATM techniques and experience and uses them to build an intra-machine interconnection network. [**Leslie83**] discussing the Universe project argues that the distinction between local and wide area networks had become largely spurious. In a world based on B–ISDN in the wide area, ATM LANs locally and DANs within machines the distinction becomes even more blurred. The main differences become the imposition of administrative and security boundaries, though it is also likely that differing *implementations* of the same basic architecture would be best for each scale.

A Desk Area Network based machine will be similar to current machines in having a number of processors, some memory modules, and a variety of peripherals. However, these will all be linked using an ATM interconnect rather than a more conventional bus. A multimedia workstation will look something like figure 4.2. Every device has to include an ATM interface onto the interconnect. At first this may seem very costly but, as the prototype DAN workstation described in the next section shows,

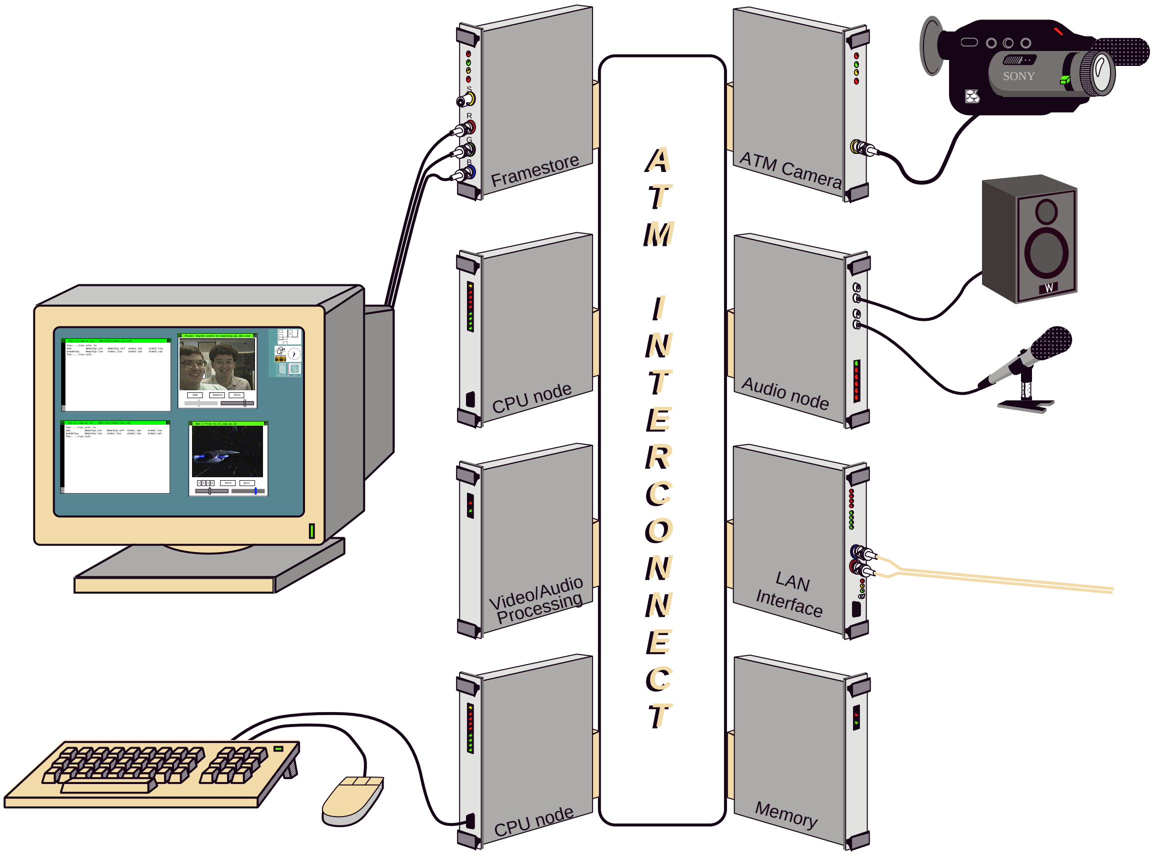

Figure 4.2: A DAN based Multimedia Workstation

this is no worse than having to have a bus interface on each peripheral card, and is directly comparable with interfacing to SCI or similar interconnects. This internal ATM connection does *not* include a full transmission system—just an interface to the interconnect. The interface can be designed with a specific task in mind and can therefore be as small as part of a Xilinx FPGA and two FIFOs. For example the camera interface can be designed for packing pixels into cells and has no need to support reassembly of disc blocks.

Using ATM as the multiplexing technique within the workstation has similar advantages to ATM on larger networks. It allows ease of management of multiple traffic classes, reduction of buffering and a simpler hardware implementation. In particular the small fixed cell size allows reassessment of the data contending for the interconnect to be both regular and frequent, reducing the chance of high priority data being delayed behind a large, less important transfer.

The data format transformation issue is removed at both the machine–LAN and the LAN–WAN boundary through the use of the ATM cell as the common data transfer unit. The only place where transformation between the natural form of the data and a different representation takes place is in the devices at the two end-points of the communication. Since these are designed for using cells this transformation may be done as a side effect of other format changes in the device. For example, in an audio capture card the CODEC just produces bytes of sampled data; packing these into cells has very little overhead, especially since they would be likely to be buffered whatever system were used.

In the wide area statistical multiplexing allows the network to have a high utilisation and still support quality of service guarantees. This clearly does not apply in the desk area, but here the operating system will have full knowledge of the data flows so management of the interconnect reduces to a scheduling problem. The control of a DAN interconnect as an operating system resource is for future study. It certainly seems feasible. Several other projects are also considering scheduling an interconnect. The AN2 local area ATM switch uses a combination of static allocation and dynamic computation to form a schedule for every cell time [**Anderson93**]. The SYMPHONY architecture for a multimedia workstation [**Bovopoulos93**], which uses the separate interconnect approach, also suggests scheduling the multimedia interconnect.

ATM is connection oriented. That is, before any communication can take place a connection must have been established between the end-points. A virtual circuit identifier uniquely identifies such a connection. This is also true within a DAN, so that whenever data is moved within the machine it is known what communication that data forms part of. Thus devices receiving data can tell, with no additional header information, where the data came from and what is supposed to be done with it. This can be particularly useful to allow sharing of more specialised devices. For example a signal processing engine may be shared by many streams that require manipulation. These may all have different operations performed on them, but the VCI is sufficient for the full context for a particular stream to be established before processing continues. Use of this technique will be seen many times in the description of the implemented DAN based machine.

The DAN is intended to be the internal interconnect of a machine, so the simplifying assumption can be made that it is reliable at the hardware layer. By this it is meant that when a source sends data either the data reaches the destination or the source is informed. It is the responsibility of the source to decide if a failed transmission should be retried or not. Clearly software buffering strategies may cause data to be discarded on its way up to higher levels, but data will not be lost or corrupted by the interconnect. This is a reasonable constraint because the technology used to create a DAN will be similar to that used in a standard workstation bus, so similar performance can be expected. A machine which loses bytes during some percentage of DMA transfers is a machine which needs to be sent for repair; similarly a DAN which loses cells should be repaired. In this respect the DAN differs from the normal view of a LAN, although protocols that assumed high reliability were regularly used on the original Cambridge Ring without problems; this was possible because of the low level acknowledgement protocol and the error rate of less than 1 bit in 10^{11} [**Needham93**].

The exact form of the ATM interconnect is not a concern of the DAN architecture. Just as rings [**Tennenhouse89**], crossbars [**Leslie91**, **Anderson93**] and buses [**Fraser92**] have all been used in ATM switches, so they may be used in a DAN machine. It is the transfer of fixed-sized cells and the asynchronous multiplexing that are the key features. The reliability requirement, which is effectively that the source must know when to retry a cell, will rule out a few switch architectures which silently drop cells internally. The use of a crossbar or other space division switch will provide for parallel data transfer, but it may well prove that a high speed ATM bus would provide a cost effective solution. Because the architecture allows this variation in implementation it is also unable to make any assumptions about being able to broadcast or multicast within the machine. In particular the ability to use techniques such as snoopy caches will be implementation dependent.

Routeing of cells to devices is the concern of the interconnect. Once there, the device must check the VCI to determine the validity of the cell. Since virtual circuits are set up all the way to the end-points, connection management is required. There are three classes of devices:

- "Dumb" devices are only able to manage data and an extremely simple control interface, such as the ability to load internal registers from a single cell message. Control messages flow on well known VCIs and require a manager on some other node of the DAN; this manager may be trusted by the device because they are both part of the same machine. The controller has responsibility for setting up the VCIs for data traffic. Since the manager is outside the data loop it is likely that the control functions can easily be performed by one of the processing engines without interfering with its other jobs.

- The second form of device has some local processing power which will enable it to perform some local management. Again the main control functions are performed by a remote node, but an RPC-like management loop is also possible.

- "Smart" devices have substantial processing power and are able to perform all management functions. Some will have the responsibility to act as proxies for interactions between "dumb" devices and the system.

The "network interface" of a DAN based machine is simply a cell router from the internal interconnect to the LAN. It is in fact very similar to a port controller on one of the LAN switches. Being at the machine–network interface it is also the barrier to the outside world and may need to implement security and data flow control. Within the machine, i.e. on the DAN, everything can be regarded as trusted and nodes can be relied not to violate their bandwidth allocation. The LAN is shared with others and more caution is needed.

Device control and reliability are the main areas for contrast between the DAN and projects such as the ORL Medusa [**Glauert93**], where devices are connected to an ATM LAN within an office. On the LAN a greater amount of processing power must be put into devices to manage control and security functions. An office with an ATM LAN switch connecting multimedia devices and a workstation appears very like a DAN based machine but the distinction is important. Indeed it may be that some of the DAN devices are not physically within the "box", but here they should be compared with SCSI devices on a traditional machine rather than LAN based devices.

The use of a DAN in a large multiprocessor machine has been considered, but in such a system the main communication is internal, either processor-to-processor or processor-to-memory. In this case the overheads of the DAN will be higher than for a specially designed interconnect, and unless such machines have a large amount of network traffic the specialised interconnect is likely to be better. The multimedia workstation class machine for which the DAN is intended will be a highly asymmetric multiprocessor machine. While there may be multiple processors which share memory the majority of communication is in the form of streams flowing between the various parts of the machine and the network.

4.5 The DAN Demonstrator

A multimedia workstation is being built based on a DAN. This is a demonstrator of the architecture and provides a test-bed for its exploration. To show the feasibility of DAN devices and to investigate their properties, a camera node and a DSP node have been built and a framestore is being designed. Other devices and processing power are provided by Fairisle Port Controllers. The next chapter describes the CPU and memory nodes developed for the DAN demonstrator. The devices that are connected to the prototype DAN cover all of the types mentioned above.

The demonstrator is based on the Fairisle switch technology and is therefore constrained to an 8 bit wide data path and a 20MHz clock rate. It seems likely that custom ICs and possibly a wider data path would be used in a full DAN implementation. The complexity of the hardware is comparable to or simpler than that required for implementing the SCI protocols for which a VLSI implementation has been produced in both GaAs [**NodeChip92**] and CMOS [**NodeChip93**].

4.5.1 Port Controller Devices

The Fairisle Port Controllers can be used as processing engines or devices on the DAN. In this role their memory size may be increased to 4Mbytes. Also, the FIQ code requires minor modifications to increase the maximum number of retries during cell transmission to provide the reliability required by a DAN. While the retry count should theoretically become infinite, in practice hardware does fail and it has been found best to abort the send after about 500 unsuccessful attempts.

Network interface nodes are provided by Port Controllers acting in their usual manner to forward cells from the local area onto the DAN. Processing may be provided using Port Controllers without a transmission system attached. The loop-back path, described in section 3.3, may be used by the processor section of the card to receive cells from the DAN, and the normal fabric insertion mechanism allows it to transmit.

Experimental ATM devices may be emulated by plugging cards into the podule expansion bus. These standard devices can then have an ATM interface provided by the Port Controller. The podule bus is a 16 bit wide expansion bus, for which a large range of devices have been produced. In particular the DAN workstation can be connected to the Ethernet using one of these podules. The podule bus is relatively slow and all data transferred between it and the DAN interconnect must move through the processor. This extra processing removes some of the benefits of end-to-end ATM to the device, and certainly the speed of the data stream is limited by the copying speed between the microprocessor section of the Port Controller and the buffer memory. However, use of these devices allows easy experimentation with different styles of device control without the need to produce specialised hardware.

4.5.2 The Camera Node

The "ATM camera" is a device which captures real-time video into an ATM cell stream. The initial version of the camera [**Pratt92**] takes RGB and sync inputs, digitises the video and outputs the cell stream with only a single line of buffering[2]. The second version [**Pratt93a**] can take composite video input and buffers several lines in order to produce a stream based on tiles of video.

The first camera falls into the dumb device class. A manager process running on one of the Port Controller processing nodes of the DAN sends the camera a control cell, which loads the timing and digitising parameters and causes single or multiple

[2]This device was implemented by Ian Pratt as a final year undergraduate project.

fields of video to be captured onto a specified VCI. Data from the control cell is latched directly into the registers on the device, and the only acknowledgement is the video stream being sent – in particular malformed control cells will cause strange results. The second version of the camera has a microcontroller which can interpret the control cell, and may support other management functions.

The management process may present an RPC-like interface to the client, or it may act in a mode whereby it emulates some Pandora Box functions and performs conversion on the outgoing video stream. This shows the flexibility of the dumb device approach; the hardware is only concerned with the data transfer path and can therefore be both simple and have high performance. The delegation of control path and management functions to software on a more powerful processing engine allows these to be made arbitrarily complex.

Clearly, the ATM camera concept could be extended to its logical conclusion by building the ATM interface into a CCD controller. This would have the benefit that extra transformations through the analogue domain could be avoided. In addition the format of the video would not be constrained to the scan-line based method which derives from the needs of CRTs. Pixel tiles are used in the version 2 camera because the 2 dimensional locality makes both compression and error recovery easier. With an ATM CCD such tiles could be directly read from the sensing array.

4.5.3 The DAN Audio/DSP Node

The audio node [**Atkinson93**] is designed both to provide audio I/O to the DAN workstation, and to be available as a general signal processing node[3]. A CODEC provides audio capture and replay at a variety of common rates. This feeds a DSP chip which connects through a Xilinx FPGA to the DAN. In addition, the Xilinx queues incoming cells for processing, and an optimised path is provided for cells to flow from the incoming queue via the processor to the output queue. The card falls into the second class of devices, since the DSP processor can perform some management functions. However, like the camera, the card is again optimised for the data path and most control functions are delegated.

4.5.4 The Framestore Node

The framestore node is currently being designed [**Pratt93b**]. It falls into the class of smart devices, having a powerful processor. Two access routes to the display will be provided: one via the processor is for graphics operations, probably using something similar to the X protocol; the second is for video streams which are directly written

[3] The Audio/DSP node was implemented by Ian Atkinson as a final year undergraduate project.

into windows on the screen. Here use is made of the VCI of the incoming stream to determine the destination window for the video. The (X,Y) offset in the frame of each block of pixels is sent in the data stream. In this way the absolute positioning information is kept locally but is rapidly determined by hardware as the cell arrives. The processor will have enough power to execute part of the distributed operating system of the machine (section 7.2).

4.6 Experience with the Demonstrator

Use of the DAN demonstrator has proved largely positive. The major problems encountered have been due to the speed of the interconnect which is entirely due to the use of conservative technology in its construction. In addition, the software and networking are in their infancy, often requiring ordered rebooting of a large number of machines to restart the system.

The framestore is still being designed so a DECStation 5000/25 workstation is currently used to provide the video output. A Yes board and transmission link are used to connect the machine as a node of the DAN. Although this potentially violates the reliability requirement of the DAN no problems have been experienced in practice. The DECStation runs the X Server software over the Wanda microkernel. The X Server includes the video extensions provided for Pandora, and a daemon running on the machine implements a software version of the Pandora Box using both Pandora and ATM camera video formats[4]. This allows interworking with the existing infrastructure in addition to the use of the higher quality video available from the new camera.

The devices that have been constructed clearly show that the ATM interface onto the DAN can be both simple and can avoid buffering. While a standard component solution could have been used the interfaces are small enough to be designed specifically for each device.

Separation of the data and control paths has proved very effective and makes it easy to provide multiple abstractions of an interface. In particular the ATM camera may be given both a Pandora compliant interface for interworking, and a generalised control interface for experimentation.

The use of ATM permitted the fine grained multiplexing required, and has ensured that the DAN based machines are well integrated with the local area network. For example the camera, which in version 1 does no data compression, can produce a single stream of data that has bursts which saturate a network link. When this data

[4]The ports of the X server and Pandora software were done by Paul Barham and Timothy Roscoe in the Computer Laboratory.

is passed through the DAN there is very little interference with the other data and control flows within the machine.

Use of the demonstrator has strengthened belief in the utility of ATM as the multiplexing technology for use both on the local area network and within the machine. The light-weight protocols developed for the local area have been shown to work well in the desk area. Discussions about protocols for use by devices on the DAN have suggested that these can be even simpler, for example by dropping the requirements for CRC checks.

4.7 Comparison with Related Work

The subject of network interfaces and end–system support for high speed networks and multimedia has had high prominence in the last year. Both the IEEE Journal on Selected Areas in Communications and the IEEE Network magazine have had special issues on this subject. Most proposals fall into the first two classes identified in section 4.3; either efficient interfaces or separate interconnects.

Two projects that are directly comparable with the DAN work are the ViewStation project at MIT [**Adam93a**] and the NetStation at ISI/USC [**Finn91**].

Like the DAN the ViewStation uses an ATM interconnect, being based on a high speed ATM switch fabric with a data rate of 700Mbit/s on each port. Network interfaces provide connections to 1Gbit/s links and, as in the DAN, it is only here that VCIs are remapped and a full transmission system used. The main device produced, called the VidBoard, is the equivalent of an ATM camera [**Adam93b**]. This is similar to the version 2 camera described above, and can perform simple manipulation in addition to just capture. A TURBOchannel interface to the fabric allows the use of DECStation 5000 and DEC Alpha AXP systems for processing power and to provide the display. This system is very similar to the DAN except for the much higher performance of the fabric, which was designed specifically for the project and is hence more suited to the workstation environment.

The NetStation is a development of the Atomic project described in section 2.1.4. It falls into the third category of proposed solutions, in extending an internal interconnect for use in the LAN. The Mosaic chips provide a switched point-to-point interconnect. By joining a workstation based on Mosaic links to the Atomic LAN the advantage of a common communication mechanism is seen, except in the rare case of communication being across the wide area. There is no need for data format transformation since variable sized packets are supported, so applications are free to use whatever size is appropriate. This use of variable sized packets may cause the scheduling problems traditionally associated with such transfers, but it is hoped to make use of multiple routes between nodes to perform load balancing at connection setup time.

Variable sized packets are proposed in the NetStation architecture to make the unit of communication used between nodes the same as the unit that the application (or device) finds natural to use. In this way the architecture potentially involves one less data transformation than the DAN, though as argued above for many devices packing into cells has very little overhead compared to packing into any other sort of buffer. Being based on the same design, the LAN also supports these variable length packets, so seamless communication is achieved. However, variable length packets have a penalty associated with them. Once transmission of a packet has started it will block other traffic until it completes. The duration of the blocking will also be variable so scheduling the interconnect becomes much harder. In the machine this problem can be alleviated by making the interconnect fast enough with respect to the maximum packet size so that the time between scheduling decisions is small. However, it is likely that CPU to memory accesses will be small (the optimum cache line size tends to be less than 64 bytes), so blocking the processor for longer than similarly sized data transfers is likely to degrade performance. It is also not clear that this solution can be applied in the local and wider areas. The use of ATM allows the use of a wide range of speeds and physical layers to build the network external to the machine.

One of the interesting contrasts between the DAN demonstrator and the NetStation is in the interface between the nodes and the interconnect. In the DAN demonstrator this has been individually designed for each node, allowing a simple but highly optimised connection for each device. The NetStation uses a Mosaic-C processor chip on each device. This both provides the interface and forms part of the interconnect routeing mechanism. In many ways the presence of a 14MIP processor in the interface for every device may prove useful, though the "Dumb" ATM devices, described earlier, show that the simple connection is often adequate. Both approaches have equal merit, and could be used in a DAN based system.

Chapter 5

Memory on the DAN

The Desk Area Network has been presented as an architecture that helps support the network communications required by multimedia workstations. In this chapter the use of the same architecture as a processor-to-memory interconnect is considered and the design of an experimental system to explore the idea is presented. A different view of the processor–memory–I/O hierarchy emerges, along with the idea of a "stream cache" which supports the second generation of multimedia applications.

5.1 Processor to Memory

Internal interconnects have traditionally been designed for processor-to-processor and processor-to-memory interconnection. On a DAN based machine the interconnect has been designed with the emphasis on machine-to-network communication. Such a machine could have both a memory interconnect and a DAN, as its I/O connector. This is similar to several proposals mentioned in the previous chapter which have both a "system" and a "multimedia" bus within a machine. However, to allow multimedia data to be treated as first class objects all devices would have to be connected to both interconnects. The management would be greatly complicated by this mechanism, for example care would have to be taken to ensure that conflicting updates were not issued simultaneously on the two interconnects. In the workstation class of machines having a single interconnect is preferable since the complexity and cost are reduced. The DAN has been demonstrated to have advantages for multimedia traffic. In this chapter its use as the single interconnect is investigated by considering processor-to-memory connections.

To study processor-memory links two components are required: a processor node and a memory node. For experimental purposes a Port Controller is used to provide the memory server and a dedicated CPU node has been designed and implemented.

Both the use of the Fairisle fabric to provide the ATM interconnect and the use of the Port Controller as the memory server will limit the system performance, but it will be argued that using purpose built hardware would result in a system with acceptable performance.

5.2 The DAN Memory-I/O Hierarchy

Traditionally the conceptual memory-I/O hierarchy found on a machine is similar to that shown in figure 5.1. Close to the processor, fast cache memory is used so that

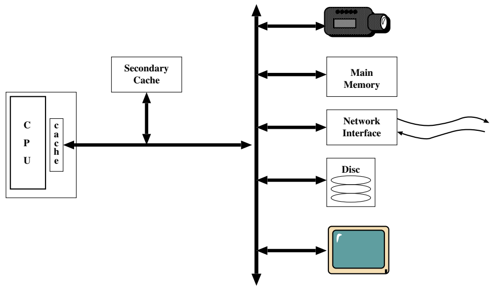

Figure 5.1: Standard Memory–I/O Hierarchy

data currently being processed may be retrieved rapidly. However I/O is treated differently. When a stream of data, for example a video clip, is to be processed it is put into main memory before being fetched by the CPU and a copy deposited in the cache. Frequently an entire sample of the data, for example a frame, has to be read before the block is available for any processing to be performed. In some machines the DMA process causes cache invalidation, in others the operating system has this task. Even in the case where memory and I/O devices are on the same bus they tend to be regarded as separate.

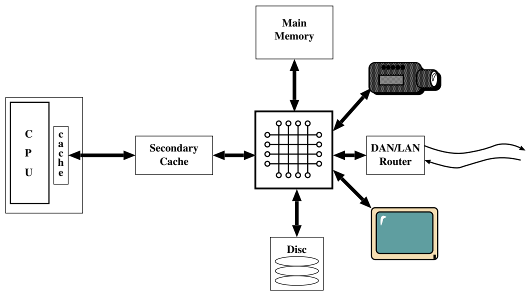

Figure 5.2: DAN Memory–I/O Hierarchy

The DAN memory system is somewhat different. The primary and secondary caches are still close to the CPU, but the cache controller is connected to the DAN interconnect. This enables I/O streams to be routed directly into the cache avoiding main memory entirely, which fits well with the whole philosophy of sending data directly to its end-point. It also has the advantage that the processor does not have to fetch the data from memory, so one level of buffering and one traversal of the interconnect is avoided. This is particularly important if general purpose processors are to be used on multimedia streams.

The use of the cache to assist with processing on a stream of data is developed in the next two sections. The problem falls into two parts, input and output. Both of these can benefit from stream use of the cache. The advantages gained by using these techniques provide strong support for using a CPU node designed along the lines suggested in this chapter.

5.2.1 Caching a Stream

When an incoming steam is cached the normal idea of temporal locality may still be applied. That is, the cache should hold the most recent data from the stream and new data should cause old to be thrown out. A section of the cache thus becomes a circular buffer holding the latest stream information. When the CPU tries to access this data there are three possible outcomes:

- The data has recently arrived in the stream. In this case the item is found in the cache and processing can continue.
- The data has not arrived yet. This is treated as a cache miss and the CPU can be halted until the required item arrives. If the data will not arrive for some (long) period of time the operating system may choose to reschedule the CPU in this case.
- The data is from longer into the past than the buffer can hold. This case should be flagged as an error to the operating system.

The first two cases have the cache behaving in a fairly standard way, though being updated rather than invalidated by the network traffic. The third is more complicated. The data from the stream has been discarded from the cache and is now lost – there was no main memory behind the cache entry. This is not considered a problem, in the case of a traditional data stream (for example coming from a fileserver) the process can generally cause a `rewind` operation, and in the case of a real-time stream the need to access the lost data suggests that processing is getting so far behind the stream that resynchronisation is required.

To allow an application to use a cached stream the operating system needs to present an abstraction of the low level behaviour. The stream data will need to be addressed in some way. Most streams will have some frame or temporal structure that the application will wish to use; this can be conveniently mapped onto a range of the process' virtual address space. Thus the process may access the stream as an array indexed by frame number.

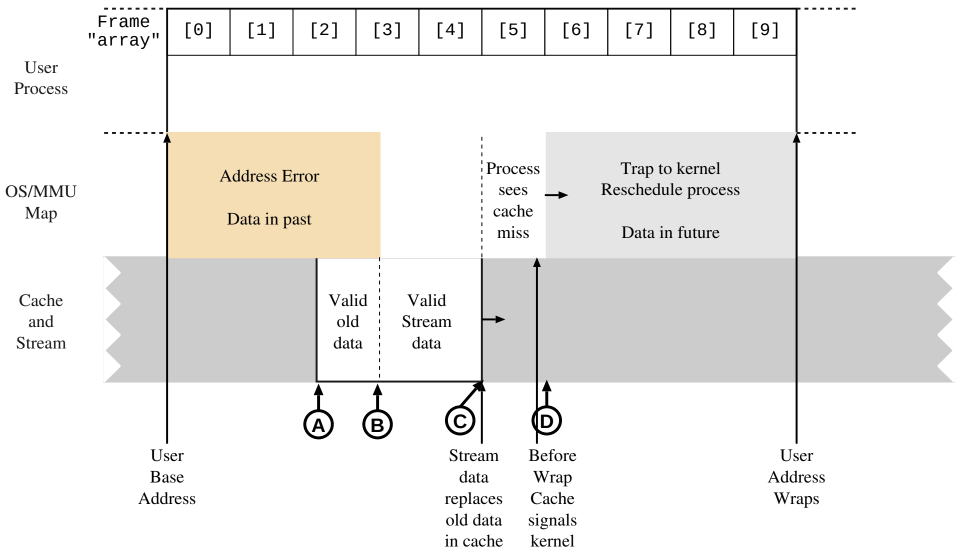

Figure 5.3: Addressing a Stream

Figure 5.3 illustrates how the address mapping may be done. User addresses will span some space, possibly having to wrap after a large time has passed. As shown, the operating system will set up the memory management unit to trap some of the range and allow other addresses to access the cache. Below point B accesses are trapped as being too far into the past to be in the cache, this boundary will move in discrete steps (probably some multiple of the page size) so some valid data will be inaccessible (from point A to B), this is the next data that will be overwritten. Addresses between points B and D are mapped through to the cache. The stream is arriving into this range, so for the lower addresses the processor sees a cache hit and retrieves the data rapidly. Above the stream arrival point (C) the processor is blocked by a cache miss until the data arrives. Past point D the system knows the data will be long enough arriving that it is worth trapping the process and rescheduling the CPU.

For this system to work the operating system must be informed by the cache when the stream is about to wrap to ensure the sliding virtual memory window is advanced. The size of the jump is one of the parameters that can be used to tune the system. If it is too big a lot of valid data becomes concealed, if too small the system will be constantly dealing with cache interrupts and moving the window. Similarly the application will need to know when to wrap round its virtual address range; this may be signalled by an event from the operating system, or be explicitly written into the program.

The second main problem is managing data loss. This is not a problem for streams originating on the same DAN, since the intra-DAN communication is reliable, but the majority of traffic will be from the lossy LAN. Loss of data that is *accessed* should be treated as an addressing exception, with the operating system and therefore user process being signalled. Loss of data that is not accessed need not be signalled (though event statistics may be useful), but the hardware has to be able to recover and ensure addressing consistency. For example when cell loss is noticed the corresponding cache line could be marked as "missing", resulting in a memory error being raised if it is accessed. The details of noticing cell loss depend on the SAR mechanism used on the stream, in the worst case buffering of a complete SAR block would be required – this could be done by some other DAN node, such as the network interface.

There are two major advantages to the stream cache system. Firstly, the data from the stream is placed where it is going to be used, namely in the CPU cache. Even if data is not accessed in strictly temporal order recent information will still be found in the cache. Secondly, the hardware manages the fine grained resynchronisation with the stream, imposing no overhead other than that necessary to wait for the data to arrive. If the window size and advance step parameters are carefully chosen there should be a large performance improvement over traditional stream reception systems. This may be compared with the technique of dallying in an interrupt routine or prior to rescheduling on a semaphore block.

5.2.2 Output from the Cache

A similar argument to that in the previous section applies in the case of stream output. It is quite likely that output streams will be built up in a random access fashion, and are therefore best formed in the CPU cache. It is commonly believed that network protocol code behaves badly with write-through caches because of the random bit field accesses that tend to be done when building headers.

As with the input case the application abstraction must be considered. When forming the outgoing stream there are two possible ways that information from frames that have already been sent may be used:

Reuse The program may wish to reuse the same data block that has just been sent, and build the next block over the top. Therefore only changes need to be written, with unwritten values being taken from the previous block. This may be particularly appealing for video applications where a picture is being built up from difference information.

Retain The previous block, or blocks, may need to be retained to allow use of their information in building up a later part of the stream. In this case the processor must be prevented from writing to blocks that have already been sent, otherwise it will alter its view of the past.

The second of these alternatives fits rather better with the addressing mechanism used for incoming streams. Indeed its use would allow an outgoing stream to be generated in place from an incoming one. It therefore seems preferable.

The mechanism used to send data on a stream will be at least partly implementation dependent. One conventional way would be to ignore the connection from the cache to the DAN and for the CPU to copy the data from the cache to a separate DAN interface. As will be described later in the chapter, this is the only mechanism available on the DAN demonstrator CPU node. This method has the disadvantage that the CPU needs to do an extra copy of the data, though as this is from the cache to a local transmission buffer it should be a fairly fast operation. Having the CPU involved in this way does allow both uses of previous blocks mentioned above.

A rather more satisfying method would be for the data to be transmitted from the cache without processor intervention, in much the same way as incoming stream data arrives. This could be done automatically by the system to achieve the desired outgoing stream characteristics; or, rather more practically, by the processor issuing an explicit send on the data block. If the processor were in control it could simply use the standard cache flush mechanism to cause the data to be sent, though the ability to mark a larger block for transmission would be preferable. Ideally once a block has been marked to be sent the CPU would be free to continue work. The implementation may require it to be halted, especially if the standard flush mechanism is used.

Operating system support via the MMU is again required both to ensure that the CPU may read but not alter data that it has sent, and to provide the address range for overlaying the stream. The address map structure will be something like that shown in figure 5.4. The same sort of sliding virtual memory window could be used as for the incoming case. Again accesses too far into the past (before point A) are trapped. The read-only region (A–C), protects history and contains the data that is currently being sent (point B which moves to meet C). In addition the operating system will need to keep track of the furthest point (D) into the future touched by the user process. There is only limited cache space so as new stream information is written some data from the past must be lost. When a new largest address access is trapped the system must compare it with the current send point plus the cache buffer size (point E). Accesses before this point will cause the window to slide, discarding old data. However, if the window were to move past this point then data that has not been sent would be overwritten corrupting the outgoing stream.

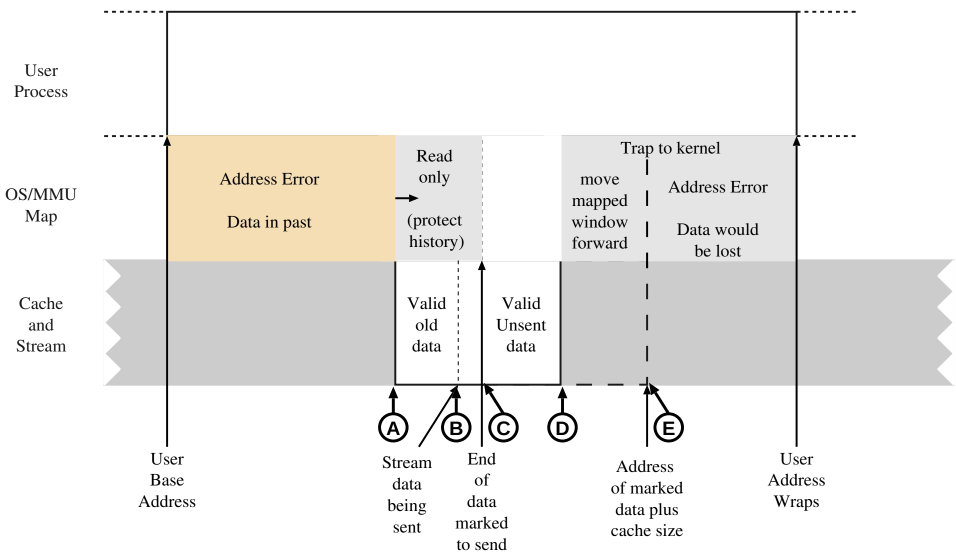

Figure 5.4: Addressing an Output Stream

These accesses need to be trapped and signalled as an exception. The process will inform the operating system when data is to be sent, so the read-only region can be extended.

5.3 The DAN Processor Node

To enable investigation of the processor-to-memory use of the DAN, a processor node was built that has all of its main memory served from a remote node. The processor node has only a small amount of local memory to enable it to boot and store the MMU page tables. This extreme view, forcing all memory to be remote, results in some problems but helps to highlight where the division between local and remote memory should be made in a real system.

The main functional blocks of the processor node are shown in figure 5.5. Although this implementation uses a particular processor, the general design should be applicable to a range of CPUs. The cache subsection manages both the secondary cache and interactions with the DAN, in designs where the processor directly supports the secondary cache this subsection becomes a lot simpler as it only has to interface the processor cache-miss bus protocol to the DAN. Whichever option is used one of the issues that will in general need to be faced by this logic is synchronising between the processor memory clock and the DAN clock. To enable consideration of this and to permit the processor and DAN sections to operate at whatever speed the final hardware supported, the node was designed to use asynchronous clocks. In retrospect this was a mistake; it adds nothing to the demonstration of the ideas of the processor node but adds significantly to the complexity of the logic design.

In particular, as noted in section 5.6, it caused a large number of subtle timing problems when adapting the hardware to investigate the stream cache ideas.

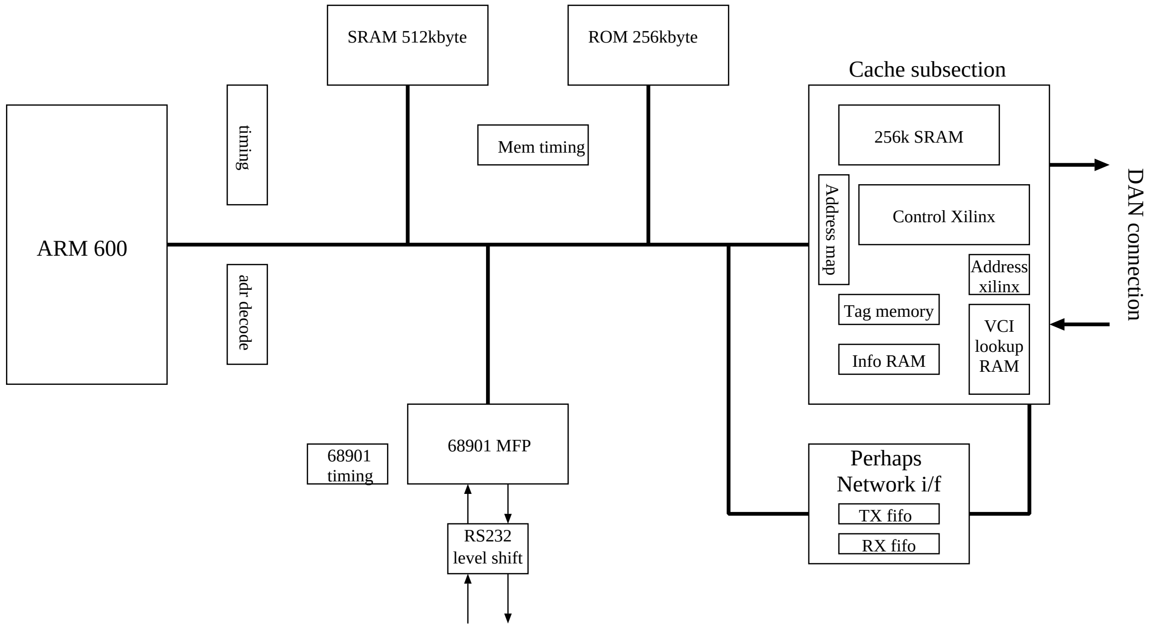

Figure 5.5: Overview of the DAN Processor Node

The DAN processor node is based on the ARM600 processor [**ARM92**]. This is a derivative of the ARM series of processors that are used in the Fairisle Port Controller, thus all the libraries and support tools existed at the start of the project. It uses the ARM6 macro cell with a 4kbyte primary cache, a write buffer, and an MMU.

The processor node initially runs the Wanda microkernel. As is argued later (section 7.2) to optimise the use of the DAN a specialised operating system is required, but it is likely this will have similar requirements to Wanda. The system requires a regular timer interrupt, mainly used to enable timeouts on semaphore and communication operations. In addition, especially for use during debugging, it is usual for the console to be available via a RS-232 serial line. To provide both of these a Motorola 68901 Multi-Function Peripheral chip has been put on the processor node [**Mot84**]. This device also provides additional timers, a parallel I/O port and interrupt management.

The card contains 256kbytes of EPROM to hold the bootstrap code and a simple monitor and test program. There is also a small amount of local RAM. When the card was being designed the size of this RAM provoked much discussion. The node is intended to make use of a remote memory server for all of its main memory so local RAM is only needed for booting. It was also thought that the MMU page tables should be local. The CPU will retain page table entries in its Translation Lookaside Buffer (TLB), but if the page tables were remote then a reference to a location neither in the TLB nor the cache could result in three cross DAN accesses to read the two levels of the page table and the data itself. I considered that the additional complexity of using DRAM was not warranted and provided 512kbytes of SRAM, using four byte-wide chips which match the EPROMs in pinout and therefore fit conveniently on the PCB. This size also allows a boot version of Wanda to run and is twice the size estimated for the set of MMU page tables needed by the running system.

As well as going through the cache the node has network access via a simple interface (the "perhaps" interface[1]). This is based on a single cell deep transmit FIFO and a single cell deep receive FIFO. This interface connects to the DAN using the same port as the cache; the "cache controller" arbitrates between them. Since there is a direct connection to the fabric the receive FIFO being not-empty can cause NACKs, preventing overflow. This interface was not designed to operate at high speed, and it is used for setting up connections to memory servers and for infrequent RPC operations.

5.4 The Cache

The main part of the processor node is the cache section. This provides access to the remote memory. The ATM cell size of 48 bytes led to a cache line of 32 bytes being chosen. The cell format for both request and reply cells is shown in figure 5.6.

ATM Header (6 bytes)			Payload (48 bytes)					
Fabric route tag	VCI	SAR Set for single cell message	Type	OP code	Adr from mem server	Adr to mem server	Spare	Cache Line Data
2 bytes	2 bytes	2 bytes	2 bytes	2 bytes	4 bytes	4 bytes	4bytes	32 bytes

Figure 5.6: A Cache Request or Reply Cell

The type field indicates that it is a memory cell and (for debugging reasons) which cache set is involved. The opcode indicates if the request from the CPU node is a read, a write, both or a reset. The reset cell causes the memory server to reinitialise the memory. It is sent either when an existing connection to the server is reused or when a new connection is opened. The cell format was deliberately made general to allow future extensions and to enable support for different memory types. Although

[1]The name may be found in the same thesaurus section as "Yes" and "Maybe".

essentially of a request–reply nature, the protocol allows a cache line to be flushed in the same cell used to request its replacement. This is only possible if both lines come from the same memory server, but since this is the common case it gives a significant reduction in total execution time.

Expansion of the cache to 512kbytes is permitted by the design, but there is only PCB space for 256kbytes. Studies of cache systems [**Hill88**, **Smith82**] have suggested that for such size the improvement in miss ratio of going from direct mapped to 2 way set associative is sufficiently small that it does not justify the increase in complexity of the cache. For this implementation of the DAN this argument is not valid since the cache line refill time is very large compared to the hit access time. Provided 2 way associativity does not require the insertion of extra processor cycles in every cache access, the small decrease in miss ratio will cause a large decrease in the average memory access time. Thus the cache was designed to be divided into two sets. Plug-in configuration links (which were replaced by switches during experimentation) allow the cache to be configured in different ways.

The cache subsystem has to work at processor speeds. Fast PALs and dedicated logic chips are used, especially for the critical timing of stopping the CPU when the cache is missed. In order to retain flexibility this fast logic is controlled by a Xilinx FPGA. This orchestrates the simpler devices; for example when a cache request cell needs to be generated the controller signals the VCI generating PAL, then the address generation system, and finally starts the data movement.

The processor has a pipelined address generation system. This gives the cache logic plenty of time to perform the parallel lookup and data selection required by an associative cache. The main problem to be faced by the control system is generation of the VCI for the request cell to a memory server. The VCI must to be derived from the memory address for which the cache miss occurred. Figure 5.7 indicates how the VCI is generated. A virtual address from the CPU is fed to the MMU which converts it into a form for the cache subsystem, decoded by having `10` as the top two bits. The rest of the address from the MMU is divided into five groups of bits:

Offset: This is fixed as the low 5 bits of the address and gives the byte offset into the 32 bytes of the cache line.

Line: The line number indicates at which address in the cache to look for the required data. The width of this field is varied, depending on how much cache memory is fitted and how many regions (see below) are required.

Tag: The cache tag is fixed at 7 bits and is used to determine if the data is in the cache at the selected line. The cache tag memory, which contains both SRAM and a comparator, is 8 bits wide and one bit is required to

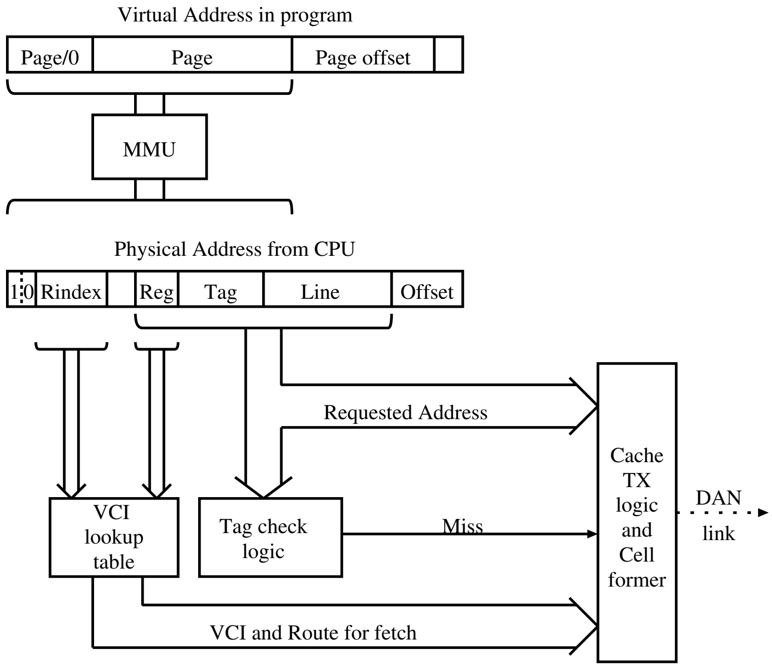

Figure 5.7: Address Translation to VCI

indicate the validity of the cache line. During experimentation some of the tag bits were linked to line bits, effectively reducing the size of the tag.

Region: The region is a field of up to three bits which is part of either the line or the tag. It is used to select which VCI to use to connect to a memory server. This permits a number of memory servers, or a number of distinct memory objects to be used. The region field *is* part of the address used to determine if an item is in the cache.

Rindex: In addition to the region, an extra 5 bits are used to determine the VCI to be used to access a memory server. These bits are *not* used by the cache - all rindex values for a given region are regarded as the same. The intention is that a region may be divided into portions with different access rights - indicated by different VCIs. The MMU is used to provide the rindex values and to perform a local check on permissions.

Cache regions may be used to force different areas of memory to use disjoint parts of the cache. For example the kernel and each user process can be effectively given its own cache. This design has the problem that the regions are of a fixed size of cache (the MMU can arrange for any size of virtual memory per region). This leads to a waste of cache space if only a few regions are used. Each region of the cache is associated with a single memory object in the server. To enable a single object to be broken down into several sub-objects the rindex bits are used. These play no part in determining a cache hit but allow a separate connection to the server to be used. Again the MMU is used to determine the logical address to rindex mapping. The region number and rindex bits form an index into a dual port memory which stores the VCI for the server connection.

The use of the rindex bits may be illustrated by an example: a region could be opened read-only and then a write lock could be taken out on a small section of it. The region would initially be mapped for reading by the MMU, using some rindex value. When the write lock is taken out (on a new connection) the locked pages will be marked read-write, the VCI table updated and a different rindex translation selected in the MMU for those pages. Any data in the CPU cache (accessed by logical address) will still be valid, and any data in the external cache will still be accessible since the address the cache sees does not include the rindex. When the write lock is released any dirty lines in the cache will be flushed, the second connection will be closed and the MMU translation will be returned to the original rindex.

The cache contains two sets. When a miss occurs the set used for the new data can be selected in one of three ways:

1. Always use set zero. This results in a direct mapped cache using only half of the memory.

2. Choose the set based on a bit of the tag. This gives a direct mapped cache using all of the memory, and is equivalent to adding an additional line bit to case (1) except that the address bit used for selection is not necessarily contiguous with the other line bits. In the case where all the cache memory is used the selection bit is contiguous, and this was the only way this setting was used during experimentation.

3. Select the least recently used set. This gives a two way set associative cache, with LRU replacement.

The memory server will have virtual circuits established with all clients, so sharing will be explicit. Upon detection of sharing the client systems may be informed, and they can take appropriate action. This suggests that the server may wish to send updates to clients - this area is still being considered. The system may be put into "flush" in which any dirty cache line touched will be flushed to the memory server.

5.5 Insights from Porting Wanda

The port of Wanda to the CPU node was relatively straightforward, since a version for the ARM600 already existed for the port controller. Apart from the Perhaps interface all the peripherals used were in the collection of previously written Wanda drivers. Initially Wanda was brought up in the local SRAM, which is large enough for the basic system to run. This preliminary work allowed the essential device drivers to be tested, but the main part of the port was to use the remote memory system[2].

One interesting aspect of the processor node is in booting the system. Traditionally a machine has to drag itself into life and obtain its system image, and it was initially expected that the same would be true of the processor node. It turned out to be somewhat simpler; all that is required is to open a connection to the memory server (which can be done with a minimal, single threaded protocol implementation) and jump to the first instruction of the kernel. At this point the hardware takes over – detecting a cache miss and suspending the processor until the memory server returns the instruction. The memory server control software, which is running on a node that has booted and is therefore in a much richer software environment, has the responsibility for fetching the processor node image from the boot server.

When Wanda was first run from the cached memory, with only the page tables and the stack used for kernel initialisation in local SRAM, everything went smoothly. The only problem seen was with the reception of network packets. The Perhaps interface has only a single cell deep receive FIFO (as was discussed earlier this has been known to cause problems for the CFR but was thought to be acceptable for the low bandwidth use expected from the CPU node). It was observed that the second cell of every packet was frequently being dropped. On reflection the explanation was obvious: when a “cell received” interrupt arrived for the first cell its service routine was executed. In many cases this routine had to be loaded into the cache from the remote memory server. While this was happening the first cell of the block was left in the receive FIFO preventing the second cell from being accepted, and eventually the network interface dropped this cell because it was causing head-of-line blocking.

The solution to the Perhaps reception problem, which could be cured by using a deeper FIFO, was to emulate a device with a larger reception buffer using the FIQ code. As on the Port Controller, any code invoked using a FIQ interrupt will run in parallel with, but with a higher priority than the Wanda system. A small FIQ routine was written to run in local SRAM which built a queue of incoming cells in the local memory. Whenever a cell arrived in the FIFO it was copied by the FIQ into this queue. Clearly if the processor was currently halted awaiting a cache line then

[2] Richard Black assisted with the port of Wanda to run from local SRAM.

running of the FIQ would be delayed, but it was always the next code executed. The FIQ posts a standard interrupt to the kernel whenever the cell queue is not empty. Obviously the double copy of the data that is needed here is something that should be avoided, but the limitation of the hardware is successfully hidden using this technique.

5.6 The Processor Node as a Stream Cache

In the discussion at the start of this chapter use of the "stream cache" was advocated for a DAN based machine. Data streams are permitted to flow directly into the CPU cache and avoid being copied through main memory. This idea was developed in parallel with implementation of the processor node. Consequently the use of stream caching was not initially supported. Modifications were made to allow a series of experiments with the new technique, though a reimplementation would be required to provide full support for both input and output streams. In particular the format of the cells in the stream needs to match that of a cache line cell.

For a stream to flow into the cache it is necessary to update the cache at times when the CPU is not stopped following a cache miss. The implementation only has a single-ported cache memory so the CPU must be halted if it attempts to access the cache while the new data is being inserted. The details of the design require that the CPU is halted for the entire duration of the cell arrival and not just for the eight cycles when data is actually written. Since most memory accesses will be to the cache this has a significant performance penalty.

The format of the stream cells is also imposed by the hardware. A more general stream cache would accept cells in a variety of AAL/SAR formats. This could be done in a relatively straightforward manner using a lookup memory accessed by the incoming VCI. The CPU node only has a lookup table accessible on the outgoing path so this method is not possible. To investigate the concepts of the stream cache the limitation of having to put a header in each cell is acceptable; and can be thought of as use of an AAL with a large per-cell overhead.

The main problem which arose in the modification of the hardware was that the processor and fabric clocks were not synchronised. While the node is operating in the normal manner the only synchronisation point occurs when the processor misses the cache and needs to exchange cells with the memory server. In this case the processor section is halted so there is no danger of incorrect interaction between the clocks. The arrival of a stream cell, on the other hand, is an asynchronous event and the state of the CPU subsection is uncertain. A large number of possible states of the CPU must be considered when responding to an arrival; missing some of them caused a lengthy debugging period during which the hardware behaved unpredictably when a stream cell arrived[3].

Outputting a stream from the CPU node is only possible by copying the data to the Perhaps interface. Using the standard cache flush mechanism, causing the CPU to be halted while each cell was sent, was considered, but this proved impossible. Two critical timing PALs were sufficiently full that no extension of the flush mechanism was possible – indeed some functionality had to be discarded when the PALs were modified to allow input streams to be cached. This section of the card needs to be redesigned, probably based on an AMD MACH 210 PAL [**AMD91**] which contains the equivalent logic to approximately 6 of the PALs currently used. Because of these problems the use of an output stream cache has not been considered in the experimental work presented in the next chapter.

[3]One of the bugs during development caused resetting of the **dirty** bit of random cache lines. While this was clearly an error it did not seem unreasonable that a stream flowing through the cache should cause things to be cleaned!

Chapter 6

Experimental Results

There are two primary goals behind the experimental programme described in this chapter. Firstly, to investigate and evaluate the practicality of using the DAN architecture for carrying memory requests as well as multimedia traffic; and secondly, to explore the advantages of having the closely coupled stream cache.

This chapter contains graphical results, supplemented by tables where needed to aid the discussion. The full numerical results may be found in Appendix A.

6.1 The Experimental Programme

To assess the usefulness of the DAN in carrying memory traffic it is first necessary to ensure that the demonstrator system behaves in a similar manner to a real system. Research into CPU caches is currently very active, with new designs and optimisations being suggested frequently. A sufficiently good match between the experiments described here and results reported in the literature will enable the conclusions of this large body of work to be applied to the design of future DAN based systems.

As was pointed out in section 2.4, the behaviour of the other system components can have as large an effect on system performance as the cache parameters. For this reason the performance of the memory servers and the interconnect are studied. Once the basic response has been noted some conflicting traffic streams, such as might be found on a DAN based machine, are introduced and their effects recorded. Coupled with the behaviour of the cache (and improved cache designs suggested by others) these results allow an assessment to be made of the effectiveness of a DAN memory system.

The final set of experiments involve the stream cache. First the impact that a stream flowing into the cache has on other tasks is studied. Then by using a simple demonstration program to receive a stream both the abstraction offered to the programmer and the system behaviour are examined.

6.2 Cache Performance

Figure 6.1 shows the system used for the initial cache performance measurements. The CPU node is connected to a DAN with two other nodes. One is a Port Controller running as a memory node and the other interfaces to the local area network.

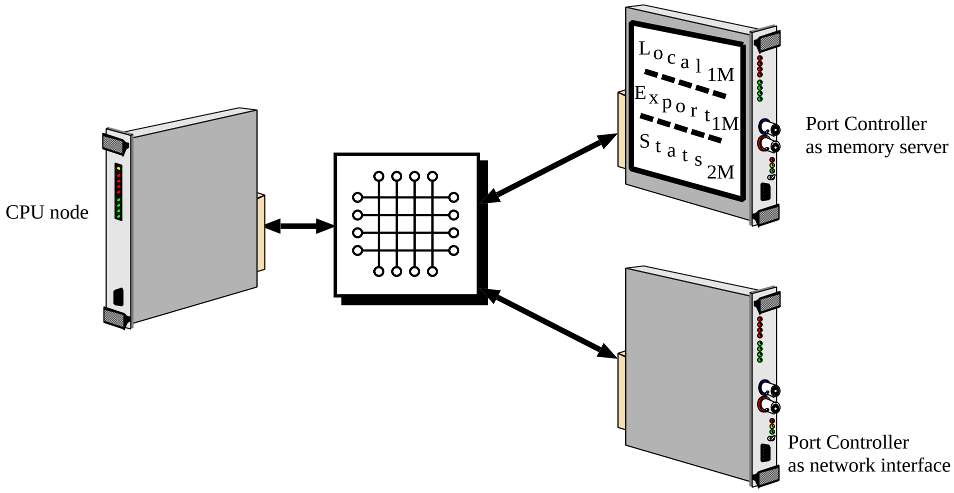

Figure 6.1: Hardware Configuration for Cache Performance Experiments

The memory server consists of a Wanda user-space process which both services request cells and maintains internal state to enable consistency checking of the requests. The Port Controller has 4Mbytes of memory, 1Mbyte of which is exported to the CPU node, 2Mbytes used to store memory access observations, and the remainder used by the local system. A configurable counter podule is connected to the expansion bus, to allow the server to record the number of accesses made to the secondary cache by the CPU node. The use of user space software with such extensive checking and recording leads to a high service time for a miss (about $400\mu s$) but this is of little consequence for the first series of experiments.

A bank of switches and a link replaced the cache size selection jumper on the CPU node. The switches select the number of lines active in each set of the cache. There are still two sets, but the link selects the replacement algorithm to use either one set, both sets LRU associative, or both sets direct mapped (i.e. the set depends on an address bit). This allows the cache size to be varied from 8kbytes to 256kbytes direct mapped, and between 16kbytes and 256kbytes set associative.

6.2.1 The PAL Compiler Example Program

The principal program which was executed on the system when taking cache performance measurements is based on the PAL compiler used in the hardware design of the CPU node. This is a good representative of a whole class of programs, in that it has an input-parse phase, an internal processing phase and an output phase. It is also sufficiently small that it and the kernel will run in the 1Mbyte of memory provided in the experimental system.

The PAL compiler[1] was run as a user-space process under the CPU node version of Wanda, and in the experiments the behaviour of the kernel was studied too. Since no other processes were running the only kernel activity was the timer interrupt and two network housekeeping threads which run infrequently.

The initial phase of the program consists of a flex and bison[2] parser which accepts a subset of the PLPL PAL description language [**AMD88**]. The input for the compiler is usually read from a file on the unix filesystem using the Wanda Dumb File System[3] (WDFS). Accessing the fileserver over the network tends to cause the behaviour of the program to be slightly unpredictable. There are too many layers of timeouts in the WDFS system, these often conflict and cause the the process to lock up for tens of seconds. To enable comparable results from a series of experiments the input was instead read from a large array in the initialised data segment.

The second stage of the compiler consists of a logic optimisation and fitting the logic into the sum of products form required for PALs. This consists of a walk of the parse tree during which the equations are rewritten.

The final stage of the compiler consists of a single pass of the tree in which the optimised equations are output in the Cambridge Hardware Description Language. The correct execution of the compiler can be confirmed by comparing this text file with that produced by the unix version. As with the input the WDFS is normally used to write the output to the unix filesystem, but during experiments it was written to an internal buffer (thus the output formating processing is performed even though the output is discarded).

---

[1] Derived from the "CBG Happy PAL Compiler" written by David Greaves.

[2] These are the GNU versions of the `lex` parser-generator and the `yacc` compiler-compiler.

[3] The WDFS was implemented by Shaw Chuang in the Computer Laboratory.

6.2.2 Experimental Measurements

The memory server runs as a user-space process, to which it was easy to add data gathering code. Information was dumped into a large memory buffer during an experiment and written out to a file on conclusion of the test.

A small change was made to the cache decode PAL on the CPU node to force a miss on every non-sequential secondary cache access. This allowed a full trace of every location (to the resolution of a cache line) touched by the example program to be taken. The trace data was suitable for using as input to the "cheetah" cache simulator[4] [**Sugumar91**, **Sugumar93a**, **Sugumar93b**]. This uses the trace to generate the miss ratios for a number of cache sizes and geometries under both LRU and optimal replacement.

After recording the full trace the decode PAL was restored to its standard behaviour of allowing data to be cached, greatly reducing the number of requests to the server, and allowing more information to be recorded about each request. For each miss the address, the time and the number of accesses since the previous miss are noted. In addition, signal points were added to the PAL compiler example program to indicate the boundaries between the phases of execution. A signal is made by accessing an address in the first 1kbytes of the memory. With 32 byte cache lines this gives 32 possible signals. The kernel was modified to use only signal zero, which is its entry point.

In the results that follow extensive use is made of the miss ratio. This is a good indicator of the behaviour of different configurations of the cache since no other timings change. The hit access time for the cache is fixed by the CPU as one "Idle" cycle during which the address is valid and the cache tags searched, followed by a single CPU cycle for each sequential location accessed. The use of the pipeline "Idle" cycle allows both a 2 way associative cache and a direct mapped cache to have the same latency. The miss access time is also the same in all cases, being fixed by the DAN and memory server speed. This time is examined in the next set of experiments.

[4] Thanks are due to the authors of this simulator who placed it in the public domain and provided me with the papers cited which describe the algorithms it uses. The only modification done was to provide a routine to read the trace format produced by the memory server.

6.2.3 Access Address

In order to understand the behaviour of the system the full set of addresses accessed was recorded. This gives a good guide to what is executing in the machine at any time. Figure 6.2 shows the address fetched for each of the misses during the execution of the kernel and example program using the smallest cache of 256 lines direct mapped. The three vertical lines indicate the changes between the phases of the example program that were described above. Using a dump of the kernel segment map the use of the various areas of memory was determined, and is shown on the right of the figure.

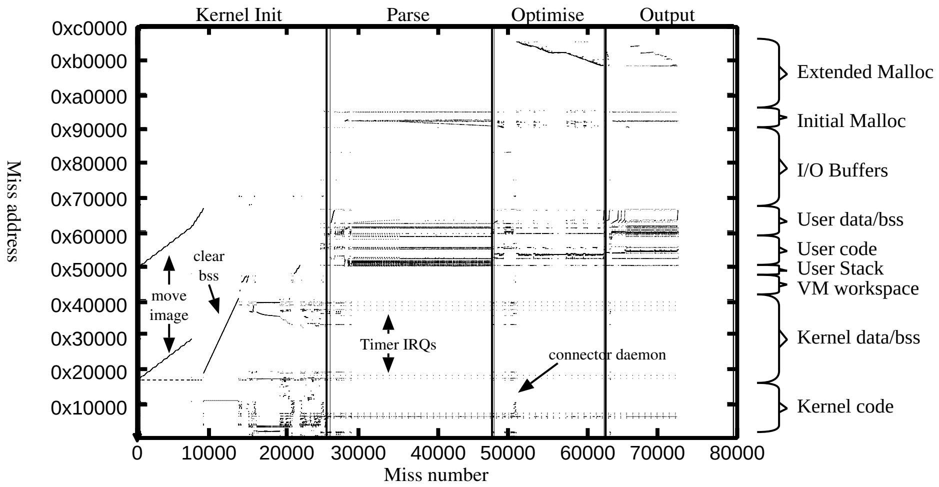

Figure 6.2: Address of Cache Miss vs Miss Number

The first 10000 misses show the kernel copying the user process to its execution address, and clearing both the user and kernel uninitialised data segments[5](`bss`). Then the kernel performs various initialisations before starting the user process. During the input stage of the compiler the parser code is frequently fetched. In the optimise phase the process has to expand its memory heap (`malloc`) space, and a new segment is seen in use. The kernel activity seen is the timer interrupt code, and the running of the network connection timeout thread at about miss number 50000.

This graph, and a related graph (which is not shown) plotting time rather than miss number on the X axis, is of interest to system designers because it shows where time is being spent, and areas where the code is thrashing the cache. For the next sets of experiments it is useful to note that once the user process has been started the majority of activity is related to it. The two kernel activities of timer interrupts and the network connection management daemon take relatively little time.

[5]Wanda defines that this segment will be zeroed before programs are started.

6.2.4 Results with No Primary Cache

For the first set of measurements the CPU primary cache was disabled forcing all accesses to be to the secondary cache. Graph 6.3 shows the results comparing the experimental values with the simulator result from a full trace. The results from the simulation are seen to indicate a higher miss ratio than those in the actual trace. This is due to the timer interrupt code. When the trace was taken, and a miss was being forced on every non-sequential access, the code ran a lot slower and hence the timer interrupts were relatively more frequent. Each run of the interrupt code will knock some user process lines out of the cache increasing the miss ratio.

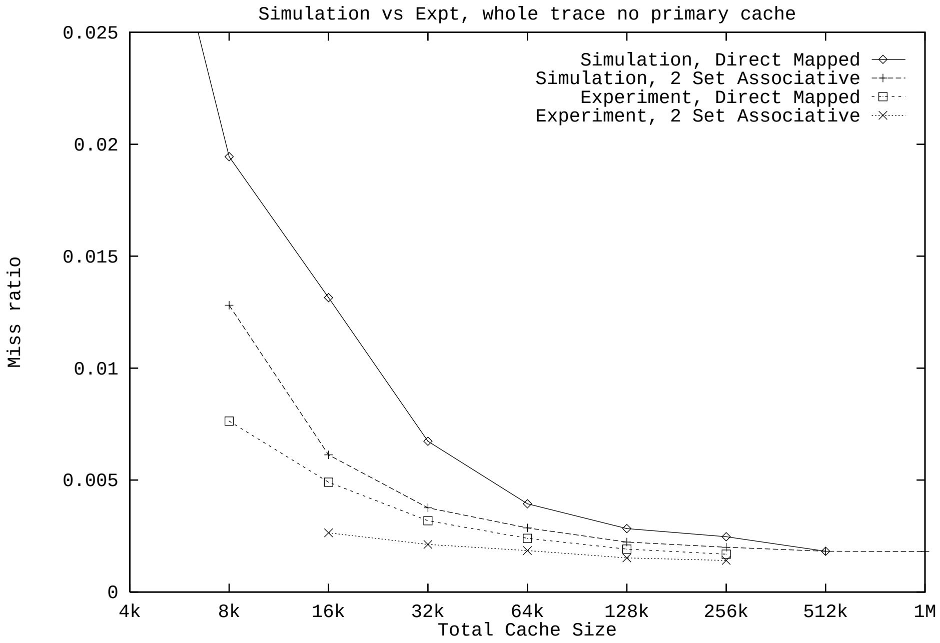

Figure 6.3: Miss Ratio with Primary Cache Disabled.

6.2.5 Results with Primary Cache

The CPU primary cache is a 4kbyte 64 way associative cache which uses random replacement, and is write-through. When enabled it is seen to miss on 9% – 14% of all accesses, so the number of accesses to the secondary cache is reduced. Table 6.1 shows some sample results for the various phases of the example program. The results show that for a direct mapped secondary cache the number of misses is reduced, as expected because data that is replaced in the secondary cache may be retained in the primary cache due to its greater associativity. However, for a 2 way associative second level cache the total number of misses changes much less; this reflects the finding reported by many researchers that moving from two way to greater associativity has only a small effect.

	No Primary		With Primary		No Primary		With Primary	
Program Section	Total accesses	Miss count	Total accesses	Miss count	Total accesses	Miss count	Total accesses	Miss count
	16K Direct Mapped				16K 2 way Associative			
Parse	477758	13297	96162	6783	476456	4176	95472	3922
Optimise	1048406	6894	106986	4372	1049005	3518	106860	3437
Output	429937	5958	82550	3480	429516	2182	82252	1963
	256K Direct Mapped				256K 2 way Associative			
Parse	478877	2255	94498	1974	478823	1135	94576	1133
Optimise	1051176	1752	106215	1744	1051185	1654	106221	1652
Output	429328	242	81491	195	428890	257	81430	259

Table 6.1: Sample Access and Miss Count Details

Figure 6.4 shows the miss ratio for various cache sizes both for the experimental capture and the simulation. In order to permit a fair comparison, accesses during kernel initialisation before the primary cache was enabled were ignored. Compared to the results with no primary cache (graph 6.3) there is a closer match between simulation and experiment. As before the difference is mainly due to the increased overhead of timer interrupts but hits on the primary cache decrease the running time, reducing this effect.

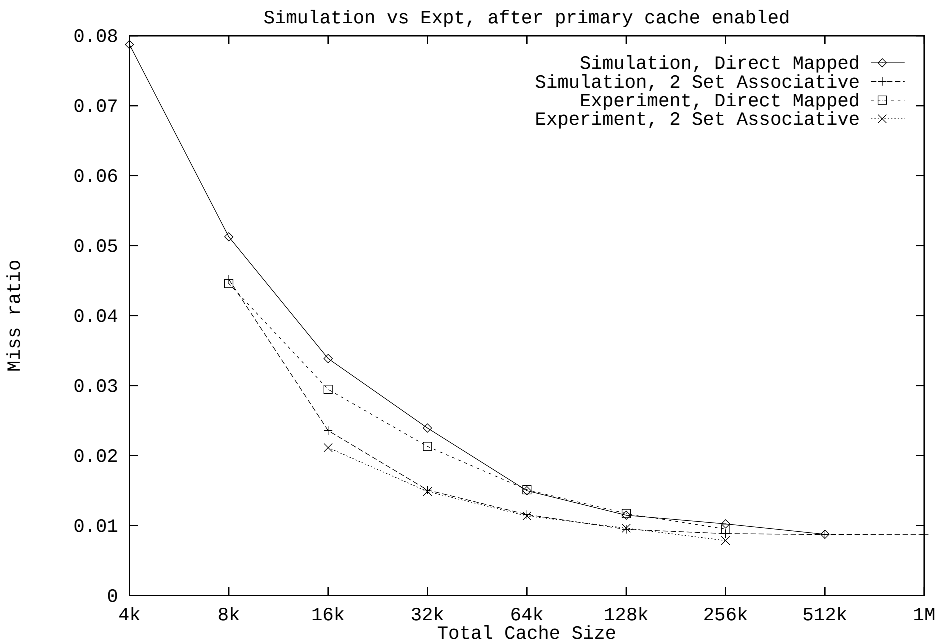

Figure 6.4: Miss Ratio with Primary Cache Enabled.

6.2.6 Simulation Results

Simulations of the cache from the captured traces are sufficiently close to the experimentally observed behaviour that it is reasonable to use the simulator to investigate cache geometries that cannot be achieved with the current hardware. Graph 6.5 shows some typical results. The graph shows miss ratios for direct mapped, 2 and 3 way set associative caches using LRU and optimal replacement algorithms. (As is expected optimal and LRU are identical for the direct mapped case where there is no choice in the line to discard.)

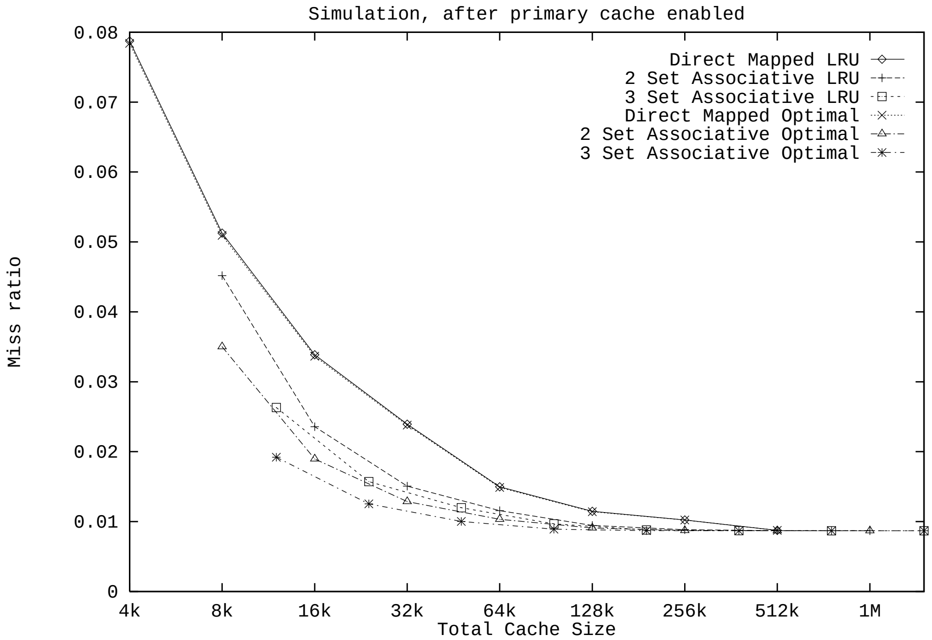

Figure 6.5: Miss Ratio Simulation for Optimal and LRU.

For changes in associativity the results are as expected. There is a fairly large gain for the move from direct mapped to two way associative and a somewhat smaller gain from a three way associative cache. As the cache gets larger the miss rates converge, reducing to about 0.8% with the primary cache enabled. It is also clear that while suboptimal for small caches, the LRU replacement algorithm is close to optimal for caches of 64kbytes and larger.

6.2.7 Comparison with Other Results

A recent study has used trace driven simulation with traces of several hundred million accesses [**Ewy93**]. It concentrated on the secondary cache parameters using a primary cache with a miss rate of 9.15%. Most of the reported results were for the SPARC architecture, but enough experiments were done on other systems to suggest that the results are valid for a range of RISC processors. The paper considers direct mapped and 4 way associative caches, and shows miss ratios that are larger for the 256kbyte cache and smaller for the 512kbyte cache than those found for the DAN CPU node. It seems probable that there are two factors influencing this – the size of the task may require a larger cache to stabilise the miss ratio and the greater length of the run which will remove startup distortions.

There is a closer match between the DAN results and the uniprocessor results reported in [**Agarwal93**]. While this paper describes a new cache design, which achieves results very close to those of a 2 way associative cache with similar hardware cost to a direct mapped one, the trace driven simulation results are given for other cache organisations as well. Both this and earlier work by the same group use the ATUM traces, captured by modified microcode on a VAX [**Agarwal86**]. This set of traces contains a variety of workloads and includes the operating system behaviour.

[**Smith87**], the reference paper on cache line sizes, includes simulations from traces taken from systems based on the Zilog Z8000, the DEC VAX 11/780, the Motorola 68000, the CDC 6400 and the IBM 370. Again the DAN results reported above match those in this paper fairly closely.

The trends from both the other work and the CPU node results are clear, indicating that after the primary cache has removed about 90% of memory accesses a large secondary cache can achieve a miss ratio of under 1%.

The match between the CPU node with the example software and results reported from more realistic traces gives some confidence that the behaviour of the demonstrator system will be closely matched by a full system in normal use. The simulation and experimental results show that changing from direct mapped to 2 way associative caches gives a reasonable return for caches of up to 256kbytes, and validates the design decision to use a 2 way associative because of the large line fetch time. The simulation results suggest that the LRU replacement algorithm is close to optimal for these sizes of cache, and that the gain of using a cache with an associativity greater than 2 is insignificant.

6.3 Performance of the DAN

A second set of measurements was made to determine the behaviour of the DAN interconnect when it is used for cache line fetches. Initially the cache line fetch time was recorded, using two versions of the software memory server and a hardware memory server. Then interfering traffic was introduced on the DAN interconnect and the effects it had on memory traffic were studied.

6.3.1 Modifications to Setup

To enable the measurement of the cache line fetch time the counter podule on the memory server Port Controller was reconfigured. The cache controller on the CPU node produces a "miss" signal which is active from the time the cache miss is detected until the processor is restarted. The counter podule is enabled by this signal to count from a reference clock at 10MHz. When the enable is removed a second counter is incremented and the primary count is latched. Thus both the number of misses and their total duration can be recorded. An estimate of the fetch time was obtained by recording the time of each fetch with the fine grained time of day clock ($1\mu s$ accuracy) and plotting the time between requests on a scatter graph. This gave about $400\mu s$, so for the number of misses seen in the previous results the primary counter had to be 32 bits wide.

6.3.2 Memory Servers Used

Two versions of the software memory server were used. The first was the user-space process that was used for debugging the CPU node and recording for the initial series of experiments. In addition to requiring all memory requests to pass through the protocol code to user-space, this server performs a lot of checking on each request. The code maintains a map of lines stored in the cache and checks the sanity of read and write requests against the size and organisation of the cache. This version of the code is therefore somewhat slow.

The second version of the software memory server makes use of the FIQ code that handles the forwarding of cells when the Port Controller is used in a switch. The same user-space server as mentioned above is used to accept the connection from the CPU node and initialise the memory with the appropriate boot image. Thereafter, control of the connection is passed to the FIQ by marking the VCI for special treatment. An incoming cache request cell is received into a cell buffer by the hardware and passed into the standard FIQ code. The mark on the VCI causes the FIQ to invoke some hand crafted assembler rather than placing the buffer on a queue. This code checks the first word of the cell to ensure that it is a valid memory read or flush-and-read request. If it is then the data is transferred or exchanged between memory and the cell buffer and the buffer is placed on the fabric transmission queue for rapid dispatch. This queue is maintained by software, but provided there is a hardware transmission slot available the next job of the FIQ will be to pass the cell to the hardware. Since the FIQ runs at top priority, and has been observed usually to be resident in the Port Controller on-chip CPU cache, this results in the fastest possible turnaround. Other types of cell are placed on the arrival queue for user-space processing, and control of the connection is returned to user-space.

A hardware memory server was made using the Null Port Controller[6]. As described in Chapter 3 this contains a bank of VRAM which is normally used as a temporary cell storage area and to contain the VCI remap tables. As a memory server the first half of each cell buffer is pre-formatted with the route, VCI and cache-cell header required, and the second half of the buffer contains the actual data. When a request cell arrives the write address from the incoming cell is used to select the buffer to which the data portion of the cell is written if the write-enable bit is set. The read address of the incoming cell is used to form the memory address of the cell to return. A standard Port Controller is used to manage the NPC.

The NPC memory server was never fully debugged. In addition, it only supports exporting a maximum of 128kbytes of memory. For both these reasons, and due to lack of support software on the CPU node, it was never used as part of a running system. However it did work sufficiently for memory accesses to be performed and the behaviour of the node to be recorded on a logic analyser. This enables it to be used for comparison in the next section.

6.3.3 DAN and Memory Server Performance

Measurements were taken of the mean cache-miss service time for four states of the CPU node. The first was taken for the operating system startup and execution of the PAL compiler; this test has approximately 38% of cache fetches also being flushes. The second measurement was taken some time later with the CPU node being "pinged"[7] at one second intervals for about a minute; this has about 20% flush-and-read requests. The final two tests were made with the cache being "thrashed" for twenty seconds, causing either continuous read requests or continuous flush-and-read requests. Each test was performed a number of times, giving repeatable results which are summarised in table 6.2. The fabric was clocked at 15MHz, giving a cell frame time of 4.2μs.

Server type	OS and PAL execution		Ping test		Read thrash		Write thrash	
	μs	frames	μs	frames	μs	frames	μs	frames
User-Space	421	100	415	98.8	413	98.3	373	88.8
FIQ	35.1	8.4	34.8	8.3	33.1	7.9	40.7	9.7
Hardware	—		—		$8.4 + x$	$2 + \delta$	$8.4 + x$	$2 + \delta$

Table 6.2: Mean Memory Service Time

The hardware tests were performed using the debug monitor to access memory,

[6]The redesign to support this was done by Gerard Smit and Daniel Gordon.

[7]Using the MSNL `mecho` program.

and by capturing the behaviour using a logic analyser. Using this setup it was not possible to accurately measure the resynchronisation times for stopping and restarting the CPU, hence the table indicates an additional time on top of that observed for the request–reply operation.

The "thrash" tests were performed by disabling interrupts on the CPU node and putting it into a loop referencing three locations that were coincident in the cache. If this loop is set just to read from the locations then every time there will be a miss because of the LRU replacement policy. Similarly, writing the locations will make the lines dirty and force both a flush and a read on every access. This allows the time for the two basic operations of "read cache line" and "flush and read cache line" to be measured. In the FIQ based server the large difference between the two is caused by the need to copy the data from the network section cell buffer into the Port Controller main memory. Access to the cell buffers is slow (observed at 700ns per word for an earlier version of the port controller [**Hayter92b**]) and the extra 7.6μs taken by the write basically consists of nine reads of the buffer (the address and 8 data words) and the eight writes to main memory. Interestingly, the user-space version is slower for reads than writes. This is because on a read it compares the current data from the cache line (which is sent in the cell whether it is to be written or not) with the data in memory for that line as a check on the dirty mechanism. The hardware version writes the data into memory as it arrives from the DAN and therefore does not show any difference between the two cases.

6.3.4 Comparison of Service Times

The service times observed are very slow compared to most real systems. There are two factors which cause this: the use of software for the memory server, and the speed of the interconnect. The hardware server experiment shows that it is possible to reduce the overhead to the two cell times required for a request cell followed by a reply cell. The use of request-reply with fixed-size cells will always impose a penalty over simple systems such as standard buses where the request consists of just presenting an address, or interconnects where a request packet is very small. However, this penalty is reduced by the ability to flush and request in the same cell.

Table 6.3 shows the read and flush times for some of the workstations used by the Systems Research Group in Cambridge[8]. Note that the write time in all of these is only the time taken for data to be written, for flush-and-read the two times should be added. The times shown are for 32 byte cache lines – for the DS 5000/25 which uses 16 byte lines this was calculated as a single setup delay followed by 8 word accesses.

[8] Unfortunately Hewlett-Packard was unwilling to release this information for its machines.

System description	CPU type	Read Line ns	Write Line ns	Source
DECStation 5000/25	R3000	960	800	[**DEC91b**]
DECStation 5000/240	R3000	640[9]	640	[**DEC91a**]
DEC 3000 Model 400	Alpha (21064)	225	225[10]	[**DEC92a**]
DEC 3000 Model 500	Alpha (21064)	200	200[10]	[**DEC92b**]

Table 6.3: Cache Line Service Times for Workstations

Clearly the times on these systems are much faster than those observed on the DAN demonstrator. However, the low end workstation is only just over eight times faster than the hardware memory server (four for flush-and-read); by moving to a 32 bit wide path through a higher clock rate switch fabric a speedup close to this could be achieved using the same level of technology as the demonstrator. The comparison with the Alpha workstations is a little unfair since these use a 32 *byte* wide memory bus and are aggressively optimised.

6.3.5 Service Time with Contention for the DAN

The DAN is intended to support real-time multimedia traffic in addition to the memory service traffic. Since it will be an important part of a real system the effect of competing streams on memory service time was studied. Because the 4×4 switch fabric used as the DAN is a full crossbar, the only way of generating competing cells is to send them to the same output of the switch as the memory server. The cells may be directed to the transmission line of the Port Controller, a process performed entirely in hardware, and then discarded. The traffic between the fabric and the transmission line will not affect the CPU which is acting as the memory server. Thus the only competition is for the output of the crossbar, as shown in figure 6.6.

Video between the ATM camera and a DECStation display node was routed from the network interface, through the DAN, and out via the memory server port. The ATM camera generates 288 bursts of 9 back to back cells for every field, then pauses during the fly-back period. This gives a peak rate of 54Mbits/s for 28% of the time. The CPU node service time measurements were then repeated. No change was noted in the average service time. This is due to the round robin scheduling used by the fabric in the presence of contention (section 3.2). With one competing stream the only way a cache line request cell could be delayed is if the previous cell sent to that switch output was also a cache request. Since the competing stream is frequently injecting cells and there will be at least a cache service time gap between requests,

[9] Pre-fetching reduces this to 320ns for the second 32 bytes of a 64 byte aligned block.

[10] Buffering in the memory control ASIC makes this value hard to determine.

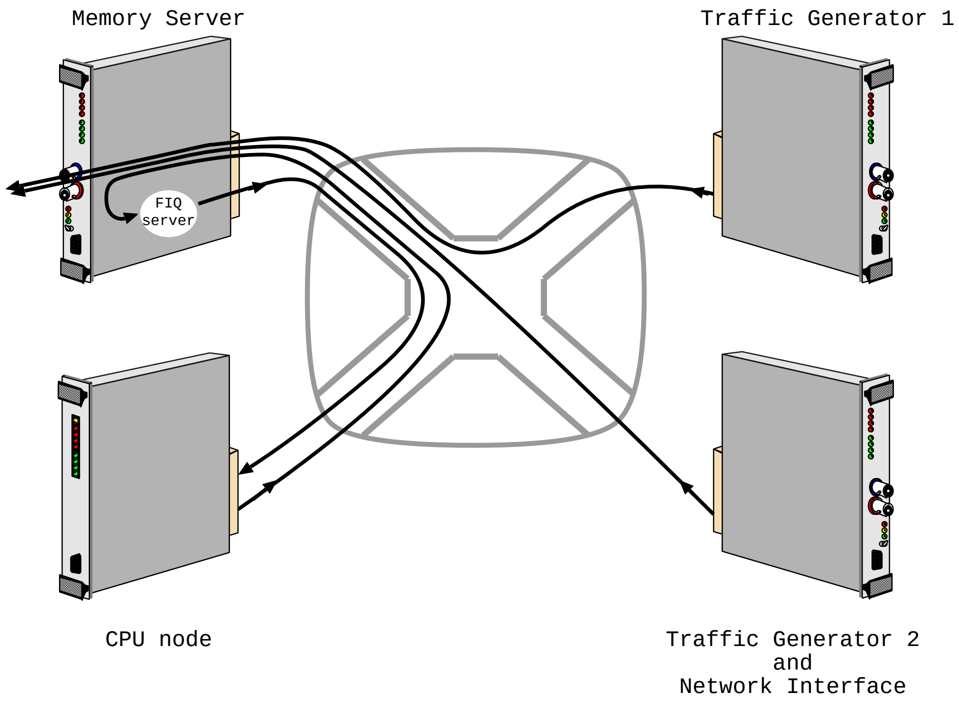

Figure 6.6: Hardware Configuration for Competing Traffic Experiments

the probability of two adjacent cache cells is very small. Thus it is highly likely that a cache cell will be accepted immediately.

In order to permit greater control over the interfering traffic, a traffic generator program was written for the Port Controller. Transmissions across the fabric are divided into cell times by the switch, which requires synchronous injection of cells. The generator can be configured to inject traffic with a specified pattern for a cycle of up to 32 cell periods. The networking code was completely replaced by the traffic generator, so the DAN had to be set up before the generators were booted and the program running on the CPU node needed to continue running for some time while the generator was started.

The test program used on the CPU node was a modified version of the "Dining Philosophers" problem, which is part of the standard Wanda test suite. This program, often used as an exercise in concurrency, models five philosophers having a meal of spaghetti. They sit at a round table with a bowl of spaghetti in the middle and a fork between each place. Each philosopher thinks for a random length of time before deciding to eat. Two forks are required to pick up the spaghetti, so the philosopher has to grab forks from both sides. The program normally terminates after each philosopher has eaten a certain amount, but for the tests it was set never to stop. All the philosopher threads run the same code, so in its basic form this program runs without causing any cache misses even in the smallest cache configuration. An extension was made to give each thread a local array, which was cleared after every thinking period - this successfully causes many cache misses.

Cache miss request cells were sent at low priority through the fabric, so they were granted the output using the round robin system with other low priority traffic and always lost to high priority traffic.

First the result found from the conflicting video traffic was confirmed by generating a single low priority conflict stream which requests every cell time. As expected this caused no effect on the average service time because of the round robin sharing.

A further experiment was then carried out using two interfering streams of low priority cells. The two sources injected traffic into the fabric based on a preset traffic pattern. In each fabric cell time they either injected a cell or remained idle. Clearly they will contend with each other leading to rejected cells. These were discarded and the pattern continued. The experiment aims to load the interconnect and its scheduling algorithm, so regularity is more useful than reliability. The results of the two streams on the cache service time are shown in table 6.4.

Pattern 1	Pattern 2	Cache Request Service Time	
1=cell sent, 0 = gap		μs	cell times
none	none	34.8	8.3
11111111	none	34.8	8.3
11111111	11111111	38.3	9.1
11111111	10101010	34.8	8.3
11111111	11101110	34.8	8.3
11111111	11111110	34.8	8.3

Table 6.4: Two Low Priority Interfering Streams

These results seem somewhat strange. Provided there is a gap in one of the streams the cache fetch time is unaffected, and when there is no gap it is increased by just under a cell time. With no gap the behaviour of the round robin schedule would be expected to increase the access time by half of one cell time: in the cell time before the request is first launched there will be a 50% chance of either stream being granted the output; if stream 1 was active then the request (effectively stream 3) will be delayed for 1 cell time while stream 2 sends a cell; but if stream 2 was active then the schedule will accept the cache request immediately. The observed delay is about 80% of a cell time. Both this increase and the lack of influence in the presence of gaps suggest that the cache requests are becoming synchronised in some way with the streams.

In order to help understand these results an experiment was performed using a single high priority interfering stream. The priority system ensures that this will always win over the cache request, simplifying the behaviour. The interference stream is generated as a burst of data followed by a gap. During the burst cells are injected into the fabric every cell time, and during the gap no cells are injected. The burst length was varied from 0 to 10 cells and the gap from 1 to 18 cells. With no gap the cache request is unable to get through the fabric so this case is uninteresting as the CPU node just stops. The results are shown in figure 6.7.

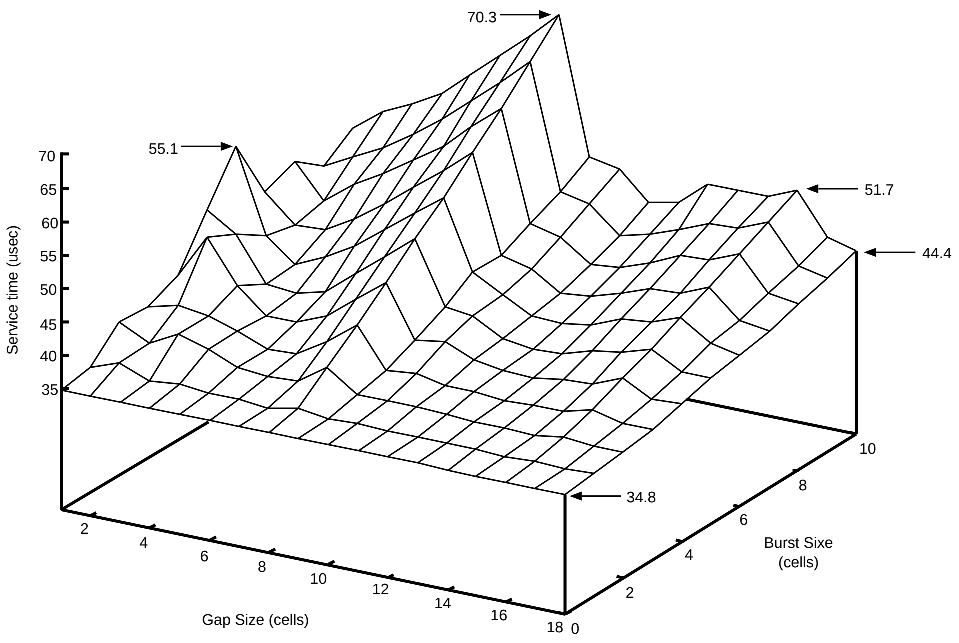

Figure 6.7: Service Time with High Priority Contention

To understand the shape of the graph it is important to note that 74% of accesses are reads with a service time of just under eight cell times and that a single cell time is very large compared to the CPU speed. Therefore eight cell times after a request there is very likely to be another request.

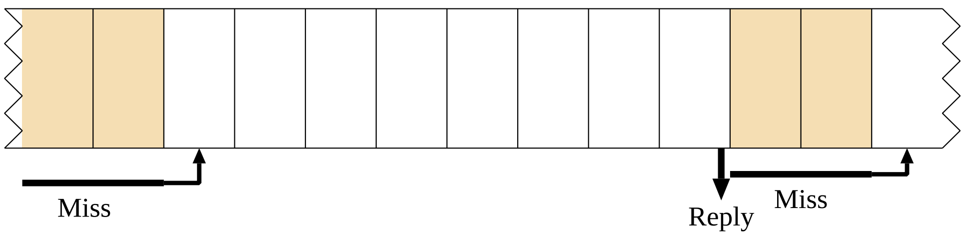

Figure 6.8: Effect of a Burst of 2 and a Gap of 8

So, as illustrated in figure 6.8, when the gap size is eight and a request succeeds in the first cell time of the gap a subsequent request will collide with the start of the next burst and be delayed. This results in the peak seen for a gap of 8. Similarly the smaller peak at a gap size of 16 where the next request succeeds but the third will be blocked. Indeed the extra delay of $36.5\mu s$ for a gap of eight and $16.9\mu s$ are very close to the expected 2:1 ratio which would be seen if requests always occurred immediately after replies resulting in every one being blocked with an 8 cell time gap and every other blocked with a 16 cell time gap.

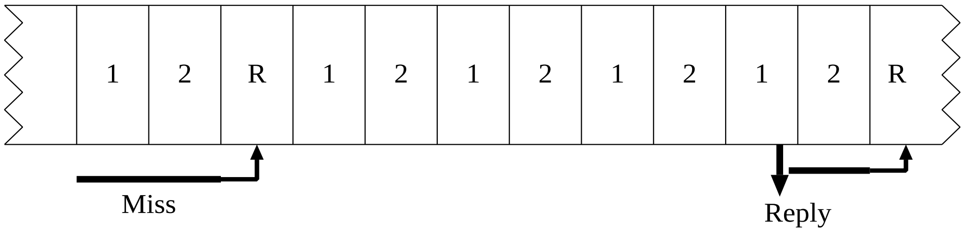

Figure 6.9: Cache Service Synchronisation

The same argument explains why the fetch time increased more than was expected in the case with two low priority streams. Recall that the half cell time delay was calculated on the premise that there was equal probability of either interfering stream being granted the fabric in the cell time prior to the request being injected. This will not be the case. As shown in figure 6.9, when a miss is detected the request cell succeeds in getting through the fabric a cycle after stream 2 has been granted. The round robin algorithm will accept stream 1 in the cell time after the request, and the streams will then alternate. The cache reply will arrive seven cycles later and the CPU will resume as soon as the data is available. Thus the most likely time for the next request is eight cell times after the previous one has succeeded. This coincides with the schedule moving to stream 2, so the request is highly likely to be delayed.

6.3.6 Comments on DAN Performance

The results show that, as expected, the cache line fetch time is influenced by the performance of the memory server, and the scheduling mechanism for the interconnect. They also show that the relationship between these is important. Figure 6.7 shows that the behaviour can vary wildly for relatively small changes in the traffic pattern in other streams.

Although somewhat incomplete, the experiments that were possible with the hardware implementation of the memory server have been useful. The prototype server requires that half of its memory is filled with pre-formated cache cell headers. An improved implementation of the server would generate the header information as the reply cell is sent, in the same way that the CPU node does when requesting. Since no reverse channel is set up through the switch, the minimum time for the memory request-reply is two cell times. The hardware implementation was shown to respond in this minimum time. In addition, it has been demonstrated that flushing the replaced cache line is achieved at no extra cost, reducing (but not eliminating) the overhead of a full ATM cell being required for each request.

Using a hardware memory server and a faster interconnect the performance of the memory server system would be good enough to be used in a practical workstation.

6.4 Using the Stream Cache

The final set of experiments was performed after the CPU node had been adapted to support the filling of the cache from a stream. To investigate this idea some simple programs were written to demonstrate the use of the stream cache – these were intended to demonstrate the concepts involved rather than to be of practical use, since in order to make full use of the ability to cache streams a large amount of operating system support is required.

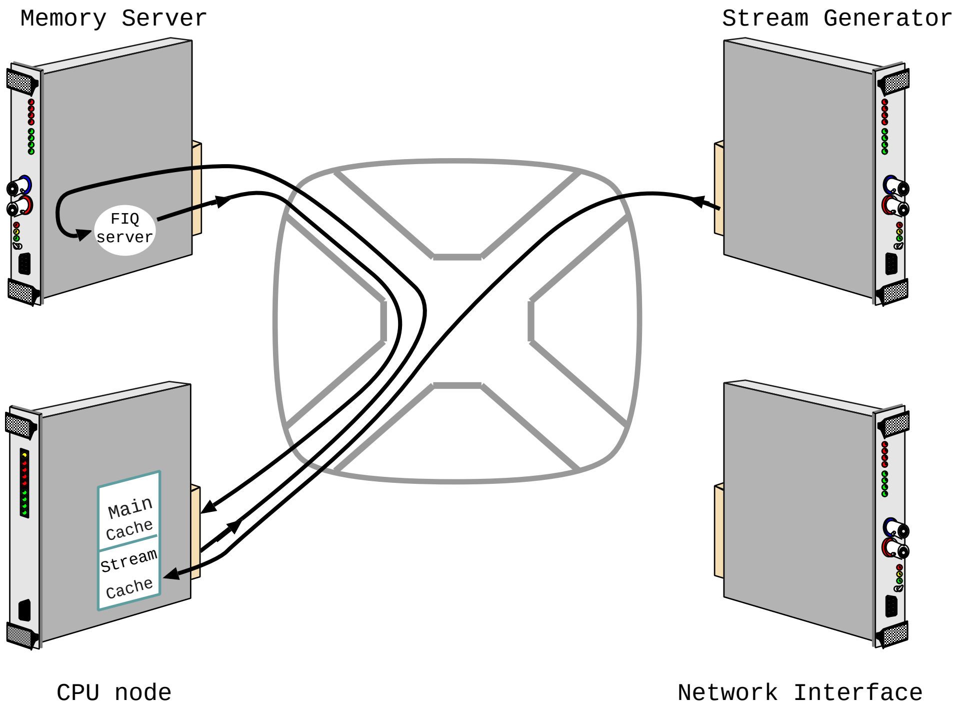

Figure 6.10: Hardware Configuration for Stream Experiments

The configuration of the DAN, shown in figure 6.10, was similar to that used in the previous set of experiments. The processor node had 256kbytes of cache, supporting two regions. Each region occupied 1Mbyte of the processor address space which was backed by 128kbytes of the cache. Region 0 was used to hold the Wanda kernel and user processes and region 1 was the circular stream buffer.

The version of the Wanda kernel running on the processor node was extended to provide minimal support for the stream cache. The region used for the stream was globally mapped into the address space seen by all user processes, as were the debug access addresses to the cache tags. Until this experiment the Wanda memory system had used 1Mbyte "sections" to map all memory. Use of these large units allowed all the translations to be retained in the CPU's TLB. In the modified kernel the stream region was allocated the full 2 level page tables, allowing translation to take place on pages of 4kbytes. A direct translation of the virtual addresses to cache region 1 addresses was set up, but initially all the pages were marked invalid. A very simple handler was added to provide a sliding VM window, along the lines suggested in section 5.2.1. The initial window is of zero length, as stream data arrives it expands to 128kbytes to match the cache size before it starts to slide through the address space. Since the hardware provides no signal of stream arrival the movement of the window is driven by accesses being made to the stream region. Because of this no attempt was made to integrate the fault handler with the scheduler to allow a full processor reschedule on a cache miss.

The MSNL connection establishment stack was modified to allow user connection requests to specify a Quality of Service type "Cache Stream" and the region for the stream to use. Connection requests of this type cause the kernel to select an appropriate VCI and configure the hardware to receive cells on that circuit into the specified cache region. Once the connection is established transmission works as usual, but data will only be received through the cache. However, a thread may issue the receive system call for the cache stream and will be blocked until an error occurs or the connection is closed.

The Port Controller used as the source of the stream runs the standard version of Wanda, with a user-space process generating the data. This process listens on a well known port for the CPU node to connect – no modification is needed to this end of the protocol stack. The user process has the responsibility of forming valid cache line cells to send on the stream; malformed cells are not detected by the receiving hardware and will cause 32 bytes from the payload to be written into the cache by forming an address from whatever bit pattern is found in the usual address field.

The memory server used is the FIQ based (faster) software server described above. One extension was made to enable it to recognise the range of addresses reserved for signalling cells and to pass these to the user process. In contrast to the case of cells which were not of type read or flush-and-read, control of the connection is retained by the FIQ. This is safe because the CPU node will be stalled until the user process replies, when control implicitly passes back to the FIQ.

6.4.1 Effects of the Stream on Other Processes

The first experiment evaluated the effect of a cached stream on other processes running on the CPU node. As was remarked in Chapter 5, the CPU must be halted for the entire arrival time of a cache cell rather than just the eight memory access cycles that might be expected. This effect is reduced by the presence of the CPU on-chip cache and write buffer, which allow processing to continue in many cases when the external cache is unavailable. In addition, the processor is permitted to access all local devices during arrival of the stream data.

The PAL Compiler example program was used, as before operating from and to memory buffers. The memory server was set to record the time from the high resolution ($1\mu s$) timer at each signal point. Thus the running time of each phase of the program could be measured. Since interrupts and background threads were also running each measurement was made a number of times with both the execution time and the number of cache misses recorded. The results given in table 6.5 do not include runs where the number of misses was exceptionally high, since this indicates other activity on the machine; inclusion of these would prevent comparisons between the different rate streams from being meaningful.

Characteristics of Stream	Stream Rate bits/sec	Parse ms	Process ms	Output ms	Total ms
No Stream	0	261	499	128	888
20 cells / 45ms	114k	261	499	128	888
200 cells / 57ms	900k	262	499	128	889
200 cells / 40ms	1.2M	262	500	129	892
1000 cells / 57ms	4.5M	264	503	134	901

Table 6.5: Pal Compiler Execution Time with Cache Stream

The results show that for streams up to 5Mbit/s the increase in running time of the example program is less than 2%.

6.4.2 Using the Stream – Spot the Ball

A demonstration was used to investigate the use of the stream cache. "Spot the Ball" receives a stream of frames and follows a "ball" as it moves around between successive frames. The "ball" consists of a 3x3 grid of pixels of a well defined value, other pixels in the frame are randomly coloured. A very simple algorithm is used for the search; it was specifically designed to show up some of the characteristics of the stream cache mechanism. The modified version of Wanda controlled the sliding VM window, but the user-space process was required to perform most maintenance tasks.

This application is clearly of no practical use, but it is closely related to tasks that might be performed in a multimedia machine. If a seminar were being relayed, for example, the machine could automatically track the speaker moving around the podium permitting both a centred view of the speaker (which would have to pan smoothly to avoid making the viewer sea-sick) and an unobstructed view of, say, the blackboard being pointed at. In this example the matching algorithm would be more complex but the principles remain the same.

The stream generator assembles the frames and transmits them a line at a time. Each frame is given a different random background, and between frames the ball is moved randomly in both dimensions. To allow checking by the searching application the location of the ball is recorded in the first two words of the frame. Each line is sent with a single WandaIPCSend system call. The implementation of the segmentation part of the protocol stack and its interaction with the FIQ make the most likely shape of the stream entering the fabric to be bursts of eight back-to-back cells.

“Spot the Ball”, running on the CPU node, defines a frame structure and treats the virtual memory range into which the stream is being received as an array of these frames. This is a very natural way to address the stream – it is simply indexed by frame number. Some care has to be taken to ensure that when the VM range is exhausted the addresses are adjusted. The incoming stream will simply wrap round and begin again at the start of the region, so all the receiving process needs to do is subtract the size of the region from the base address it uses for its “array”.

The algorithm used by the spotting process is to start from the location found for the ball in the previous frame and spiral outwards checking individual pixels until one of the correct colour is found. This is then used as a centre point about which the surrounding pixels are checked. Clearly this is inefficient, but it is very easy to follow the behaviour of the program and hence the system. Provided it is not trapped by the kernel for using an address outside the VM window, the system will stall if the data accessed has not arrived. For the purposes of the experiment, rather than stalling the CPU, it could be made to execute a loop in which it spins checking the cache tags for the arrival of the data and incrementing a counter. This count gives a measure of the time spent waiting for the data. It would have been preferable to have read a timer before and after the stall to get a calibrated time measurement, but the only timer available was only eight bits wide and this is of insufficient resolution to record both the normal short stalls and the occasional long ones.

The frame size was set to 64×64 with 32 bit pixels, giving 16kbytes per frame. The sliding VM window was thus set to move in 16kbyte blocks, causing a “future access” address fault to occur only the first time a frame was touched. The source was set to generate 10 frames per second, giving a data rate of 1Mbit/s. In one sampled frame the search followed the path shown in figure 6.11. The start point was (32,32) and the ball, shaded in the figure, had moved to (30,36). In each square the dally count of the first access to that location is shown. The delay waiting for the first location (marked 'X') includes the interframe gap, so is not comparable, and is not recorded.

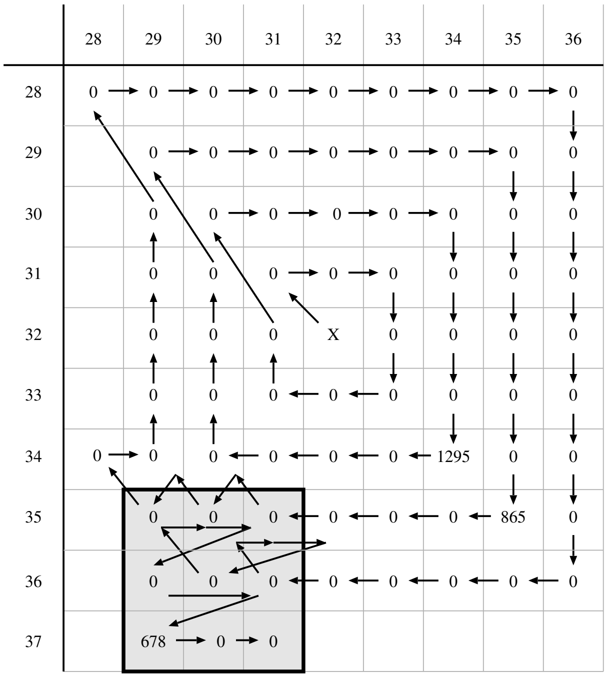

Figure 6.11: Time (in CPU loops) for Search Access

The results give a clear picture of the operation of the stream cache. All lines prior to the first location (i.e. with a y component less than 32) will have arrived before it so the access delays to pixels in those lines are zero. The time spent searching round the first turn of the spiral is sufficient for the data in the line below the 'X' to have arrived so the CPU does not stop. By the second turn of the spiral the CPU is beginning to get ahead of the stream and it is seen to stall for a while waiting for the lower right pixel (34,34). The same effect is seen on the third turn, but the stall at (35,35) is shorter because the CPU has done more work traversing the longer path. Just after this the ball is encountered for the first time and a defect of the algorithm becomes clear—it searches in locations that the ball cannot possibly occupy. This extra work and the fourth cycle round the spiral take long enough that the CPU is not stalled in the lower right access to (36,36). The ball is then discovered, with a final stall when checking for its lower left corner (29,37).

Once the ball has been located the program has no interest in the rest of the data for that frame and it will progress to the next frame. It is at this point that it is trapped for doing an access far enough into the future to warrant rescheduling the process. The simple handler just adjusts the sliding window and returns control to the user process dally loop. Despite this simplicity, control is given to the kernel and it is clear where an improved scheduler would fit in. The trapping of past accesses was also observed, by making an access more than eight frames into the past.

6.4.3 Comments on the Stream Cache

An adaptation of Eoin Hyden's JPEG decompression algorithm, mentioned in section 2.3, shows use for both stream input and output. The main work in the program is converting the data from the frequency to the spatial domain using the inverse discrete cosine transform. An input frame consists of a series of 8x8 blocks of frequency components which often contain a large number of zeroes. The algorithm first places the average luminance values into the output frame and then works on adding the higher frequency information starting with the most detailed blocks. If the CPU time available to the process is less than that required for all the frequency components to be decoded, this method ensures that a full frame is produced with the less detailed areas blurred. Importantly from the stream cache point of view is that both the input and output frames are randomly accessed, so both want to be in the CPU cache for maximum performance.

The use of the input stream cache has proved beneficial both in allowing a natural programming model to be used for multimedia data types and in keeping the processor closely synchronised with the incoming data. Many streams possess a natural frame structure and therefore easily fit into this model. Data that arrives in variable length blocks, such as compressed video, or in small frames will be harder to fit in to the sliding window model, though both could be done using fixed size memory blocks that are only partially filled. Even in these cases the ability of an application programmer to address the stream as a series of increasing memory locations is still powerful.

Chapter 7

Further Work

The investigation of the Desk Area Network architecture in this dissertation has centred on the behaviour of the hardware and the validation of the concepts. This chapter suggests some further work that should be done. With the completion of the second version of the camera and the framestore in the next few months several projects in the Computer Laboratory will begin using the demonstrator machine allowing additional feedback.

7.1 Extensions to the Existing Demonstrator

Several architectural features of the existing DAN demonstrator warrant further research. The main area where theoretical design work reported in this dissertation needs practical investigation is in the use of stream output from the cache. As was outlined in section 5.6 a major redesign of the cache subsystem on the processor node will be required to achieve this. This redesign could be extended to remove the requirement for the stream cells to use the cache cell format and allow general SAR blocks to be used.

A topic that has not been addressed is the protocols which should be used on the DAN, both for the control interface and for data transport. These have been created in an ad hoc manner for each of the DAN devices. Based on experience with the demonstrator, the feasibility of implementing a generally applicable protocol for device control should be considered. Protocols for the DAN will need to allow for synchronisation between streams from different devices. The topic of multimedia stream synchronisation is discussed by [**Sreenan92**].

In the original proposal of the architecture an ATM disc device was suggested. No further consideration has been made of this device, except for the observation that prototype hardware could easily be made by plugging a SCSI interface into a Port Controller podule slot. It is not clear what interface such a device should present. It is possibly the case that rather than an "ATM disc" the node should be an "ATM filesystem", and have sufficient CPU power to enable the lowest level storage service (or "byte segment custode" in the OPERA terminology [**Bacon93**]) to be implemented. [**Jardetzky92**] provides a discussion of fileserver design for continuous media.

7.2 An Operating System for the DAN

Both the architectural design of the DAN and the specific implementation suggest the need for operating system support to exploit the architectural features of a DAN based machine. There are three areas where this is required: device control, interconnect scheduling, and stream cache management. The "DANOS" project has recently started at the Computer Laboratory to consider these.

The DAN devices have benefited from separating management and control from the data flow part of the device. However, all the devices and their controllers are separate in the current system – there is no overall distributed operating system managing the whole machine. If such a system were implemented it would be running over a highly heterogeneous multiprocessor machine, with nodes of vastly different processing power. An operating system that can do this will be required to provide the support that applications seem likely to need. For example with a "smart" framestore device an application might want to arrange that one thread is executed on the framestore processor while the main program is running on a processor node.

Scheduling of the interconnect has been mentioned several times. The operating system will have knowledge of all the data flows in the machine, since it will be responsible for allocating the resources that they will use. It can use this information to compute a schedule for the interconnect to ensure all flows meet the quality of service offered to them. This will require an interconnect that can accept such a schedule.

Caching of streams, and the closeness of the connection between the cache and the network, is one of the main advantages of the DAN architecture. The use of regions of cache may reduce the need for cache flushes on context switches, will allow high priority processes to retain their cache between activations, and (though not in the demonstrator) allows the system to allocate an appropriate size of cache for the memory object being cached. The operating system needs to be aware of the behaviour of both ends of a cache connection, and in the case of a stream, needs to control addressing of the data by the user process. In particular the cache-aware operating system will be able to reschedule the processor in the case of a cache miss if it knows that the information will not be available for a large number of machine cycles.

7.3 A Higher Performance DAN

The existing DAN demonstrator has fairly low performance, principally due to the use of existing components and the avoidance of custom logic chips. One area for future consideration is the production of a higher performance machine.

The interconnect forms the heart of any DAN based machine. For the demonstrator the Fairisle fabric used carries data along a byte wide path at 20MHz clock rate. Experiments using version 3 of the Port Controller have shown that with no additional design effort the hardware can be run at higher speeds. (A Port Controller was seen to send cells through the fabric and loop-back path at 28MHz.)

There are two ways in which the interconnect performance could be improved: go faster, or go wider. The SCI and QuickRing systems show that with existing technology it is possible to drive links of similar widths to the existing DAN with the data clocked at up to 500MHz. The driver circuitry becomes more complex at these speeds but interface chips can be produced. Making a wider link may cause problems with the physical connectors needed for each device, but buses of 32 and 64 bits width are becoming common, and ATM switch designs based on a cell wide bus have been suggested. One of the penalties of a very wide connection is in its use within a device, where it is unlikely that more than 32 or 64 bits at a time will be useful. It would almost certainly prove easier to have an interface chip to multiplex/demultiplex from a 32 or 64 bit bus on the device to a fast narrower link (indeed the Dolphin SCI NodeChip does exactly this), than going from the same bus to a very wide link.

A higher performance interconnect would make the use of the DAN for processor-to-memory connections more appealing. A hardware memory server could use standard interleaving techniques to form a very high speed memory system, though the use of triple ported DRAMs as seen on the Null Port Controller server could prove as effective. It is by a technique similar to the triple ported memory that the Rambus DRAMs (RDRAMs) achieve their speed. A single address access pulls an entire row of the DRAM array into a “cache” formed from the sense amplifiers. From this “cache” data can be read almost as quickly as from a fast static RAM.

Together with the high performance memory system a higher performance CPU node could be made. The use of a processor which directly controls the secondary cache would greatly simplify the node. Direct CPU control is also likely to enable pipelining to improve the hit access time to the secondary cache. [**Ewy93**] shows that for large secondary caches the hit access time is of far greater importance in the performance of the machine than the access time for main memory. So tuning the speed of the secondary cache may well offset the relatively long access time across the DAN. It would also be interesting to see the effects of using a stream cache on a high performance processor such as the Alpha, where fairly complex manipulations are possible.

7.4 The DAN as an Application Platform

The Pegasus project will use the DAN as a platform to support the Nemesis multimedia kernel. The project aims to provide systems level support for the second generation multimedia applications for which the DAN provides hardware support. Results from the Pegasus project will enable the tradeoffs made in the DAN architecture to be more closely examined. No reports of the cache behaviour of multimedia applications has been seen, so an interesting project would be to do a comparison between these and existing traces.

The application which will be produced to demonstrate the Pegasus system is an "Automated Digital TV Director", this will control an audio/video conference [**Mullender92**]. The use of stream caches should greatly assist the video analysis in this application.

Earlier work on the architecture of a complete distributed multimedia system has led to the identification of a number of system requirements [**Nicolaou90**]. The DAN architecture solves many of the problems of "inappropriate workstation design" that were described. The Integrated Multimedia Applications Communication (IMAC) architecture should fit very well on a DAN based system; in particular there is a close match between the IMAC model of streams and the mechanisms provided in the DAN for data flows.

Chapter 8

Conclusion

The **Desk Area Network** has been presented as an architecture for a workstation to support the next generation of multimedia applications. A set of requirements was derived by considering existing multimedia systems and the expected behaviour of new applications. These may be summarised as: a need for a high bandwidth interconnect closely coupled with the network interface; use of a single interconnect to allow multimedia and traditional data to have the same status; sharing of the interconnect at a fine granularity; and avoiding buffering or transforming the format of the data. The thesis of this dissertation is that an interconnect based on the Asynchronous Transfer Mode will meet the requirements.

A demonstrator of the DAN architecture has been built and used to validate this thesis. It has shown that such a system is capable of supporting both multimedia and more traditional processor-to-memory traffic. To enable processing of data streams the "stream cache" was proposed. This is an extension of the secondary CPU cache to allow data from a stream to be placed directly into the cache, thereby avoiding unnecessary passage through main memory. It has been shown that the stream cache allows the processor to resynchronise to the incoming stream with very little overhead, and that the operating system may provide an application with a simple abstraction of the stream and reschedule the CPU when the process is awaiting data. The need for operating system support for the DAN was noted, but not addressed.

The performance of the demonstrator is limited by the technology used in its production. Consideration of more aggressive designs used in related work lead to the conclusion that a DAN can be used to produce a high performance system. Similarly, given a more powerful processor the somewhat contrived example programs could be replaced with useful multimedia manipulation applications.

The presentation and processing of multimedia data on personal workstations will grow as high speed network access becomes more common. To support this growth, novel architectures such as the DAN will need to be adopted.

Appendix A

Tables of Results

This chapter contains tables giving the results of the experiments described in Chapter 6. Full discussion of the results, along with a graphical presentation, may be found in that chapter.

A.1 Cache Behaviour

The first set of results are for the experiments described in section 6.2. The cumulative secondary cache access, miss and miss-with-flush counts are given for the signal points at the transitions between phases of the PAL compiler program. Table A.1 gives the results with the primary cache disabled and table A.2 the results when it is enabled.

PAL compiler event	Access Number	Miss count	Flush count	Access Number	Miss count	Flush count
	8k Direct Mapped					
start	2799104	25642	12152			
parse end	3276604	47757	16420			
optimize end	4325666	62571	19387			
end	4755781	72593	20668			
	16k Direct Mapped			16k 2 way Associative		
start	2836923	20876	10762	2977921	16198	10024
parse end	3314681	34173	13505	3454377	20374	11171
optimize end	4363087	41067	15757	4503382	23892	12797
end	4793024	47025	16624	4932898	26074	13194
	32k Direct Mapped			32k 2 way Associative		
start	2972586	16390	9967	2998258	15009	9607
parse end	3449828	24575	11978	3474842	17200	10448
optimize end	4497734	28897	13734	4524150	19835	11822
end	4927201	31407	14217	4953346	21038	12352
	64k Direct Mapped			64k 2 way Associative		
start	3021579	14616	8991	3029468	13923	8547
parse end	3498598	19754	10760	3507653	15592	9284
optimize end	4548471	22757	12176	4557527	17807	10331
end	4977677	23892	12723	4986825	18502	10834
	128k Direct Mapped			128k 2 way Associative		
start	3044459	13355	6628	3046903	11337	6622
parse end	3521890	15864	7554	3525336	12893	7622
optimize end	4570820	18401	9064	4576211	14800	8682
end	5000135	19165	9437	5005557	15251	8974
	256k Direct Mapped			256k 2 way Associative		
start	3102220	12872	2770	2968158	10863	2401
parse end	3581097	15127	3817	3446981	11998	3337
optimize end	4632273	16879	5350	4498166	13652	4521
end	5061601	17121	5427	4927056	13909	4756

Table A.1: Results With No Primary Cache

PAL compiler event	Access Number	Miss count	Flush count	Access Number	Miss count	Flush count
	8k Direct Mapped					
start	984277	21072	11514			
end parse	1082262	33678	14272			
end optimize	1190226	40035	16481			
end	1273823	45224	17085			
	16k Direct Mapped			16k 2 way Associative		
start	987470	18121	10481	991561	15981	9955
end parse	1083632	24904	12250	1087033	19903	11072
end optimize	1190618	29276	14103	1193893	23340	12709
end	1273168	32756	14571	1276145	25303	13078
	32k Direct Mapped			32k 2 way Associative		
start	989275	15981	9850	994882	14993	9604
end parse	1084973	20918	10983	1089886	17144	10441
end optimize	1190922	24125	12550	1196551	19782	11821
end	1273410	26205	13038	1278665	20961	12353
	64k Direct Mapped			64k 2 set Associative		
start	992315	14273	8852	994386	13912	8543
end parse	1087447	17323	9843	1089441	15582	9287
end optimize	1193832	19923	11185	1196179	17804	10341
end	1275554	20932	11726	1277862	18499	10836
	128k Direct Mapped			128k 2 set Associative		
start	992700	13261	6576	995018	11339	6631
end parse	1087552	15497	7499	1089524	12888	7625
end optimize	1193732	17721	8961	1195857	14793	8683
end	1275767	18392	9335	1277553	15243	8973
	256k Direct Mapped			256k 2 way Associative		
start	993980	12858	2760	993918	10862	2403
end parse	1088478	14832	3798	1088494	11995	3337
end optimize	1194693	16576	5331	1194715	13647	4521
end	1276184	16771	5402	1276145	13906	4757

Table A.2: Results with Primary Cache

The next set of results presented are the miss ratios produced by the cache simulator for the trace data. The results with the primary cache disabled were obtained from considering the whole trace. The other results come from considering accesses after the primary cache has been enabled.

Lines per Set	Direct Mapped	2 Set LRU	3 Set LRU	4 Set LRU
128	0.037237	0.012810	0.006638	0.004590
256	0.019442	0.006127	0.003893	0.003301
512	0.013151	0.003766	0.002968	0.002701
1024	0.006736	0.002864	0.002465	0.002244
2048	0.003942	0.002233	0.002085	0.002006
4096	0.002833	0.002001	0.001837	0.001816
8192	0.002474	0.001821	0.001815	0.001815
16384	0.001820	0.001815	0.001815	0.001815

Table A.3: Simulation, Whole Trace, Primary Cache Disabled

Lines per Set	Direct Mapped	2 Set LRU	3 Set LRU	4 Set LRU
128	0.078744	0.045178	0.026309	0.018211
256	0.051260	0.023580	0.015718	0.013380
512	0.033873	0.015077	0.011978	0.010715
1024	0.023935	0.011559	0.009690	0.009047
2048	0.014976	0.009445	0.008807	0.008712
4096	0.011463	0.008837	0.008708	0.008698
8192	0.010242	0.008701	0.008698	0.008698
16384	0.008744	0.008698	0.008698	0.008698

Table A.4: Simulation, Trace after Primary Cache Enabled, LRU Replacement

Lines per Set	Direct Mapped	2 Set OPT	3 Set OPT	4 Set OPT
128	0.078395	0.035000	0.019203	0.013866
256	0.050942	0.018961	0.012539	0.010657
512	0.033655	0.012860	0.010028	0.009132
1024	0.023803	0.010321	0.008920	0.008717
2048	0.014882	0.009082	0.008710	0.008698
4096	0.011414	0.008735	0.008699	0.008698
8192	0.010233	0.008699	0.008698	0.008698
16384	0.008744	0.008698	0.008698	0.008698

Table A.5: Simulation, Trace after Primary Cache Enabled, Optimal Replacement

A.2 Conflicting Traffic

Table A.6 gives the cache line service time for high priority contending traffic. This experiment is described in section 6.3.5.

Gap	Burst Size										
Size	0	1	2	3	4	5	6	7	8	9	10
1	348	356	395	391	414	484	551	458	477	444	473
2	348	372	374	402	480	458	428	417	427	465	506
3	348	353	397	396	416	392	394	419	460	489	526
4	348	358	383	383	378	385	415	444	485	518	552
5	348	353	365	366	378	400	441	476	514	548	588
6	348	354	360	367	398	436	473	511	545	585	625
7	348	350	362	394	431	469	507	543	582	621	663
8	348	359	392	428	467	504	540	579	619	661	703
9	348	351	361	371	389	411	439	436	457	477	503
10	348	353	361	374	396	406	411	425	445	469	494
11	348	351	360	366	378	381	389	397	415	431	453
12	348	351	358	365	371	376	386	400	419	441	462
13	348	351	357	362	369	378	394	414	435	457	497
14	348	352	357	363	374	390	411	432	454	476	498
15	348	350	354	363	377	397	415	435	456	477	498
16	348	351	360	375	395	412	432	453	474	495	517
17	348	350	356	363	373	387	403	410	426	439	455
18	348	351	356	364	376	386	394	403	417	430	444

Table A.6: Service Time with High Priority Contention (0.10μSec)

Bibliography

[**Adam93a**] Joel Adam, Henry Houh, and David Tennenhouse. *Experience with the VuNet: A Network Architecture for a Distributed Multimedia System*, September 1993. To be presented at the 18th Conference on Local Computer Networks. (pp 29, 40)

[**Adam93b**] Joel Adam and David Tennenhouse. *The Vidboard: A Video Capture and Processing Peripheral for a Distributed Multimedia System*. In Procedings SIGGRAPH, August 1993. (p 40)

[**Agarwal86**] Anant Agarwal, Richard Sites, and Mark Horowitz. *ATUM: A New Technique for Capturing Address Traces Using Microcode*. In Proc 13th Annual International Symposium on Computer Architecture, June 1986. (p 65)

[**Agarwal93**] A Agarwal and S Pudr. *Column-Associative Caches: A Technique for reducing the miss rate of Direct-Mapped Caches*. Computer Architecture News, 21(2), May 1993. Procedings 20th Annual International Symposium on Computer Architecture, May 16 – 19 1993. (pp 16, 65)

[**Alnes93**] Knut Alnes. *800 MB/s SCI "MP" System Runs in Dolphin Lab*. Email to SCI mailing list, June 1993. (p 6)

[**AMD88**] Advanced Micro Devices. *PLPL Software User Documentation*, version 2.2 edition, 1988. (p 59)

[**AMD91**] Advanced Micro Devices. *MACH Family Data Book*, Q1 1991. (p 56)

[**Anderson93**] Thomas Anderson, Susan Owicki, James Saxe, and Charles Thacker. *High Speed Switch Scheduling for Local Area Networks*. Technical Report 99, DEC Systems Research Center, April 1993. (pp 34, 35)

[**ARM92**] Advanced RISC Machines Limited, Swaffham Bullbeck, Cambridge, UK. *ARM 600 Datasheet*, 1.0 edition, 1992. (p 49)

[**Atkinson93**] Ian Atkinson. *A Digital Signal Processor and Audio Node for the Desk Area Network.* University of Cambridge, Computer Laboratory Part II Project, July 1993. (p 38)

[**ATMUNI93**] *ATM User-Network Interface Specification Version 2.1.* ATM Forum Technical Committee, Signalling Subworking Group Draft, May 1993. (p 10)

[**Bacon93**] Jean Bacon, John Bates, Sai Lai Lo, and Ken Moody. *OPERA: Storage and Presentation Support for Multimedia Applications in a Distributed ATM netork Environment.* Technical Report 295, Cambridge University Computer Laboratory, April 1993. (p 81)

[**Beeler92**] Reto Beeler. *The Fairisle VME Interface.* In ATM Document Collection. University of Cambridge Computer Laboratory, February 1992. Systems Research Group Technical Note. (p 26)

[**Berc93**] Lance Berc. *J-Video.* Personal communication, 1993. (p 14)

[**Berenbaum93**] Alan Berenbaum, Joe Dixon, Anand Iyengar, and Srinivasan Keshav. *A Flexible ATM–Host Interface for XUNET II.* IEEE Network, July 1993. (p 13)

[**Black93a**] Richard Black. *FDL cell format and Meta signalling.* In ATM Document Collection 2 (The Orange Book), chapter 5. Cambridge University Computer Laboratory, February 1993. Systems Research Group Technical Note. (p 13)

[**Black93b**] Richard Black. *Framing and Sequencing.* In ATM Document Collection 2 (The Orange Book), chapter 3. Cambridge University Computer Laboratory, February 1993. Systems Research Group Technical Note. (p 13)

[**Blair93**] Gordon Blair, Andrew Campbell, Geoff Coulson, Francisco Garcia, David Hutchinson, Andrew Scott, and Doug Shephard. *A Network Interface Unit to Support Continuous Media.* IEEE Journal on Selected Areas In Communications, 11(2):264–275, February 1993. (pp 14, 31)

[**Bovopoulos93**] Andreas Bovopoulos, R Gopalakrishnan, and Saied Hosseini. *SYMPHONY: A Hardware, Operating System, and Protocol Processing Architecture for Distributed Multimedia Applications*. Technical Report WUCS-93-06, Washington University in St. Louis, March 1993. (pp 31, 34)

[**CCITT90**] CCITT. *Broadband Aspects of ISDN*. CCITT Draft Recommendation I.121, May 1990. (p 10)

[**Cohen92**] Danny Cohen, Gregory Finn, Robert Felderman, and Annette DeSchon. *ATOMIC: A low-cost Very High Speed LAN*. Technical Report ISI/RR-92-291, University of Sothern California/Information Sciences Institute, September 1992. (p 9)

[**Cohen93**] Danny Cohen, Gregory Finn, Robert Felderman, and Annette DeSchon. *ATOMIC: A High–Speed Local Communication Architecture*. FTP from `ftp.isi.edu`, July/August 1993. Submitted to Journal of High Speed Networks. (p 10)

[**DEC91a**] DEC. *DECStation & DECSystem 5000 Model 240 Technical Overview*, 1991. DEC Order code EC-N0194-51, availible by ftp from `gatekeeper.dec.com`. (p 69)

[**DEC91b**] DEC. *Maxine System Module Functional Spec*, rev 1.2 edition, February 1991. (p 69)

[**DEC92a**] DEC. *DEC 3000 Model 400/400S AXP Technical Summary*, 1992. DEC Order code EC-N0093-51, availible by ftp from `gatekeeper.dec.com`. (p 69)

[**DEC92b**] DEC. *DEC 3000 Model 500/500S AXP Technical Summary*, 1992. DEC Order code EC-N0094-51, availible by ftp from `gatekeeper.dec.com`. (p 69)

[**DEC92c**] Digital Equipment Corporation. *DECChip 21064-AA RISC Microprocessor Preliminary Data Sheet*, April 1992. (p 16)

[**Dixon92**] MJ Dixon. *System Support for Multi-Service Traffic*. Technical Report 245, Cambridge University Computer Laboratory, January 1992. Ph.D. dissertation. (pp 19, 29)

[**Dol92**] Dolphin SCI Technology. *Cbus Specification Version 2.0*, August 1992. (p 6)

[**Druschel93**] Peter Druschel, Mark Abbot, Michael Pagles, and Larry Peterson. *Network Subsystem Design.* IEEE Network, July 1993. (p 28)

[**Dutton92**] TA Dutton, D Eiref, HR Kurth, JJ Reisert, and RL Stewart. *The design of the DEC 3000 AXP Systems, Two High performance workstations.* Digital Technical Journal, 4(4), Special Issue 1992. (p 30)

[**Ewy93**] Benjamin Ewy and Joseph Evavs. *Secondary Cache Performance in RISC Architectures.* ACM Computer Architecture News, 1993. (pp 64, 83)

[**Fairbairns93**] Robin Fairbairns. *Esprit BRA 6865 — Pegasus: Summary Report, Kernel Work Package.* To become a Pegasus Project Report, September 1993. (p 14)

[**Finn91**] G Finn. *An Integration of Network Communication with Workstation Architecture.* ACM SIGCOMM, Computer Communications Review, 21(5), October 1991. (pp 10, 32, 40)

[**Fraser92**] AG Fraser, CR Kalmanek, AE Kaplan, WT Marshall, and RC Restrick. *XUNET 2: A Nationwide Testbed In High–Speed Networking.* In Proceedings IEEE INFOCOM, volume 2, pages 0582–0589, May 1992. (p 35)

[**Fraser93**] AG Fraser. *Early Experiments with Asynchronous Time Division Networks.* IEEE Network, January 1993. (p 10)

[**Glauert93**] Tim Glauert and Stewart Wray. *Medusa.* Presentation given to the Cambridge University Computer Laboratory Systems Research Group, May 1993. (p 36)

[**Gordon92**] Daniel Gordon. *The Maybe Board, Design and Operation.* In ATM Document Collection. University of Cambridge Computer Laboratory, February 1992. Systems Research Group Technical Note. (p 26)

[**Gordon93a**] Daniel Gordon. *The 4×4 and 8×8 Fabrics.* In ATM Document Collection 2 (The Orange Book), chapter 25. Cambridge University Computer Laboratory, February 1993. Systems Research Group Technical Note. (p 22)

[**Gordon93b**] Daniel Gordon. *Null Port Controller Design.* In ATM Document Collection 2 (The Orange Book), chapter 20. Cambridge

University Computer Laboratory, February 1993. Systems Research Group Technical Note. (p 25)

[**Greaves90**] D Greaves, D Lioupis, and A Hopper. *The Cambridge Backbone Ring.* Technical Report 90/2, Olivetti Research Limited, February 1990. Also in Proc. IEEE Infocom 90. (p 21)

[**Greaves93**] David Greaves and Leslie French. *The Olivetti Research 'Yes V2' option module.* In ATM Document Collection 2 (The Orange Book), chapter 33. Cambridge University Computer Laboratory, February 1993. Systems Research Group Technical Note. (p 26)

[**Gustavson93**] David Gustavson. *SCI news mailing E35a.* Electronic mail, July 1993. (p 6)

[**Hayter91**] MD Hayter and DR McAuley. *The Desk Area Network.* ACM SIGOPS Operating Systems Review, 25(4), October 1991. Also Cambridge University Computer Lab Tech Report 228, May 1991. (p 28)

[**Hayter92a**] Mark Hayter. *Fairisle/Yes/Maybe ATM Taxi link Protocol.* In ATM Document Collection. University of Cambridge Computer Laboratory, February 1992. Systems Research Group Technical Note. Updated version in the ATM Document Collection 2. (p 26)

[**Hayter92b**] Mark Hayter and Richard Black. *Fairisle Port Controller Design and Ideas.* In ATM Document Collection. University of Cambridge Computer Laboratory, February 1992. Systems Research Group Technical Note. (p 68)

[**Hayter93a**] Mark Hayter. *FPC3 xilinx Xi5, Design and notes.* In ATM Document Collection 2 (The Orange Book), chapter 19. Cambridge University Computer Laboratory, February 1993. Systems Research Group Technical Note. (p 27)

[**Hayter93b**] Mark Hayter and Richard Black. *Fairisle Port Controller Design and Ideas.* In ATM Document Collection 2 (The Orange Book), chapter 16. Cambridge University Computer Laboratory, February 1993. Systems Research Group Technical Note. (p 23)

[**Hayter93c**] Mark Hayter and Richard Black. *FPC2 xilinx chip – Xi3, Design and Software.* In ATM Document Collection 2 (The Orange Book), chapter 18. Cambridge University Computer Laboratory, February 1993. Systems Research Group Technical Note. (p 27)

[Hill88] Mark D Hill. *A Case for Direct Mapped Caches.* IEEE Computer, December 1988. (pp 16, 51)

[Hopper78] A Hopper. *Local Area Computer Communication Networks.* Technical Report 7, Cambridge University Computer Laboratory, 1978. Ph.D. dissertation. (p 10)

[Hopper88] A Hopper and RM Needham. *The Cambridge Fast Ring Networking System.* IEEE Trans. Computers, 37(10), October 1988. (p 21)

[Hopper90] A Hopper. *Pandora - an experimental system for multimedia applications.* ACM Operating Systems Review, 1990. (pp 13, 29, 31)

[Hyden93] Eoin Hyden. *Nematoid experiments.* Personal Communication, September 1993. (p 14)

[IEEE.P1596] *Scalable Coherent Interface.* IEEE Standard P1596-1992. (p 5)

[Jardetzky92] Paul Jardetzky. *Network File Server Design For Continuous Media.* Technical Report 268, Cambridge University Computer Laboratory, October 1992. Ph.D. dissertation. (p 81)

[Leslie83] IM Leslie. *Extending the Local Area Network.* Technical Report 43, Cambridge University Computer Laboratory, 1983. Ph.D. dissertation. (pp 11, 32)

[Leslie91] IM Leslie and DR McAuley. *Fairisle: An ATM network for the local area.* Proc. ACM SIGCOMM, September 1991. (pp 21, 22, 35)

[Leslie93] Ian Leslie. *The EISA ATM Interface.* In ATM Document Collection 2 (The Orange Book), chapter 24. Cambridge University Computer Laboratory, February 1993. Systems Research Group Technical Note. (p 26)

[Lunn93] Andrew Lunn, Andrew Scott, and Doug Shepherd. *ATM, AAL5 and Multimedia Devices.* Distributed Multimedia Research Group Technical Report MPG-93-06, Lancaster University, 1993. (p 31)

[**McAuley90**] DR McAuley. *Protocol Design For High Speed Networks.* Technical Report 186, Cambridge University Computer Laboratory, 1990. Ph.D. dissertation. (pp 10, 19, 29)

[**MIPS91**] MIPS Computer Systems. *Introduction to the R4000 Microprocessor*, 1991. Mfg. Part Number M8-00041. (p 16)

[**Mot84**] Motorola Semiconductors. *The MC68901 Multi-Function Peripheral*, January 1984. (p 49)

[**Mullender92**] SJ Mullender, IM Leslie, and DR McAuley. *Pegasus Project Description.* Technical Report, September 1992. Pegasus Paper 92-1. Also Cambridge University Computer Laboratory Technical Report 281. (p 83)

[**Nat93**] National Semiconductor. *QR0001 QuickRing Data Stream Controller Preliminary Rev 14*, July 1993. (p 8)

[**NCLATM92**] *Network Compatible ATM for Local Network Applications.* Discussion document circulated by Apple Computer, Bellcore, Sun Microsystems and Xerox., April 1992. This has been superseded by [**ATMUNI93**]. (p 10)

[**Needham82**] R. M. Needham and A. J. Herbert. *The Cambridge Distributed Computer System.* Addison-Wesley, Reading, Ma., 1982. (p 28)

[**Needham93**] Roger Needham. *Computers and Communications*, September 1993. Seminar given at the 3rd Pegasus Workshop, Cambridge. (p 35)

[**Newman89**] P Newman. *Fast Packet Switching for Integrated Services.* Technical Report 165, Cambridge University Computer Laboratory, 1989. Ph.D. dissertation. (p 22)

[**Nicolaou90**] Cosmos Nicolaou. *A Distributed Architecture for Multimedia Communication Systems.* Technical Report 220, Cambridge University Computer Laboratory, December 1990. Ph.D. dissertation. (p 83)

[**NodeChip92**] *NodeChip Functional Specification.* Dolphin SCI Technology Information Sheet, 1992. (p 37)

[**NodeChip93**] *DST511A CMOS NodeChip.* LSI Logic and Dolphin SCI Technology Advance Information Data Sheet, August 1993. (p 37)

[**Pasquale92**] Joseph Pasquale. *I/O System Design for Intensive Mutitmedia I/O.* In Third Workshop on Workstation Operating Systems,, Key Biscayne, Florida, April 1992. (p 29)

[**Pratt92**] IA Pratt. *The ATM Camera.* University of Cambridge, Computer Laboratory Part II Project, July 1992. (p 37)

[**Pratt93a**] Ian Pratt. *A brief description of the ATM Camera v2.* Personal Communication, August 1993. (p 37)

[**Pratt93b**] Ian Pratt. *The DAN Framestore.* In ATM Document Collection 2 (The Orange Book), chapter 29. Cambridge University Computer Laboratory, February 1993. Systems Research Group Technical Note. (p 38)

[**Pratt93c**] Ian Pratt and Eoin Hyden. *Port Controller Performance Measurements.* In ATM Document Collection 2 (The Orange Book), chapter 30. Cambridge University Computer Laboratory, February 1993. Systems Research Group Technical Note. (p 27)

[**Prudence93**] Mike Prudence. *Sapphire ATM Hub ERS.* Dawn Project Memo, HP Laboratories, Bristol, UK, April 1993. (p 25)

[**Przybylski88**] Steven Przybylski, Mark Horowitz, and John Hennessy. *Performance Tradeoffs in Cache Design.* In Proceedings 15th Annual International Symposium on Computer Architecture, 1988. (p 16)

[**Rambus92**] *Rambus Architectural Overview.* Rambus Inc, Mountain View, California, USA, 1992. (pp 7, 8)

[**Ray93a**] Steven Ray. *Re: Toshiba Press Release & Rambus DRAMs.* Electronic mail communication, September 1993. (p 8)

[**Ray93b**] Steven Ray. *Toshiba Press Release & Rambus DRAMs.* USENET News article in group `comp.arch Message-ID: <slrCCqo4K.LyK@netcom.com>`, September 1993. (p 7)

[**RD24Status93**] *Application of the Scalable Coherent Interface to Data Acquisition at LHC.* CERN/DRDC 93-20, May 1993. A status report of the European Organization for Nuclear Research project group RD24. (pp 6, 7)

[**Redell92**] Dave Redell. *Sun's Coco system: 9/8/92 BayCHI Talk at PARC.* Report on [**Tang92**], September 1992. (p 14)

[Sakata93] Shiro Sakata. *B-ISDN Multimedia Workstation Architecture.* IEEE Communications Magazine, August 1993. (p 31)

[Seitz92] Charles L. Seitz. *Submicron systems architecture, semiannual technical report.* Technical Report Caltech-CS-TR-92-17, caltech, 1992. (p 9)

[Smith82] Alan Smith. *Cache Memories.* Computing Surveys, 14(3):473–530, September 1982. (pp 15, 51)

[Smith87] Alan Smith. *Line (Block) Size Choice for CPU Cache Memories.* IEEE Transactions on Computers, C-36(9), September 1987. (p 65)

[Sreenan92] Cormac Sreenan. *Synchronisation Services for Digital Continuous Media.* Technical Report 292, Cambridge University Computer Laboratory, October 1992. Ph.D. dissertation. (p 80)

[Støa93] Dag Støa, Kjersti Moldeklev, and Øivind Kure. *ATM-SCI Interface — An Overview.* Dolphin SCI Technology and Norwegian Telecom Research Information Sheet, February 1993. (p 31)

[Sugumar91] Rabin Sugumar and Santosh Abraham. *Efficient Simulation of Multiple Cache Configurations using Binomial Trees.* Technical Report CSE-TR-111-91, CSE Division, University of Michigan, 1991. (p 60)

[Sugumar93a] Rabin Sugumar and Santosh Abraham. *cheetah(1) – Single-pass simulator for direct-mapped, set-associative and fully associative caches.* Unix Manual Page, 1993. (p 60)

[Sugumar93b] Rabin Sugumar and Santosh Abraham. *Efficient Simulation of Caches under Optimal Replacement with Applications to Miss Characterization.* In Proceedings ACM SIGMETRICS Conference, 1993. (p 60)

[Tang92] John Tang. *Why Do Users Like Video? A Case Study in Developing Usable Multimedia Collaborative Technology.* Presentation given at the September 1992 meeting of the San Francisco Bay Area ACM SIGCHI, September 1992. (p 14)

[Temple84] S Temple. *The design of a Ring Communication Network.* Technical Report 52, Cambridge University Computer Laboratory, 1984. Ph.D. dissertation. (p 21)

[**Tennenhouse86**] D. L. Tennenhouse. *The Unison Data Link Protocol Specification.* Unison Project Document UC022, October 1986. (p 13)

[**Tennenhouse89**] DL Tennenhouse and IM Leslie. *A Testbed for Wide Area ATM Research.* In Proc. ACM SIGCOMM, September 1989. (p 35)

[**Tennenhouse92**] D Tennenhouse, M Ciholas, and J Davin. *Telemedia, Networks and Systems Group Annual Report.* Technical Report AR-001, Laboratory for Computer Science, MIT, 1992. (p 14)

[**Xilinx92**] *The Programmable Gate Array Data Book.* Xilinx Inc, 1992. (p 17)