Devices on the Desk Area Network

P Barham, M Hayter[1], D McAuley and I Pratt
University of Cambridge Computer Laboratory

March 1994

Abstract

The Desk Area Network was proposed as an architecture suitable for a multimedia workstation. This paper describes how the architecture has evolved and the demonstration workstation that has been constructed[2].

1 Introduction

Common usage of the term "multimedia" is to describe systems which incorporate both traditional computer data forms, such as text and graphics, and data forms such as audio and video. This presents new problems due to both the isochronous requirements for audio and video and the significantly higher bandwidth that is needed to transport video. The wish to link machines which use this mix of data types is one of the driving forces behind the development of high speed networks based on the Asynchronous Transfer Mode (ATM); these aim to provide the necessary performance and guarantees about the timeliness of data transfers.

The Asynchronous Transfer Mode makes use of small fixed size cells to carry data through the network. It is connection oriented and each cell contains a label to identify to which of the multiplexed communications the cell belongs. The emerging standards [**CCITT90**, **ATMUNI93**] subdivide the label into a virtual path and a virtual channel; within this paper, this subdivision is not significant and so the term circuit identifier is used to mean the whole of the label. As scheduling of both links and switches is performed on a per cell basis, the small size enables preemption to be performed at a fine granularity – such techniques are also common in many modern high speed workstations, where buses are designed to enable cache access to preempt long DMA transfers. This property is used in the network to offer a range of Qualities of Service (QoS) and support a spectrum of traffic, from isochronous to very bursty.

Designers of multimedia workstations must address many of the same problems. In particular there is a large amount of data, which is often sensitive to jitter, moving between the peripherals of the end-system and the network interface. This phenomena has been recognised for many years and has been termed "Intensive I/O" [**Pasquale92**]. Currently multimedia systems concentrate on the presentation of the data to a human user, therefore requiring tight timing constraints. However, the ever increasing CPU power available in a desktop machine also enables applications which process such data in real time; while some of these may be more tolerant to jitter than the human eye, they can only serve to increase the bandwidth requirements.

A workstation architecture which aims to effectively support these forms of applications should therefore enable multimedia data to be delivered directly to peripheral end-points with minimal or no interaction by the CPU, while continuing to permit processing of the data by the CPU where required.

The approach that we have developed is to extend the use of ATM to the device and processor interconnect within a multimedia workstation [**Hayter91**]. This interconnect, which we call a

[1]Now at DEC Systems Research Center.

[2]This work was supported by a grant from the Science and Engineering Research Council.

Desk Area Network (DAN), uses ATM cells as the basic unit of data transfer. The aim is to provide improved system performance by the use of a device and processor interconnect which:

- is amenable to implementation as a space division switch fabric,
- provides frequent preemption points,
- allows a simple and efficient high bandwidth connection to an ATM based network.

Similar work is in progress in the ViewStation project [**Adam93**], and [**Pfeifer93**], where again ATM is used as the basis for a multimedia machine. The assumption that future high speed networks will be packet based [**Cohen92**] has led to an approach like the DAN but using packet switching internally [**Finn91**].

This paper presents the architecture, describes the demonstrator that has been built and presents results from the initial experiments performed using the system.

2 The DAN Architecture

A Desk Area Network based machine is similar to current machines in having a number of processors, memory modules, and a variety of peripherals. However, the DAN applies a system wide multiplexing mechanism and channel identification to the interconnection of these modules based on ATM cells; hence the physical units transferred are cells, and they are associated with channels and routed by the labels in their headers.

A multimedia workstation based on a DAN is shown later in the paper in figure 4. Every device is equipped with an interface to the ATM interconnect. Such an attachment is no more complicated than the equivalent bus interface (and even simpler than over-complicated buses such as EISA). One reason for building the prototype DAN, rather than simply pontificating about it, is to illustrate this point. Furthermore, the internal ATM connection does *not* need to include a complete transmission system, with bit serial communication, clock recovery, clock alignment, fault isolation, buffering etc. Synchronous clocked unidirectional parallel buses can be used due to the physical proximity of the devices. As with standard peripherals, the interface logic may be designed with a specific task in mind and can therefore be as small as a few PALs and a pair of FIFO memories.

2.1 ATM as an interconnect

Using ATM as the multiplexing technique within the workstation has similar advantages to the use of ATM in larger networks. Different channels (even multiple channels between the same two devices) can easily have associated with them different characteristics, the channel identifier providing an index which may be used to select contention priority and queuing disciplines, and thus different Qualities of Service.

As already mentioned, many bus based systems (e.g. TURBOchannel, PCI and SBUS), which support an arbitrary sized data transfer between devices, insist that the bus be made available at regular intervals for preemption. Often the particular interval chosen is based on that of the cache / memory transaction. However, using any small fixed-size data transfer unit also achieves this goal. The choice we have made for the DAN is aimed at streamlining the passage of data through the network interface; hence we choose the same size as has been standardised for the local and wide area ATM networks. As will be seen, the choice of 48 data bytes fits well with the implementation of 32 byte cache line transfers when due consideration is given to the bits for the transaction type, and the related read and write addresses.

The use of fixed size transfer units introduces the need to segment larger blocks of data at the source and re-assemble them at the destination. In the case of a single communication this is trivial, but the scheduling of the interconnect can cause multiple blocks to be arriving in an interleaved manner. The reassembly operation may therefore be fairly costly. Many ATM network interface designs provide hardware assistance for conversion to standard protocols. In the DAN it is often sensible to implement device (or media) specific ATM adaptation layers within the device rather than requiring the host interface or the main CPU to implement a plethora of different adaptation layers.

In the wide area statistical multiplexing of a large number of streams is claimed to allow the network to function at high utilisation and support quality of service guarantees, albeit statistical ones; these "guarantees" are based on assumptions of independence of the sources and are usually only asymptotically valid. Such arguments clearly do not apply in the desk area where the number of streams is small and where they are often highly correlated. Within the DAN, the management entity for the switch is the operating system, and the required scheduling of the interconnect is tightly coupled to the scheduling of processes on the processor(s). The integration of these scheduling mechanisms, and the treatment of the interconnect as generic operating system resource is under study. Several other groups are also considering scheduling an interconnect where statistical predictions cannot be made with great confidence. 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 separate main and multimedia buses, also suggests controlled scheduling of the multimedia interconnect.

2.2 Implementation choices

The DAN is intended to be the internal interconnect of a machine, so, in common with bus based systems, the simplifying assumption is made that the hardware provides a reliable path for communication between devices. This may of course involve all the techniques seen in bus based systems to ensure a sufficient level of reliability; for example, parity, ECC, transaction retry (hence blocking), etc. The benefit of this reliability is to greatly simplify the control of the system – no need for a 7 layer stack of protocols to read a register on a device. This trade off is made for the DAN as for bus based systems; as the technology and complexity are similar, so similar reliability can be expected.

The exact implementation 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. However, the reliability requirement, which is effectively that the source must know when to retry a cell, rules out a few "hail-mary" switch architectures, which silently drop cells internally.

As with network switches, the use of a crossbar or other space division fabric will enable concurrent data transfer between different pairs of devices, but it may well prove that a high speed slotted bus would provide a cost effective solution for a small number of nodes. As the architecture allows this variation in implementation, certain functions (e.g. cache coherency) are not defined as part of the DAN architecture, rather they are implementation dependent.

2.3 Device control

The label size defined for the standard ATM cell is sufficiently small[3] that it is normal to use ATM as a connection oriented technique; hence, before use, context indexed by the label[4] is established in each component of the network traversed by the channel. This is also true within the DAN, where a context is established within devices and, if the stream is passing between a device and the network, within the network interface and network.

Thus devices receiving data can use the VCI to access any channel specific information; the example of a VCI per window, where the context is the coordinates of the window and clip mask, is described in the framestore section. Use of this technique will be seen many times in the description of the implemented DAN based machine.

Within the DAN, the OS is responsible for access control and protection and, if it is doing its job properly, can ensure that only trusted components configure nodes for communications; hence internally, DAN nodes can be relied upon not to use bogus VCIs or violate their bandwidth allocation. The parallels can be seen here with the management of many traditional operating system resources; for example, the discs implementing the backing store of a virtual memory system are controlled by privileged code and then trusted not to DMA to random memory locations.

The use of virtual circuits requires that some entity perform connection management for the DAN nodes. 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 single cell messages on a predefined VCI. The source of such control messages would be a trusted manager on some other node of the DAN. The controller has responsibility for allocating VCIs for data streams and configuring the device. Usually the manager would be outside the data path so that it is likely that the control functions can easily be performed by one of the processing engines without undue interference with its other jobs.
- "Supervised" devices have some local processing power which will enable them to perform some local management, for example they might allocate their own VCIs. Again the main connection establishment functions are performed by a more competent processor node, but the control interactions between manager and device are at a higher level.
- "Smart" devices have substantial processing power and are able to perform all of their own management functions; some of these nodes will also act as proxies for interactions between "dumb" devices and the network and operating system.

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. 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 from an engineering viewpoint is important.

3 The DAN Demonstrator

A multimedia workstation based on a DAN has been largely completed. This is a demonstrator of the architecture and provides a test-bed for its exploration. To show the feasibility of DAN devices

[3]One good reason for the small size is to maintain reasonable bandwidth efficiency.

[4]Even without any QoS considerations, at least routing context is required.

and to investigate their properties, a camera node and a DSP node have been built. A framestore node has been designed and until the hardware is complete is being emulated by a DECStation 5000/25. To investigate the use of the DAN as a processor-memory interconnect, in addition to its use for interconnecting multimedia devices, a CPU and memory node have been constructed.

The demonstrator is based on the Xilinx FPGA based switch element designed for the Fairisle ATM LAN [**Leslie91**] and is therefore constrained to an 8 bit wide data path and a 20MHz clock rate, providing nominally 160Mbps full duplex per port. It seems likely that custom ICs and possibly a wider data path would be used in a full DAN implementation. The complexity of the interface hardware is comparable to that required for implementing the base SCI protocols for which high speed VLSI implementations have been produced in both GaAs [**NodeChip92**] and CMOS [**NodeChip93**].

3.1 Network Interface

The "network interface" of a DAN based machine is simply a cell router from the internal interconnect to the LAN. It is very similar to a port controller (or line-card) of an ATM LAN switch. However, being at the machine–network interface it is also the barrier to the outside world and will need to implement security (e.g. drop cells with bogus VCIs) and data-flow control.

The demonstrator network interface uses the port controller that was developed for the Fairisle ATM switch [**Leslie91**]. This has minor software modifications, but the hardware is identical, and it performs very similar queuing in the DAN as it would in a switch. This supports full duplex operation at 100Mbps between the network link and the internal interconnect. More details on the performance of this device has been described in [**Black94**].

The network interface node is the boundary between the DAN with its simplifying reliability property, and the LAN where more complicated protocols must be used for error detection etc. As streams cross the boundary into the DAN, for example, complete AAL5 PDUs could be assembled and the CRC checked. By doing this it is possible to ensure that only complete and uncorrupted PDUs are let onto the DAN, and that they are within their resource allocation (e.g. mean and peak data rates). This can remove the need for complicated reassembly, error detection, and protection mechanisms on individual nodes in the DAN.

3.2 The Camera Node

The "ATM camera" is a device which captures and digitizes real-time video into an ATM cell stream. Two versions of the camera have been produced and both are described as they show examples of "dumb" and "supervised" devices.

The first version of the camera was completed in April '92. As an example of a "dumb" device, it shows how simple a DAN node can be.

The node contains no processor and is controlled via a single cell message that it receives on a fixed VCI. The cell configures parameters such as the "grab" window, scaled size, colour mode and output VCI. This control cell is generated by a manager process running on a processor node. The manager exports an RPC like interface to clients. Multiple clients can connect to the manager and request different video streams on different VCIs. The process will deal with scheduling the different requests and sending the appropriate configuration cell to the camera. Note that the video data is never touched by the management process.

Additionally, format conversion processes can be inserted into the stream of video data. In particular, a process that converted V1 camera video into Pandora [**Hopper90**] format was demonstrated as part of the software emulation of a Pandora Box described later.

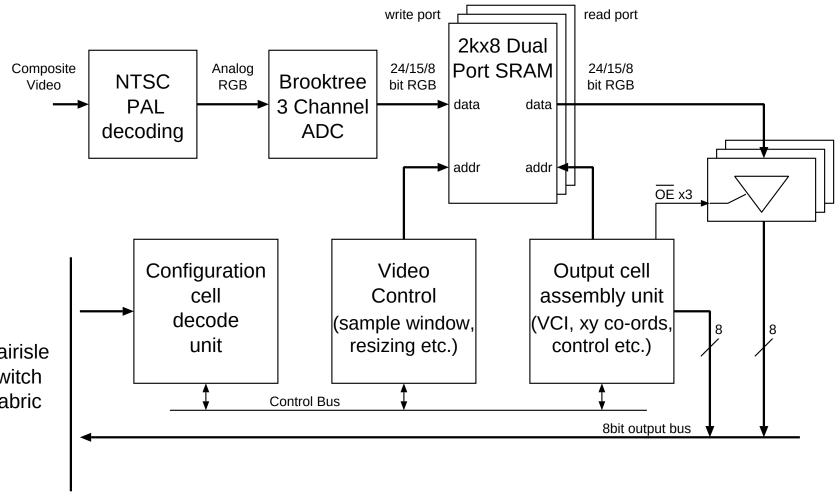

Figure 1: Datapath diagram for DAN Camera V1

The V1 camera digitizes in either 8, 15 or 24 bits RGB into a 2 line buffer. This provides decoupling between the video and network clocks. The cells composing each line of digitized video are preceded by a cell containing a line number and frame count.

With only a 2 line buffer, this type of camera generates very bursty traffic. A typical application might require a 352x288[5] 24bit RGB picture at 15 fps. This requires a continuous rate of 36Mb/s. However, the V1 camera with its 2 line buffer will generate bursts of over 130Mb/s during active areas of the frame. The availability of inexpensive "video fifos" (used to provide freeze frame capability in VCRs) allowed this problem to be overcome in the version 2 camera. By buffering an entire video frame, transmission at the mean rate can be achieved.

The video format was also changed for the V2 camera. It was decided to package the video in multiples of 8x8 pixel tiles. This has a number of advantages over line based methods: having a fixed minimum unit is useful for hardware implementations – by making this unit 64 pixels in an 8x8 block we do little to restrict the range of possible picture sizes; two dimensional blocks are also the basis for many image compression schemes – 8x8 blocks were chosen for both JPEG and MPEG, so our tiled video scheme lends itself well to compression with these methods. Were the DAN interface to be built into a CCD controller, the 8x8 tiles could be read directly from the CCD element, avoiding the scan line to tile conversion that is necessary when dealing with conventional video formats designed for display on CRTs.

While the camera V1 was a "dumb" device, the second version of the camera uses a simple 8-bit microcontroller to provide more complex control functions; for example, it is possible to download a schedule which is repeatedly executed, which defines for each video field which video source to use, the XY resolution required, depth, coding (RGB, YUV, JPEG), and destination (i.e. VCI).

3.3 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[6]. A CODEC provides stereo audio capture and playback in a number of common rates and formats up to stereo 48kHz 16 bit PCM (DAT format). The CODEC feeds an Analog Devices 16bit integer DSP which connects to the DAN via

[5]This size is known as CIF, Common Intermediate Format

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

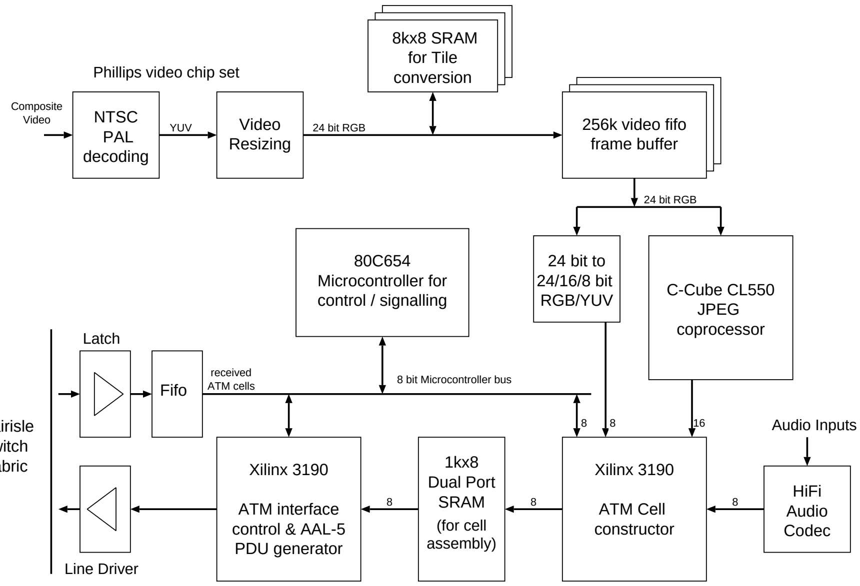

Figure 2: Datapath diagram for DAN Camera V2

4k of dual ported SRAM and a control Xilinx FPGA.

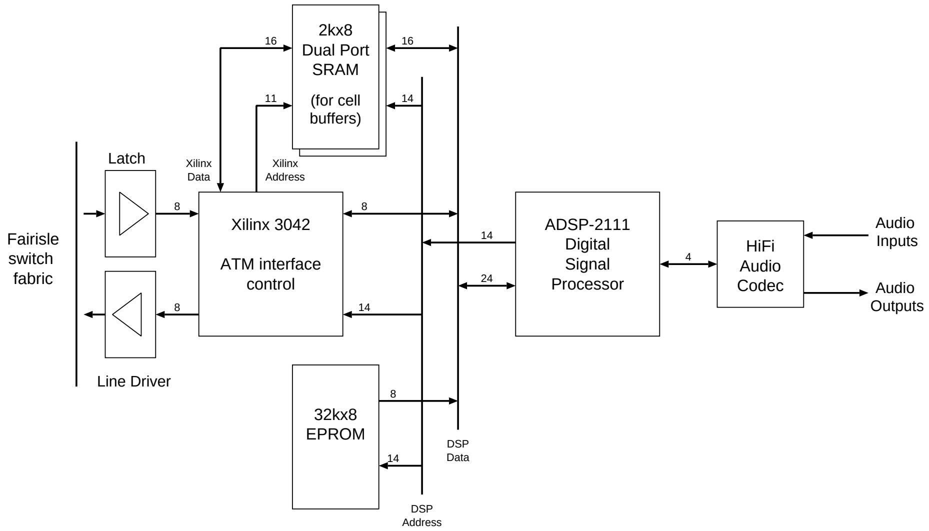

Figure 3: Datapath diagram for DAN DSP/Audio node

Half of the dual ported memory contains a FIFO queue of cells received from the DAN. Arriving cells are presented to the DSP which must explicitly free the cell buffer before it is reused. If there are no free cells then the DAN interface refuses incoming cells providing back-pressure to the source.

A further queue is used for cells to be transmitted across the interconnect. However, cell buffers on this list may come from anywhere in the SRAM. This allows the DSP to process cells in situ in the receive buffer, and then thread them back on the transmit queue. This leads to efficient stream processing without data copy. Since the threading operation consists of a single write to the SRAM and a small amount of internal bookkeeping the 12MHz DSP can perform the null operation (i.e. simply forwarding the data) at more than the 160Mbps interconnect rate.

The DSP node falls into the “supervised” class of devices, since the DSP processor can perform some simple management functions. However, like the camera, the card is again optimised for the data path and most control functions are delegated to a general purpose processor node.

3.4 The Framestore Node

This section describes the final design of the DAN framestore which was arrived at after evaluation of various levels of functionality using software emulation on a DECStation 5000/25. Section 4.4 describes the opposite extreme of implementing the complete windowing system in the framestore.

The DAN Framestore performs a single rendering operation, that of taking a number of 8x8 pixel tiles from within an arriving PDU, and copying them to the display. The VCI of the incoming cells identifies the destination window, and the associated VC context contains parameters such as the position of the window on the display, and the pixel format (e.g. 8 bit mono, RGB 3:3:2, RGB 8:8:8).

While the hardware version of the DAN framestore will support simultaneous display of windows with different depths, the frame buffer in the DECStation is only 8 bits deep so most formats require conversion in software before display.

Due to the reliable nature of the DAN, reassembling complete PDUs can be avoided, and the data contained in the incoming cells copied directly to the framebuffer. In the case of video streams from remote sources, the network interface node can ensure that only complete and uncorrupted PDUs are let onto the DAN. While dropping corrupt PDUs will result in small areas of the picture not being updated, inter-frame correlations often make these hard to detect.

The framestore does not provide explicit support for more traditional rendering operations, such as drawing lines and displaying text. Such operations are performed on a processor node. This has a local pixel map and can generate the list of modified tiles as it renders into this map. Any of these *video differences* which are in a visible region of the window are sent to the framestore for display. This minimises the number of primitives which must be supported by the framestore, simplifying hardware implementation, and also has the benefit of making video and graphical windows indistinguishable.

Clipping of windows is performed on a per-pixel basis by the DAN Framestore. Each client, communicating on a separate VCI for each window, is offered a “virtual write-only display” abstraction. This obviates the need for a shared service in the display datapath to enforce protection barriers, and simplifies the process of resource allocation by preventing “QoS Crosstalk” between applications – the resources used by each application for rendering its windows are naturally accounted to that application process rather than being some unknown proportion of the shared service.

Clients are responsible for all updates to their own windows, though for most applications a shared library with a set of default rendering operations will be indistinguishable from the more conventional server-based approach. Clients with more specialised needs are free to implement their own rendering policies. For example, a client may delay flushing updates of large amounts of text to the framestore until it believes that no more output will be received for a while.

Clipping is achieved using an overlay plane to hold a tag for each pixel. Each window (and corresponding VCI) is also allocated a tag. Each incoming pixel on a particular VCI is only copied to the framebuffer if the tags are identical. The tag space is relatively small and is allocated using a map colouring algorithm. This method allows both video and graphical windows to be clipped against each other to arbitrary boundaries; a useful feature often unsupported by other multimedia peripherals.

The software emulation on the DECStation proved capable of clipping a number of arbitrary shaped video and graphical windows against each other with very little loss of performance, the main bottleneck proving to be the speed at which the processor was capable of draining the receive FIFOs of the ATM interface.

3.5 The Processor Node

The processor node developed for the DAN [**Hayter93**] allowed investigation of two areas: the use of the interconnect to carry CPU-memory traffic and to allow processing to take place on multimedia data types. It is noticeable that in many existing systems multimedia data are second class, the only operations that can usually be performed in real-time are capture, replay and presentation. Where other processing is done it tends to be performed by specialised hardware; for example compression/decompression of video. The DAN processor node aims to provide high speed access to such data and hence permit more general processing on these data.

The DAN processor node consists of an ARM600 chip, composed of the ARM6 processor core, MMU, primary cache and write buffer, and a large external secondary cache (256kbytes in 32 byte cache lines). The secondary cache is normally configured as two way associative with LRU replacement, but may also be set as direct mapped. On a miss the external cache controller sends a single cell request to a memory server, and receives a single cell reply containing a line of data. The secondary cache is copy-back, and if data needs to be flushed it may be piggybacked onto the request.

Two different memory servers were implemented for experimentation with cache line transfers across the DAN. The most commonly used one was implemented in software on a Fairisle port controller, as this enabled simple recording of addresses and data transfers during experiments. The second server was implemented in hardware and demonstrated the expected two cell response time but has not been widely used as it is less useful from an experimental point of view.

4 Results and experience

The raw results obtained from the various devices are presented first, then the results from a range of experiments performed using the processor node and devices in a variety of configurations.

4.1 The Camera Node

When capturing live video (e.g. 768 x 288 pixels per field, 50 fields per second for PAL sources[7]) at 24 bits per pixel, each scan line of pixel data is captured in $64\mu s$. In this mode, digital pixel data is being generated by the ATM camera at a sustained rate of 265Mbps, peaking at over 288Mbps. The video FIFOs on the ATM camera are capable of holding an entire frame which may then be transmitted as back to back cells with only a small gap between frames waiting for the next vertical sync pulse. Without taking into account protocol overheads, it is clear that the ATM camera can easily saturate a 160Mbps port on the switch fabric used in the DAN demonstrator.

Using the video FIFOs on the version 2 camera, constant bit rate video streams between 1 and 80Mbps have been generated. For example, a 352x288 pixel image in 16 bits per pixel, 25 fps generates 88Mbps of video averaged over a scan line. The frame buffer enables this to be smoothed to a continuous rate of 41Mbps by inserting gaps between PDUs.

This data can be carried through the interconnect and has been shown not to adversely interfere with cache line traffic (see section 4.6.2).

[7] Most of our sources of video are interlaced at 2 fields per frame.

4.2 The DAN Audio/DSP Node

The major use of the Audio/DSP node has been as a source and sink of audio rather than as a stream processor. Interoperability tests with Sun and HP workstations using various encodings were performed by recording the data from one system and replaying on the other. Simultaneous capture and replay by the audio node was also demonstrated. When used in its role as an audio source and sink the current software always runs the CODEC at 48kHz sample rate, and the DSP either sub-samples on capture or interpolates on replay to provide lower sampling rates. This enables multiple streams to be supported at different qualities, and has been found useful for jitter removal.

A second test was used to stress the DAN interface with the node in its DSP configuration. A virtual circuit was created across the DAN from the DSP node's transmit system to its receive buffer. The loop was completed by instructing the DSP to transmit on this circuit any cells that were received. Once a single cell was injected into this loop the ability of the DSP to perform null processing on a stream at the full interconnect speed was demonstrated by observing that the DSP transmitted a cell in every cell time. Thus, the DSP node could be used to log simple statistics on data passing though it.

4.3 The Framestore Node

Software emulation of the framestore has been used to enable the demonstration of the DAN and experimentation with various levels of framestore functionality. Unfortunately, the performance of the emulation running on the DECStation has been limited by the speed at which it can service its simple ATM interface. However, it is still capable of sinking over 15Mbps of 8bit RGB video into up to 16 overlapping, arbitrary shaped windows. The hardware version is designed to be able to sink video at the full interconnect rate.

4.4 Pandora Emulation

An early demonstration of the DAN used the ATM camera and framestore to provide an emulation of the Pandora's Box based multimedia workstation [**Hopper90**]. This enabled interworking with the multimedia infrastructure at both the Computer Laboratory and with Olivetti Research Ltd.

In this configuration, the DECStation was used as a highly intelligent frame buffer, and ran a complete X-server over the Wanda microkernel. The Pandora extensions were built into the X Server allowing unmodified Pandora applications to talk to the machine as though it were a genuine Pandora's Box. A separate daemon interpreted the Pandora control protocol intended for the Box and generated the correct requests to the DAN devices in order to create the required audio and video streams.

Local video and audio streams were able to use the native DAN formats between the camera and the framestore, in particular allowing larger colour video streams to be displayed. However, remote streams were transformed to conform to the Pandora protocols. This format conversion was performed using one of the processor nodes on the DAN.

Using this configuration, the DAN workstation was shown to interwork with standard Pandora's boxes allowing use of all of the normal video conferencing and video mail applications, at times handling up to 20 video streams simultaneously.

This configuration was found to be unsatisfactory as it was not possible to maintain application quality of service through the shared X server, not least because it is difficult to identify and partition the resources used by each application.

4.5 A Complete System

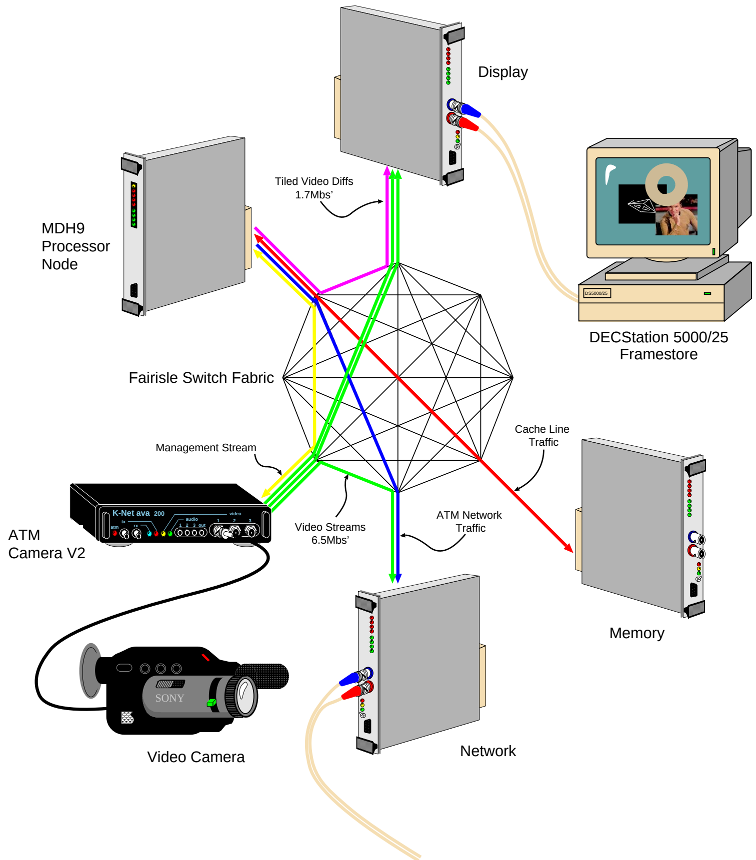

Figure 4: The experimental configuration

A complete demonstration system was constructed using the devices described above. A number of overlapping video windows from different sources were displayed on the framestore. To test the clipping algorithm in the framestore, one of these was a circular window with a hole in the middle through which two other windows could be seen.

A software library was written for the processor node which allowed clients to make use of the write-only virtual display capability of the framestore. Each client uses X-like primitives to render its own windows directly into the framestore, removing the need for an X server in the display datapath.

The various streams shown in figure 4 are as follows:

- 2 live video streams from the ATM Camera to the framestore. Each stream was 320x256 8 bits per pixel, 12 frames per second from the ATM camera. Each stream would peak at 32Mbps, however the ATM camera transmission rate was slowed by inserting gaps between each PDU, so that the average rate over a frame was 14Mbps causing each frame to be received over a period of 37.5ms. This corresponds to a sustained rate of 6.5Mbps per stream.
- 1 audio stream from the ATM camera to the framestore. Using a sample rate of 48kHz mono, 8 bit samples, this give a data rate of 384kbs. This was displayed as an oscilloscope trace in a window on the framestore.

- 1 stream of tiled video differences from the CPU node to the framestore. The processor node was running a simple 3D wireframe animation in a window 320x256 (1280 tiles), achieving 107 fps with on average 28 8x8 pixel tiles being updated per frame. This generates about 1.7Mbps.
- 1 Memory server handling cache line requests from the processor node peaking at about 600 cache lines/second (150kbps of useful cache data, 250kbps of data across the switch fabric in each direction). The surprisingly small Cache / Memory bandwidth requirement comes from the relatively large second level cache on the CPU node.
- A stream between the processor and the network interface supporting traditional network IO e.g booting, remote file system access, RPCs and TCP/IP traffic.
- Several low bit rate management streams between the processor node and the various devices.

Other experiments required streams whose source or sink was not on the DAN. Examples of this include video and audio streams from the ATM camera to other workstations on the ATM network, and displaying incoming video from remote ATM cameras. In both of these cases the streams do not need to pass through either the processor or main memory (although this is perfectly possible if some processing of the stream is required).

As in all other experiments, cache line service times were unaffected by the presence of the other streams.

4.6 The Processor Node

The main observations made with the CPU node were to explore the practicality of using the ATM based interconnect for cache traffic. Comparison with studies of cache behaviour reported by other authors [**Smith82**, **Smith87**, **Hill88**, **Agarwal86**, **Agarwal93**, **Przybylski88**] showed that the example loads used in these experiments were realistic.

Further experiments were performed to observe the effects on the cache traffic for interfering data streams.

4.6.1 Cache / Memory behaviour

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 a 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 running no user processes but being "pinged"[8] 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 1. In this experiment the fabric was clocked at 15MHz, giving a cell frame time of $4.2\mu s$.

The hardware tests were performed using the debug monitor to access memory, and by capturing the behaviour using a logic analyser. Using this setup it was not possible to accurately measure the small resynchronisation times[9] for stopping and restarting the CPU on top of that observed for the request–reply operation.

[8] This used the local ATM protocol, which defines an echo service through which a single cell message was passed.

[9] This time is due to switching between the interconnect clock and the CPU clock and is estimated as 100–200ns.

Server type	OS and PAL execution		Ping test		Read thrash		Write thrash	
	μs	frames	μs	frames	μs	frames	μs	frames
Software	35.1	8.4	34.8	8.3	33.1	7.9	40.7	9.7
Hardware	—		—		8.4	2	8.4	2

Table 1: Mean Memory Service Time

The "thrash" tests were performed by disabling interrupts on the CPU node and putting it into a loop referencing three locations that were co-resident 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 software based server the large difference between the two is caused by the need to copy the data from the network cell buffer into the main memory. Access to the cell buffers is slow (observed at 700ns per word for an earlier version of the port controller [**Hayter92**]) and the extra $7.6\mu s$ 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. 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.

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 buses or interconnects where the request consists solely of the address. However, this penalty is reduced in the DAN by the ability to flush and request in the same cell.

Table 2 shows the read and flush times for some of the workstations used by the Systems Research Group in Cambridge. 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 and are the best case times (for the DS 5000/25 which uses 16 byte lines this was calculated as a single setup delay followed by 8 word accesses).

System description	CPU type	Read Line ns	Write Line ns	Source
DECStation 5000/25	R3000	960	800	[**DEC91b**]
DECStation 5000/240	R3000	640[10]	640	[**DEC91a**]
DEC 3000 Model 400	Alpha (21064)	225	225[11]	[**DEC92a**]
DEC 3000 Model 500	Alpha (21064)	200	200[11]	[**DEC92b**]

Table 2: 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 and using an ASIC to allow a higher rate switch fabric, this performance is achievable on a DAN. The comparison with the Alpha workstations is a little unfair since these use a 32 *byte* wide memory bus and are aggressively

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

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

optimised.

4.6.2 Multimedia streams and the cache

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. The implementation of the demonstrator is based on a crossbar switch element, so traffic was generated contending for the same crossbar output as the memory server. This traffic was marked so that it was discarded by the hardware of the memory server with no performance impact.

The interference traffic used came from both an ATM camera video source and from a traffic generator. In general the experiments[12] showed the effects of the conflicting stream on the cache service time was mainly influenced by the scheduling strategy of the interconnect. For example in one experiment using video traffic with a peak rate of 54Mbps at the same priority as the cache lines, there was has *no effect* on the cache line fetch time – while at first surprising this result can be explained by the round robin scheduling within the switch fabric.

A more interesting result was found by using a traffic generator to create a interfering stream with higher priority at the contention point. 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. Clearly, with no gap the cache request is unable to get through the fabric and the CPU is unable to access memory. The results are shown in figure 5.

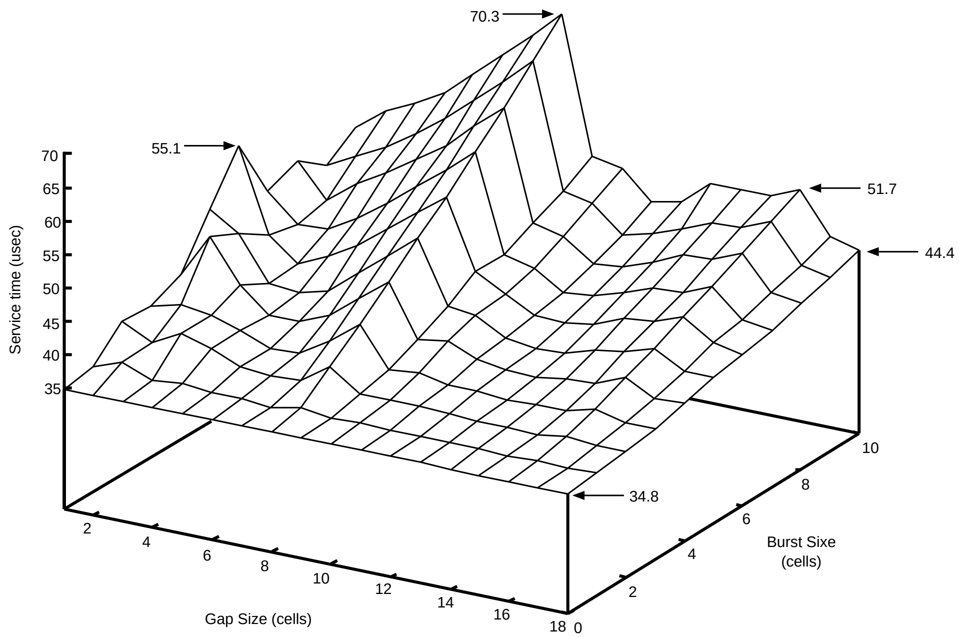

Figure 5: Service Time with High Priority Contention

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

[12]These experiments are fully reported in [**Hayter93**]

to the CPU speed. Therefore eight cell times after a request there is very likely to be another request. Hence 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.

The conditions for this experiment are clearly contrived, especially marking the contending stream as higher priority than the CPU cache line access, but it serves to show that with a little thought about the scheduling used on the interconnect there is no obstacle to simultaneously carrying both multimedia and cache traffic on a DAN.

The principal gain from using caches is that the data currently being manipulated and the code being executed are both found in fast memory close to the processor. To obtain the same benefit for streams of multimedia data, use can be made of the close connection between the cache and the DAN – stream data from devices can be placed directly in the cache as it arrives. In this case, unlike the usual behaviour of a cache, data appears regardless of whether it has been previously accessed. The CPU therefore has fast access to the data as soon as it arrives, if the data is not yet present a cache-miss is seen and the processor stalls. The operating system on the machine must deal with two other cases: "data in the past", where the access is to part of the stream which has been lost from the buffer; and "data in the future", where the access is to data which will not arrive on the stream for some time. The first of these should be raised as an to the application as an exception, since it indicates that the process is unable to keep up with the incoming data rate. The second is likely to occur frequently if only parts of a frame are of interest, and should result in the processor being rescheduled. This idea is explored more fully in [**Hayter94**].

An example use for this system is in what we term a "Spot the Ball" type problem. This is where the task is attempting to locate some feature in the incoming stream and track it from frame to frame. For example, the ball in a football match, the speaker's head in a seminar, or the probe on the end of a robot arm. In all these cases hints from previous frames may be used, and it is likely that only a small amount of the current frame will be examined. However, the access may be fairly random — depending on the search algorithm used. Implementation of a simplified "Spot the Ball" has been used to investigate the working of this system.

5 Conclusion

The Desk Area Network demonstrator built has provided a test-bed for the various DAN ideas. It has shown that interfacing to an ATM based interconnect is no more complex than to a standard bus, and that the DAN is capable of effectively supporting both multimedia and cache traffic.

Further work is underway to build an operating system and windowing system for the DAN, that makes greater use of its various properties.

6 Bibliography

References

[**Adam93**] 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. (p 10–2)

[**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 10–12)

[**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. (p 10–12)

[**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. (p 10–3)

[**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 10–6)

[**ATMUNI93**] *ATM User-Network Interface Specification Version 2.1.* ATM Forum Technical Committee, Signalling Subworking Group Draft, May 1993. (p 10–1)

[**Black94**] RJ Black, IM Leslie, and DR McAuley. *Experiences with Building ATM Switches for the Local Area*, 1994. Submitted to ACM SIGCOMM'94. (p 10–5)

[**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. (p 10–3)

[**CCITT90**] CCITT. *Broadband Aspects of ISDN.* CCITT Draft Recommendation I.121, May 1990. (p 10–1)

[**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 10–2)

[**DEC91a**] DEC. *DECStation & DECSystem 5000 Model 240 Technical Overview*, 1991. DEC Order code EC-N0194-51, availible by ftp from `gatekeeper.dec.com`. (p 10–13)

[**DEC91b**] DEC. *Maxine System Module Functional Spec*, rev 1.2 edition, February 1991. (p 10–13)

[**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 10–13)

[**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 10–13)

[**Finn91**] G Finn. *An Integration of Network Communication with Workstation Architecture.* ACM SIGCOMM, Computer Communications Review, 21(5), October 1991. (p 10–2)

[**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 10–3)

[**Glauert93**] Tim Glauert and Stewart Wray. *Medusa*. Presentation given to the Cambridge University Computer Laboratory Systems Research Group, May 1993. (p 10–4)

[**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 10–1)

[**Hayter92**] 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 10–13)

[**Hayter93**] MD Hayter. *A Workstation Architecture to Support Multimedia*. Technical Report 319, Cambridge University Computer Laboratory, November 1993. Ph.D. dissertation. (pp 10–9, 10–14)

[**Hayter94**] MD Hayter and DR McAuley. *Multimedia Streams and the CPU cache*, February 1994. Submitted to ASPLOS'94. (p 10–15)

[**Hill88**] Mark D Hill. *A Case for Direct Mapped Caches*. IEEE Computer, December 1988. (p 10–12)

[**Hopper90**] A Hopper. *Pandora - an experimental system for multimedia applications*. ACM Operating Systems Review, 1990. (pp 10–5, 10–10)

[**Leslie91**] IM Leslie and DR McAuley. *Fairisle: An ATM network for the local area*. Proc. ACM SIGCOMM, September 1991. (pp 10–3, 10–5)

[**NodeChip92**] *NodeChip Functional Specification*. Dolphin SCI Technology Information Sheet, 1992. (p 10–5)

[**NodeChip93**] *DST511A CMOS NodeChip*. LSI Logic and Dolphin SCI Technology Advance Information Data Sheet, August 1993. (p 10–5)

[**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 10–1)

[**Pfeifer93**] T Pfeifer. *Micronet Machines. New Architectural Approaches for Multimedia End-Systems*. In Proceedings of 4th International workshop on OS support for Digital Audio and Video, November 1993. (p 10–2)

[**Przybylski88**] Steven Przybylski, Mark Horowitz, and John Hennessy. *Performance Tradeoffs in Cache Design*. In Proceedings 15th Annual International Symposium on Computer Architecture, 1988. (p 10–12)

[**Smith82**] Alan Smith. *Cache Memories*. Computing Surveys, 14(3):473–530, September 1982. (p 10–12)

[**Smith87**] Alan Smith. *Line (Block) Size Choice for CPU Cache Memories*. IEEE Transactions on Computers, C-36(9), September 1987. (p 10–12)

[**Tennenhouse89**] DL Tennenhouse and IM Leslie. *A Testbed for Wide Area ATM Research*. In Proc. ACM SIGCOMM, September 1989. (p 10–3)