MITSUBISHI ELECTRIC RESEARCH LABORATORIES
http://www.merl.com

Network Latency in DART and Windows NT

John Howard, Neil McKenzie, Olivier Voumard, Ross Casley

TR98-03a December 1998

Abstract

This paper reports on I/O latency measurements made using an experimental network adapter based on Mitsubishis M65433 ĎARTc̈hip. DART has achieved its design goal of very low application-to-application latency by mapping buffers and ring queues directly into virtual memory and thus avoiding all operating system latency.

MERL – A Mitsubishi Electric Research Laboratory

Performance of the M65433 "DART" ATM Network Interface

John H Howard
Olivier Voumard
Neil McKenzie
Ross Casley (VSIS, Inc)

TR98-03a July 21, 1998

Abstract

This paper reports on I/O latency measurements made using an experimental network adapter based on Mitsubishi's M65433 "DART" chip. DART has achieved its design goal of very low application-to-application latency by mapping buffers and ring queues directly into virtual memory and thus avoiding all operating system latency.

Revision History:

1. Initial version, May 22, 1998
2. Greatly expanded, June 25, 1998
3. Re-format charts, July 21, 1998

Introduction

The M65433 DART[1] ATM Network Interface chip integrates ATM segmentation and reassembly functions with a PCI interface and various innovative features for a very low latency host interface, traffic shaping, and message processing. It is designed to work together with a M65434 SONET STS-3C Framer Transmission Convergence chip and external RAM on a local bus controlled by the M65433 to make a PCI-based network interface card, or NIC.

Specific performance-related DART features include:

- Onboard virtual address translation, allowing the I/O adapter and the application to communicate directly through application virtual memory.
- A ring queue architecture for flexibly transferring empty and full buffers between application and adapter. Both the buffers and the ring queues themselves occupy the shared virtual memory space. It was hoped that this architecture would allow application to application latencies comparable to network hardware latencies by bypassing all software bottlenecks, and at the same time support full network bandwidth for bulk data transfers. We had set a latency target of 20 microseconds.
- Ability to map certain I/O adapter registers into the application address space, further facilitating direct communication between adapter and application (this feature was not implemented perfectly in DART, but we were nevertheless able to evaluate its performance by temporarily allowing a security hole in the driver.)
- A host request mechanism which allows the host system's CPU to handle address translation faults and certain other adapter events efficiently in the interrupt handlers. It was specifically hoped that the HRQ mechanism could support ATM ABR resource management cell processing in software with a host system overhead of no more than 10%.
- A "message processing" co-processor which flexibly allows hardware-assisted transactions between two host systems. We did not evaluate this feature due to lack of resources.

An initial prototype interface board was developed and evaluated at Vsis, Inc in Sunnyvale at the end of 1997, then delivered to MERL in February 1998 for further performance evaluation. A number of board problems were discovered, leading us to redesign the board. Performance measurements made with the original boards were re-validated and extended using the new boards.

Performance results are divided into two parts; those related with the fundamental timing of the chip itself, and those related to the success of the DART architecture in the context of a host operating system, Windows NT 4.0.

It should be emphasized that resources were too limited to perform a comprehensive performance or functional evaluation. We have not evaluated message processing at all, have barely touched the traffic management features, and have not attempted a conventional NDIS driver. Nevertheless, we feel confident that DART meets its basic objective of supporting direct application access to a network adapter with application-to-application latencies less than 20 microseconds and throughput approaching the theoretical maximum. RM cell processing overhead can be as high as 30%, but is likely to be substantially less than 10% in normal operation.

Network latency

Low latency has been a primary goal of most recent network architecture research. It is generally recognized that improving throughput by providing more buffering comes at the cost of increasing latency as well. On the other hand, if we can decrease latencies we can not only improve throughput without additional buffering, but at the same time reap other benefits such as improved real-time performance and efficient processing of client-server traffic, and parallel processing using networks of workstations.

Architectures (for example Application Data Channels[2], Hamlyn[3], and U-Net[4]) for achieving low latency generally seek to avoid data copying by receiving and transmitting data directly from buffers located in the application's virtual address space. This requires mechanisms to:

- use virtual rather than physical addresses in the network adapter (DART incorporates a TLB like that in U-Net[5] for this purpose)
- exchange full and empty buffers between kernel and application with little or no kernel involvement (DART incorporates five ring queues for this, for each of two different applications, and also can give the applications direct access to adapter registers)
- demultiplex incoming messages "on the fly" in the adapter in order to deliver them to the correct application (DART uses the connection-orientation of ATM to associate VC's with applications.)

The DART architecture was presented previously. This paper reports specifically on performance obtained from the M65433 chip which implements that architecture. Not surprisingly, that performance is remarkably sensitive to the properties of the PCI bus. Moll and Shand[6] present some relevant insights into PCI bus performance and NT interrupts.

Base Hardware Performance

Experimental Setup

The test platform consisted of two Pentium workstations running Windows NT Workstation 4.0, service pack 3. The two workstation were connected to MERL's internal Ethernet. Measurements were made using an ATM fiber either looped back to a single workstation or directly connected between the two. Additional details are given in Table 1.

	266 MHz workstation	**300 MHz workstation**
Motherboard	Intel PD440FX	Intel AL440LX
Bios	AMI 1.00.05.DT0	Phoenix 4.0 release 6.0
Chipset	82440FX PCI set	82440LX AGP set
Processor	266 MHz Pentium II®	300MHz Pentium II®
Number of CPU(s)	1	1
Primary Cache	32 KB	32 KB
Secondary Cache	512 KB	512 KB
Memory	128 MB SDRAM	64 MB SDRAM
Graphic card	ATi Rage Pro Turbo	Matrox Millennium II AGP
Hard drive	Maxtor 85108 A8, 4.3GB	Quantum Fireball ST2.1A, 2.1GB
Ethernet adapter	3Com EtherLink III PCI (3C590)	3Com Fast EtherLink XL (3C905)
ATM adapter	DART	DART

Table 1: Measurement platform

The measurement software consisted of an elementary device driver, plus an application program which did the actual measurements. The driver took advantage of the adapter's flexible addressing capabilities and of various memory mapping mechanisms provided by Windows NT to expose adapter registers and local memory directly to the application program, and also to expose application virtual memory to the adapter. Once all of this was set up, the device driver got out of the way. The test application program controlled the adapter directly to send and receive data.

Time Stamp Counter

Timing measurements used the Pentium's Time Stamp Counter. This 64-bit counter increments once per processor clock cycle and is readable by any user- or kernel-mode program. In a few cases we also used an

oscilloscope to measure intervals between certain physical events such as cell framing signals in the UTOPIA physical interface between the SAR and TC chips. Reading and storing the Time Stamp Counter require about 0.17 μsec (44 cycles) on the 300 MHz platform, which is noticeable at the resolution we are using. Including additional code to scale the value read and store it in a histogram raised the measurement overhead to 0.57 μsec (170 cycles).

Figure 1 gives a histogram of the intervals between successive TSC readings and shows that there are several noise sources in TSC measurements. The very strong peak (99.99 percent of the samples taken) at 0.57 μsec represents the usual case of no interfering events between two successive Time Stamp Counter readings. There is a triplet of secondary peaks at about 3.4, 4.1, and 5.1 μsec, which we speculate are due to timer interrupts (for more detail see the section on "Interrupt processing overhead".) These peaks were present even with the network and as many devices as possible disabled and the measurement program run at a high priority. Additional samples scattered up to and past 30 μsec probably represent other operating system interrupts. There were some samples larger than 33.3 μsec which were discarded in this histogram.

The various peaks in the histogram are spread out somewhat, indicating the presence of jitter in the timer measurements. Jitter sources include SONET framing, internal polling loops in both the chip and the measurement program, processor cache misses, and operating system overhead. Since there may be jitter in both starting and ending times, it is important to look at multiple samples and disregard noise when using the Time Stamp Counter.

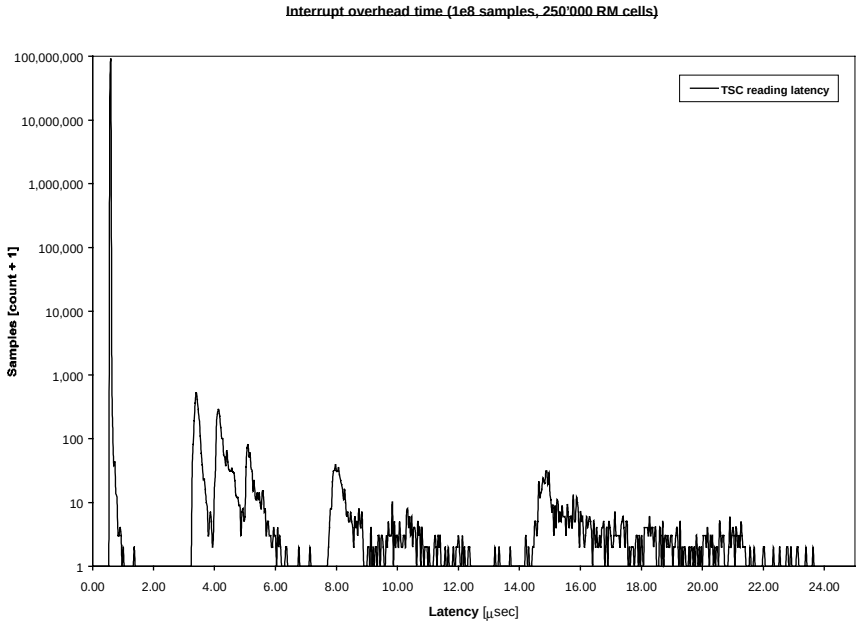

Figure 1: Time Stamp Counter baseline distribution

Latency for short frames

Latency is the time period starting when the application program enqueues data be sent (TXin entry stored by application) and ending when it is fully delivered to the receiving buffer and the receiving application is notified (RXdone entry stored by NIC). We measured the interval as the difference between Time Stamp Counter readings immediately before setting the TXin entry and immediately after observing the creation of the RXdone entry, using a wait loop which polled the RXdone queue. No I/O interrupt was used. Time stamp counter intervals were accumulated in a histogram. The starting times were varied randomly to avoid resonances between intrinsic events such as cell times, SONET framing, and built-in polling loops.

Figure 2 gives the histogram of the latency to send very short (0 byte) AAL5 frames. The average latency is 17.7 μsec with spread, or jitter, of about 3 μsec.

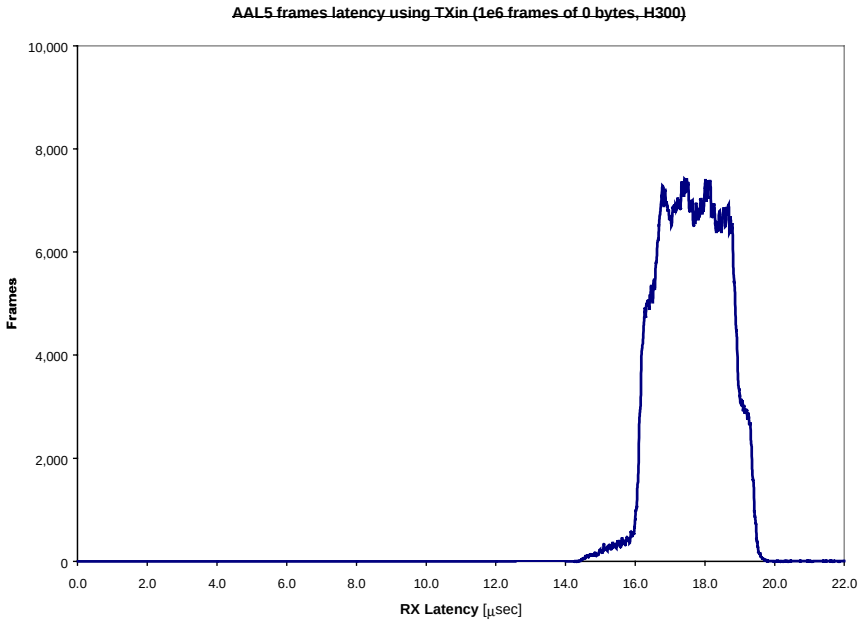

Figure 2: AAL5 frame latency

Latency breakdown

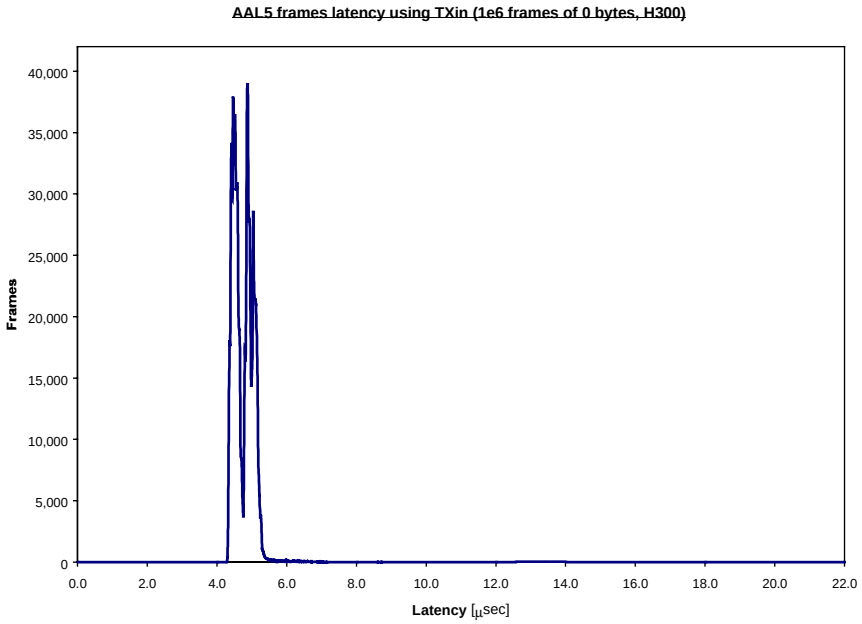

Figure 3: Transmit Latency

Figure 3 shows the time required for transmit processing only, that is, from TXin queue request until the TXdone queue response which indicates that transmission complete. (Note that the M65433 indicates that transmission is complete when the cell(s) to be transmitted have been fetched from main memory, not when it is actually sent.) The mean is 4.8 μsec with a jitter of about 1 μsec. The distribution has two peaks separated by about 0.5 μsec.

Oscilloscope measurements show a physical transmission time of about 7.2 µsec, measured from the UTOPIA TxSOC (start-of-cell) signal to the corresponding RxSOC. Additional oscilloscope measurements using a mixture of software and hardware markers produced an estimate of 8.1 µsec from the time the TXin queue entry was created until the TxSOC signal is observed. Combining, we estimate a latency budget of:

time [µsec]	Activity
4.8	TXin to TXdone (send overhead)
3.3	TXdone to TxSOC (transmit FIFO)
7.2	TxSOC to RxSOC (physical latency)
2.4	RxSOC to RXdone (receive overhead)
17.7	TOTAL

Table 2: Latency breakdown (AAL5 frames)

Raw Cell Latency

The M65433 also supports a special "raw cell" mechanism intended to bypass normal scheduling and deliver cells directly to the output FIFO.

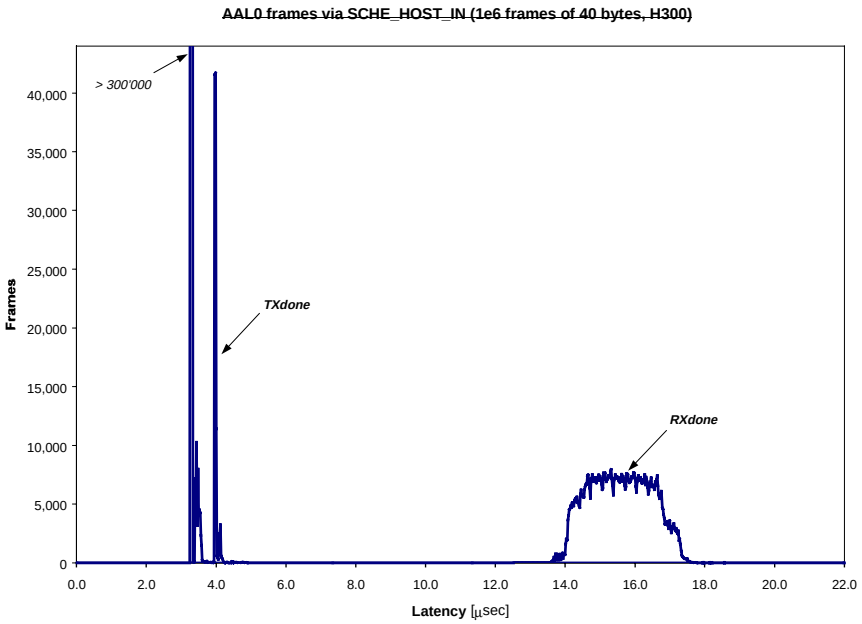

Figure 4: Raw cell latencies

Figure 4 shows latencies for the raw cell mechanism. In this figure the transmission and reception histograms are superimposed. The left peak represents the sending time (measured from when the SCHE_IN_HOST register is set until the location specified by the SCHE_IN_NOTIFY_P register is cleared), while the right peak represents full transmit latency, from SCHE_IN_HOST until the RXdone queue entry is created. In comparison with the AAL5, latencies decreased by 2.3 µsec and the jitter was unchanged. The 2.3 µsec difference appears to be entirely attributable to TXin/TXdone processing overhead.

Latency and throughput as a function of payload size

Latency increases linearly with payload size, as shown in Figure 5. The slope is 2.852 µsec per cell, close to the theoretical time of 2.83 µsec to send a cell. Figure 6 presents throughput as a function of frame size, contrasting it with the theoretical maximum of 135.63 Mbit/sec. For large frames the throughput approaches 95% of the maximum.

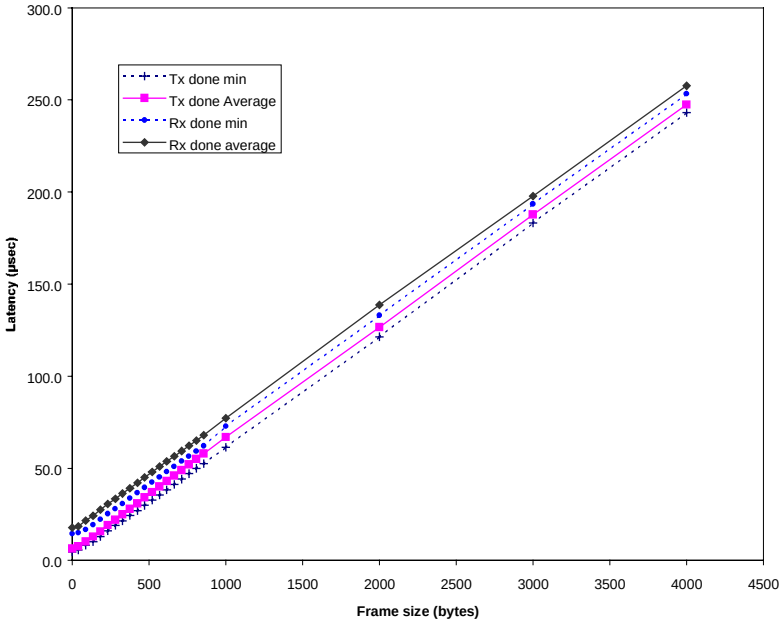

Figure 5: Latency as a function of payload size

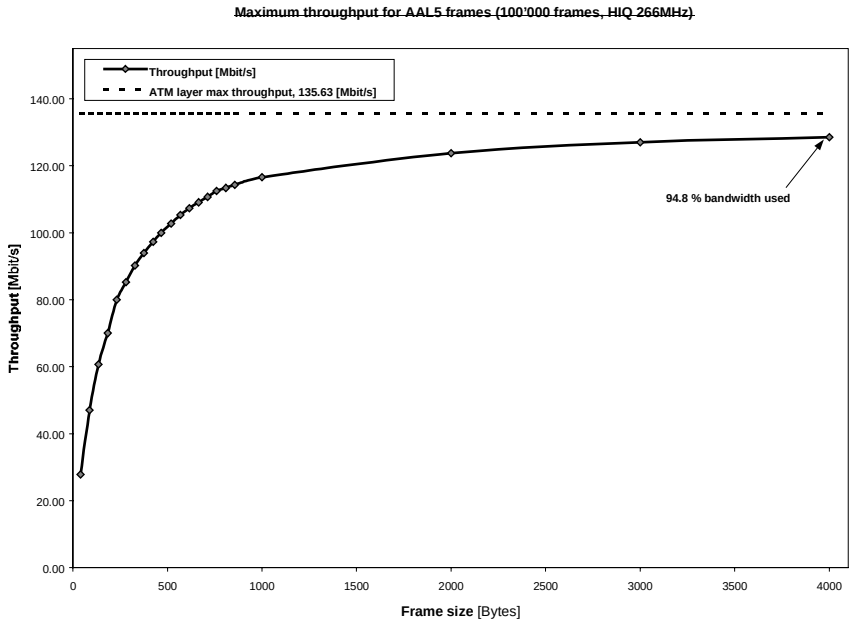

Figure 6: Throughput as a function of frame size

Two-machine round trip latency

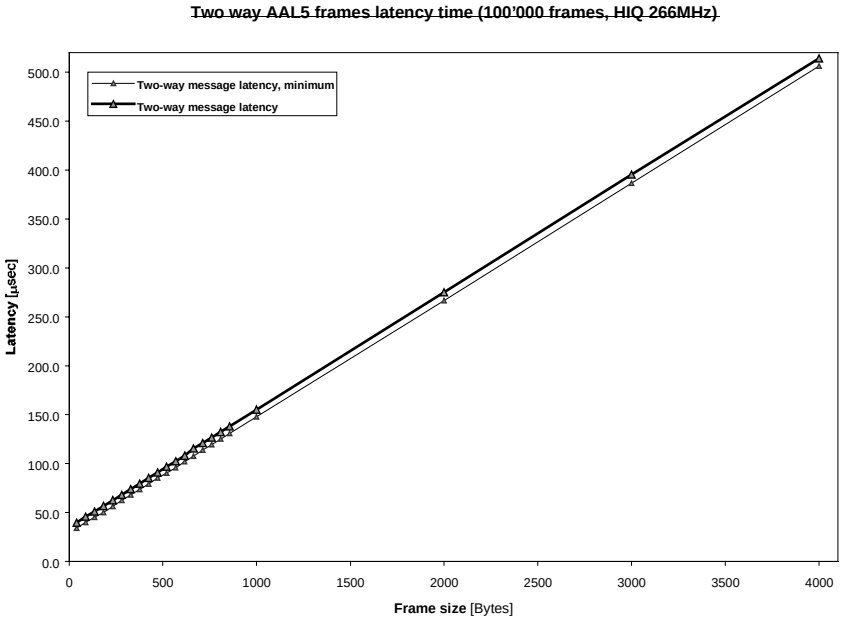

Figure 7: Two machine latency

Finally, we measured round-trip latency between two systems. In this case, the application program in the second system returns incoming messages to the originator, which receives them and measures total round-trip time. Total round trip time, not counting processing time in the remote machine, is reported in Figure 7. For a payload size of 40 bytes the round trip time is 39.7 µsec.

Software performance implications

A key goal in DART's Direct Access Architecture is to minimize latency and overhead by bypassing the operating system kernel whenever possible. This section presents measurements of various kernel functions on our host platform and relates them to the relevant DART features.

I/O Control Overhead

Table 3 presents the time to perform system I/O calls (IOCTL operations) and contrasts them with direct access to the adapter.

Operation performed (with measurement overhead removed)	time [μsec]	average [cycles]	minimum [cycles]
Measurement overhead	0.17	47	44
Direct read of adapter register by application	0.43	127	121
Direct write of adapter register by application (including LOCK OR to force synchronization)	0.32	95	91
Null IOCTL (serviced in dispatch routine)	8.79	2637	2552
IOCTL w/parameters, set & test one register	13.85	4157	4045
Dummy I/O (serviced through startIO & DPC)	17.05	5115	5023
Dummy I/O w/parameters, set & test register	19.54	5862	5704

Table 3: Basic timing measurements, 300 MHz platform

It takes 0.17 μsec to read and store the Time Stamp Counter, even without accumulating a histogram. This overhead value is subtracted from the remainder of the readings in the table. The register read and write depended on Windows NT's ability to map PCI addresses into the virtual address space. It is necessary to include a serialization instruction (LOCK OR) after a register write in order to avoid caching, and also to help deal with a bug in the chip's PCI interface. The null IOCTL simply returns as quickly as possible from the driver's dispatch routine; the IOCTL with parameters sets and tests a single adapter register, and the dummy I/O is actually a null IOCTL which is serviced through Windows NT's standard Start I/O and completion routines without any actual I/O.

If one assumes that one register read and write (totaling 0.6 μsec) may be needed to start an I/O directly, then direct register access is more than 20 times faster than a conventional start I/O. DART needs no register operations to send a frame, but it does need one register write to for the application to notify the adapter that RXfree and TXdone queue entries can be re-used. These writes can be amortized over several frames. If the registers are not directly accessible by the application, then there may still be some gain to be obtained by initiating physical I/O from the dispatch routine rather than a start I/O, but it is not nearly as large as with direct register access.

Interrupt processing overhead

The time to process an interrupt was measured by comparing two histograms of time intervals, a baseline and a second one with a source of interfering interrupts. Figure 8 overlays the baseline histogram (previously presented in Figure 1) with the output of the same measurement program run in the presence of interrupts caused by a stream of Resource Management (RM) cells provided by a second machine across the ATM link. These RM cells are processed through the Host Request Queue directly in the driver's Interrupt Service Routine without any higher-level interrupts or notification.

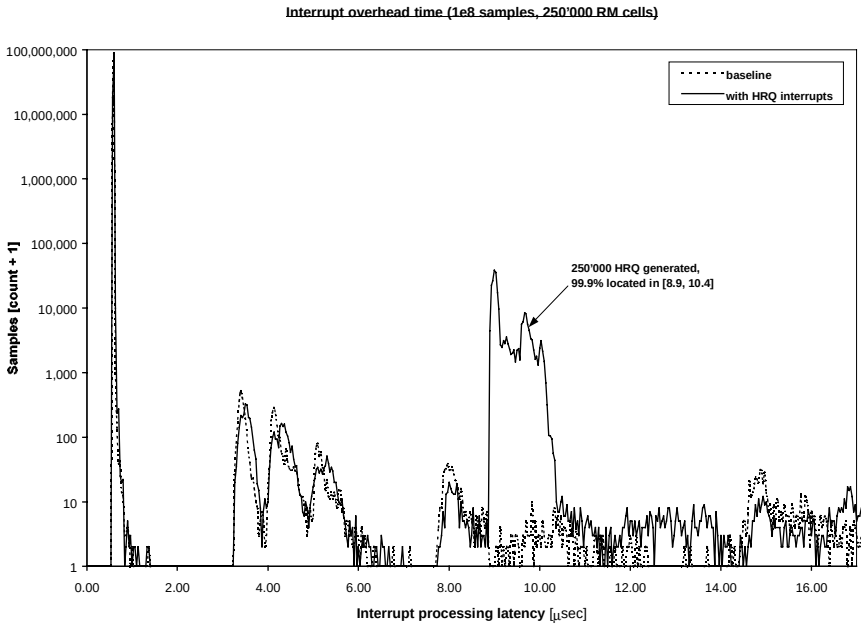

Figure 8: Interrupt processing overhead

The overlay shows a clear additional peak beginning at 8.9 µsec and containing about 249K events (we generated 250K cells.) Subtracting a 0.4 µsec measurement overhead (including histogram accumulation), this indicates that it takes about 8.5 µsec to accept, process, and return from a HRQ interrupt. This is consistent with our estimate of 6 to 10 µsec to process an interrupt, made while originally designing the HRQ mechanism[1]. Welsh[7] reports an interrupt entry overhead of about 3 µsec on a 133MHz Pentium running NT, measured in software using the TSC, and Moll and Shand[8] report about 5 µsec latency measured by hardware to enter the ISR on a 200 MHz Pentium Pro.

ATM traffic management using the ATM Forum's ABR (Available Bit Rate) service involves generating one RM cell out of every 32 cells sent. The DART architecture specifies three HRQ events for every RM cell: one just after the RM cell is generated, one at the receiving end to turn the RM cell around, and a third at the sender to receive the returned RM cell. If each of these generated a separate HRQ interrupt we would spend 26.7 µsec for every 32 cells, and if all the traffic were ABR (unlikely) running at full speed, this overhead would be incurred once every 90 microseconds, for a 30% overhead, on our current platforms. This is above our target of 10%. However, DART also has a mechanism for batching HRQ requests by imposing a minimum interval between interrupts, at a cost of somewhat longer latency. We estimate that it would take about 17 microseconds to process a batch of three HRQ requests, reducing overhead to below 20%. Faster CPU's will eventually bring this number into the acceptable range, and in addition the 100% ABR scenario is unlikely.

Conclusions

One of the basic assumptions of DART was that a zero copy architecture could produce very low latencies. DART exceeds its latency goal of 20 microseconds by a comfortable margin: small frames sent using the standard (AAL5) mechanism had an application to application latency of 17.7 microseconds (with a jitter of 3 microseconds.) Sending a single cell with the raw cell mechanism was even faster, with an average latency of 15.4 microseconds.

Throughput approached 95% of the theoretical maximum of 135.63 MBps for large frames.

We did not quite attain our goal of 10% overhead for processing RM cells using the host CPU. The worst case overhead could be as high as 30%, which we can reduce to 20% by imposing a small delay in responding to RM cells. However, this overhead occurs only if the network is fully saturated with ABR-only traffic, and drops in proportion to the amount of ABR traffic present.

[1] R. Osborne, Q. Zheng, J. Howard, R. Casley, D. Hahn, and T. Nakabayashi, *DART - A Low Overhead ATM Network Interface Chip*, **Hot Interconnects IV: A Symposium on High Performance Interconnects**, Stanford University, Palo Alto, CA, August 1996. MERL Technical Report TR96-18.

[2] Druschel, P., *Operating System Support for High-Speed Communication*, **Comm ACM 30**, 9, September 1996, pp 41-51.

[3] Buzzard, G., Jacobson, D., Marovich, S., & Wilkes, J., *Hamlyn, a high-performance network interface with sender-based memory management*, **Hot Interconnects III,** August 1995. HP Labs Technical Report HPL-95-86, http://www.hpl.hp.com/techreports/95/HPL-95-86.html.

[4] Welsh, M., Basu, A., & von Eicken, T., *ATM and Fast Ethernet Interfaces for User-level Communication*, **Third International Symposium on High Performance Computer Architecture,** Feb 1997. http://www.cs.cornell.edu/U-Net/.

[5] Welsh, M., Basu, A., & von Eicken, T., *Incorporating Memory Management into User-Level Network Interfaces*, **Hot Interconnects V,** August 1997.

[6] Moll, L. & Shand, M., *Systems Performance Measurement on PCI Pamette*, **Symposium on Field-Programmable Custom Computing Machines** (FCCM 97), April 1997.

[7] Welsh, M., Basu, Anindya, and von Eicken, T., Incorporating Memory Management into User-Level Network Interfaces, *Hot Interconnects IV*, Stanford University, August 1997.

[8]Moll, L., and Shand, M., Systems performance measurement on PCI Pamette. *FPGAs for Custom Computing Machines (FCCM'97)*, IEEE, April 1997.