A Study of I/O Architecture for High Performance Next Generation Computers

Anurag Sah[1,2]
Dinesh C Verma[1,2]
Vojin G Oklobdjiza[1]

TR-91-008

January, 1991

Abstract

We describe an I/O architecture for a high performance next generation computer. The architecture proposed in this paper makes special provisions for communication networks. In order to allow for the expected multi-media and time-critical components of future computer usage, we propose the concept of *logical buses* which gives the illusion that there are a number of dedicated buses between the components of a system. A logical bus has a number of performance parameters associated with it, and the system architecture ensures that the performance parameters for each logical bus are satisifed during the operation of the system.

This research has been supported in part by AT&T Bell Laboratories, Hitachi, Ltd., the University of California under a MICRO grant, the National Science Foundation under Grant No. CDA-8722788, and the International Computer Science Institute. The views and conclusions in this document are those of the authors and should not be interpreted as representing official policies, either expressed or implied, of any of the sponsoring organizations.

[1]University of California at Berkeley, Berkeley, California 94720 USA

[2]International Computer Science Institute, 1947 Center Street, Berkeley, California 94704-1105 USA

1. Introduction

Computer Architecture has almost been synonymous with processor design in recent days, and very little mention of I/O archtecture design is found in literature. This is at odds with the fact that I/O comprises a large part of the total work performed by a computer system. Part of the explanation is that most peripherals, as a matter of fact all peripherals in the present-day computers are very slow when compared to the CPU's/memory's capacity to handle data. Thus a uniform interface similar to the channels used in IBM System 360 appeared sufficient to take care of them.

In the near future, the situation with regards to the peripherals is very much likely to change. There are two main reasons for this, first the traditional role of the computer as a data-cruncher is no longer true. The modern computer is more likely to participate in user interaction through voice, video (commonly called multi-media) than any of its ancestors. The peripherals used in these interactions have speed requirements which are comparable to the rate at which the CPU demands data/instructions from the memory. In many cases these devices require data rates in excess of what a typical CPU would require of the memory.

Future systems will therefore have some interfaces which will require high bandwidths and also some whose requirements would be minimal. Applications running on some components, e.g., video or multi-media displays, have rather strict requirements on the delays data towards (or from) that component must have. If simulation results from a remote site are displayed on a graphics terminal, and the processing unit requires to read some information from the disk, the quality of the display can be affected significantly due to interference. We believe that a good I/O architecture must provide mechanisms to support all these components with acceptable performance. In short, any future system must provide for a wide diversity in throughput and performance requirements by individual system components. An architectural concept insulating the data communication of the different components of a computer will therefore prove to be very useful in future systems.

In the next section, we describe the concept of *logical buses*, our solution to the I/O problem in future systems. Section 3 will describe a possible way to implement this concept The next section will discuss how this architecture can be made to scale according to requirements. Since no physical bus can support an unlimited number of logical buses, we briefly discuss the software tests that can let the operating system know when to stop establishing new logical buses. We present some simulation results and finally discuss the merits and limitations of this scheme and directions for future work.

2. I/O Architecture

The goal in the proposed I/O architecture is to accommodate computer networks within the framework of a traditional computer architecture. We would like to incorporate the network as an integral part of our system. The right way to proceed, in our opinion, is to incorporate good features of a communication network into the I/O architecture of a high performance computer.

The most important function (and some may say the only function) of an I/O architecture is to move data efficiently between the central processor unit and the peripherals. Data transfer is the objective of computer communication networks as well, and we feel that some concepts used in computer communication networks can be ported over to the I/O architecture of computer systems. The proposed I/O architecture attempts to utilize the concept of *virtual circuits*[1] in the design of a traditional computer system. A virtual circuit provides an illusion to two distant users

that they have a dedicated communication link between them, even though they may be sharing the physical resources of the network with many other users. In the I/O architecture of a computer system, such sharing of resources may result in substantial cost benefits. The other alternative would be to provide dedicated hardware links between different components of a system. A good decision about how many and which hardware links to provide can only be taken if we can predict the characteristics of I/O traffic with a reasonable degree of accuracy. In the view of the expected increase of multi-media component of computer communication, traffic characteristics may change so dramatically so as to make the present design inadequate. Thus, we must design a flexible architecture.

Virtual circuits, as used presently, can not be directly used in computer systems because data transmitted on them can experience wide and unpredictable delays. In a network spread over a wide area (or even a local area network), this may be tolerable and even unavoidable, but certainly undesirable for a computer system. In order to make the concept acceptable in the frame-work of a computer system, we must be able to provide acceptable performance on our proposed mode of data transfer. Fortunately, it is feasible to provide such guarantees in computer networks [2][3] where the communication abstraction is called a *real-time channel.* We shall attempt to provide similar abstractions within our I/O architecture.

2.1. Logical Buses

The I/O architecture is based on the concept of logical buses. A logical bus will be a connection between two components of a computer system. Each logical bus will have its own performance attributes. Thus any two components in the I/O architecture will continue to have the performance better than what they desire irrespective of other components sharing the same physical hardware. This feature will prove to be useful when part of the data to be moved by the I/O architecture may have stringent performance requirements, or there may be a wide variety of data traffic. As an example, one may need to have a graphic display of simulations running remotely across a wide area network and to support simultaneously possible accesses to disks and other peripherals. Interactive displays, especially visual displays will have rather strict delay requirements, and need to be given priority over other kind of accesses. However, there may be other kinds of traffic (we do not yet know of) which may have other kinds of requirements. Thus, the performance attributes of a logical bus will need to be specified using a general description. A specification of the maximum possible time that a packet may have on a logical bus seems to be a simple and natural way of specifying these requirements.

For each of the logical buses in the system, we have a unique identifier. We identify a logical bus (also called *session* in this text) consisiting of three parts, the first part denotes the source or the sending module on the session, and the next part denote the receiving module. The remaining part is used for having possible multiple sessions between the same pair of devices. Thus we can have multiple sessions for each pair of devices.

The next section shows a method to implement the abstract notion of a logical bus.

3. Bus Implementation

The bus structure consists of device controllers connected to the bus. These device controllers make requests for obtaining the bus. The arbiter is responsible for handling bus contentions and allocating the bus to one pair of controllers. If a logical bus already exists between the two controllers, packet is transmitted with this logical bus identifier to the receiving controller.

We want our I/O architecture to be compatible with the existing devices. In order to achieve this goal, our device controllers are divided into two portions, a device-specific controller and a device independent controller. The device independent controllers understand the concept of logical buses and session identifiers while the device-specific controller is responsible for the translation of device requests into a format understandable by the raw device.

On the receipt of a packet to be sent on a particular logical bus, the device independent controller (which requires access to the data bus) interrupts the arbiter with a priority depending on the performance requirements of the logical bus. On the acknowledgement of this interrupt, the sender puts the data on the bus specifying which device controller it is meant for.

We now describe the functions of the various components of the bus architecture in alightly more detail. For the sake of simplicity, we will describe a very simple bus organization in this section, and describe how it can be scaled in Section 4.

3.1. Bus Organization

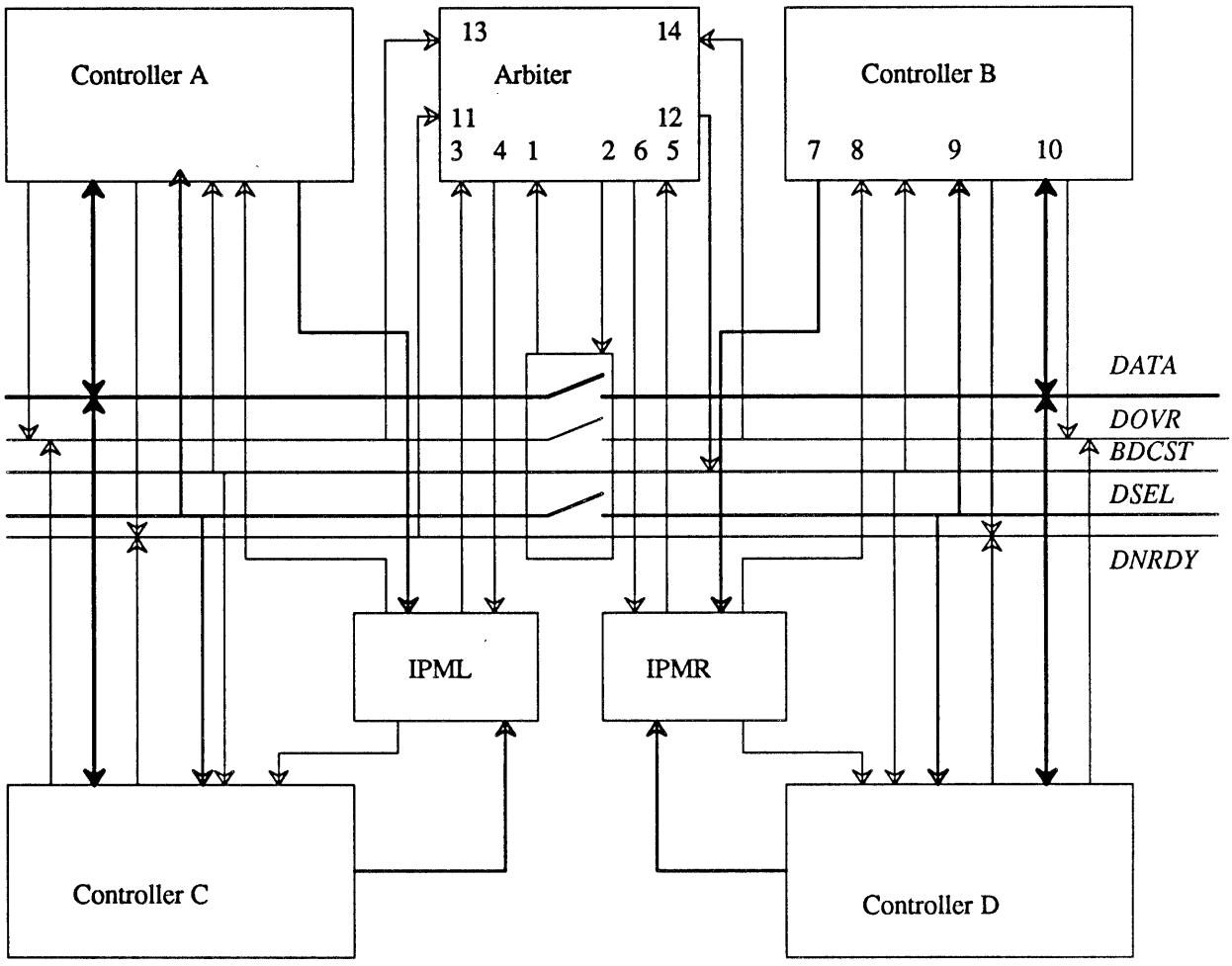
The I/O Bus architecture that we propose (Figure 1), is divided into two parts, which may be connected by means of a switch. When the switch is in an open state, the two sides of the bus may be used independently and concurrently. When a connection needs to be established between two modules, one on either side of the switch, then the arbiter is responsible for closing the switch. We call the two sides of the bus as the E-bus (the external bus) and the C-bus (the core bus). Both the C-Bus and the E-Bus have been provided with their own Interrupt Priority modules, which resolve requests for obtaining the bus based on priority levels of the interrupts generated by the requesting controllers.

The arbiter resolves contention for the C-Bus and the E-Bus. Both Interrupt Priority Resolution Modules send their highest priority interrupts as well as one line for cross-connection. The cross connection bit indicates whether a cross connection is desired or not i.e. a connection between two controllers , one on C-Bus and one on E-Bus.

The arbiter sends appropriate IACK signal to the Interrupt Priority Resolution modules (IPM) when the buses are available. If the arbiter wishes to allow a cross-connection, then it sends control signal to the switch and places it in the ON-state. It then sends the IACK to the IPM of the cross-connection requesting controller. The IPMs forward the IACK to the controller selected on the basis of highest priority. We explain the IPMs in the next section.

3.2. Interrupt Priority Modules

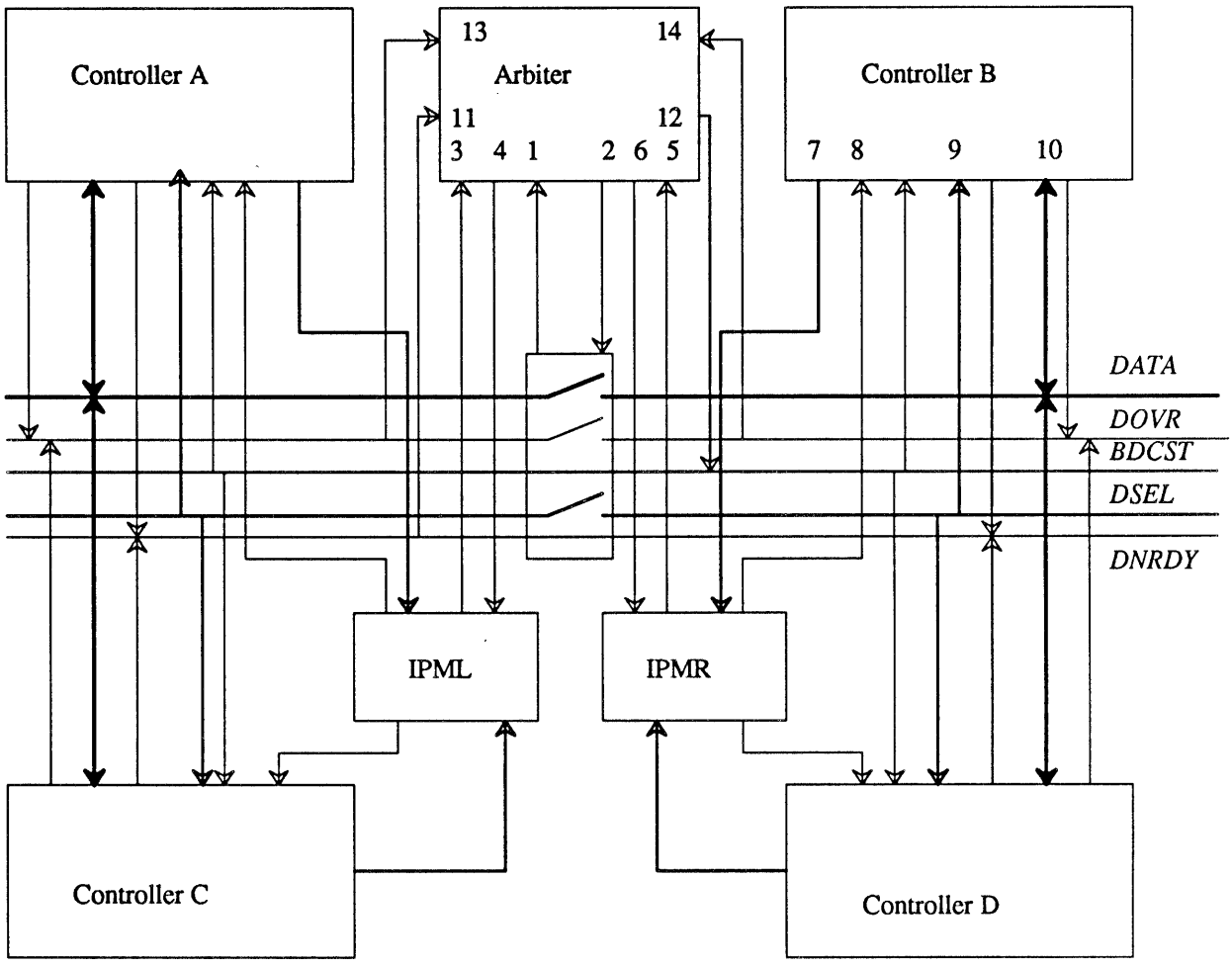
As explained earlier, there are two IPMs , one for the C-Bus and one for the E-Bus. Each is responsible for resolving requests for its bus as well as possible cross-connection, based on the priority of the interrupts generated by the various requesting controllers. The comparator submodule selects the highest priority interrupt and outputs the line number on which this interrupt occurs. The bits of line numbers are used for multiplexing the priority and cross connection bit of the selected interrupt line and sending them to the arbiter. The line number also gets buffered on to an Interrupt Selected Line Stack. This is because the arbiter sends the IACK only when the bus is free. In the meantime other interrupts may be generated. When the IACK arrives from the arbiter, the line number at the top of the stack is chosen to demultiplex the IACK on to the appropriate IACK line (to the different device controllers).



1 SSTAT - Switch Status
2 SCNTRL - Switch Control
3 INTL - Interrupt Left
4 IACKL - Interrupt Ack. Left
5 INTR - Interrupt Right
6 IACKR - Interrupt Ack. Right
7 INT - Controller Interrupt
8 IACK - Controller Ack.
9 DSEL - Device Select
10 DATA - Data Lines
11 DNRDY - Device Not Ready
12 BDCST - Broadcast Indicator Line
13 DOVRL - Data Over Left
14 DOVRR - Data Over Right

IPML - Interrupt Priority Module Left
IPMR - Interrupt Priority Module Right

Figure 1



1 SSTAT - Switch Status
2 SCNTRL - Switch Control
3 INTL - Interrupt Left
4 IACKL - Interrupt Ack. Left
5 INTR - Interrupt Right
6 IACKR - Interrupt Ack. Right
7 INT - Controller Interrupt
8 IACK - Controller Ack.
9 DSEL - Device Select
10 DATA - Data Lines
11 DNRDY - Device Not Ready
12 BDCST - Broadcast Indicator Line
13 DOVRL - Data Over Left
14 DOVRR - Data Over Right

IPML - Interrupt Priority Module Left
IPMR - Interrupt Priority Module Right

Figure 1

There are two instructions for I/O on the internal bus. IN moves data from devices on internal bus to a register of the CPU. OUT moves data from a register of the CPU to devices on the internal bus. The DMA module has been entrusted with more functions than is currently done. It is responsible for generating control sequences for programming the device controller registers (on control sessions). The CPU just programs the registers of the DMA module. The DMA module generates the control sequence from this point onwards while the CPU can be busy with its own work.

4. Scalability

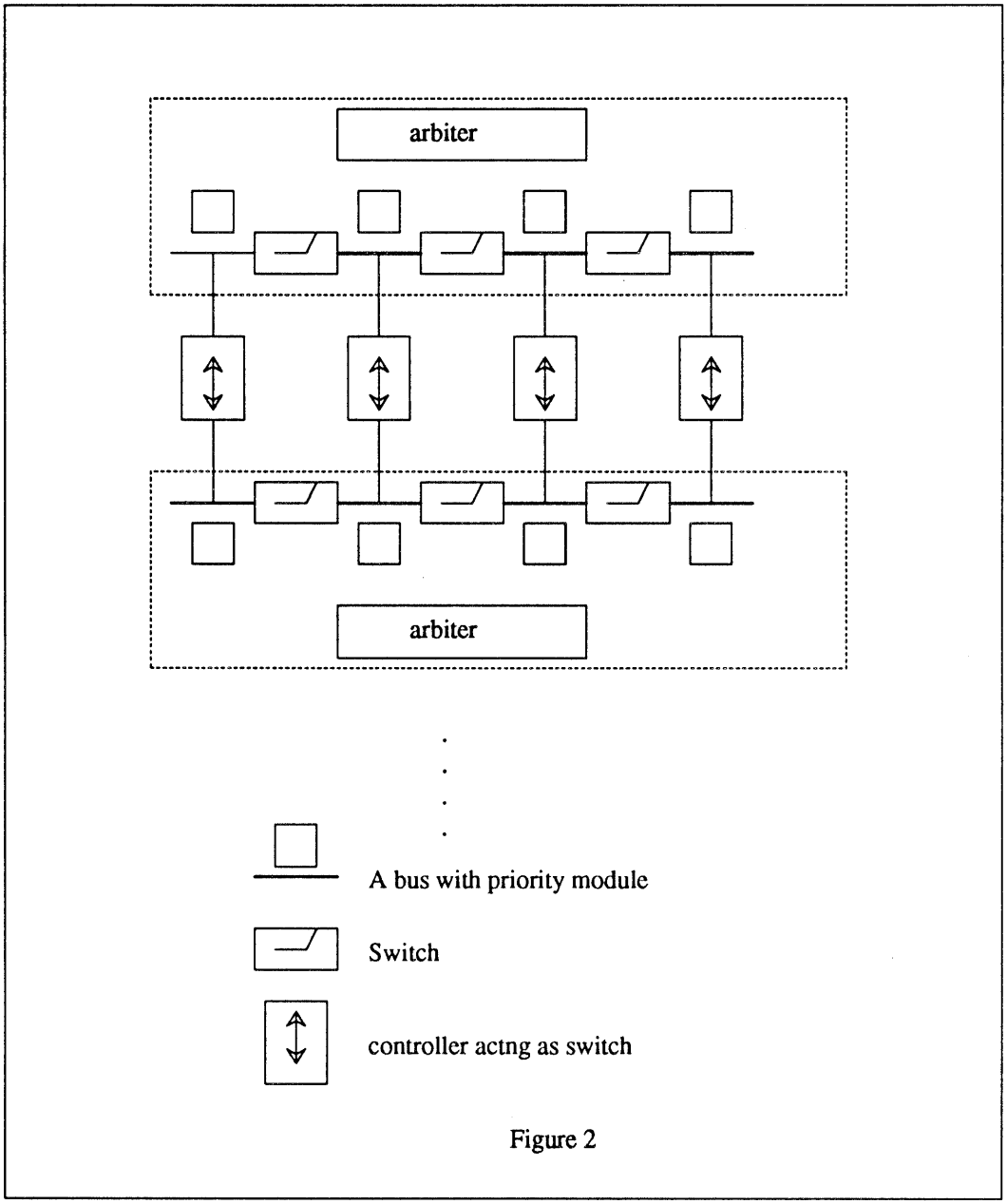


Figure 2

An I/O architecture for the future systems must be scalable. At the same time, it should allow for different devices communicating across the bus to have as high a bandwidth as possible. Our I/O architecture can be looked upon as a series of buses which are connected by a

series of switches. In the detailed example of Section 3, we had two buses (the C-Bus and the E-Bus) which operated independently unless devices wished to communicate across the switch. At the cost of complicating the arbiter logic, we can extend our scheme to accommodate 4 buses, connected together by means of 3 switches. Devices on the same bus can use the full bandwidth of the system. If they wish to communicate on a device on another bus, we would need to turn on some switches,

The basic buses (like the C-Bus or E-Bus) can be arranged in horizontal and vertical levels (Figure 2). Let us call all the basic buses at the same horizontal level as a bus group. Each bus group can have a maximum number of device controllers, limited only by the number of bits in session identifier. However, these devices can be placed on a number of different basic buses. Each bus group has a arbiter which is responsible for granting access across the switches in that group. Each basic bus in a bus group has its own interrupt priority module.

For communication between bus groups, we can develop a similar arbiter. However, we can simplify the architecture by using a controller to act as a switch between different bus groups. This allows the number of bus groups in the system to grow indefinitely while each bus group is limited to thirtytwo devices.

Buses in the same bus group have identical bandwidths, while buses in different groups can have different bandwidths. Thus network controllers can be placed in one bus group, while terminals can be placed in another bus group and connected to the first by means of terminal controllers.

Placement of devices in such an architecture is important for an efficient operation. Devices which tend to communicate at high bandwidth to one another frequently should be placed on the same bus. Devices which communicate rarely with one-another but require high bandwidth should be placed in the same bus group but on different basic buses. A vertical growth should be allowed for related communication channels.

What are the features of a controller that has to act as a switch between vertical layers. This controller is responsible for mapping the session identifier from the previous layer into a session id for its layer. It can be composed from two controller modules, each component cntroller being as described in Section 3.3. Each one of these modules in responsible for the one-way data communication in one of the two possible ways data can be carried. Keeping different identifiers for each level is advantageous since it allows more sessions per level. This ensures that we will never run out of session identifier space, no matter how deep our vertical hierarchy grows.

A interconnection as described above may allow multiple paths from a sending module to a receiving module. It may potentially lead to out of order packets. However, all packets on the same session (logical bus) are going to follow the same path. This is because the session identifier uniquely determines the receiving module at each horizontal layer of buses. If the mapping between session identifiers is one-to-one as well, then the problem of out-of-order packets will not arise. However, packets traveling on two logical buses can arrive out of order. We do not address this issue now because it is not obvious if this is a serious problem nor do we know of a simple way to avoid this problem. If neccesary, a sequence number field in the packet header can be used for time-stamping or ordering packets.

5. Ensuring Performance

The description of sessions is incomplete without a description of the mechanisms in order to ensure that performance will be met for a session. There are two aspects to meeting the performance bounds of a given session. The first part consists of ensuring that the bandwidth of a bus is not overallocated, and the other part consists of giving priority to the sessions with more urgent performance requirements over sessions with less stringent requirements.

We shall not mention the software tests in detail. Obviously, many different mechanisms can be used to ensure this criterion. One such scheme is described in [4]. Based on the average and peak throughput requirements of the session, the probability that the session is active (contending for the bus) is calculated. It is possible from this information to compute the probability that a given combinations of sessions is active. For some combinations of the For some of the possible combinations (e.g. when all the sessions are contending for the bus) the requirements of some channels may not be satisfied. This can be verified by a small tracing through of the contention resolution protocol. It is then verified that the combined probability of each such combination is acceptable to the concerned sessions. As an example, it may be acceptable to the video display if its performance is not met in one out of every hundred combinations. Each set of devices have their own performance requirements, and sessions corresponding to them will have the appropriate performance characteristics.

In order to give preference to the sessions with strict requirements, we give priority to sessions with more stringent delay bounds. The priority submodule (see Section 3.3) assigns to every packet a deadline which is obtained by adding the time of its arrival to the delay bound required by the session. The bits of the deadline determine the priority of the packet on the session.

In order to establish sessions, we assume some default sessions which will be established for every pair of devices in the same bus group at boot time. For each bus group, we assume that one of the cpu-memory blocks will be responsible for the establishment of sessions. There will always exist a session from this "session master" to all other controllers on the same layer. This session can be used for starting any further session establishment requests.

6. The Simulations

In this section we present simulation results of our scheme. The traffic has been characterised by mathematical models. These models provide a reasonable estimate of the traffic patterns expected in the real system. The two traffic models that have been used relate to the two kinds of traffic considered, conventional traffic and the time critical, delay sensitive traffic.

Conventional traffic has been generated by a two state model of packet inter-arrival times. In the two state model, the traffic is assumed to follow a poisson distribution with two means. These means are related to each other by roughly a factor of 100. The two state model has a state transition probability(p_tr) associated with it. The two states correspond to the two means. State transition occurs with a probability of p_tr. The justification of this model is that conventional traffic occurs in bursts. A period of high traffic rate is followed by a period of low rate which represents effective silence. This is true of current computer traffic which is highly bursty. The performance of the scheme is measured by bus utilisation and the delay associated with the transactions.

Time critical traffic is harder to characterise. Not enough studies have been carried out which can reflect the nature of the traffic pattern. Thus a model of traffic is hard to find. We

conservatively approximate this traffic by the following model. The paradigm is based upon the fact that there will be a regular arrival time between frames. This is because information update for visual appeal has to be carried out only at rates suitable for human perception. Hence the traffic is modelled as arriving at regular intervals. The interval is calculated on the basis of peak thruputs that can be provided by the different external bus controllers. Each type has its own interarrival rate. This is also corrected for the load factor. The size of the frame is dependent upon the compression schems used. Since suitable compression algorithms are still in the phase of being studied, we approximate the frame size by a regular poisson process with a large mean. The large mean reflects the huge amount of information contained in a frame.

The performance metrics of importance are the bus throughput and the traffic delays encountered. These are functions of two independent variables. The first is the load conditions encountered by the buses. This is reflected in the interarrival means associated with the traffic generation models. In practice, the load signifies an increase in bus utilisation until the point that the bus saturates. The second parameter is the cross transaction probability, namely the probability that the transactions on the bus-system occur between controllers which are located on different buses, C-bus and E-bus. Larger is this probability, greater is the reduction in total bandwidth available on each side.

Figure 3

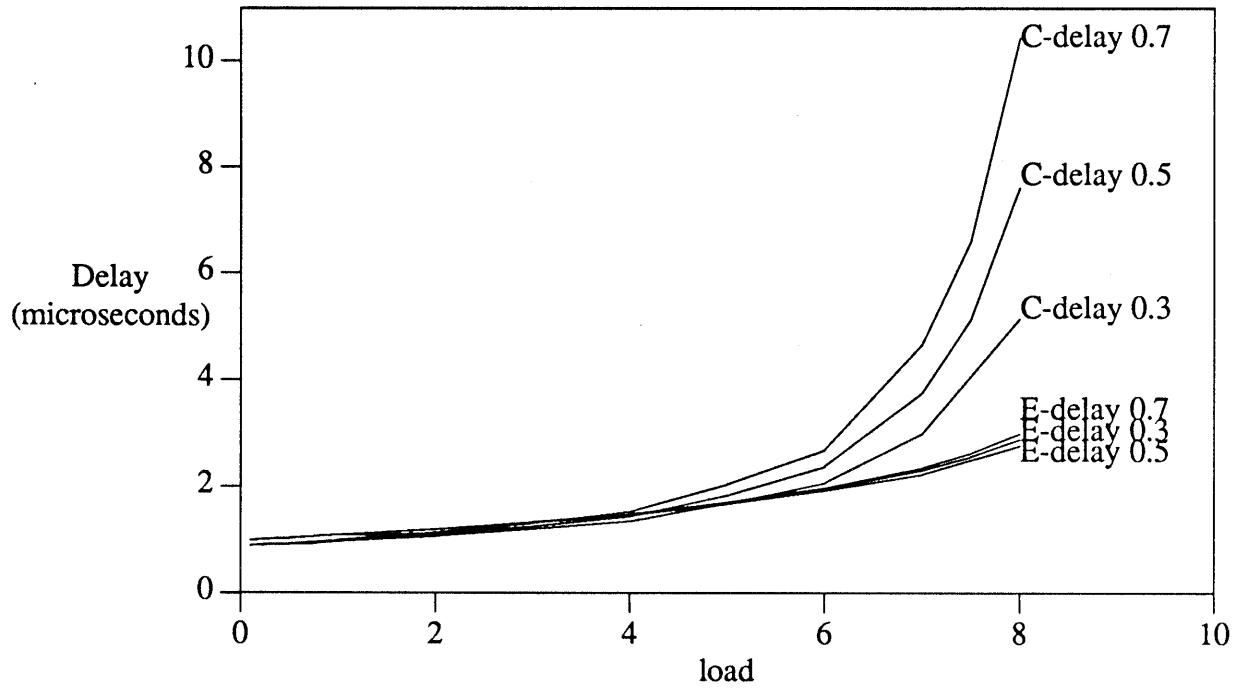


Figure 3 presents the results for the delays encountered. Both the delays in the core bus and the external bus have been plotted as a function of the load. The delays have been shown for three cross transaction probabilities, 0.3, 0.5 and 0.7. The load represents the number of controllers of a particular type that can be provided. More the number of the controllers, greater is the load. A load of x signifies that x controllers operating at peak bandwidths can be provided. The load is restricted by bus saturation. The delay for the core bus is seen to increase rapidly beyond a load factor of 4. This tendency grows as the crossover probability increases. This represents an

ability on the part of the delay-sensitive traffic to obtain the bus at a higher priority than the conventional traffic. The priority is decided by the deadline associated with the packet. For this simulation study, we simplify things by associating a constant higher priority with the delay-sensitive traffic packets. The delay associated with the external bus however does not increase significantly. Also, the delay does not vary much across various cross-transaction probabilities. This is a benefit for delay sensitive traffic. If we are ready to justify the cost of increasing the delay on packets which do not have delay deadlines, then the session initiation can be done without much botheration as to the nature of the transactions that will ensue during the course of the session lifetime. This is particularly true for small load factors.

Figure 4

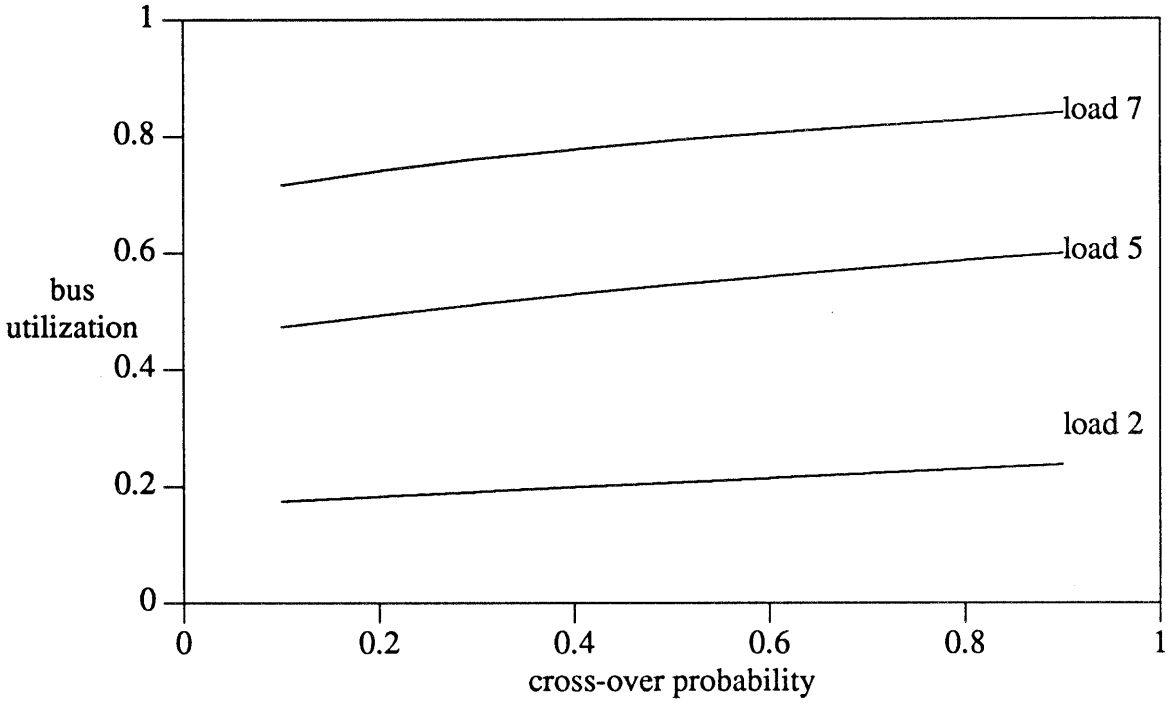
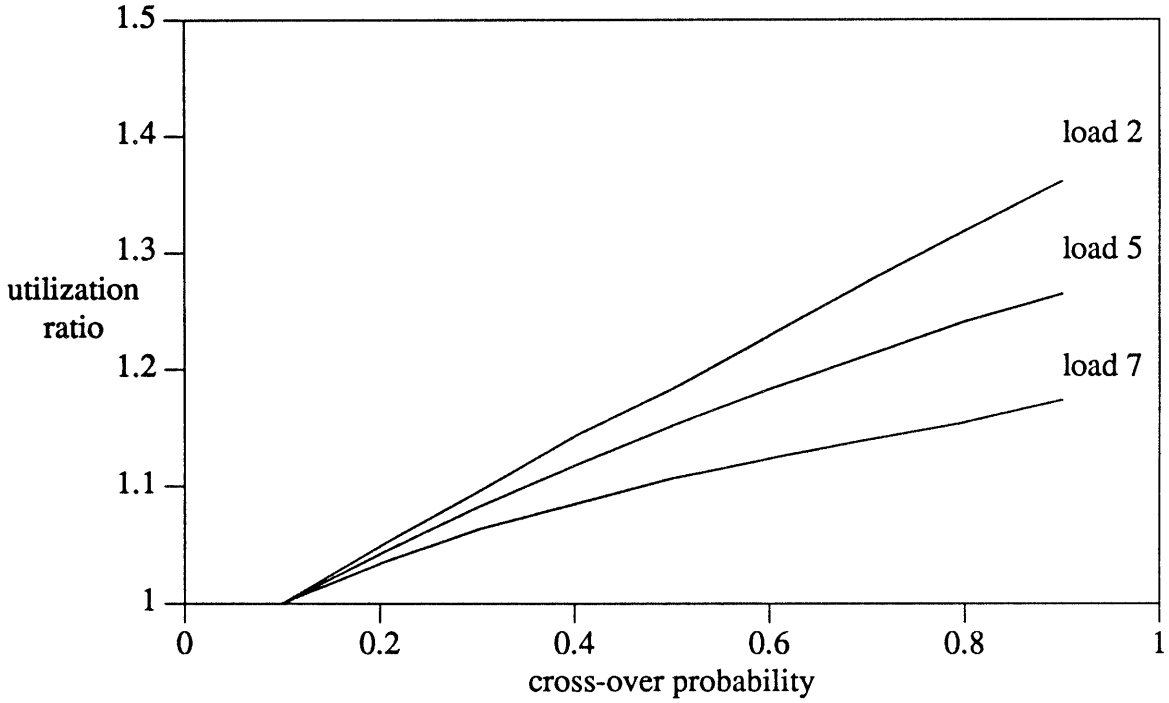


Figure 4 presents the average bus utilisation results for three load factors, 2, 5, 7. The variation has been plotted against different cross-over probabilities. The utilisations increase at almost the same rate, though obviously in different regimes of utilisations. This is extremely helpful for the session initiation software which has to decide whether a session can be guaranteed requested delay requirements. From the session table that it has set up, it can calculate the probability of cross-transaction and the current load expected. A linear or close to linear approximation can then decide the resulting average bus utilisation. This can then be used to decide whether the session can be provided with sufficient bandwidth to meet its requested traffic load characteristics. Figure 5 presents the average utilisation ratio. This reflects the percentage change in utilisation with a variation in the crossover probability for three different load factors, 2, 5 and 7. The percentage change decreases for a higher load than for a smaller load, as the cross transaction probability increases. This figure also reflects the deflection from linearity of the average utilisation.

Figure 5



7. Conclusions and Future Work

We believe that our architecture has certain advantages over the conventional architectures[5][6]. We can scale to a potentially unlimited extent in the vertical direction by adding a layer of buses to this architecture. Moreover, in each bus group, we are able to isolate and provide high bandwidth to a number of devices communicating simultaneously. The flexibilty in computing the priorities enables us to give preference to time-critical devices over non-critical devices.

This design is only the first step in the study of the I/O architecture for future systems. It would be intersting to see if we can characterize the workload conditions under which the scheme of logical buses will perform better than the conventional channel architecture. The design of some of the components is not well defined at this stage, specifically the conflict resolution module in the controller. Finally, we have to examine the compatibility issue in more detail. We have assumed that the device dependent controllers will enable us to use the existing equipment but we need to verify this assumption by examining some specific devices.

We are currently planning on a VHDL simulation to study the circuits timing and synchronization problems involved in desgining a bus as presented in Section 3.

8. References

[1] A. Tennenbaum, "Computer Networks, Vol 1", pp. 187-195.

[2] D. P. Anderson, "A software architecture for network communication", *Proc. 8th Int. Conf. Distrib. Comp. Sys.*, San Jose, CA, pp. 376-383, June 1988.

[3] D. Ferrari, "Real-Time Communication in Packet Switching Wide-Area Networks", Tech. Rept. TR-89-022, International Computer Science Institute, Berkeley, May 1989.

[4] D. Ferrari and D. C. Verma, "A scheme for Real-Time Channel Establishment in Wide Area Networks", Tech. Rept. TR-89-036, International Computer Science Institute, Berkeley, June 1989.

[5] J. Levy, "Buses, the Skeleton of computer structures", in Computer Engineering by G. G. Bell, J. C. Mudge and J. E. Mcnamara, pp 269-300, Digital press, 1978.

[6] J. Buzen and A. Shum, "I/O Architecture in MVS/370 and MVS/XA", CMG Transactions, Vol. 54, Fall 1986, pp. 19-26.