AN ABSTRACT OF THE THESIS OF

James Carver Hill (Name of student) for the Ph. D. (Degree)

In Electrical Engineering (Major) Presented on Aug. 15, 1969

Title: Priority Structures for Multiaccess Memory Systems

Abstract approved Redacted for Privacy (Signature)
Professor L. N. Stone

The quest for greater computer-system speed has brought about a continuing increase in the parallelism of these systems. In particular, the number of processors and other devices which are required to access the same core-memory — not for economy, but for the advantages to be derived from using the memory as an exchange for data and control information — has increased markedly.

That portion of such multiaccess systems which allocates the memory accesses available at a given instant of time among the devices then requesting access to memory is denominated the priority structure. These attributes are desirable in a priority structure:

1. It should service all requesting devices within the limits of their patience; i. e. the latency of each memory-

accessing channel should be less than the patience of the corresponding device.

2. It should be efficient in that all accesses available from the core-memory are made available to devices.
3. It should be modularly constructed and reconfigurable so that no single component is absolutely necessary for system operation; i. e. it should introduce no intrinsic point of permanent articulation.

Priority structures used heretofore, principally strict-priority and first come-first serve, do not exhibit all of these characteristics: First come-first serve systems, although efficient, are incompatible with devices of small patience unless intrinsic points of permanent articulation are allowed. Strict-priority systems, are generally not 100% efficient, have variable channel latencies, but are favorable to the elimination of articulation points. Their inadequacies are not a result of the idiosyncrasis of a particular hardware implementation but are shown to be, in each case, a fundamental shortcoming of the service discipline itself, when applied to multiaccess memory systems.

An alternate discipline, limited-latency, is developed which does exhibit all of the desired characteristics. In an example of this discipline, each multiplexor, at each level, in the memory-access structure is required to contain a cycle-counter, the states

of which are decoded so as to guarantee each input channel some fraction of the total number of cycles available to that multiplexor. Necessary, or desirable, restrictions on the sequence of guaranteed-fractions to be so obtained are derived and their consequences are discussed.

Finally, it is shown that the limited-latency structure is a general schema, independent of ad hoc hardware considerations, which can serve as the basis for particular designs. (It does, in fact, include first come-first serve and strict-priority as special cases.)

Priority Structures for Multiaccess Memory Systems

by

James Carver Hill

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Doctor of Philosophy

June 1970

APPROVED:

Redacted for Privacy

Head of Department of Electrical Engineering

in charge of major

Redacted for Privacy

Dean of Graduate School

Date theses is presented Aug. 15, 1969

Typed by Norma Sayegh for James Carver Hill

ACKNOWLEDGMENTS

This dissertation was inspired by a problem encountered in the course of my work at the Lawrence Radiation Laboratory of the University of California. The general support of the laboratory personnel—typing, drafting, reference procurement, and technician assistance—is greatly appreciated. However, I wish to particularly thank my colleagues John O'Brien, James Moore, and Norma Sayegh for patiently listening to many verbal presentations of the dissertation as it developed and for reading and criticizing the first written draft.

The author is especially grateful to Professor L. N. Stone, Head of the Department of Electrical and Electronic Engineering at Oregon State University, for his assistance in identifying the problem and developing the exposition of the solution.

LIST OF TABLES

LIST OF FIGURES

TABLE OF CONTENTS

PRIORITY STRUCTURES FOR MULTIACCESS MEMORY SYSTEMS

INTRODUCTION

The innovations, inventions, and improvements in computer technology in the last 15 years have increased the speed of computer systems both directly and indirectly.

The direct application of technological achievements has resulted in an increase in the intrinsic speed of computer system components. Faster circuits, due primarily to the evolution and perfection of manufacturing techniques in solid-state electronics—faster core memories, due primarily to the evolution and perfection of manufacturing and testing techniques for extremely small cores—faster rotating memories, through the perfection of the mechanical-finishing and plating processes for the rotating surface and the refinement of head-manufacturing techniques—are three examples.

Indirectly, improved technologies—particularly in the area of circuits—have increased the speed of computer systems by decreasing the cost of components while increasing their reliability. Decreased cost and increased reliability of components have made large, parallel systems feasible. Such systems are capable of greater useful speed because they allow many operations (input, output, calculation) to proceed simultaneously instead of seriatim (3, p. 1).

The apparent speed of computer systems (or, if you prefer, turn-around time) has also been greatly improved by the indirect application of technology. In particular, interactive time-sharing systems—which dramatically reduce turn-around time—owe much of their success to high-speed rotating memories and a proliferation of fast registers within the processor (6, p. 1 - 15).

Due to diminishing returns in overall system speed realizable by improvements in intrinsic speed, recent approaches to increased system speed have necessarily concentrated on improving useful and apparent speed.[1] Consequently, the simultaneous activity within systems has been greatly increased. This increased simultaneity greatly increases the demand for central-memory cycles and creates attendant problems; these problems are referred to as problems of multiaccessing (11).

This thesis is concerned with the particular problem of allocating the available memory cycles among the many devices competing for this commodity in a multiaccess system.

[1] For example, the Control Data 7600 is constructed with discrete-component circuits, not integrated circuits. However, its iterative, parallel architecture yields a very high useful speed (7).

DESIRABLE ATTRIBUTES OF A PRIORITY STRUCTURE FOR MULTIACCESSING

There are three salient considerations which should govern the design of the priority structure which allocates memory cycles in a multiaccess system:

1. The worst-case wait-time for a memory cycle at a particular channel must not exceed the patience of the device connected to that channel.
2. The structure should contain no intrinsic point of permanent articulation.
3. The system should be "efficient" in the sense that all available memory cycles not required by synchronous devices may be utilized by asynchronous devices.

In this chapter we will relate each consideration to the associated multiaccessing problems.

Patience

In an excellent paper by Wallace and Rowswell the patience of a channel is defined as the maximum time which can safely elapse between the emission of a request by the channel and the completion of

the corresponding memory access (10). For clarity in subsequent parts of this thesis, however, some modification of their definition is desirable.

Consider the case of channels serving synchronous devices with one word of buffering (in addition to any assembly or disassembly registers); it is clear that the reciprocal of the transfer rate of the synchronous device is the "maximum time which can safely elapse" (minus some overhead, perhaps, such as the minimum allowable time between clearing and setting the data flip-flops).

However, in the case of channels serving synchronous devices with more than one word of buffering, the amount of time which may "safely elapse" depends upon the number of buffer registers which happen to be full (or empty) at the instant a particular request is issued. If, for example, data from the device must be emptied into a buffer register which is still full, then the "safety rule," for the request corresponding to the word in the still-full register, has obviously been violated. Staudhammer, Combs, and Wilkinson speak of such over-writer (or re-reads) as transfer-timing errors (9).

In the case of channels serving asynchronous devices there may be no maximum to the wait-time the device can tolerate (infinite patience) but the channel itself handles every request within some finite time. Therefore for the sake of clarity we will, from now on, speak of the

patience of devices and the latency of channels and, in the light of the preceeding observations, define them as follows:

> The patience of a device is the maximum time which may elapse between the emission of every request and the completion of the corresponding memory access without the occurrence of transfer timing errors.
>
> The latency of a channel is the maximum time which can elapse between the presentation of a request to that channel and completion of the corresponding memory access.

The patience of a device should exceed the latency of the channel serving the device.

Articulation

The term "point of articulation" is borrowed from graph theory. In a graph-theoretic context it means a node which, when removed (along with its incident edges) from a connected graph, disconnects the graph (1, p. 67). In the context of systems, "articulation point" is to be understood as meaning a component of the system which, when it fails, entirely removes some capability from the system. If we say that the system has failed when it can not perform all of its normal functions and that it has not failed when it can perform all normal functions, albeit at a reduced rate, then it follows that the failure of an articulation point produces a system failure.

There are two cases of interest. If, after the failure, the configuration of the system can be changed to an "equivalent" configuration which does not include the failing articulation point and thereby allow the system to begin operation again, the articulation point will be called a point of temporary articulation. If, on the other hand, no such reconfiguration is possible, the articulation point will be called a point of permanent articulation.

For illustration let us consider a simple computer system consisting of two arithmetic processors with individual consoles; two-tape-unit controllers, each connected to four tape units; and eight core-memory modules, each having four ports. Each processor and each tape-unit controller is connected to a unique port.

If the tape-unit on which the system tape is hung fails, the system will fail; however, it is not difficult to move the system tape to another tape-unit and then restart the system. Consequently, the tape-unit holding the system tape is merely a temporary articulation point.

This system need not contain any permanent articulation points, but it may. It is possible to create them in the software. It would be possible to write the system program in such a way that a minimum of five tape-units would be required; in that case, both tape-unit controllers would become permanent articulation points. Furthermore, it is possible to create permanent articulation points by carelessness. For

example, if the number of correct copies of system tape is allowed to decrease to one, that tape is, until a copy of it is made, a permanent articulation point. However, in all that follows it will be assumed that a system contains no such extrinsic articulation points but only the intrinsic articulation points of the hardware.

As an illustration of an intrinsic point of permanent articulation, suppose that we replace the four-port memory modules of the example system with one-port memory modules in conjunction with a four-channel multiplexor. (Each processor and each tape-unit controller is now connected to a unique multiplexor channel.) Obviously, the multiplexor is an intrinsic point of permanent articulation.

Equally obvious is the desirability of systems containing no articulation points, for these are systems which do not fail as the result of the failure of any one component. Such systems can be designed; however, a goal much more easily and economically attained is the design of systems without intrinsic points of permanent articulation.

A system without permanent articulation points will avoid protracted periods of downtime since it need be down only long enough to reconfigure the system, not to repair it. This characteristic is particularly desirable in large time-sharing systems because the expense (and irritation) of downtime is proportional to the number of users deprived

of the services of the system.

Access Efficiency

The priority structure of a system is a mechanization of the designer's policy for the distribution of the central-core-memory accesses available at a given instant of time among those devices then requesting memory cycles. Under worst-case conditions of addressing conflict (all requesting devices trying to access locations in the same memory module), the cycles available in that instant of time will be limited to the number which one memory module can provide. Usually the primary objective of the designer's priority policy is to assure satisfactory operation of synchronous devices under these adverse conditions, i.e., it is a rationing policy for times when memory accesses are in short supply.

As we will show shortly, however, worst case conditions are unlikely to prevail for most of the time. It is, therefore, a worthwile secondary objective of the priority policy to avoid wasting (not utilizing) available cycles at times when the supply of memory cycles is greater than that required by the synchronous devices.

Naturally the priority structure itself can not utilize memory cycles but it can be favorable or unfavorable to the inclusion of fast, asynchronous devices to act as "cycle-sinks" in the system. Therefore,

we will define the access efficiency of a priority structure as being the time average of the ratio of the number of memory accesses available to the devices, to the number of memory cycles available to the priority structure.

Although the access efficiency can be calculated without actually knowing the number of memory accesses available (if it is possible to ascertain what fraction of available cycles the priority structure must waste), estimating the number of accesses available in a general system is a worthwile exercise in that it provides insight into some problems of multiaccessing.

The Availability of Memory Accesses

Consider a multiaccess system with the following properties, which we will assume to prevail for the remainder of the paper.

The central core-memory is comprised of M identical modules. Each of these modules has precisely one input-output channel which connects the module of memory directly to a time-division multiplexor called a port structure. See Figure 1.

Each port structure provides access to its module of memory for P external channels. The corresponding external channels from each port structure are connected in common and called a port (the port structures are uniquely numbered and a portion of the address supplied

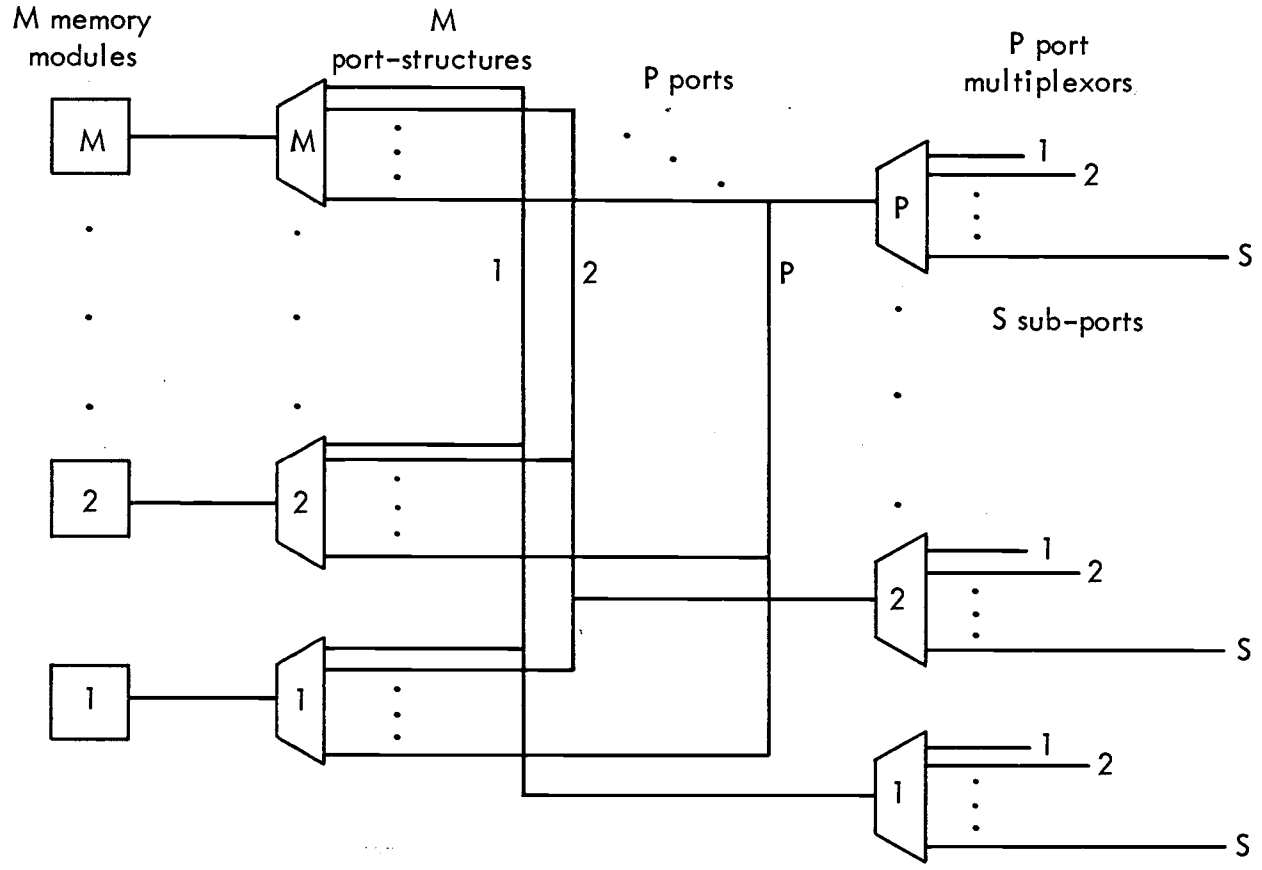


Fig. 1. General system configuration.

by any memory-accessing device is decoded to activate only one of the M port structures per device request). Thus, a port is an input-output channel which can access any word of memory through one multiplexing level.

Each port is connected directly to a second-level time-division multiplexor called a port multiplexor. A port multiplexor provides access to its port for S external channels. Every memory-accessing device in the system has exclusive use of one or more of these channels called sub-ports. Thus, a sub-port is an input-output channel which can access any word of memory through two levels of multiplexing.

The number of memory accesses available per unit time depends upon the speed of the memory modules and the degree of parallelism in the accessing hardware, i.e., the number of ports and modules.

If there were only one port, either the transfer rate of the entire system would be limited to the rate of a single memory module,[2] or the memory would have to be partitioned so that all of memory would no longer be addressable by any device in the system (contrary to our definition of a port). Therefore, the desirability of having more than one port is patent.

[2] Interlacing and overlapping are ignored since they can be accounted for by multiplying factors when considering average performance.

If the number of ports were equal to the number of memory modules, it would be theoretically possible to keep all memory modules busy all of the time and thus approach the absolute maximum utilization of the memory. In practice, however, some of the devices accessing memory will necessarily be idle an appreciable fraction of the time (consider the latency of discs and drums, for example) making it impossible to sustain such a high utilization (except in a system having an extremely undersized memory). Consequently having the number of ports equal to the number of modules is superfluous.

In trying to proportion M and P in a particular system we can not simply assume that the number of accesses available per unit time will be equal to P x R provided $P \leq M$, where R is the cycle rate of a memory module. Indeed the existence of address conflicts, more than one port addressing the same memory module, will appreciably reduce the number of cycles available.

To investigate this problem, let us begin by making the optimistic assumption that all address states are equally probable. An address state is a string of integers

$$a_1, a_2, a_3, \ldots a_n \qquad a_i \in (1, 2, 3, \ldots M)$$

where a_i is the module being addressed by port i. We assume every port is addressing some module.

Let U_n be the probability that precisely n of the M modules are being addressed (one or more of the modules is being addressed by more than one port). Let R be the cycle-rate of one memory module.[2] Then T, the average number of memory accesses available per unit time, is given by

$$T = R \sum_{n=1}^{M} n\, U_n. \qquad \text{Equation 1}$$

Hellerman (5) has considered the case where M equals P, and derived the following formula:

$$T = R \sum_{k=1}^{M} \frac{k^2 (M-1)!}{M^k\, (M-k)!}. \qquad \text{Equation 2}$$

His model for computing U_n is to determine the probability that the first n integers of the address string are distinct. The model is simple but somewhat pessimistic in that it essentially equates any string of D distinct integers, D less than M, with an M-minus-D-way address conflict.

Since we are also interested in cases where M is not equal to P, we will approach the problem in a different manner. We will compute U_n by dividing the number of ways P ports can address M modules so that precisely n addresses are distinct, by the number of ways P ports can address M modules. Or, in classical terminology, U_{P-i} is the number of arrangements of P (distinguishable) marbles in M

cells, so that precisely P-i cells are occupied,[3] divided by the total number of arrangements of P marbles in M cells.

Now the number of arrangements of P marbles in M cells so that precisely P-i cells are occupied is equal to the number of ways of chosing P-i cells from M cells, times the number of arrangements of P marbles in P-i cells so that P-i cells are occupied, i.e.,

$$A(P, M, P-i) = \binom{M}{P-i} A(P, P-i, P-i). \qquad \text{Equation 3}$$

But

$$A(P, P-i, P-i) = \sum_{k=0}^{P-1-i} (-1)^k \binom{P-i}{k} (P-i-k)^P \qquad \text{Equation 4 (2, p. 58)}$$

And the total number of arrangements is M^P; therefore,

$$U_{P-i} = M^{-P} \binom{M}{P-i} \sum_{k=0}^{P-1-i} (-1)^k \binom{P-i}{k} (P-i-k)^P \qquad \text{Equation 5}$$

consequently

$$R^{-1} T = M^{-P} \sum_{i=0}^{P-1} \sum_{k=0}^{P-1-i} (P-i) \binom{M}{P-i} (-1)^k \binom{P-i}{k} (P-i-k)^P \qquad \text{Equation 6}$$

Values of this estimate (Equation 6), as well as Hellerman's formula, are plotted in Figure 2 to illustrate the reduction in available accesses due to address conflicts. It should again be pointed out

[3] We do not say "...that i cells are unoccupied," a more familiar but very different problem indeed. Consider $P < M$.

that the assumption of equally probable address states is optimistic. In practice, address conflicts can reduce the availability of cycles considerably more than is indicated by Figure 2.

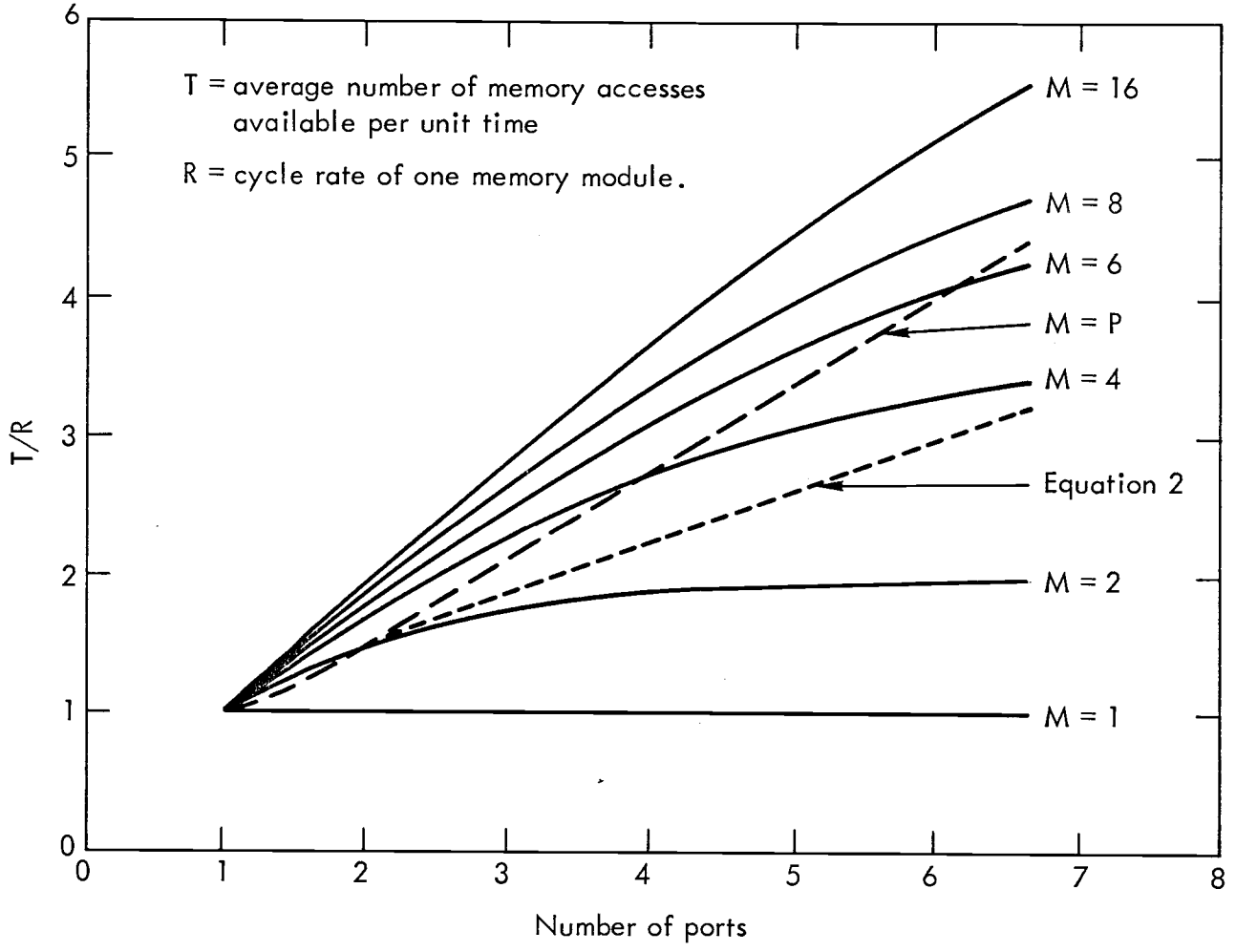


Fig. 2. Plot of T/R versus the number of ports for various numbers of modules, showing the reduction in available accesses due to address conflicts.

ANALYSIS OF COMMON MEMORY PRIORITY STRUCTURES

The common disciplines for memory-access allocation are:

1. First come-first served
2. Round-robin
3. Strict-priority
4. A combination of the above

In analyzing these four approaches we shall investigate the effect of the discipline upon system efficiency and articulation in addition to determining its effect upon the patience required of devices.

Only the last consideration, patience, has received a modicum of attention in the literature (9, 10). No doubt, efficiency and articulation have not been studied as thoroughly as patience (latency), because they must be measured relatively while latency can be more decisively evaluated. Devices either receive service within the limits of their patience or they do not. System articulation and efficiency are important considerations, nevertheless, and the author contends that it is unreasonable to ignore them simply because they are somewhat intangible.

First Come-First Served

In a first come-first served system it is true that it is impossible to ascertain in all cases which of a number of requests was first initiated, but it is also true that this uncertainty does not affect the latency, efficiency, or articulation of a system employing this discipline.

Latency

Let us assume that we have a two-level multiplexing scheme like the one described in the preceding chapter. If there are N_i devices connected to the i^{th} port multiplexor, then each of them may have to wait at most N_i multiplexor cycles to complete any request. Similarly, each port multiplexor may be required to wait a maximum of P memory cycles (P is the number of ports) to complete one of its requests. Consequently the latency of every device using the i^{th} multiplexor is given by

$$L'_i = P N_i h, \qquad \text{Equation 7}$$

where h is the memory cycle-time. However, in most of our work we will use the memory cycle-time as the unit of time, in that case we have

$$L = P N_i. \qquad \text{Equation 8}$$

Consequently, if the system must service one or two high-speed synchronous devices it will be necessary to connect them to a port multiplexor for which N_i has been made suitably small.

Efficiency

Note from Equation 8 that the latency of any channel is not affected by the rate of any device in the system. Consequently, high-rate asynchronous devices are well tolerated. Their presence as cycle-sinks can result in an access efficiency of 100%; however, it may be that a multiplexor servicing only one or two small-patience devices can not also service a cycle-sink because the corresponding increase in N_i might make the latency of the channels servicing the synchronous devices too large. In that case, the access efficiency of the system would be reduced since all cycles available to the multiplexor when the synchronous devices were idle would be wasted.

Articulation

If we wish to eliminate multiplexors and ports as permanent points of articulation it is necessary to provide each device in the system with two access paths to memory (assuming every device is necessary for system operation): one path to be used during normal operation and an alternate path to be used whenever it is necessary to reconfigure

the system in order to bypass an inoperative port or port multiplexor. Naturally, we want the latency of every sub-port to be independent of the system configuration.

A simple example of a first come-first serve system that is reconfigurable in the preceding sense is given in Figure 3.

If, for example, multiplexor B fails, reconfiguring the system consists of enabling devices b1 and b2 in multiplexor C and enabling devices b3 and b4 in multiplexor A. Observe that reconfiguring the system does not change the latency of any sub-port; unfortunately, this is not a general property of first come-first serve structures.

If we let d_i represent the number of devices switched from the j^{th} to the i^{th} multiplexor when the j^{th} multiplexor fails, then, in order to make all latencies insensitive to reconfiguration, we require

$$P N_i = (P - 1)(N_i + d_i), \qquad \text{Equation 9}$$

where

$$\sum_{\substack{i=1 \\ i \neq j}}^{P} d_i = N_j,$$

for all i and any j in (1, 2, 3 ... P). Since Equation 9 must hold for all i, we can sum any P - 1 of the P corresponding equations and obtain

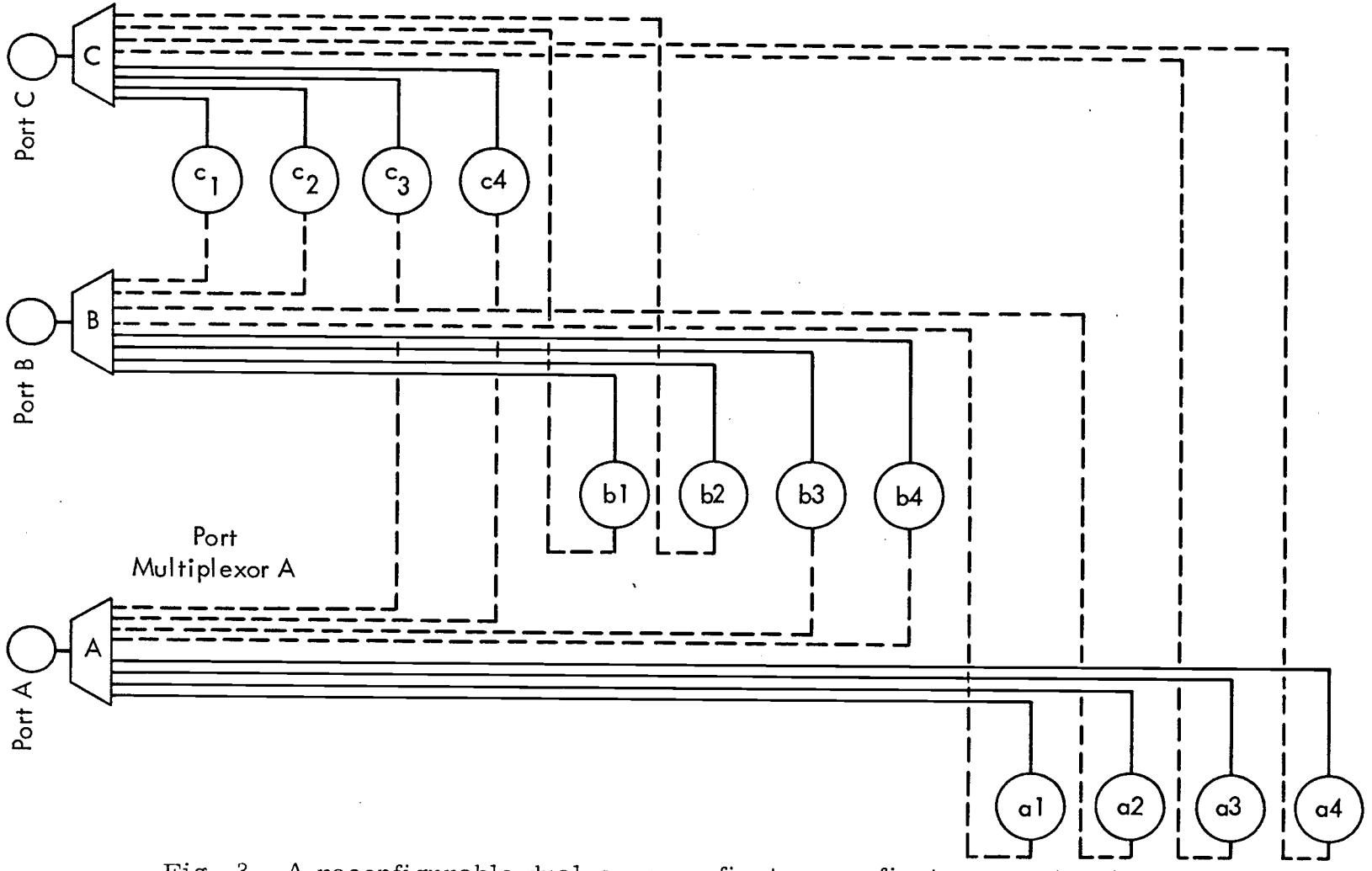


Fig. 3. A reconfigurable dual-access, first come-first serve structure.

$$\sum_{\substack{i=1 \\ i\neq j}}^{P} P N_i = \sum_{\substack{i=1 \\ i\neq j}}^{P} (P N_i + P d_i - N_i - d_i), \qquad \text{Equation 10}$$

$$\therefore \quad \sum_{\substack{i=1 \\ i\neq j}}^{P} (P - 1) d_i = \sum_{\substack{i=1 \\ i\neq j}}^{P} N_i, \qquad \text{Equation 11}$$

$$\therefore \quad (P - 1) N_j = \sum_{\substack{i=1 \\ i\neq j}}^{P} N_i \quad \text{for all } j. \qquad \text{Equation 12}$$

Let $$N = \sum_{i=1}^{P} N_i$$

$$\therefore \quad P N_j = N \quad \text{for all } j. \qquad \text{Equation 13}$$

From Equation 13 it follows that the same number of devices, n, must be connected to each multiplexor; it then follows from Equation 9 that P - 1 must divide n. Consequently, the conditions under which a first come-first serve system can be satisfactorily reconfigured are seen to be very restrictive and incompatible with a need to provide a variety of sub-port latencies in order to adequately serve a variety of devices.

Summary

We may characterize a two-level, first come-first serve structure as one capable of high efficiency but not compatible with devices of small patience unless intrinsic points of permanent articulation are allowed.

Round-robin

In principle, a round-robin structure works as follows: The request lines of each sub-port (port) is examined in turn. When a requesting sub-port (port) is encountered it is allocated the next multiplexor (memory) cycle. Upon completion of that cycle the scan of request lines resumes, beginning with the next in the order of scan.

Since the scanner can never be more than $N_i - 1$ positions away from the request line of any sub-port connected to the i^{th} multiplexor, a cycle corresponding to a request on sub-port i will begin no later than $N_i - 1$ multiplexor - cycle - times after the issuance of the request. Therefore, it will complete after N_i cycles. Similar remarks apply to the requests of multiplexors to the port structures. Consequently, the latency of all sub-ports connected to the i^{th} multiplexor is given by

$$L = P N_i. \qquad \text{Equation 14}$$

Thus, all the observations concerning the efficiency and articulation of the first come-first serve structure are equally applicable to the round-robin structure. Therefore, the systems are equivalent for our purposes.

Strict-Priority

Latency

Let us determine the latency of the sub-port of priority i in a strict-priority system. When a request is issued on sub-port i, service commences immediately if the memory (including the access structure) is not busy. If the memory is busy, service for sub-port i will commence at the end of the cycle in progress provided there are no outstanding requests of higher priority (i.e. requests on sub-ports with numbers less than i) at that time. When there is such a queue of higher priority requests,the response to the request of priority i is of course deferred until the number of elements in the queue is reduced to zero.

Unfortunately, we can not simply say that the maximum time to reduce the queue to zero is i-minus-one cycle-times, because as soon as service for a device begins it is free to issue another request and re-enter the queue. We must be slightly more subtle. But before we proceed to ponder the maximum queue-service time let us make some useful distinctions concerning the components of latency.

The latency of sub-port i is comprised of two parts, the waiting period, w_i, and the service period, h. The service period, h,

(approximately the memory cycle-time) is assumed to be constant for all cycles and is used as the unit of time in all that follows. The waiting period, the time elapsed between the issuing of a request and the beginning of the corresponding memory access, also has two components: the set-up period and the queue-service period.

The set-up period is the time spent completing any memory access (possibly of priority lower than i) initiated before the occurrence of the request on sub-port i.

The queue-service period is the time spent servicing requests of priority higher than i. Of course the queue-service period is not a constant, so in order to determine the latency of a sub-port we must know the maximum length of its queue-service period.

By definition, there are no outstanding requests of priority higher than i at the beginning of a maximum queue-service period. Consequently, the maximum time spent servicing requests of priority f (where $f < i$) during W_i, the maximum waiting period for sub-port i, is no more than the smallest integer (h is the unit of time) greater than or equal to W_i divided by t_f, the minimum time between requests on sub-port f (or the reciprocal of R_f, the maximum rate at which the device connected to sub-port f can issue requests). Note that w_i is precisely the period in which the arrival of requests of priority higher than i can delay the service of i.

Consequently the maximum queue-service period for sub-port i is given by

$$Q = \sum_{u=1}^{i-1} \left\lceil \frac{W_i}{t_u} \right\rceil, \qquad \text{Equation 15}$$

where $\lceil x \rceil$ means the least integer greater than or equal to x.

The maximum set-up period is of course 1 (in units of h).

It follows that the maximum waiting period for sub-port i is the minimum solution (occurrence of the first zero in the queue length) of the integer equation[4] (10, p. 66)

$$W_i = \sum_{u=1}^{i-1} \lceil W_i R_u \rceil + 1. \qquad \text{Equation 16}$$

Therefore, the latency of the i^{th} sub-port is given by

$$L_i = W_i + 1. \qquad \text{Equation 17}$$

Note that, unlike the first come-first serve system, the latency of the i^{th} sub-port in a strict-priority system is dependent upon the transfer rates of higher priority devices. This dependency is disadvantageous in two ways. First, it lengthens the system design process, since the choice (from a set of functionally similar units with different data rates) of the $n + 1^{st}$ device, along with the design of its controller, must often be deferred until the choices for the first n devices are firm. Second, the interdependence of

[4] Wallace and Rowswell have given an explicit bound for W_i which is derived in Appendix I.

sub-channel latencies may lead to the development of a "tuned" system in which even a slight increase in the transfer rate of one device may cause transfer-timing errors for some other device. The difficulty in linking cause and effect for such subtle failures is formidable.

Equation 17 appears to provide the means for making certain that the patience of a device exceeds the latency of its channel.

Unfortunately, the exactness of Equation 17 is illusory in any system in which the sub-ports connected to devices are used to transfer control words in addition to data words. This dual use of a sub-port, a virtual necessity in paged systems, creates a variation in the request rate on the sub-port. This variation is referred to as the "galloping problem."

As a very specific example, which will illustrate several general aspects of the problem, consider the case of reading a contiguous block of pages from disc, which must go to non-contiguous pages in core. Clearly, the disc-controller must have access to an ordered list of control words specifying the starting core-address and word-count of each contiguous portion of the transfer-block (or an equivalent table of information). For the sake of argument let us assume that there is a table of words each containing a word-count, and a starting address corresponding to a contiguous portion

of the transfer-block. Let us call these words pointer words. Let us also assume that each pointer word contains two flags: one to indicate indirect addressing (i.e., the pointer word is to be replaced by the contents of its address field, this operation may be nested), and another to tag the last pointer word in a job.

Let us further assume that the (or a) control processor initiates a disc-job with two control words—one, called the comdad word, which contains the command to the disc (read, write, search, etc.) and the disc-address of the information to be transferred; and another called the locator word, which contains the address of the first pointer word of the job. The comdad word and the locator word must be sent to the disc-controller by some processor-to-controller channel external to the memory system, or the disc-controller, when not busy, must periodically examine in core, some predetermined (by switches, perhaps) flag word for an indication that the contents of the comdad and locator words contain the parameters for a new job (the address of comdad could be contained in the flag word; the address of locator could be this address plus one). (See Appendix II for a discussion of the implied interlock problem.)

Since it weakens the assertion that the dual use of sub-ports as both control and data channels is necessary in paged systems, let us postulate that initiation of disc-jobs is accomplished without

the necessity of memory access through the sub-port connected to the disc-controller. However, once the job has started, the $n + 1^{st}$ pointer word must be read from core during the time the data transfers corresponding to the n^{th} pointer word are being made. As a result a control word transfer must be inserted between two data-word transfers, thus violating the constant-rate consideration upon which the latencies of lower-priority sub-ports are determined according to Equations 16 and 17.

One might naively suppose that the problem could be avoided by connecting the controller to two sub-ports, using one for data and one for control. But what priority is to be given to the control sub-port?

If it is a high priority, what rate is assigned to it so that Equation 17 can be used with practical exactness? With a high priority assignment the control sub-port represents an asynchronous use of a priority position intended for a synchronous device of constant rate.

If the control sub-port for the disc is assigned a low priority the $n + 1^{st}$ pointer word may not be obtained before all the words of the n^{th} part of the transfer-block have been transmitted, resulting in what are effectively transfer-timing errors. The possibility of such errors is not remote; the word-count in the n^{th} pointer word

could be small and the $n + 1^{st}$ pointer word might be indirected one or more levels.

At this point one might attempt to avoid the problem in another way. All the pointer words for a job could be obtained from core (at a rate no greater than the normal disc rate) and stored in the controller before beginning the data transfer. How many pointer words will there be? The number of pointer-word buffers in the disc-controller becomes a somewhat artificial restriction on the number of chained links in a data transfer. Such a restriction could markedly reduce the efficacy of paging if the number of allowed links was not large enough.

Still, there must be a way out. It is not so much the additional number of transfers taken by the control sub-port that aggravates lower priority members of the system, but the fact that they may come at any time, and in particular, at just that moment when the patience of a lower priority device is about to expire. Then why not put two or three words of buffering (with asynchronous control) in lower priority devices so that they may adjust to occasionally shorter transfer periods on higher priority sub-ports? Such buffers installed in the j^{th} device would, for the $j + 1^{st}$ and following devices, compound the problem of indeterminate transfer periods on higher priority sub-ports.

Of course the possibility of transferring the pointer words to the controller by some means external to the memory system still remains. Suppose the controller interrupts the control processor whenever a control word is required and, in response to the interrupt, the control processor transmits the control word to the disc-controller over a processor-to-controller channel external to the memory system. Even if we assume (unrealistically) that the processor will not, itself, have to retrieve the pointer word from core, the time to process an interrupt will be many (eight or more) times greater than the time of a single memory cycle. Consequently, with all devices interrupting the control processor at unpredictable times in order to obtain necessary control words, the ensuing interrupt-service-priority problem becomes greater than the memory-access priority problem which we are trying (unsuccessfully) to avoid.

The problem can not be avoided. The flexibility and sophistication (particularly paging) of large multiaccess systems makes the dual use of memory access channels for data-word transfers and control-word transfers inescapable. The resulting uncertainty in transfer rates makes latency calculations indeterminate and increases the danger of developing a "tuned" system when memory-access distribution is controlled by a strict-priority discipline.

Access Efficiency

In Figure 4, the circles marked a1, a2, a3, b1, and b2 are synchronous devices with patience exceeding the latency of their respective sub-ports. Device b3 is a high-speed asynchronous device which can utilize all the accesses available to it, i.e., a cycle-sink. The lowest priority sub-port services b3 and the highest priority sub-port services a1; the discipline is, of course, strict-priority.

The presence of b3 guarantees that the lower-priority port, B, will never be idle. The higher priority port, A, will be idle whenever a1, a2, and a3 are idle—a situation which must arise if the latencies of the sub-ports for b1, b2, and b3 are to be finite (Equation 16). Consequently there are accesses available to A which cannot be utilized.

If a cycle-sink, a4, were connected to the lowest priority sub-port of the higher-priority multiplexor in order to utilize the excess cycles available, the latencies of the sub-ports servicing devices b1 and b2 would increase beyond the patience of these devices. It follows that only the lowest-priority port, of those ports serving one or more limited-patience devices, can have a cycle sink among its devices.

Consequently the access efficiency of a strict-priority system cannot be 100%, unless all limited-patience devices and one

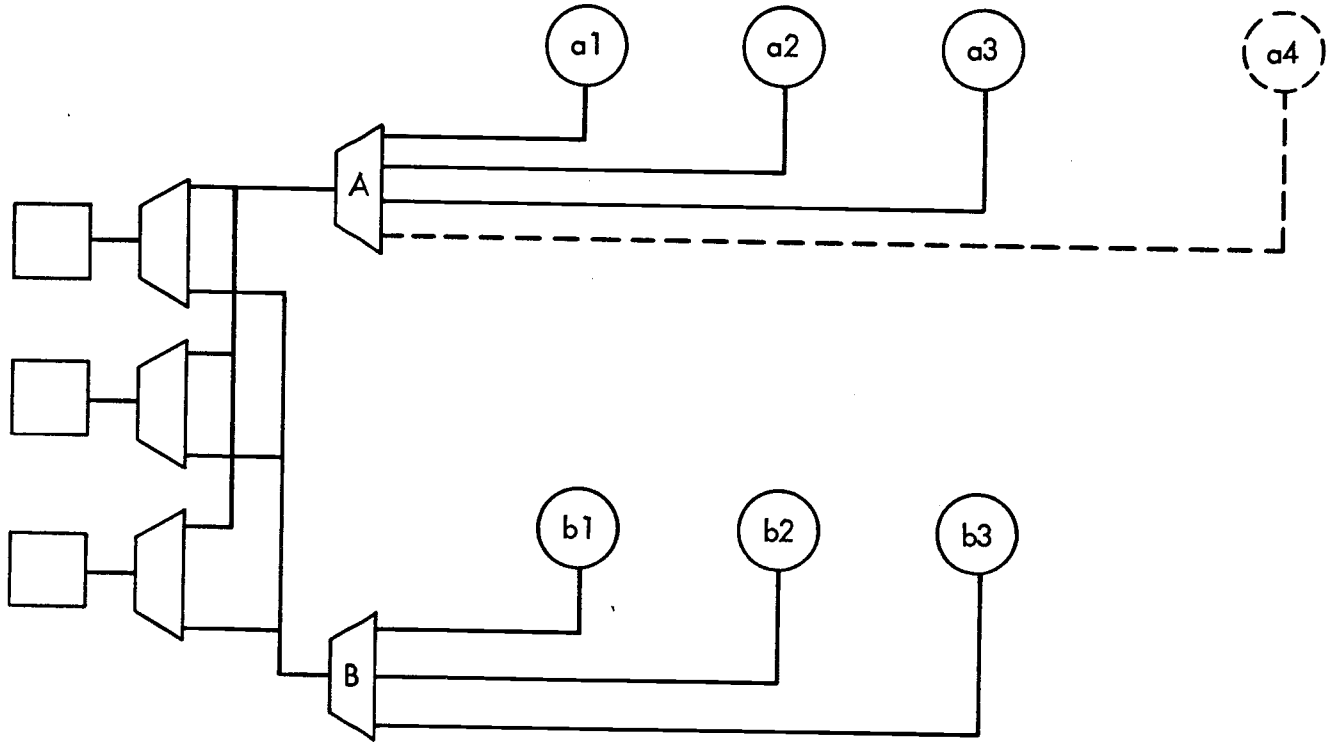


Fig. 4. A strict-priority structure containing a cycle-sink.

cycle-sink are serviced through the highest priority multiplexor (in which case that multiplexor would be a point of at least temporary articulation).

An upper bound for the reduced access efficiency in a given system configuration is readily determined. If S ports, of a total of P ports, service devices of limited patience, then those devices can utilize at most the number of cycles available from one memory module since they all have finite patience and Equation 16 must hold. If a cycle-sink is connected to the lowest-priority port of the S ports it can, in addition, utilize at most the cycles available from one memory module.[5] Thus, the S ports can utilize at most 2/P times the total number of cycles available to all ports; the remaining P - S ports can, of course, utilize at most (P - S)/P times this total. Therefore, the access efficiency of a strict-priority discipline, expressed as a fraction, is given by

$$E = \frac{P - S + 2}{P} \qquad \text{for } (2 \leqslant S \leqslant P).$$

Equation 18

Lock-Out

It is also true of strict-priority systems that only one cycle-sink can be usefully connected to a multiplexor since, if there

[5] Again we ignore interlacing and overlapping because they can be accounted for by multiplying factors.

were two, the lower priority one would never obtain any cycles. This lock-out situation can arise even if the cycle-sinks are not connected to the same multiplexor; sometimes it is hazardous.

Imagine two processors in a dual-processor system interacting in the following way. Processor A is waiting for processor B to finish a computation. Consequently, A (behaving as a cycle-sink) is continually testing a control word in memory to be modified by processor B upon completion of the computation. However, B can never gain access to the control word if it has lower priority than A. (Such an obvious hazard can remain undiscovered as long as B always indicates completion of its computation before A tests the control word.)

Articulation

Consider a multiaccess system with a strict-priority discipline in both the port structures and multiplexors as illustrated in Figure 5. In Figure 5 the upward-directed arrows mean that for both the port structures and the multiplexors the relative priority of the input channels increases from bottom to top. With this ordering in mind, observation of the figure reveals that the relative priorities of the devices, D1 through D12, corresponds to their device numbers with D1 having the highest priority.

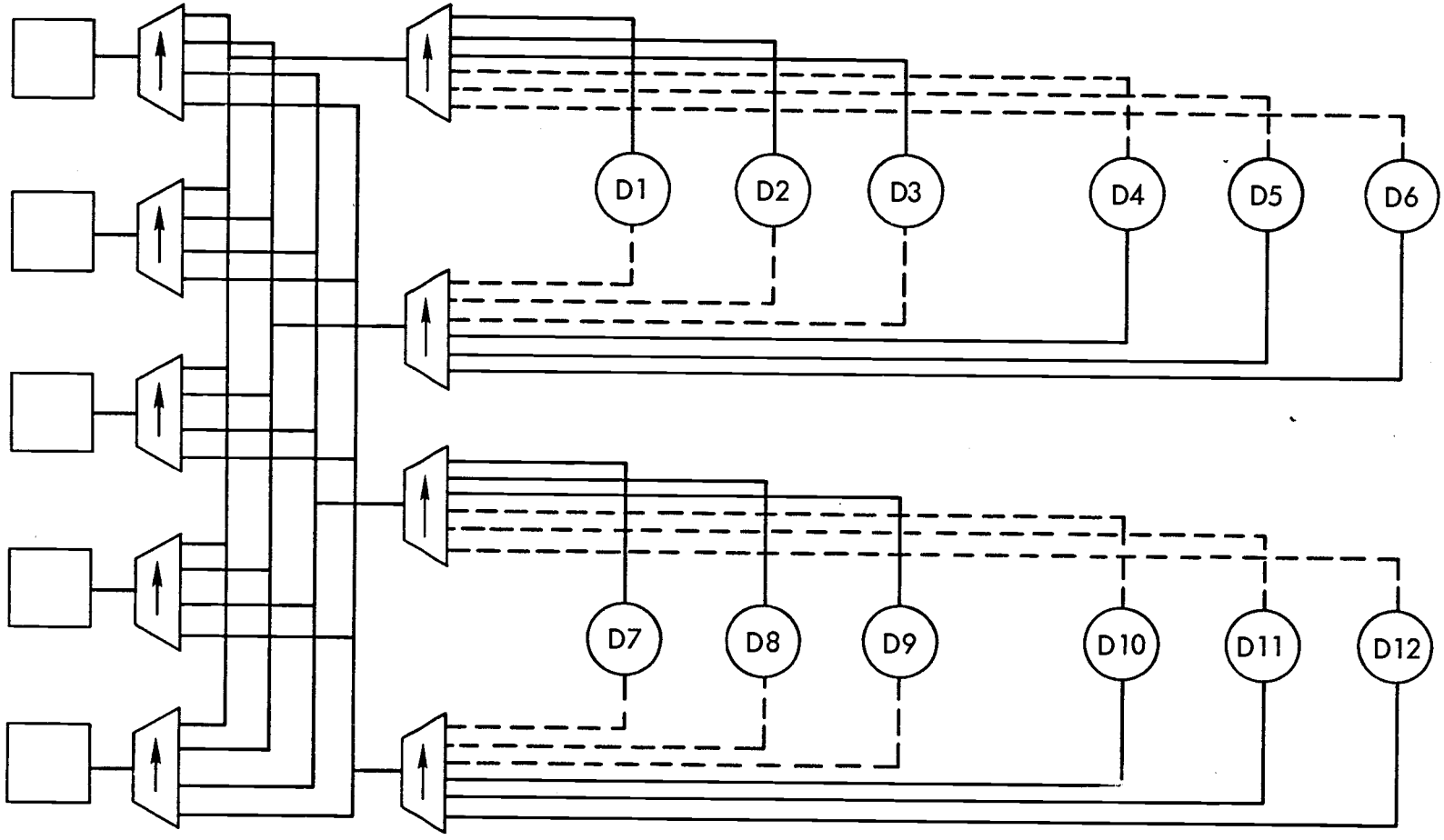


Fig. 5. A reconfigurable dual-access, strict-priority structure.

Note that the multiplexors are arranged in pairs and that the devices associated with each multiplexor-pair are separated into a high-priority group and a low-priority group.

The normal access paths (solid lines) of the high-priority group are the high-priority sub-ports of the high-priority multiplexor and the normal access paths for the low-priority group are the low-priority sub-ports of the low-priority multiplexor. The alternative access paths (dotted lines) of the high-priority group are the high-priority sub-ports of the low-priority multiplexor and the alternative access paths for the low-priority group are the low-priority sub-ports of the high-priority multiplexor. As a consequence of this arrangement, if any multiplexor or port fails, merely activating the alternative access paths of the associated devices will restore all of them to the system without changing the relative priorities of any of the devices. Since the relative priorities do not change, the latencies of the sub-ports will not change (Equation 17); therefore, the patience required of each device is not increased by reconfiguring the system. Consequently, a strict-priority multiaccess memory system can be implemented without introducing intrinsic points of permanent articulation.

Closure

In summary, the strict-priority structure is characterized by variable sub-port latencies and less than 100% efficiency, although it is favorable to the elimination of intrinsic points of permanent articulation.

Combined Schemes

Strict-priority and round-robin schemes are sometimes combined with the objective of serving synchronous devices within the limit of their patience through priority sub-ports, while allowing asynchronous devices to compete for the remaining cycles on the equitable basis established by the round-robin discipline.

Such a combination does have real merit as a system. Because it is no longer absolutely necessary to preserve the latency of the round-robin sub-ports when reconfiguring the system (as they no longer serve devices with limited patience), a dual-access scheme can be employed to eliminate any of the round-robin ports or multiplexors as intrinsic points of permanent articulation.

Naturally, the one or more pairs of strict-priority multiplexors need not introduce permanent articulation points. (A single port having priority over all others would of course be a permanent point of articulation.)

In brief, the mixed structure eliminates the lock-out problem of the strict-priority discipline and the articulation problem of the round-robin discipline. However, reduced efficiency and the more serious galloping problem of the strict-priority structure will still be present.

THE LIMITED - LATENCY STRUCTURE

In this chapter we will introduce a priority structure which allows synchronous devices to be serviced within the limits of their patience without introducing the galloping problem, allows 100% access efficiency without introducing the lock-out problem, and favors an architecture free of intrinsic points of permanent articulation.

We will first illustrate the essentials of this structure, called the limited-latency structure, by example and then discuss how the structure of the example might be implemented. Next, the attributes of generalized limited-latency structures will be developed. After the necessary preliminaries mentioned above, the chapter will conclude with discussions of latency, efficiency, and articulation in the limited-latency structure.

An Example of a Limited-Latency Structure

For the purpose of illustration, but without loss of generality, let us postulate a system with six ports, in which the multiplexor connected to each port serves 16 sub-ports.

Now let us assign priorities 1/2, 1/4, 1/8, 1/16, 1/32, 1/32 to the ports 0 through 5 respectively, where the priority 1/2 assigned to port 0 means that port 0 is guaranteed at least half of the available

memory cycles. Thus, precisely all of the available memory cycles are committed, since the sum of the priorities equals one. (The means for accomplishing this will be discussed in the following section.) Naturally, a particular port may usurp more than its allotted fraction of cycles if the other ports demand less than their guaranteed fraction.

Likewise the sub-ports on each multiplexor are numbered 0 through 15 and are assigned priorities (with respect to the multiplexor) $1/2$ through $1/2^{15}$.

It is immediately clear that sub-port 0 on multiplexor 1 has an over-all priority of $1/4$ with respect to the memory system; and in general the device on sub-port n of multiplexor p has over all priority of $2^{-(n + p + 2)}$.

In the utilization of the system, synchronous devices are assigned to sub-ports for which the sum $(n + p + 2)$ is suitably small. The asynchronous devices are connected to the remaining sub-ports.

It should be observed that, except for extreme values, there will be more than one sub-port having a particular $(n + p + 2)$ sum. Consequently, identical, or equivalent, pieces of equipment need not be connected to the same multiplexor; this is favorable to the elimination of articulation points.

The number of system articulation points can be further reduced by a variation of the preceding scheme. Instead of assigning the ports priorities 1/2, 1/4, 1/8, 1/16, 1/32, 1/32, we assign two equal-priority ports to each of three priority levels; the priority levels have priority 1/2, 1/4, 1/4 resulting in priorities of 1/4, 1/4, 1/8, 1/8, 1/8, 1/8 for the six ports which we now renumber 0A, 0B, 1A, 1B, 2A, 2B. The priority of the sub-ports served by each multiplexor is similarly revised and the 16 sub-ports are renumbered 0a, 0b through 7a, 7b.

Within a particular priority-level, say level M, either of the two sub-ports (ports) Ma or Mb can obtain all the cycles available to that level if the other sub-port (port) is inactive. (The means for accomplishing this will also be discussed in the following section.)

In order to appreciate the advantages of this revised structure let us imagine that the system has been interconnected in the following way:

Each device is connected to two multiplexors on the same port-priority level but it is "enabled" in only one of the two multiplexors. A device connected to sub-port Ma in the A multiplexor is connected to sub-port Ma in the B multiplexor, etc. Half of the devices are enabled in multiplexor A and half the devices are enabled in

multiplexor B.

For a system connected in this way, the devices have the over-all priorities shown in Table I. Each fraction in the table represents a device guaranteed that fraction of memory cycles. The asterisks indicate the position a devices would occupy in the table if its alternate access path were enabled.

Suppose the multiplexor connected to port 1B fails; all the devices enabled in 1B can be switched to multiplexor 1A. Multiplexor 1A will now be able to pre-empt twice as many cycles as before, since multiplexor 1B will be taking no cycles. As a result, the effective port priority of multiplexor 1A is doubled, yielding the over-all priorities shown in Table II.

Observe that each device still has the same over-all priority. Thus the existence of a dual access scheme which eliminates intrinsic points of permanent articulation in a particular limited-latency structure is demonstrated.

TABLE I. NORMAL SUB-PORT PRIORITIES.

Port Priority \ Sub-port		0a	0b	1a	1b	2a	2b	...
		1/2		1/4		1/8		
		1/4	1/4	1/8	1/8	1/16	1/16	...
0A	1/4	1/8	*	1/16 #	*	1/32	*	
1/2		.	.	.	.	.	.	
0B	1/4	*	1/8	*	1/16	*	1/32	...
1A	1/8	1/16	*	1/32	*	1/64	*	
1/4		.	.	.	.	.	.	
1B	1/8	*	1/16	*	1/32	*	1/64	...
2A	1/8	1/16	*	1/32	*	1/64	*	
1/4		.	.	.	.	.	.	
2B	1/8	*	1/16	*	1/32	*	1/64	...

1/4 x (1/8 + 1/8)

TABLE II. ALTERNATE SUB-PORT PRIORITIES.

Sub-port / Port Priority / Priority	0a	0b	1a	1b	2a	2b	...
		1/2		1/4		1/8	
	1/4	1/4	1/8	1/8	1/16	1/16	...
0A (1/4)	1/8	*	1/16	*	1/32	*	
0B (1/4)	*	1/8	*	1/16	*	1/32	...
1A (1/4)	1/16	1/16	1/32	1/32	1/64	1/64	...
1B (-)	*	-	*	-	*	-	
2A (1/8)	1/16	*	1/32	*	1/64	*	
2B (1/8)	*	1/16	*	1/32	*	1/64	...

Implementation

In strict-priority disciplines the multiplexor (or port structure) is required to give the imminent cycle to the highest priority device of those then requesting. Typically this discipline is implemented by having a select signal propagate down a register of request flip-flops until a flip-flop in the set state is encountered. Suitable logic prevents the signal from propagating further, and the forthcoming cycle is given to the device corresponding to the set flip-flop first encountered. The n request flip-flops (RF_0, RF_1 ... RF_{n-1}) corresponding to the n input channels (ports or sub-ports) are encountered in the order q_0:

$$q_0 = RF_0, RF_1 \cdots RF_{n-1}.$$

Priority is determined by position in the order of propagation; the request flip-flop closest to the source of the select signal corresponds to the device of highest priority.

The limited-latency structure can be implemented by a variation of the preceeding scheme. First consider the consequence of inserting the select signal at some logic node other than the one immediately before (in the sense of signal propagation) the channel 0 flip-flop. If, for example, the select signal is inserted immediately before the channel 1 flip-flop, the request flip-flops are encountered in the permuted order RF_1, RF_2, RF_3 ... RF_{n-1}, RF_0, which is denoted

by q_1. The channel 1 request flip-flop is appended to the end of the sequence under the assumption of "end-around" propagation of the select signal.

Thus, by proper choice of the insertion node for the select signal any desired cyclic-permutation of the original order, q_0, can be obtained. Let the n possible permutations be designated

$$q_0, q_1, q_2, \cdots q_{n-1}$$

where the subscripts indicate the number of the first encountered request flip-flop.

Let us assume that the insertion node of the select signal is varied from cycle to cycle by a logical network so that each of the orders $q_0, q_1, \cdots q_{n-1}$ is the order of propagation for some fraction of the total number of cycles. Let f_i correspond to the fraction of time q_i is the order of propagation, and let $\{F\}$ correspond to the sequence $f_0, f_1, f_2 \cdots f_{n-1}$. We must indicate how the logic network steers the select signal so as to produce a required sequence $\{F\}$.

For a moment only, let us not consider the general case but devote our attention to the sequence of the example,

$$f_i = 2^{-(i+1)} \qquad \text{for } i = (1, 2, 3, \ldots n-2), \qquad \text{Equation 19}$$

$$f_i = 2^{-i} \qquad \text{for } i = n-1 \qquad \text{Equation 20}$$

The implementation of this sequence requires, in each multiplexor and port structure, a priority-setting apparatus consisting of a binary counter and a combinatorial decoding network. Each counter is incremented once for every cycle taken by the associated multiplexor or port structure. The outputs of the counter flip-flops are used as inputs to the decoding network which steers the select signal so as to produce q_0 on every other cycle taken by the unit, q_1 on every fourth cycle taken etc. A particular order, q_i, is selected whenever

$$\overline{C}_0 = 1 \quad \text{for } i = 0 \qquad \text{Equation 21}$$

$$\overline{C}_i \bigcap_{u=0}^{i-1} C_u = 1 \quad \text{for } i = 1, 2, \ldots n-2 \qquad \text{Equation 22}$$

$$\bigcap_{u=0}^{i} C_u = 1 \quad \text{for } i = n - I \qquad \text{Equation 23}$$

where C_u is a boolean variable equal to the state of the counter flip-flop with weight 2^u.

Note that if a simultaneous-transition counter is used, the logical condition for selecting q_i is the same as the set condition for the counter flip-flop having weight 2^i (except for $i = n - 1$). Consequently the sequence of the example is particularly economical to implement. Table III exhibits the relationship between the counter flip-flops and the decoder outputs for the example sequence with $n = 4$.

TABLE III. THE STATE DECODING NECESSARY TO IMPLEMENT THE BINARY SEQUENCE FOR THE FOUR-CHANNEL CASE.

C_3	C_2	C_1	C_0	q_0	q_1	q_2	q_3
0	0	0	0	1	0	0	0
0	0	0	1	0	1	0	0
0	0	1	0	1	0	0	0
0	0	1	1	0	0	1	0
0	1	0	0	1	0	0	0
0	1	0	1	0	1	0	0
0	1	1	0	1	0	0	0
0	1	1	1	0	0	0	1
1	0	0	0	1	0	0	0
1	0	0	1	0	1	0	0
1	0	1	0	1	0	0	0
1	0	1	1	0	0	1	0
1	1	0	0	1	0	0	0
1	1	0	1	0	1	0	0
1	1	1	0	1	0	0	0
1	1	1	1	0	0	0	1

For completeness we must indicate how the cycles available to one priority level can be divided between two sub-ports so that either can obtain them all if it is the only one requesting, but neither can obtain more than half when both are requesting. All that is required is a flip-flop and some additional logic for each pair of sub-ports (ports). The flip-flop, called the AB flip-flop is complemented at the completion of every cycle obtained by either member of the pair. The logic—with inputs: the state of the AB flip-flop, the state of the request flip-flops for the two sub-ports (ports), and the select signal corresponding to the particular priority level—simply steers the select signal to give sub-port a priority over sub-port b if the AB flip-flop is in the A state and vice versa.

A more general implementation would be to return to the original scheme of one priority level for each sub-port (port) and add additional steering logic for the select signal at each node-pair so that the order of propagation is

$$q_i = RF_i, RF_{i+1}, RF_{i+2} \cdots RF_{i-2}, RF_{i-1} \quad \text{for even } i, \qquad \text{Equation 24}$$

and

$$q_i = RF_i, RF_{i-1}; RF_{i+2}, RF_{i+1}; \cdots RF_{i-2}, RF_{i-3} \quad \text{for odd } i. \qquad \text{Equation 25}$$

Thus, if j is even and $f_j = f_{j+1}$, the cycles will divide equally as required. However, when an equal division of the available cycles between the pair of sub-ports is all that is required, the preceding AB flip-flop scheme is more economical.

Attributes of Generalized Limited-Latency Structures

No doubt a sequence of priorities corresponding to the binary sequence seems somewhat ad hoc. We will now determine what other sequences will work.

The sequence $\{U_p\}$ is to represent the priorities of the P ports with respect to the port structures, and the sequence $\{V_n\}$ is to represent the relative priorities of the N sub-ports associated with each multiplexor. Since U_j represents the fraction of total available memory cycles guaranteed to port j,

$$\sum_{j=0}^{P-1} U_j = 1; \qquad \text{Equation 26}$$

similarly

$$\sum_{k=0}^{N-1} V_k = 1. \qquad \text{Equation 27}$$

It then follows that

$$\sum_{j=0}^{P-1} U_j \sum_{k=0}^{N-1} V_k = \sum_{j=0}^{P-1} \sum_{k=0}^{N-1} U_j V_k = 1,$$

Equations 28 & 29

which expresses the fact that the sum, over all sub-ports, of guaranteed-fractions-of-available-memory-cycles must equal one. Consequently we are required to restrict the sequences $\{U_p\}$ and $\{V_n\}$ so that the corresponding series each converge to one. This is our first

restriction on the terms of the priority sequences.

Until now it has been understood, but not stated, that if a channel is guaranteed 1/4 of the cycles it has top priority once every 4 cycles instead of on 3 consecutive cycles out of every 12, for example. The assumed distribution is a natural one, for it yields the minimum latency obtainable with any allocated fraction of cycles. Consequently we now state the requirement: The cycles guaranteed to a particular channel (port or sub-port) are to be uniformly distributed in the train of cycles obtained by the multiplexing unit (port-structure or multiplexor).

From this requirement it follows immediately that each term of $\{U_i\}$ and each term of $\{V_i\}$ must be of the form $1/a_i$, where a_i is a positive integer not equal to zero. This is our second restriction on the terms of the priority sequences.

However it is clear that the terms 1/3 and 1/4 can not be included in the same priority sequence (the two corresponding channels would at times be "guaranteed" the same cycle) so there must be additional constraints on the terms of the sequences. Choose any two terms from a priority sequence, $1/a_i$ and $1/a_j$, where a_i and a_j are both natural numbers. We desire that channel a_i be guaranteed one cycle every a_i cycles and channel a_j be guaranteed one cycle every a_j cycles without conflict for any i

and j in (0, 1, 2, ... n-1), i.e., for any pair of terms. How does this requirement restrict the choice of the sequence of integers $\{a_i\}$?

First observe that, since the train of cycles to be allocated can be placed in one-to-one correspondence with the integers, the given problem is equivalent to determining the restrictions on the integers a_i and a_j so that the set A_i, comprised of every a_i^{th} integer beginning with some integer b_i, and the set A_j, comprised of every a_j^{th} integer beginning with some integer b_j, are disjoint for any i and j in (0, 1, 2, ... n-1). In other words, a set A_i is the set of integers congruent to b_i modulo a_i;

$$A_i = \{a_i \ni a_i \equiv b_i \bmod a_i\}, \qquad \text{Equation 30}$$

and we require

$$A_i \cap A_j = \phi \quad \text{for all } i, j \in (0, 1, 2 \ldots n-1) \qquad \text{Equation 31}$$

Let us assume $A_i \cap A_j \neq \emptyset$ for some i and j, then for some pair of integers x and y

$$x\, a_i + b_i = y\, a_j + b_j. \qquad \text{Equation 32}$$

Without loss of generality we assume

$$i < j \Rightarrow a_i \leqslant a_j$$

so that

$$\frac{a_j}{a_i} = q_{ij} + \frac{r_{ij}}{a_i} \quad \text{where } q_{ij} \text{ and } r_{ij} \text{ are integers;} \qquad \text{Equation 33}$$

$$\therefore \quad x\,a_i + b_i = y\,q_{ij}\,a_i + y\,r_{ij} + b_j \qquad \text{Equation 34}$$

$$\therefore \quad b_i \equiv y\,r_{ij} + b_j \pmod{a_i}. \qquad \text{Equation 35}$$

If

$$r_{ij} = 0 \qquad \text{Equation 36}$$

and

$$b_i \not\equiv b_j \pmod{a_i} \text{ for all i and j except i=j,} \qquad \text{Equation 37}$$

then there is no i and j such that Equation 35 holds. Therefore the restrictions—

$$(a_i, a_j) = a_i \qquad \text{Equation 38}$$

(the greatest common divisor of a_i and a_j is a_i) and

$$b_i \not\equiv b_j \pmod{a_i}. \text{ —} \qquad \text{Equation 39}$$

are sufficient to assure

$$A_i \cap A_j = \emptyset \quad \text{for all i, j, (Equation 31)}$$

as required. These are our third and fourth restrictions on the terms of the priority sequences.

The restriction of Equation 38 can always be met by choosing the elements of the set $\{a_i\}$ in the following way:

$$a_0 = x \qquad \text{Equation 40}$$

$$a_i = y_i\,a_{i-1} \qquad \text{for } i > 0 \qquad \text{Equation 41}$$

where x and all the y_i are arbitrary positive integers.

The restriction of Equation 39 can always be met by letting

$$b_0 = 0$$

and choosing b_i from S_i,

$$S_i = \overline{\bigcup_{u=0}^{i-1} A_u} \quad \text{for } i > 0. \qquad \text{Equation 42}$$

Clearly this can always be done for, if

$$S_i = \emptyset \qquad \text{Equation 43}$$

for some $i = j$
$j \in (0, 1, 2 \ldots n-1)$,

then

$$\sum_{\substack{i=1 \\ i \neq j}}^{n-1} 1/a_i = 1, \qquad \text{Equation 44}$$

which violates the first restriction placed on the terms of the priority sequences.

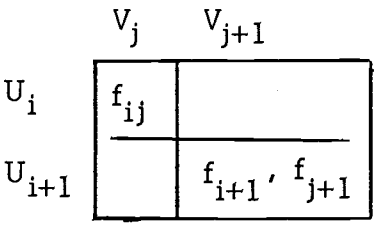
To complete our generalization of the limited-latency structure, we must ascertain what if any, additional restrictions must be placed on the terms of $\{U_i\}$ and $\{V_j\}$ in order to implement a dual-access architecture without intrinsic points of permanent articulation.

Consider the general over-all priority table, Table IV. If a particular multiplexor fails, the i^{th} or $i + 1^{st}$, let us say, then we require the fraction of cycles guaranteed to all sub-ports be unchanged when the devices originally enabled in multiplexor i are enabled in multiplexor i + 1 (or vice versa).

TABLE IV. GENERAL CASE, OVER-ALL PRIORITY TABLE.

Port Priority / Sub-port Priority	$V_0 + V_1$ (V_0, V_1)	$V_2 + V_3$ (V_2, V_3)	~	$V_{n-2} + V_{n-1}$ (V_{n-2}, V_{n-1})
$U_0 + U_1$ (U_0, U_1)	f_{00} / f_{11}	f_{02} / f_{13}	~	$f_{0, n-2}$ / $f_{1, n-1}$
$U_2 + U_3$ (U_2, U_3)	f_{20} / f_{31}	f_{22} / f_{33}	~	$f_{2, n-2}$ / $f_{3, n-1}$
. . .			~	
$U_{P-1} + U_{P-2}$ (U_{P-2}, U_{P-1})	$f_{P-2, 0}$ / $f_{P-1, 1}$	$f_{P-2, 2}$ / $f_{P-1, 3}$	~	$f_{P-2, n-2}$ / $f_{P-1, n-1}$

Originally we have the arrangement

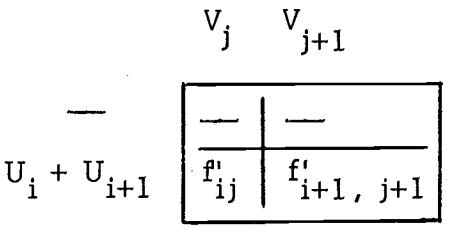


where

$$f_{ij} = U_i \times (V_j + V_{j+1}) \quad \text{Equation 45}$$

$$f_{i+1,\,j+1} = U_{i+1} \times (V_j + V_{j+1})$$
$$i, j \in (0, 2, 4 \ldots n-2). \quad \text{Equation 46}$$

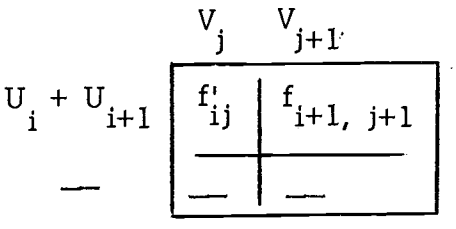
After reconfiguration we have one of the arrangements,



where

$$f'_{ij} = (U_i + U_{i+1}) \times V_j \quad \text{Equation 47}$$

$$f'_{i+1,\,j+1} = (U_i + U_{i+1}) \times V_{j+1} \quad \text{Equation 48}$$



for either arrangement.

We require

$$f_{ij} = f'_{ij} \quad \text{Equation 49}$$

and

$$f_{i+1,\,j+1} = f'_{i+1,\,j+1}; \quad \text{Equation 50}$$

$$\therefore \quad U_i \times (V_j + V_{j+1}) = (U_i + U_{i+1}) \times V_j \quad \text{Equation 51}$$

and

$$U_{i+1} \times (V_j + V_{j+1}) = (U_i + U_{i+1}) \times V_{j+1}. \quad \text{Equation 52}$$

It then follows that

$$\frac{U_i}{U_{i+1}} = \frac{V_j}{V_{j+1}} \quad \text{Equation 53}$$

which must hold for all i and j. Consequently we must make the restriction

$$\frac{U_i}{U_{i+1}} = \frac{V_j}{V_{j+1}} = R, \quad \text{a constant,}$$

$$\text{for all } i, j \in (0, 2, 4, \ldots n\text{-}2).$$

Equations 54 & 55

This is our fifth restriction on the terms of the priority sequences.

It is important to observe that the dual-access requirement has in no way restricted the ratios

$$\frac{U_i + U_{i+1}}{U_k + U_{k+1}} \quad \text{and} \quad \frac{V_i + V_{i+1}}{V_k + V_{k+1}}$$

where $i \neq k$, and $i, k \in (0, 1, 2 \ldots n\text{-}2)$.

Suppose that sub-port pairs were not arranged as shown in Table 4 but as shown below:

	V_j	V_{j+1}
U_j		$f_{i+1,\,j+1}$
U_{j+1}	f_{ij}	

This arrangement leads to the requirement

$$\frac{U_{i+1}}{U_i} = \frac{V_j}{V_{j+1}}. \qquad \text{Equation 56}$$

It is desirable to allow both arrangements within the same system so that the multiplexor connections for a single pair of devices can be changed for diagnostic purposes. Consequently we require

$$\frac{U_i}{U_{i+1}} = \frac{V_j}{V_{j+1}} = \frac{U_{i+1}}{U_i} = 1 .$$

for all i, j, ϵ(0, 2, 4...n-2).

Equations 57, 58 & 59

Equation 60 represents our sixth restriction on the terms of the priority sequences.

There is additional justification for this last restriction. Note that in the normal configuration a device can pre-empt all of the fraction of cycles guaranteed to two sub-ports (and in the alternate configuration it usurps a fraction of cycles guaranteed to two ports), i.e., $V_j + V_{j+1}$ (or $U_i + U_{i+1}$). As before, we want these cycles to be uniformly distributed in the train of all cycles, in order to minimize latency. A uniform distribution could be obtained, if $a_j = a_{j+1}$ (as it must if Equation 59 is to hold), by requiring

$$b_{j+1} = b_j \pm a_j/2. \qquad \text{Equation 60}$$

But unless we require a_j to be an even integer we can only require

$$b_{j+1} = b_j \pm \lceil a_j/2 \rceil. \qquad \text{Equation 61}$$

This is our seventh restriction on the terms of the priority sequences. Naturally, the "uneveness" of the distribution, in the case of a_j odd, must be taken into account in determining sub-port latencies.

In summary we can state that the sequence $\{U_i\}$ of port priorities in the limited-latency structure must meet these six restrictions:

1. $\sum_{i=0}^{P-1} U_i = 1.$

2. $U_i = U_{i+1}$ for i even.

3. $U_i = 1/a_i$ $a_i \in (1, 2, 3, 4 \ldots)$.

4. $(a_i, a_j) = a_i$ where $i < j \Rightarrow a_i \leq a_j$.

5. $b_i \neq b_j \pmod{a_i}$ for $i \neq j$ and $a_i \equiv b_i \pmod{a_i}$.

6. $b_{i+1} = b_i \pm \lceil a_i/2 \rceil$ for i even and $a_i \equiv b_i \pmod{a_i}$.

Identical restrictions apply to the sequence $\{V_j\}$ of sub-port priorities.

Note that the restrictions on $\{U_i\}$ and $\{V_j\}$ were derived from necessary or desirable properties of the limited-latency structure itself and in no way reflect limitations or idiosyncrasis of a particular hardware implementation of the structure. On the contrary, the implementation used in the example, namely the exhaustive partitioning of the states of a counter into n disjoint sets, can readily be applied to sequences other than the binary sequence.

Sub-port Latencies in the Limited-Latency Structure

The latency of a sub-port in a limited-latency structure is not precisely $(U_i (V_j + V_{j+1}))^{-1}$ as one might infer from Equation 45. Recall that if a particular device is not requesting service its

"guaranteed" cycles may be usurped by other devices. It can happen, then, that the j^{th} device will assert its request line just as one of "its" cycles is being given to some other device. Consequently, device j will have to wait—one cycle-time for the just-missed cycle to complete, plus $\left[(V_j + V_{j+1})^{-1}\right] - 1$ cycle-times before the beginning of "its" next cycle, plus one cycle-time to be serviced—a total of $\left[(V_j + V_{j+1})^{-1}\right] + 1$ multiplexor cycle-times. (Since we have not restricted the a_i of the priority sequences to even integers, we must use the least integer greater than or equal to $(V_j + V_{j+1})^{-1}$ in calculating latencies.)

The multiplexors themselves fare no better with respect to the port-structure. Even if we assume a multiplexor makes its $n + 1^{st}$ request immediately after its n^{th} request has been acknowledged, it will still "just-miss" some of "its" cycles because successive requests coming from different devices are most likely addressed to different memory modules, and the priority-setting apparatuses in their different port structures are not likely to be in phase. Therefore, we may proceed as before and conclude that each multiplexor cycle-time can be as long as $U_i^{-1} + 1$ memory cycle-times for the i^{th} multiplexor.

It follows that the latency for sub-port ij is given by

$$L_{ij} = (U_i^{-1} + 1)\left(\left[(V_j + V_{j+1})^{-1}\right] + 1\right), \quad \text{Equation 62}$$

which, recalling the restriction $V_j = V_{j+1}$ for even j, reduces to

$$L_{ij} = (U_i^{-1} + 1)\ ((2V_j)^{-1} + 1) \qquad \text{Equation 63}$$

or

$$L_{ij} = (2U_i\ V_j)^{-1} + U_i^{-1} + (2V_j)^{-1} + 1 \qquad \text{Equation 64}$$

if V_j^{-1} is an even integer.

The preceding expression gives the latency of sub-port ij in the normal configuration; by similar reasoning the latency for the alternate configuration is given by

$$L'_{ij} = (\left[(U_i + U_{i+1})^{-1}\right] + 1)\ (V_j^{-1} + 1), \qquad \text{Equation 65}$$

which reduces to

$$L'_{ij} = ((2U_i)^{-1} + 1)\ (V_j^{-1} + 1) \qquad \text{Equation 66}$$

or

$$L'_{ij} = (2U_i\ V_j)^{-1} + (2U_i)^{-1} + V_j^{-1} + 1 \qquad \text{Equation 67}$$

if U_i^{-1} is an even integer.

When $U_i = V_j$, $L_{ij} = L'_{ij}$ and there is no ambiguity in specifying the latency of the sub-port. But, in general, there will be sub-ports in the system for which $U_i \neq V_j$. For these sub-ports, in order to comply with our definition of latency, we must consider the larger of L_{ij} and L'_{ij} to be the latency of sub-port ij.

It is apparent, certainly, that the latencies for the fastest pair of sub-ports in a limited-latency system[6] will be greater than

[6] Latencies of 12 memory cycle-times correspond to the sequences 1/3, 1/3, $U_2 \ldots U_{P-1}$ and 1/3, 1/3, ... V_{n-1}.

the latency of the highest priority sub-port in a strict-priority system. However, with a limited-latency structure, the latency of a particular sub-port is a function only of well-defined parameters and is independent of system variables, such as the transfer rate of devices connected to other sub-ports.

This insensitivity to variations in system activity eliminates the galloping problem. In a limited-latency structure the controllers for synchronous devices may contain several registers for data-word buffering plus one or two registers for control-word buffering. It then becomes the burden of the device controller to insert control-word requests into the stream of its data-word requests at propitious times—judged from local considerations only—when all data-word registers are full on an out-of-memory transfer, for example. In practice, a device trying to process an intricately chained data list may or may not be successful in obtaining all necessary control words, but at least its hyperactivity (pathological, perhaps) can not cause transfer-timing errors in some other device.

Access Efficiency of the Limited-Latency Structure

The limited-latency structure's characteristic insensitivity of sub-port latency to system variables is beneficial to access

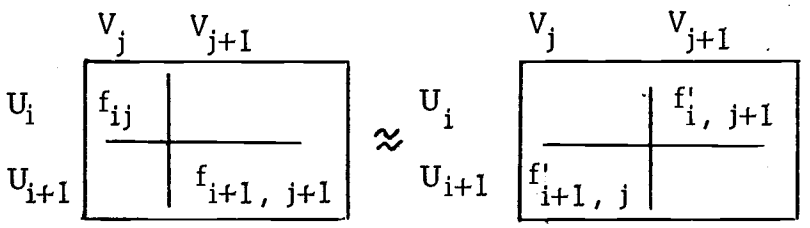
efficiency. Cycle-sinks can be connected to any number of sub-ports on any number of ports, easily producing an access efficiency of 100%. Even if all devices in the system were cycle-sinks, the lock-out problem would not arise; the available cycles would simply be allocated in proportion to the over-all sub-port priorities.

Articulation in the Limited-Latency Structure

The restrictions (listed at the end of "Attributes of Generalized Limited-Latency Structures") placed on the terms of the priority sequences $\{U_i\}$ and $\{V_j\}$ were largely derived from the requirement that the structure must have a dual-access scheme which preserves latency for all sub-ports when the system is reconfigured and thus introduce no intrinsic points of permanent articulation. The requirement is met. However the dual-access scheme of the limited-latency structure has two advantages not found in the scheme of the strict-priority structure.

First, the limited-latency structure tends to produce multiple channels with the same latency, which encourages the incorporation of more than one unit of a particular type of device into the system. Naturally a plurality of equivalent devices, as opposed to unique units, tends to make the devices themselves temporary, instead of permanent, articulation points.

The second advantage is that in the limited-latency structure a single pair of devices can be reconfigured, i.e.,



$$f_{ij} = f'_{i+1,\,j} \qquad \text{Equation 68}$$

$$f_{i+1,\,j+1} = f'_{i,\,j+1}, \qquad \text{Equation 69}$$

with little or no change (depending upon the relative demands of the pair of devices involved) in the relative number of cycles being obtained through each multiplexor. Thus, in the event of subtle, intermittent failures associated with a particular sub-port, it is a simple matter to change configurations and thereby determine if the problem is multiplexor- or device-oriented. In a strict-priority system; however, groups of devices must be switched from multiplexor to multiplexor so as not to permute the relative priorities of the devices. See Figure 5. In the worst cases (the highest priority device of the higher priority multiplexor or the lowest priority device of the lower priority multiplexor is suspect), all devices from both multiplexors must be enabled in only one of them. The cycles available to the idle multiplexor are entirely wasted, making the test-run expensive in terms of reduced activity for any asynchronous

devices involved.

Summary

The limited-latency structure is favorable to high access efficiency and the elimination of articulation points. Furthermore the patience required of devices is explicitly defined and constant for all devices using the structure. However, the fastest allowed device can not be as fast as it could be in a strict-priority structure.

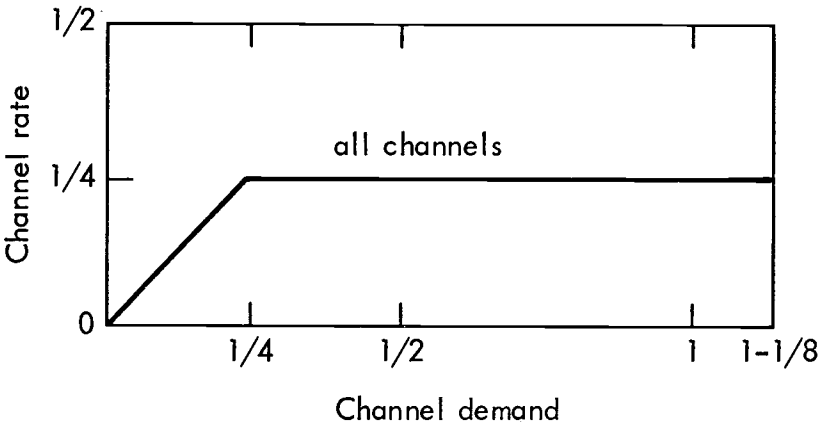
ADDITIONAL CONCLUSIONS

The Spectrum of Priority Structures

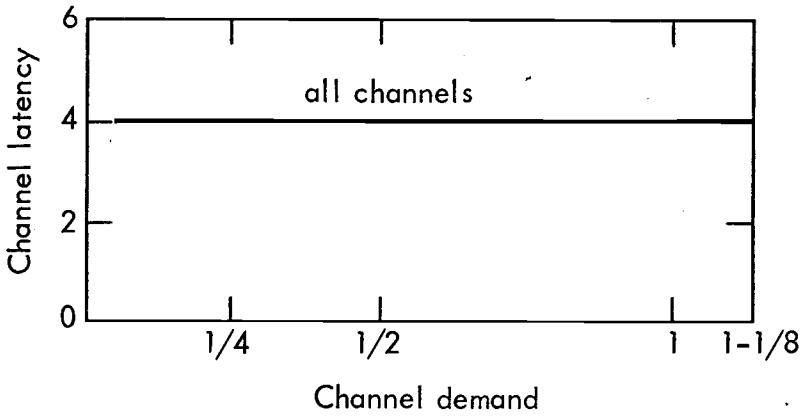
We have already shown that the limited-latency structure is generally superior to other structures with respect to the ancillary goals of access efficiency and freedom from permanent articulation points. However, there are still some observations to be made, particularly with respect to the primary goal of a priority policy, serving each device within the limits of its patience.

It must be assumed that every priority structure will meet its primary goal for as long as the number of devices and their rates do not deviate from the values specified at the time the system was defined. However, evolving systems are the rule rather than the exception, so let us compare the three principle disciplines in a dynamic situation. We will compare the disciplines when the rate of devices (the demand of channels) is varied.

Consider three implementations of a one-level (for the sake of simplicity), four-channel priority structure, first come-first serve, strict-priority, and limited-latency. Again for the sake of simplicity, we will assume that all channels are demanding the same number of accesses per unit-time. The number of accesses actually allocated to a particular channel, channel rate, and channel latency are plotted

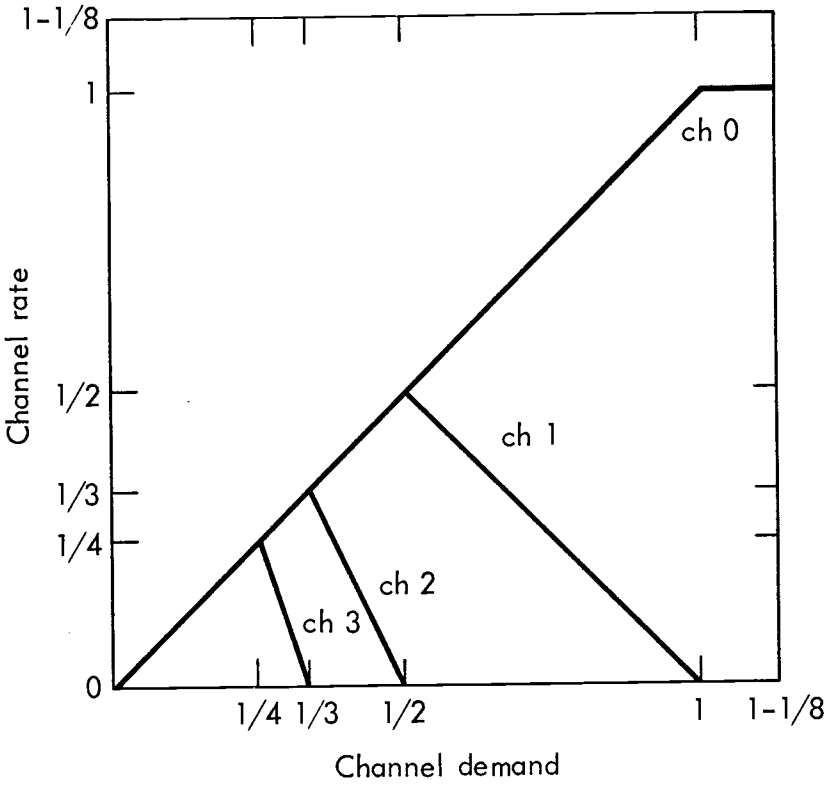


6a. Rate versus demand, both in units of memory cycle-rate.

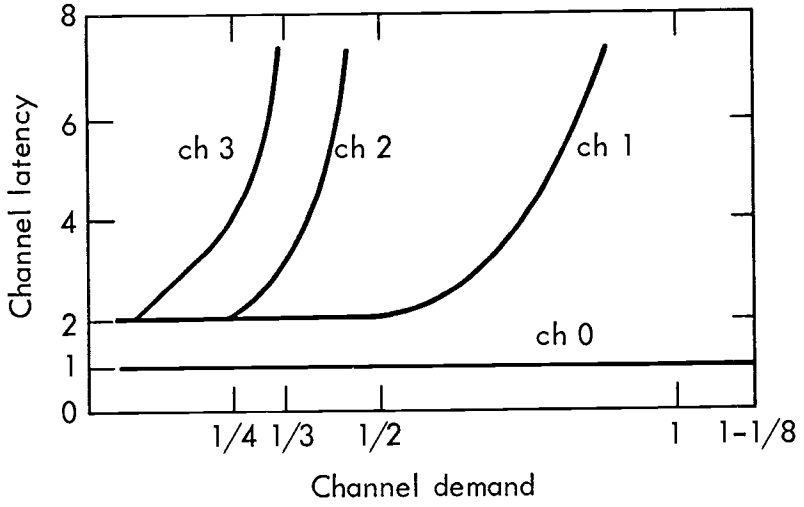


6b. Latency in units of memory cycle-times versus demand in units of memory cycle-rate.

Fig. 6. Rate and latency versus channel demand (uniform over all channels) in a one-level, four-channel, first come-first serve structure.

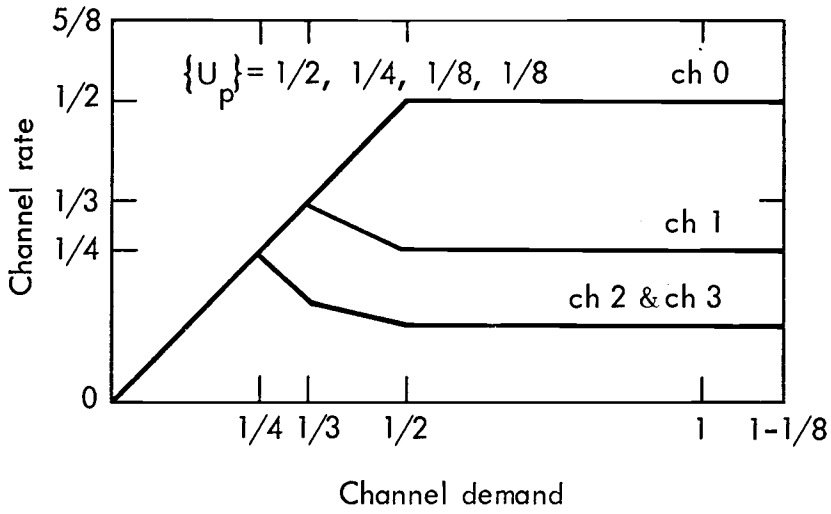


7a. Rate versus demand, both in units of memory cycle-rate.

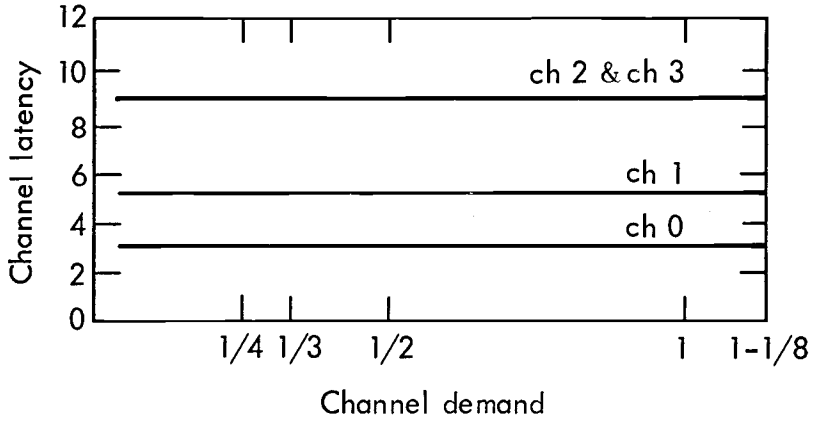


7b. Latency in units of memory cycle-times versus demand in units of memory cycle-rates.

Fig. 7. Rate and latency versus channel demand (uniform over all channels) for four channels in a one-level strict-priority structure.



8a. Rate versus demand, both in units of memory cycle-rate.



8b. Latency in units of memory cycle-times versus demands in units of memory cycle-rate.

Fig. 8. Rate and latency versus channel demand (uniform over all channels) for four channels in a one-level limited-latency structure.

against this uniform channel demand for all three structures in Figures 6, 7, and 8.

As expected, the channel rate increases to meet channel demand for all channels, for all structures, until the sum of the rates equals the transfer capability of the memory. It is not until demand exceeds capability that the structures exhibit their inherent characteristics. (In studying these curves we should not limit our concept of increasing demand to that increase of a few percent per month which represents growth in system acceptance and utilization. We must keep in mind the large variations in demand, several hundred percent, that occur from millisecond to millisecond as a result of the asynchronous operation of multiple memory-accessing devices.)

The first come-first serve structure divides the available accesses equally among the channels no matter how great the demand. No channel receives service equal to its demands, but then no channel is entirely deprived of service, either. The latency of all channels is constant and independent of system demand.

In the strict-priority structure, as demand increases, service to the lowest-priority channel is decreased to zero, as demand increases further, service to the second-lowest-priority channel is decreased to zero. This successive elimination of lower-priority channels continues until the highest-priority channel has usurped the entire

service capability of the memory. When we note the steeply-rising latency curves for this structure and imagine the demand-operating point varying widely and rapidly, it is not difficult to appreciate the subtlety of the anomalies which can arise in such a system, as discussed in chapter 3.

The rate vurses demand curves for the limited-latency structure are interesting in that they illustrate the fact that cycles not requested by the highest-priority channel are distributed among the lower-priority channels in proportion to the corresponding terms of the priority sequence. For example, when the demand on all channels is equal to 5/12 R, the 1/12 R of unused cycles guaranteed to the highest-priority channel are distributed among the lower-priority channels so that channel 1 receives 1/24 R cycles and channels 2 and 3 each receive 1/48 R cycles in proportion to the terms 1/4, 1/8, 1/8 of the priority sequence.

Note that the rate curves for the limited-latency structure are initially similar to those of the strict-priority structure in that the curves for the lower priority channels have a negative slope, but that as channel demand is further increased they have zero slope similar to those of the first come-first serve structure. These similarities suggest that the limited-latency structure is a compromise between the other two. However, it may be more accurate to view

strict-priority and first come-first serve as special cases of the general limited-latency structure.

Clearly, the first come-first serve structure of our example could have been implemented as a limited-latency structure having the priority sequence (1/4, 1/4, 1/4, 1/4). Similarly, the strict-priority structure could have been implemented as a "limited"-latency structure having the priority sequence (1, 0, 0, 0), a sequence obtained as a limit when a_1, a_2, and a_3 are allowed to approach infinity. Some additional thought allows us to make the observation that any first come-first serve or strict-priority structure can be realized as a particular limited-latency structure.

Of course, some of the problems associated with the original schemes would remain; a limited-latency system with priority sequence (1, 0, 0 ... 0) would be vulnerable to galloping and lock-out, for example. However, the purpose of the observation is not to support a proposal that all priority structures actually be built as limited-latency structures with particular parameters; its purpose is to illustrate the fact that the limited-latency structure actually encompasses a spectrum of possible structures which includes first come-first serve and strict-priority. Consequently, with a design approach based on the principles of the limited-latency structure there is some hope of producing systems that can be adapted to

unanticipated requirements; this hope is not realistic with an ad hoc design approach.

Central Versus Local Control

In their paper Wallace and Rowswell (10) introduce an ideal priority structure as a comparison standard. The ideal priority structure allocates each imminent memory access to the requesting device having the smallest amount of unexpired patience. Thus, if there exists some order of servicing the requesting devices so as not to introduce transfer-timing errors, the ideal priority structure will service them in precisely that order. Consequently, if there exist conditions under which the ideal structure would fail to serve every device within the limits of its patience, then any other structure must necessarily fail under those conditions also; hence, the name ideal.

Indeed, the structure is ideal in two respects; it will, if possible, always serve each device within the limits of its patience, and its tolerance of cycle-sinks makes it capable of 100% access efficiency. However, the structure can not escape being a point of permanent articulation. This conclusion is patent when we note that the task of comparing the patience fragments of all devices can not be partitioned among several independent units. Even if we were willing to accept such a vulnerable system (no more vulnerable than some systems

actually in use), it is impossible to actually build an ideal priority structure because it requires a counter (or other time-keeper) for every device and an immense tree of comparison logic, all of which must operate with essentially zero gate-propagation delay.

In contrast to the central-control approach of the ideal priority structure, the limited-latency structure makes all allocation decisions on the basis of local considerations only, for example, the state of the cycle-counter (just as devices within the structure choose the times to make their control-word requests on the basis of local considerations only). While the limited-latency approach does solve some problems like galloping and lock-out, which are not strictly local in nature, without introducing centralized control, it can not optimize the utilization of the memory to the degree theoretically obtainable with the ideal structure. However, the local autonomy of the limited-latency scheme leads to a modular system favorable to the elimination of permanent articulation points.

In the opinion of the author, future designs of large computer systems will exhibit an increasing tendency to favor modularity and freedom from permanent articulation over minimization and maximized resource utilization.

Closure

We have developed a general priority structure for multiaccess systems that meets all three of the salient requirements enumerated in chapter 2. Therefore, we can conclude that responsiveness, efficiency, and freedom from points of permanent articulation are, fortunately, not contradictory system attributes. However, only those systems which have these three characteristics as design goals at the outset are likely to exhibit them as final characteristics.

BIBLIOGRAPHY

1. Busacker, Robert G. and Thomas L. Saaty. Finite graphs and networks: an introduction with applications. New York, McGraw-Hill, 1965. 294 p.

2. Feller, William. An introduction to probability theory and its applications. 2d ed. Vol. 1. New York, Wiley, 1957. 461 p.

3. Fernbach, Sidney and Joseph E.Wirsching. Parallelism in computing systems. Livermore, 1965. 15 p. (University of California. Lawrence Radiation Laboratory. UCRL - 12293-T)

4. Grosswald, Emil. Topics from the theory of numbers. New York, Macmillan, 1966. 298 p.

5. Hellerman, H. On the average speed of a multiple module storage system. Institute of Electrical and Electronic Engineers Transactions on Electronic Computers 15: 670. 1966.

6. Leclerc, Jean-Yyes. Memory structures for interactive computers. Ph. D. Thesis. Berkeley, University of California, 1966. 79 numb. leaves.

7. McLaughlin, R. A. The CDC 7600. Datamation 15: 61-64. Jan., 1969.

8. Saaty, Thomas L. Elements of queueing theory with applications. New York, McGraw-Hill, 1961. 423 p.

9. Staudhammer, John, C. A. Combs and Gordon Wilkinson. Analysis of computer peripheral interference. In: Proceedings of 22nd National Conference, Association for Computing Machinery, 1967. Washington, D.C. Thompson Book Company, 1967. p. 97-101. (Association for Computing Machinery. Publication P-67)

10. Wallace, C. S. and B. G. Rowswell. Competition for memory access in the KDF9. The Computer Journal 10: 64-68. 1967.

11. Wegner, Peter. Machine organization for multiprogramming. In: Proceedings of 22nd National Conference, Association for Computing Machinery, 1967. Washington, Thompson Book Company, 1967. p. 135-150. (Association for Computing Machinery. Publication P-67)

12. Wilkes, M. V. and R. M. Needham. The design of multiple-access computer systems. Part 2. The Computer Journal 10: 315-320. 1967

APPENDICES

Appendix I: A Bound for the Maximum Waiting Period for the i^{th} Channel in a Strict-Priority Structure

In a strict-priority structure consisting of n channels, the waiting period of the i^{th} channel is given by the minimum solution of the integer equation

$$W_i = \sum_{u=1}^{i-1} \left[W_i R_u \right] + 1 \qquad \text{Equation A1}$$

where $[x]$ means the least integer greater than or equal to x, R_u is the maximum rate at which the device connected to the u^{th} channel can issue requests, and time is in units of a constant service time (10, p. 66). See chapter 3, page 26.

If W_i is the least integer solution of Equation A1, then for any other integer m, $1<m<W_i$,

$$(W_i - m) < \sum_{u=1}^{i-1} \left[(W_i - m) R_u \right] + 1. \qquad \text{Equation A2}$$

We substitute Equation A1 into A2 and obtain

$$1 - m + \sum_{u=1}^{i-1} \left[W_i R_u \right] < 1 + \sum_{u=1}^{i-1} \left[(W_i - m) R_u \right]. \qquad \text{Equation A3}$$

$$\therefore \quad \sum_{u=1}^{i-1} \left[W_i R_u \right] - \sum_{u=1}^{i-1} \left[(W_i - m) R_u \right] < m. \qquad \text{Equation A4}$$

But all terms are integers; therefore,

$$\sum_{u=1}^{i-1} \left\{ \left[W_i R_u \right] - \left[(W_i - m) R_u \right] \right\} \leqslant m - 1. \quad \text{Equation A5}$$

Since each of the terms $\left\{ \left[W_i R_u \right] - \left[(W_i - m) R_u \right] \right\}$ is either zero or a positive integer, at most $m - 1$ of them are non-zero.

Now, if we let $m = k$, $k \epsilon (1, 2, 3 \ldots i-1)$ for $k = 1$ we have

$$\sum_{u=1}^{i-1} \left\{ \left[W_i R_u \right] - \left[(W_i - 1) R_u \right] \right\} \leqslant 0 \quad \text{Equation A6}$$

but there are, at most, zero non-zero terms, i.e. there are at least $i-1$ values of u, such that

$$\left[W_i R_u \right] = \left[(W_i - 1) R_u \right]. \quad \text{Equation A7}$$

Similarly for $k = 2$ we have

$$\sum_{u=1}^{i-1} \left\{ \left[W_i R_u \right] - \left[(W_i - 2) R_u \right] \right\} \leqslant 1. \quad \text{Equation A8}$$

Therefore, at most, one of the terms is non-zero. Consequently there are at least $i - 2$ values of u, such that

$$\left[W_i R_u \right] = \left[(W_i - 2) R_u \right]. \quad \text{Equation A9}$$

Continuing the argument, we arrive at the conclusion that for $k = n$ there are at least $i - n$ values of u, $\{u^*\}$, such that

$$\left[W_i R_{u^*} \right] = \left[(W_i - k) R_{u^*} \right]. \quad \text{Equation A10}$$

Therefore, by choosing one such value of u^* for each value of k, and taking care not to use any value of u^* more than once (easily

accomplished by starting with k = i-1 and working "backwards" to k = 1), we can assemble a set of k equations

$$\left[W_i\, R_{u^*}\right] = \left[(W_i - k)\, R_{u^*}\right] \text{ for } k = 1,\ 2,\ 3 \ldots i-1.$$

By adding all the equations of the ensemble together we obtain

$$\sum_{k=1}^{i-1} \left[W_i\, R_{u^*}\right] = \sum_{k=1}^{i-1} \left[(W_i - k)\, R_{u^*}\right], \qquad \text{Equation A11}$$

where $\{u^*\}$ is some permutation of the integers 1, 2, 3 ... i-1. Consequently,

$$\sum_{k=1}^{i-1} \left[W_i\, R_{u^*}\right] = \sum_{k=1}^{i-1} \left[W_i\, R_u\right] = W_i - 1; \qquad \text{Equations A12 \& A13}$$

therefore,

$$W_i = \sum_{k=1}^{i-1} \left[(W_i - k)\, R_{u^*}\right] + 1. \qquad \text{Equation A14}$$

To obtain a bound on W_i we replace $[x]$ by $(x + 1)$ in Equation A14 and solve for W_i, resulting in (10, p. 66)

$$W_i \leqslant \frac{i - \sum_{k=1}^{i-1} k\, R_{u^*}}{1 - \sum_{k=1}^{i-1} R_u}, \qquad \text{Equation A15}$$

where we require $\sum_{k=1}^{i-1} R_u < 1$, which means the channels of

priority higher than i have not preempted the entire service capability.

The term $\sum_{k=1}^{i-1} k R_{u*}$ will be minimized by that permutation for which

$k_i < k_j \Rightarrow R_i \geq R_j$. Consequently, if the channels are originally numbered in order of decreasing rate we have the bound

$$W_i \leqslant \frac{i - \sum_{u=1}^{i-1} u R_u}{1 - \sum_{u=1}^{i-1} R_u}, \qquad \text{Equation A16}$$

developed with less detail by Wallace and Rowswell.

Appendix II: The Interlock Problem

In this paper it has been assumed that all memory cycles take the same amount of time. However, it is quite common for memories to have pause cycles in addition to read and write cycles. When a pause cycle is initiated by a processor, a word is read from memory, but the restore portion of the cycle is suspended until the processor puts a resume signal (accompanied by new data) on the input bus to memory.

Consequently, a pause-type cycle keeps the memory busy longer than a simple read or write by an amount of time equal to the time elapsed between the completion of the read portion of the cycle and the arrival of the resume signal at the memory. Since two-way bus-propagation delays plus some processor response-time are included in that elapsed time, pause cycles may be several times as long as a normal read or write.

Staudhammer, Combs, and Wilkinson (9) have shown that the expected waiting-time and mean-time-to-transfer-timing-error for other devices in a strict-priority system are significantly sensitive to the mean length of the time-per-instruction that the memory is kept busy by the control processor. (And certainly any pause cycle significantly longer than a normal memory cycle would nullify

minimum channel-latency calculations.)

In systems with only one control processor (which would of course be an intrinsic point of permanent articulation) it is possible to simply force all pause cycles to be separated into a normal-read followed by a normal-write, forsaking only the increase in processor-speed that the pause capability provides for such operations as adding one to the contents of some memory location.

However, systems which contains more than one control processor use the pause cycle as an essential interlock which makes possible the unambiguous examination and modification of control flags in memory by any one of the processors (12). This necessary function of the pause cycle must be replaced in any scheme which forbids pause cycles. The most obvious solution is to incorporate a "read-interlock" cycle into the design of the memory control (in place of the pause cycle).

A "read-interlock" cycle fetches the contents of the addressed word during the read portion of the memory cycle and immediately restores some special pattern during the write portion of the memory cycle. Of course, the control processors must be so designed that reception of the special pattern, when trying to fetch a flag word, forces some appropriate response (such as re-execution of the flag-fetch intruction). Naturally, if the special pattern is not received,

the processor can safely examine and modify the flag word and subsequently restore the modified flag word to memory with a normal-write cycle, since any other processor examining the flag word in the interim will receive the special pattern.

There is essentially no difference between a pause cycle and a read-interlock cycle in terms of hardware cost, but a pause-cycle scheme can have a deleterious effect upon the operation of the priority structure of the system.