(4)

SRM/16

THE UNIVERSITY OF NEWCASTLE UPON TYNE

COMPUTING LABORATORY

A Switch for Connecting Computer Components

C.G. Bell
H.C. Lauer
B. Randell

3rd June, 1971

In a PDP-11 system as currently defined by DEC [1,2], there is exactly one central processor which communicates with many peripheral devices and memory units over a single bus, called the Unibus. The system architecture, however, provides no means for connecting several processors together into one system. In this memo, we discuss the possible design of a "switch" to accomplish this connection.

The design is motivated by several considerations. On one hand, we want a switch which allows several devices to have access to shared devices on a word-by-word basis. In this sense, the switch is a dynamic one, and the term "multiplexor" might be more characteristic of its function. On the other hand, we require a switch which allows the static reconfiguration of computer components into one or several computer systems. In this sense, the term "configurator" might be more appropriate.

For both aesthetic and practical reasons, we require that the switch be as general as possible without sacrificing performance, that it be both reliable and failure-tolerant, that it be easily expandable to accommodate more components, and that it demand few, if any, hardware modifications to standard PDP-11 components. It became apparent, when the authors were investigating various possible designs, that a switch which preserves the logical structure of the Unibus would be both elegant and practical.

In the following material, we will review some of the considerations which led to our design. Then we will discuss the design itself and comment on its implications.

Functional Requirements

The switch design was originally motivated by the needs of two research projects, but its applications are manifold. In the proposed project on Highly Reliable Computer Systems at the University of Newcastle, multiple processor systems, networks of separate but cooperating computers, and systems which share only a few components must be investigated. We prefer not to let the research be prejudiced by a given kind of hardware configuration. Furthermore, we must have the ability to remove a faulty component and to service it "off-line" with whatever other components are needed to service it, whilst the remainder of the system operates normally, protected from the repairing and testing. The Network Project at Carnegie-Mellon University requires the ability to allocate a subset of the components to a user for testing parts of operating systems on "bare hardware", isolated and protected from the other components. In both projects, the switch must permit systems with more than one processor. It must also provide the ability to test new, undebugged hardware without interfering with other work.

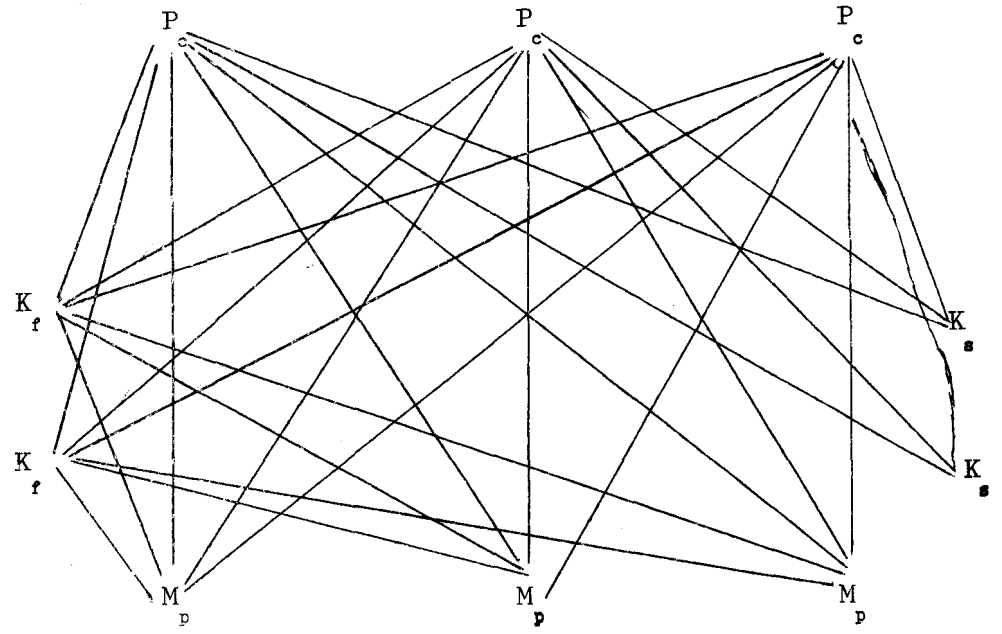
The possible applications of the switch beyond these two research projects are too numerous to mention. As a static switch, its principal benefit is in systems where reconfiguration and/or expansion is done frequently. In its dynamic function as a multiplexor, it will enable the single-processor PDP-11 system to be expanded into a multiple-processor system.

Because the static function of the switch is to partition the set of components into separate, possibly untested, systems it must not be under the control of any of the switched devices. Rather, it should be regarded as a "manual" switch with relatively long periods (i.e., ½-hour or more) between reconfigurations. The problem of connecting and disconnecting components under program control in a multiple processor system is an interesting one, but it can and should be investigated separately. The "manual" character of our switch provides the necessary protection when the mechanisms of any component (except, possibly, the switch itself) fail due to hardware or program faults.

In addition to connecting components into systems, we will also demand that our switch facilitate the renaming of these objects. For example, if all of the core memory were divided between two separate systems, each system might want its address space to begin with location zero and increase linearly. The switch must map the logical memory names from the two systems into the absolute memory addresses recognised by the hardware components. As with the setting of the switch itself, this mapping must be static and not under programmed control of any component in order to protect the hardware configuration from faults. Thus this mapping is quite distinct from the program-controlled mapping functions, such as provided by paging and segmentation facilities, that might be used in the system or systems that are configured with the switch. (The design of such program-controlled mapping functions, with particular reference to the problems of tolerating hardware faults and program bugs, is one of the topics that will be tackled in the projected research project on highly reliable systems.)

Structure of the Switch

Given that a collection of processors, memories and peripheral control units are to be connected together, any of several accepted methods might be used. Figure 1 depicts a system in which each pair of communicating components is connected by a separate, physical link, e.g. the Modular One system. The notation used is the PMS-notation of Bell and Newell[3].



P_c - Central Processor

M_p - Primary memory

K_s - Controller for slow I/O device (e.g. card reader)

K_f - Controller for fast device (e.g. disk)

Figure 1

Note that a distinction has been made between "slow" peripheral controllers (K_s) and "fast" ones (K_f). The slow controllers are incapable of transferring information to and from memory directly; they must interrupt a processor to accomplish this function. Thus, there are no links from any K_s to any M_p. The fast devices, such as disks and tapes, can be programmed to transfer data to or from memory directly, hence the links between each K_f and M_p. There must also be links between the K_f and the processors P_c for the purpose of transmitting control information.

If there are p processors, m memories, k_1 fast controllers, and k_2 slow controllers, then the number of cables and connection pairs required is

$$p * (k_1 + k_2 + m) + m*k_1$$

Each is expensive and a source of unreliability.
(Not shown are the connections necessary for communication among the processors themselves. The PDP-11 provides no such facility and it would be impractical to alter the processors to allow direct communication. However, special slow controllers can be built to relay messages and interrupts between processors, thereby providing a convenient communication mechanism within the philosophy of the existing system.)

In order to provide the facilities we require for reconfiguration, the system of Figure 1 would require a toggle in each link to allow it to be "cut" (i.e., to forbid any communication over it) manually. It would also require a rotary switch, or its electronic equivalent, to perform the renaming function. This rotary switch would specify the sets of addresses by which each device on a link would address the other. The dynamic function of the switch is implemented by machinery in each component to resolve conflicts in simultaneous communications over more than one link.

A second method of connecting components is via the distributed crossbar network, as illustrated in Figure 2 and as found on the IBM System/360 Model 67 and other systems. The

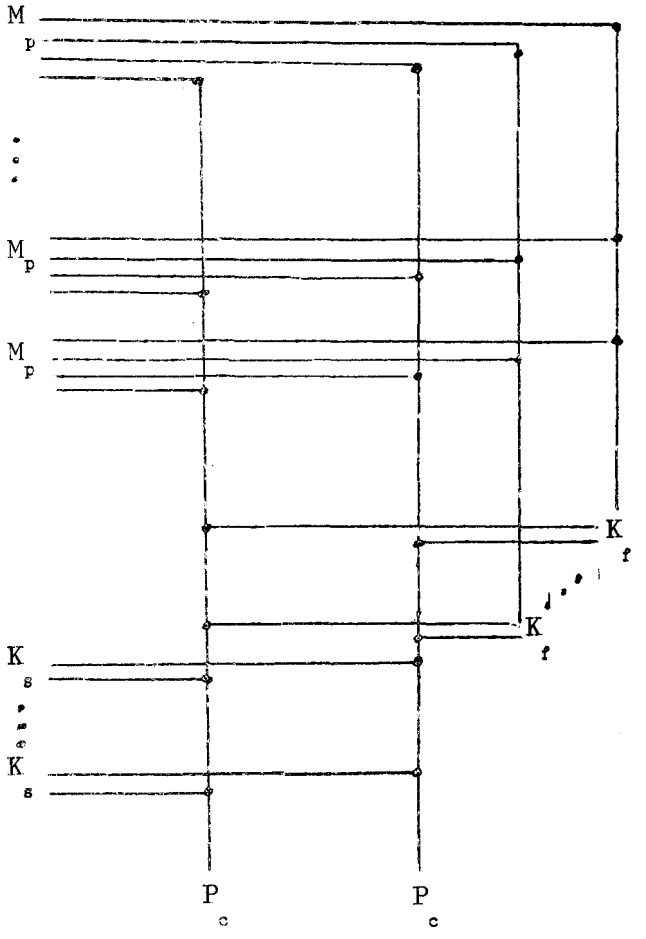


Figure 2

junctions shown in the figure as "T-junctions" actually loop into each device. I.e., the vertical line from a P_c is actually represented by a set of cables, the first from P_c to the first K_s, the second from that K_s to the next, and so on, to connect all K_s, K_f, and M_p. Thus the number of cables and connection pairs is

$$p*(\underline{k}_1 + \underline{k}_2 + \underline{m}) + k_1 * \underline{m}$$

the same as before.

In this configuration, the static function of the switch is implemented by a toggle and rotary switch in each horizontal line, while the dynamic function is implemented as in Figure 1.

In Figures 1 and 2, the switching machinery is distributed over the various components being switched. A third alternative is to centralize the switching function into one module to which all components would be connected. Figure 3 illustrates such a scheme in which the vertical lines are trunk lines and the horizontal lines are the links to the individual components. The trunks are capable of setting up a conversation between any pair of components capable of maintaining one by making the connections at the appropriate crosspoints (marked by "x" in the figure).

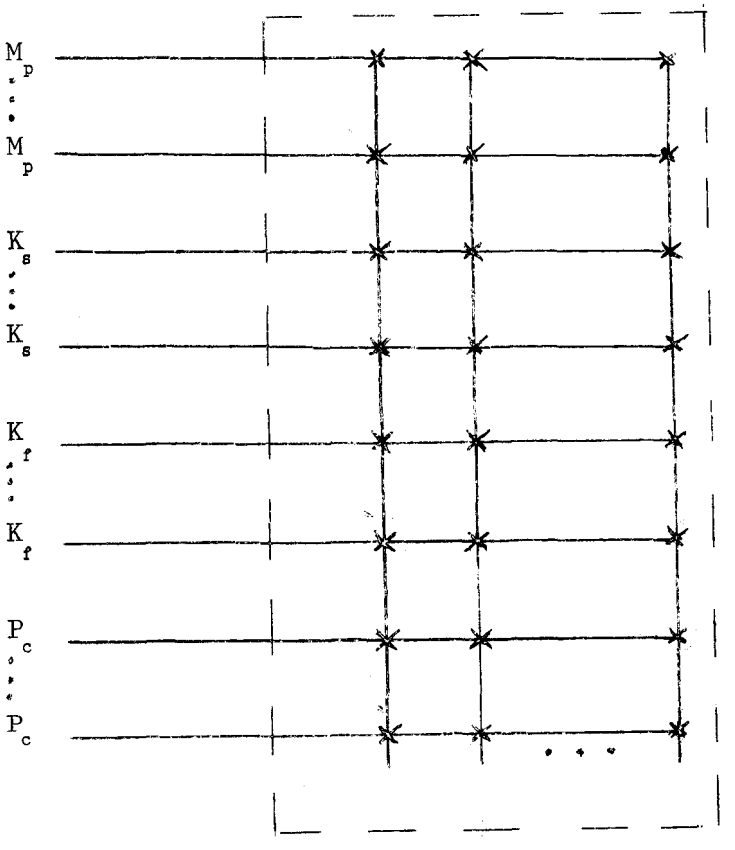


Figure 3

The making and breaking of these connections in response to requests from components fulfills the dynamic function of the switch. The static function is accomplished by permitting or prohibiting such connections. Thus, for example, the set of components can be partitioned into two disjoint systems by allocating trunks to the partitions and only permitting connections to be made between a component and a trunk in the same partition.

The number of cables and connection pairs in this configuration is

$$\underline{m} + \underline{p} + \underline{k}_1 + \underline{k}_2 ,$$

a considerable saving over the previous examples.

Not shown in Figure 3, but very necessary, is the arbitration mechanism which resolves conflicts when there are several requests on each trunk line. The PDP-11 Unibus is essentially a trunk line, and such a mechanism is contained in (but not functionally integral with) the processor. This suggests an alternate version of Figure 3 which is more in keeping with the PDP-11 system structure, as illustrated in Figure 4. The control units labelled K_B are these arbitration mechanisms, and are capable of resolving conflicts on each vertical Unibus. Still necessary is the mechanism for resolving conflicts on the horizontal links. These are labelled K but are physically and functionally part of the switch. ^{c}

As with Figure 3, the number of cables and connections is

$$\underline{m} + \underline{p} + \underline{k}_1 + \underline{k}_2 .$$

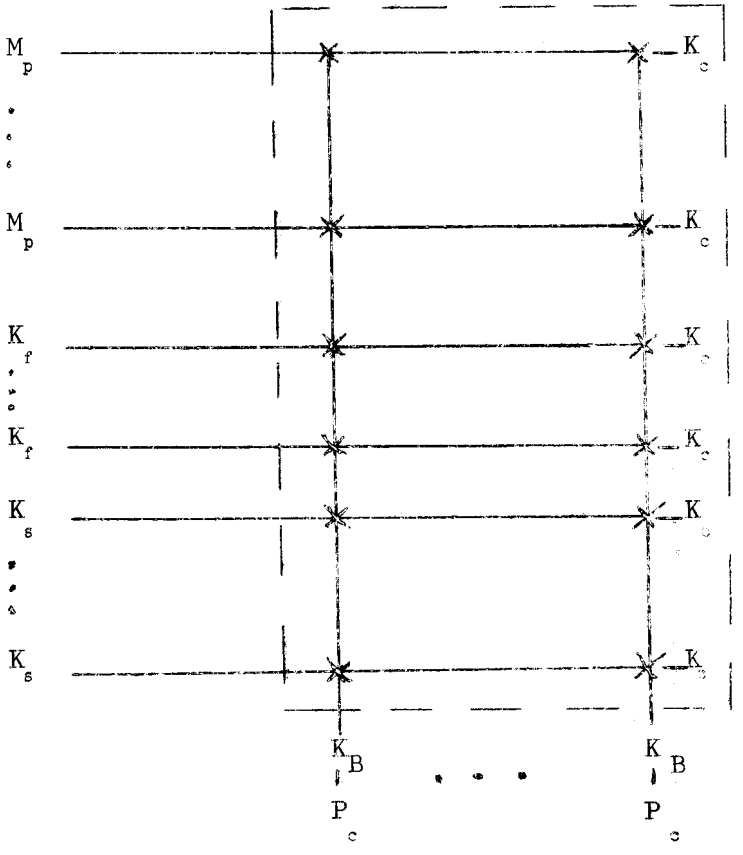


Figure 4.

Characteristics of the Unibus

In order to be able to understand the functions of the various parts of the switch, a short review of the PDP-11 Unibus structure is in order. There are several different kinds of lines on a Unibus to carry different kinds of signals. These lines include:

1) Address Lines
2) Data Lines
3) Control Lines
4) The Interrupt Line
5) Request Lines
6) Grant Lines

All of these are arranged in a linear fashion with the bus arbitration control unit at one end and an electrical termination at the other end. A central processor is connected to the Unibus through the arbitration unit, but all other devices (i.e., memories, controllers, etc.) are connected in a uniform way. The appropriate lines from each device are connected in parallel across all of the Unibus Lines except the grant lines. The grant lines loop into each device and signals are transmitted in a pass-the-pulse manner. Thus, a Unibus has the structure of Figure 5.

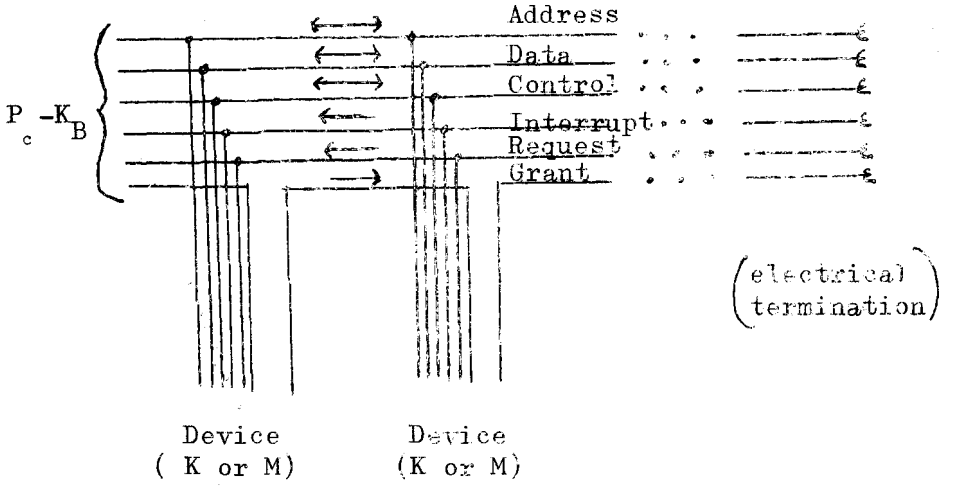


Figure 5.

A component connected to the Unibus can be either active, passive, or both. An active device is one which can request control of the bus, place an address on the address lines (thereby broadcasting it to all devices), and either transmit or receive data or exercise a control function. A passive device is one which cannot request control of the bus; it can only recognize addresses and either accept or return data. In the PDP-11 system, a central processor is a strictly active device, while a core memory is strictly passive. The controllers for peripheral devices (i.e. K and K_s) are both active and passive: they are active in the sense that once they are operating they can move data to and from core directly or transmit interrupts on the interrupt line; they are passive in the sense that their control registers are treated just like memory cells.

The arrows in Figure 5 indicate the possible directions of information flow on the Unibus. Since active and passive devices may be connected in any order, addresses must be transmitted in both directions. The data lines are naturally two-way since any active device can either read or write. Of the various control lines, some are bidirectional, some are unidirectional from the bus arbitration controller to devices, and some are unidirectional from devices to the bus arbitration controller. The interrupt line is unidirectional from the devices; i.e., only the processor can recognize an interrupt. An active device which requires the use of the bus must transmit its request to the bus controller on one of the unidirectional request lines. When the controller grants the request, it replies on the corresponding grant line. This signal is passed sequentially from device to device until it is intercepted by the requesting device. Thus if two devices have requests on the same request line, the one nearer the processor gains control.

The granting of requests depends, in part, upon the internal priority of the processor. I.e., a request from a device for control of the bus is honoured only if it is a higher priority than that of the priority register in the processor. Because of the nature of the grant lines, devices nearer the processor are of higher priority than those farther away. I.e., they are able to intercept the appropriate grant signal as it passes by.

This mechanism insures that at most one of the active devices is in control of the Unibus at any one time, and that conflicting requests are resolved.

The Horizontal Links

We have seen that the Unibus structure provides the necessary conflict resolution machinery on the vertical lines of Figure 4 (provided, of course, that the devices retain their Unibus compatability). There are two other kinds of conflicts which must still be resolved: 1) the case when requests on two vertical Unibuses are made to the same device; and 2) the case when a device requesting control of a Unibus has several to choose from. These must be handled by the parts of Figure 4 labelled K_c and which, for want of a better name, we call the basic modules of the switch.

In order to specify the basic modules, we must first consider the structure of the horizontal links. Unless we are prepared to modify each device (M_p, K_s, or K_f) which we connect to the switch, we had better guarantee that the electrical and logical characteristics of these links are identical to those of the Unibus, at least so far as the individual devices are concerned. One way to do this (indeed, the traditional way in hardware design) is to tailor each horizontal link to the device it serves. The alternative, more in keeping with the PDP-11 design philosophy, is to guarantee that each horizontal link always looks like a Unibus.

This has many advantages. For example, all horizontal links are identical and thus independent of the type of devices connected. Furthermore, all standard PDP-11 devices can be connected without modification. More important, the connections of a horizontal link to the various vertical links can have a form very much like the connection of a Unibus to various devices. Thus, the arbitration machinery for resolving conflicts on a horizontal link can have much the same form as that of K_B, the Unibus arbitration control. (Unfortunately, it does not yet appear that we can use K_B directly).

Another advantage of making each horizontal link look like a Unibus is that several devices can be connected to one link, as in Figure 6.

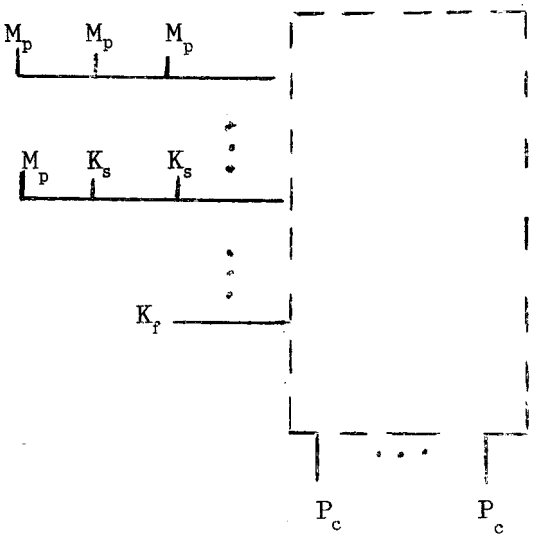


Figure 6.

This is exactly the way multiple devices are connected in the existing PDP-11 system. It makes possible a substantial reduction in the number of horizontal links, and therefore in the number of basic modules in the switch.

(Note that in configurations such as those of Figures 1 and 2, special controllers, called I/O controllers or multiplexors, are introduced to permit the connection of several devices to one node of a bus. Figure 7 depicts a conventional way of doing this in the system of Figure 2.)

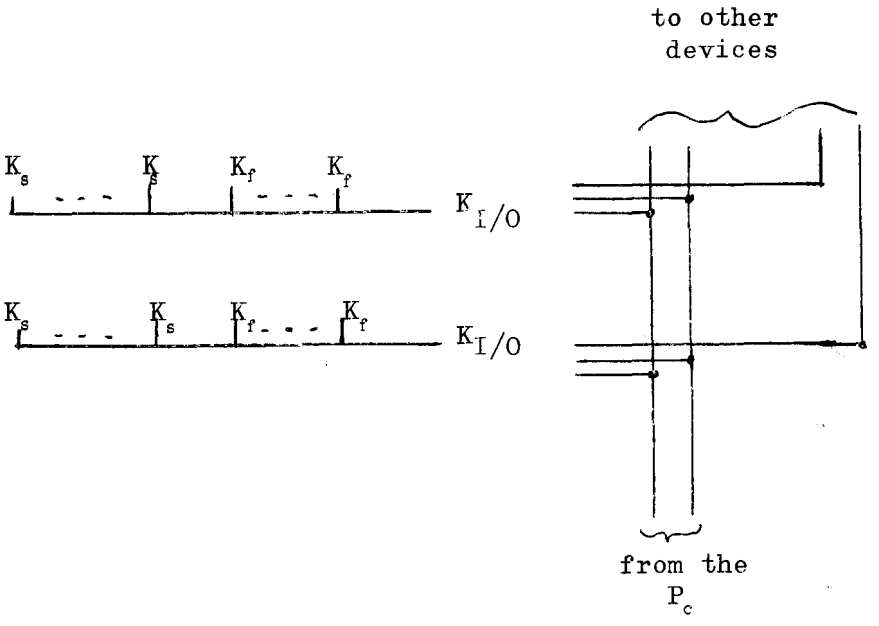


Figure 7.

An observation to be made is that the essential difference between the horizontal and vertical links is that the vertical links terminate in the bus arbitration mechanism (i.e., K_B, or the left end of Figure 5) while the horizontal links terminate with an electrical termination (the right end of Figure 5). Thus while each type of link is a Unibus, they represent "opposite ends" of the Unibus.

We can now summarize the structure of the switch as a device for connecting Unibuses while maintaining their logical and electrical characteristics. The dynamic function of the switch, i.e., as a multiplexer of communications among system components, is served by the Unibus architecture itself. The static function, that of permitting or prohibiting connections and of renaming components, is served by toggles at each of the crosspoints. The resulting switch combines the full flexibility of the Unibus itself with the requirements of multiple processor systems, shared devices, and arbitrary reconfigurability.

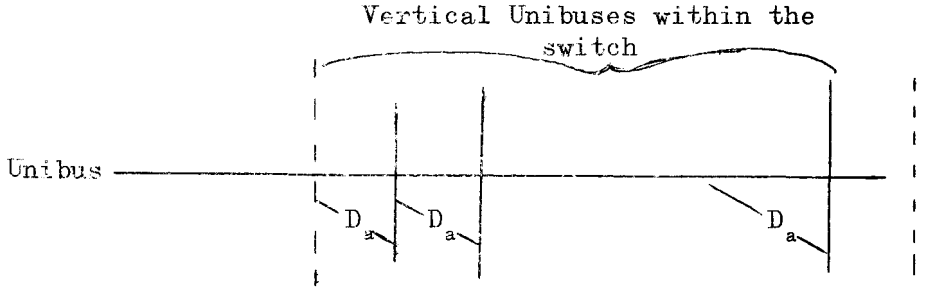
Addressing and the Static Function

When any device has control of a Unibus and it wants to communicate with another, it broadcasts the address of the second on the address lines. All devices inspect that address, but only its owner replies, thereby establishing the communication link. This is complicated by our switch structure. For example, addresses broadcast on a vertical link must be recognized by the appropriate crosspoint and be rebroadcast on the proper horizontal link. At the same time, that crosspoint must establish a connection so that the two buses look like a single continuous Unibus. Consequently, there is a heavy responsibility within each basic module for recognizing and rebroadcasting addresses.

How this is implemented often imposes serious constraints on the flexibility of the system and the type of development which can be carried out on it. We require a scheme which is both general and rational and which provides maximum flexibility with device name assignments. Giving all components fixed, unique, addresses in a uniform name space is a general scheme, but it poses many problems similar to those of absolute addresses in software. I.e., every time we want to change something, it is necessary to track down all of the instances of its name. Since these can be buried in both programs and data, it is an impossible task.

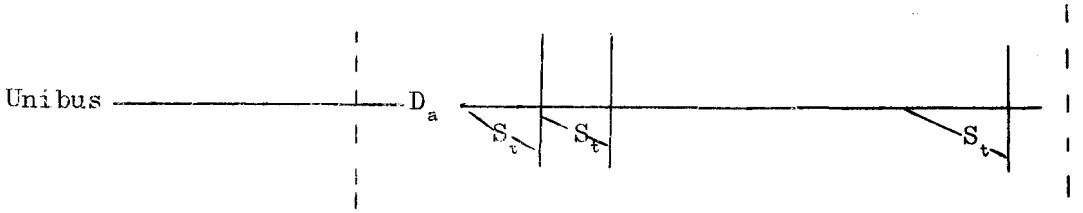
An alternative scheme is to provide an address translation mechanism as part of each crosspoint mechanism or basic module. Then each horizontal link would have the same small, but fixed, set of names assigned to it. Whenever an address is broadcast on a vertical link, the basic module rebroadcasts a translated version on the horizontal link. The actual mapping function is specified in part of the static or manual setting of the switch. This has several advantages. First, identical devices can be connected to different horizontal links and can be replaced by "off-the-shelf" spares when faulty; no renaming is necessary at replacement time. Second, if the switch is used to partition the set of components into several systems, each can have whatever naming conventions it chooses. In particular, two such systems can be made completely identical, in particular with respect to the names of their components.

While it would be convenient to have separate address recognition and mapping facilities at each crosspoint (as in Figure 8a), this would be expensive and impractical. A suitable compromise has a uniform facility across an entire horizontal link within the switch. Of course, it is still necessary to have a toggle at each crosspoint in order to permit or forbid connections. Figure 8b illustrates the basic module of a horizontal link in which the address mapping (symbol D_a, meaning data operation in the PMS notation) is implemented.



Note: Data cperation "D_a" is the address mapping function.

(a)



Note: D_a is as above, S_t is the toggle which permits or prohibits connections.

(b)

Figure 8.

With this arrangement, the devices connected to one horizontal Unibus can be connected or disconnected from any vertical Unibus as a group, and they can be renamed within the system as a group. The actual range of the address mechanism has not yet been resolved. Clearly, it must be capable of recognizing the addresses of one memory module and translating them into hardware addresses, presumably starting at zero. But if peripheral devices could also be connected to the same Unibus, it might be desirable to recognize and map a second range of addresses, as well.

Dynamic behaviour of the Switch

The basic module in the switch is the set of cross-point connections and the address mapping for one horizontal Unibus, i.e., Figure 8b. Expandability is achieved in the vertical direction by duplicating these modules. As we will see below, expandability in the horizontal direction (i.e., in the number of processors) will be more difficult because of implementation reasons.

The basic module must perform different functions on the different lines of the Unibus in response to the signals on those lines. These functions will be grouped into three categories:

1) to connect a vertical Unibus to a horizontal Unibus when an appropriate address appears on the vertical lines and to resolve conflicts among the latter;

2) to connect a request line (see Figure 5) from a device on the horizontal bus to the corresponding line on an appropriate vertical bus, thereby transmitting the request to a processor - bus controller; and

3) to connect a horizontal bus to a vertical bus in response to the granting of a request.

Our basic rule is that whenever a connection is made, the horizontal and vertical link together present the appearance of one Unibus. Whenever a connection of the third type is made, the horizontal and vertical links present the appearance of one vertical Unibus, capable of being connected to another horizontal link. If there are p processors, hence p vertical buses, and $m + k$ horizontal buses, then the number of communication paths which can be established simultaneously is

$$\min (p, m + k).$$

Figure 9 illustrates three simultaneous conversations, one between a processor and memory, the second between a processor and a slow controller, and the third between a fast controller and a memory. The dashed lines represent the apparent Unibuses

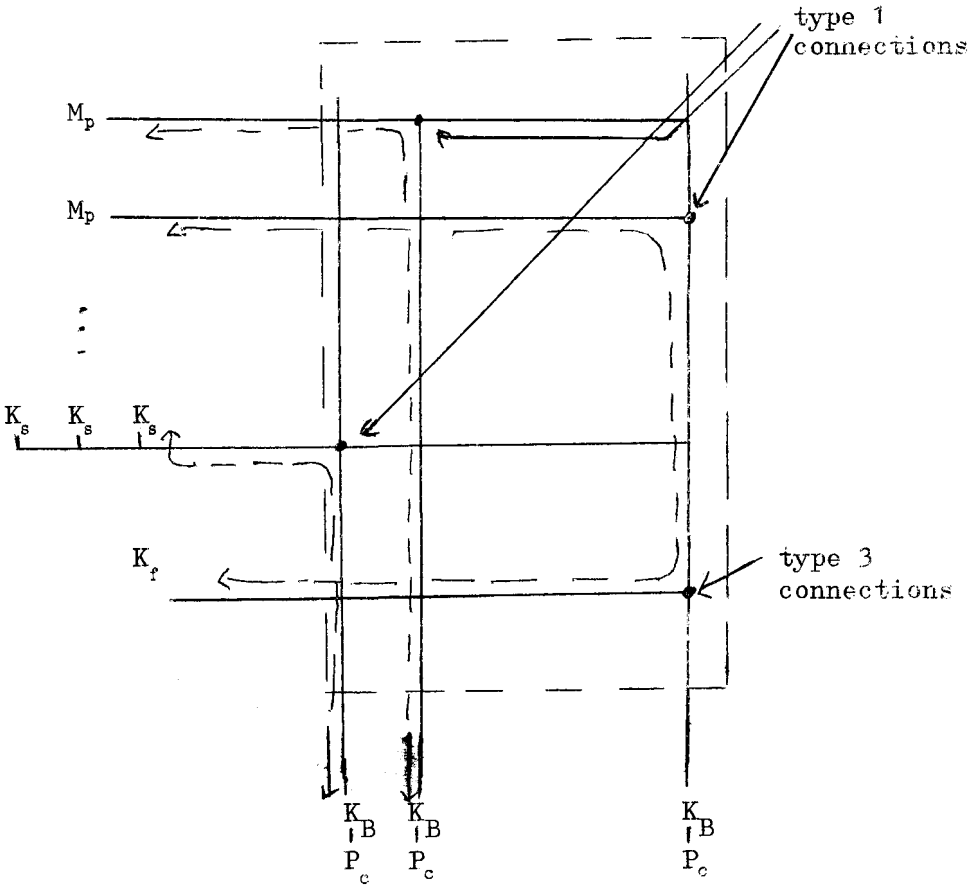


Figure 9.

created by the type 1 and type 3 connections. Note that the rightmost processor is temporarily unable to use its Unibus while the $K_f - M_p$ conversation is taking place. This is the same circumstance as occurs in single-processor PDP-11 systems.

Connections of type 1) are the simplest. For each vertical bus which is presenting an address and appropriate control signals, the basic module selects one according to some priority or round-robin rule, ignoring all lines for which the crosspoint toggles forbid a connection. It then connects the data lines for two-way information transfer and relays the appropriate control signals to and from the horizontal bus. The connection is broken after the data transfer is completed.

Connections of type 2) are more complicated. Any request for bus control (i.e. either a non-processor request or a priority interrupt request) from any device on a horizontal Unibus must be routed to the bus arbitration controller on one of the vertical lines. Clearly, if a certain crosspoint connection is not permitted (by virtue of the toggle setting), the request cannot be routed that way. However, it may be desirable that the request be restricted to one of a subset of processors for which the connection is enabled. For this reason, we need another set of toggles

at the crosspoints to permit or forbid the routing of bus requests through them.

Typically, there will be more than one vertical bus to which a bus request can be connected. The basic module must then choose one of these, presumably based on the priority of the various processors. Thus, it must have access to the three priority bits of each processor state word. These are, apparently, not difficult to get, but they must be routed to the switch on separate wires since there is no room for them on the Unibus. When a choice is made, that request line is connected to the request line of the vertical Unibus. Alternatively, an "address" mapping structure for translating interrupts from the horizontal to vertical Unibus can be implemented. This would be conceptually similar to (but much simpler than) the address recognition facilities of Figure 8b.

Connections of type 3) now become relatively straight-forward. When a bus controller grants a request, it sends out a pulse on a bus grant line (see Figure 5). This is relayed from device to device, or basic module to basic module, until it reaches the requestor. Thus a basic module must be prepared to accept and divert a bus grant signal whenever its horizontal Unibus has the corresponding request line raised. When it does so, it connects the address, data, interrupt, and control lines of the horizontal bus to those of the vertical bus.

An important constraint is placed on connecting to a horizontal Unibus those active devices which can initiate their own data transfers (i.e., the K_f). Since these devices broadcast addresses, they will be inspected by all other devices on that Unibus, plus all of the devices on a vertical bus to which a type 3 connection has been made (including the other crosspoints on that vertical bus). This can lead to confusion in short order if a device on the horizontal bus happened to have the same hardware address as specified by D_a in some other basic module. In fact, any attempt by K_f to communicate with a device on the same horizontal bus will be troublesome because the address translation mechanism (i.e., the D_a of Figure 8b) will not be invoked. This problem can be solved by always assigning fast controllers to different horizontal Unibuses from any devices with which it corresponds.

Control of the Switch

The switch we have just described serves both "dynamic" and "static" functions. Its dynamic functions are controlled by the requests and communications on the various Unibuses, and the speeds of these functions are the speeds of the Unibus operations (i.e., $\sim$ 100 nsec). The static functions are embodied in the crosspoint toggles and address maps, and they are under the direct control of an operator. Thus switching rates are of the order of minutes, hours, or days.

There are several interesting methods of providing operator control of the switch. Any of them can be implemented independently from the switch itself if the controls take the form of logic signals to basic switch modules. (I.e., for each crosspoint, logic signals indicate whether or not the "toggle" is closed and whether or not bus requests may pass through that crosspoint). Similarly, other logic signals should indicate which range of addresses is to be accepted by that module. The function of whatever control panel is chosen is to produce those logic signals, preferably only in the right combinations.

A simple control panel would consist of toggle switches for the crosspoints and rotary switches for the address selection. This has the disadvantage that reliability of the switch becomes limited by the reliability of these mechanical components and that there is no protection against improper settings (in the address translation). An apparently more reliable form of switch is the plastic cylinder containing magnets used on the IBM 2314 disk drives. These are less prone to faults because there are no moving parts, and they can be made relatively idiot-resistant by encoding distinct mappings into separate cylinders so that duplication is impossible.

Both schemes involve manually setting up the configuration whenever necessary. This would probably be practical when the number of links, both horizontal and vertical, is small. But as this number grows, the set-up time, cost, and complexity go up rapidly. The task of manually setting the switch becomes much like that of programming machines with plugboards. Furthermore, it becomes practically impossible to reconfigure one subset of components while a disjoint subset is running, for fear of throwing the wrong switch.

An alternative scheme is to use a small, dedicated computer to control the switch. It would set the various toggle and address translation functions as registers in its own environment. Properly programmed, such a processor would insure that all of the settings were consistent and would provide a convenient way of reconfiguring the components attached to the switch.

This use of a computer is closely related to another use suggested for the PDP-11, namely that of "driving" the operator's console. Since the leads to lights and switches on a PDP-11 control panel terminate at a plug on the back of the machine, they can be connected to the dedicated machine. Then it becomes possible to control several CPU's at once and to cause complicated operator procedures to be executed with a single command. The potential in this form of control for both the switch and the processors is enormous.

Comments

In order to keep costs, performance, and reliability within reasonable bounds, it is necessary, in part, to minimize the number of connections and drivers between Unibus or twisted pair signals and logic signals within a board. Since each Unibus has 56 signals in it, the cost of these connections could go up very rapidly. Thus, although we have conceptually drawn our horizontal and vertical links as Unibuses, we will implement several crosspoints within one circuit board. However it is done, failures of any part of the switch must be made to look like the failure of a horizontal or vertical link. Then that link can be "removed" by reconfiguring the system, and repairs can be made at a later time. Similarly, it should be possible to connect or disconnect a Unibus without turning power off in the whole switch.

It would be desirable to make the switch expandable in both the horizontal and vertical directions. This is relatively easy in the vertical direction if the vertical Unibuses come out of the top of the switch in the form of a standard PDP-11 Unibus connection. Then another switch could be plugged in. Expansion in the horizontal direction is a more difficult problem, and is one we have not solved. It is complicated by the observation that looking into the switch from a horizontal link, we see a Unibus arbitration mechanism. If we expanded in the horizontal direction, the intermediate switches would have to be different from the end switch, as only one could present that appearance.

It is probably possible for deadlocks to occur in the switch as the design stands now. These would happen when communications would travel up one bus to a device while it is interrupting the processor in another bus. Certainly, the attempts to gain expandability in the horizontal direction and to connect switches together would compound these problems. We have not studied them seriously, but we believe that they can be readily solved.

Note that there is nothing about this design that either demands or prohibits any PDP-11 processor from having private devices, not under the control of the switch. Furthermore, there is nothing to demand that a vertical Unibus terminate at a processor; all that is necessary is a bus arbitor, K_B. Thus a configuration such as Figure 10 is possible.

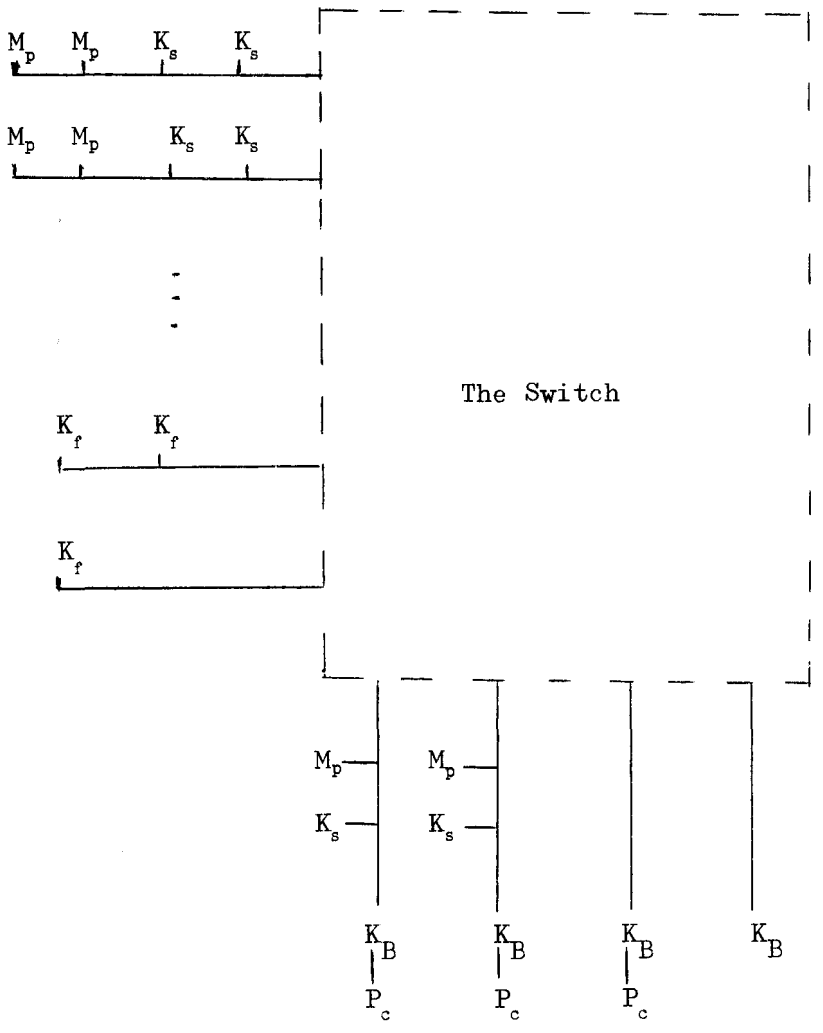


Figure 10

In this figure, two processors each have a private memory and a private slow device (say a teletype). One processor has neither. The fourth vertical Unibus has no processor (i.e., a "null processor") and is included to provide extra bandwidth to support the communication between fast devices and memory. In this way, a disk or drum does not need to lock out a processor from its own Unibus, as it does in Figure 9.

We made but brief mention of the problem of communication and interrupts between processors. This can be done easily, efficiently, and cheaply by building special devices (K_I) to connect to two Unibuses at the same time. Then one processor interrupts another by writing to the register of K_I over its Unibus. It then raises an interrupt on the other Unibus, which is answered by the second processor. This is illustrated in Figure 11.

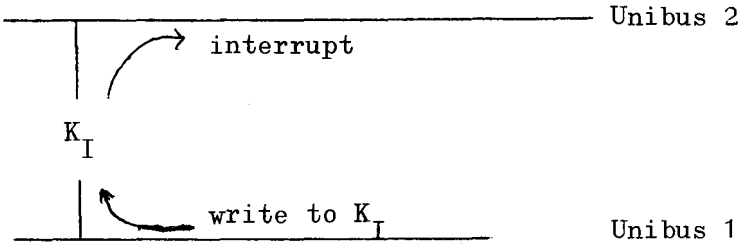


Figure 11

Calendar clocks and interval timers can also be connected to Unibuses and provide interrupts in the same way.

Acknowledgements

This memo is the result of discussions among the authors and with J. Eve, C.R. Snow, A.J. Mascall, J. Givens, K. Herron and others during Professor Bell's recent visit to Newcastle.

References

1. Digital Equipment Corporation, PDP-11 Handbook, Second Edition 1969.

2. G.G. Bell, et al, "A new architecture for mini-computers- The DEC PDP-11", Proceedings of the Spring Joint Computer Conference, 1970, pp 659-675.

3. G.G. Bell and A. Newell, Computer Structures: Readings and Examples, McGraw-Hill, 1971.