BUS BASICS

R.W. Dobinson
CERN, Geneva, Switzerland

The aims of this course

1. To introduce the general concepts of buses used to interconnect processors, peripherals, and instrumentation.
2. To classify, albeit crudely, different types of bus systems.
3. To give some examples of past, present, and future buses as applied to the field of high-energy physics, including some details of software.
4. To provide references for further study.

The non-aims

1. To discuss, in more than a passing fashion, electrical properties of bus systems, mechanics, and cooling — not that these are unimportant, but rather because they would seem to be out of place at a computing school.
2. To talk about serial buses, in particular local area networks. There is another lecture course on this topic and I will therefore limit my comments to trying to locate such buses within the over-all "bus family".
3. To explain the precise details of any bus. Each bus has upwards of 60 signal lines, and there are at least 10 popular buses each with their own, often complicated, rules for use — the problems are clear!

1. INTRODUCTION: WHAT IS A BUS? (SOME EXAMPLES)

Historically, buses arose from the need to interconnect the component parts of electronic computers. Data transfers were required between CPU, memory, and peripherals, and whilst in principle all required paths could be implemented with dedicated point-to-point wiring, it rapidly became clear that such an approach was expensive and clumsy compared with linking the components with one or a few shared data paths or buses (Figs. 1a,b). It should be noticed that in replacing the point-to-point wiring of Fig. 1a by the single bus of Fig. 1b, we have traded cost and simplicity for a potential speed penalty arising from a lack of the intrinsic parallelism present in point-to-point wiring. This is brought home by noting that DEC, after introducing a single bus (or UNIBUS) concept in their early PDP-11 machines, were unable to maintain this approach for later high-performance systems, and were driven to using several different buses in machines such as the 11/70 and 11/45 (Fig. 2).

In high-energy physics, buses arose from the need to interface scalers, ADCs, TDCs, chamber electronics, etc. to minicomputers in a cost effective way. It was clearly impossibly expensive to interface each electronic channel separately to a minicomputer, and therefore a way of multiplexing access from the mini to any one of several channels contained in a "module" was evolved (Fig. 3). As an additional advantage, power and cooling could be provided for all channels packaged mechanically in a module. However, this still required the interfacing to be done for every different module, and if one type of mini were replaced by another, then one essentially had to go back to the drawing board and re-interface all the required instruments. A very attractive solution to the problem is shown in Fig. 4. All equipment is interfaced to an instrumentation bus, which is then itself interfaced to the computer I/O bus. This is the CAMAC philosophy: plug-in modules are housed in a standard cooled crate with access not only to a defined data bus for transmission of information but also to common power supplies. Changing the mini now means changing the interface to the instrumentation bus but not re-interfacing every instrument. Another type of instrumentation bus appeared in the mid 1970's. This bus, the Hewlett Packard GPIB, had similar advantages to those of CAMAC in that instruments could be interfaced in a computer-independent way; however, it did not provide mechanical housing, power supplies, or cooling (Fig. 5).

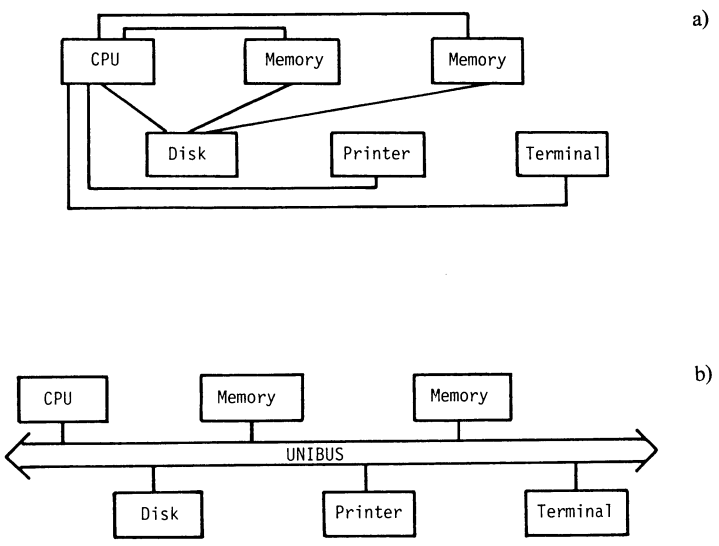
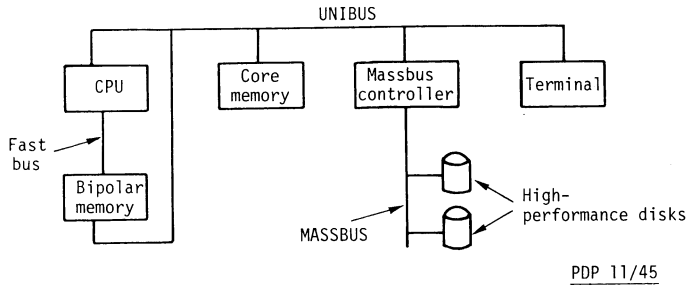


Fig. 1 a) Component parts of typical computer linked via dedicated point-to-point wiring
b) PDP-11/20, components all linked by UNIBUS



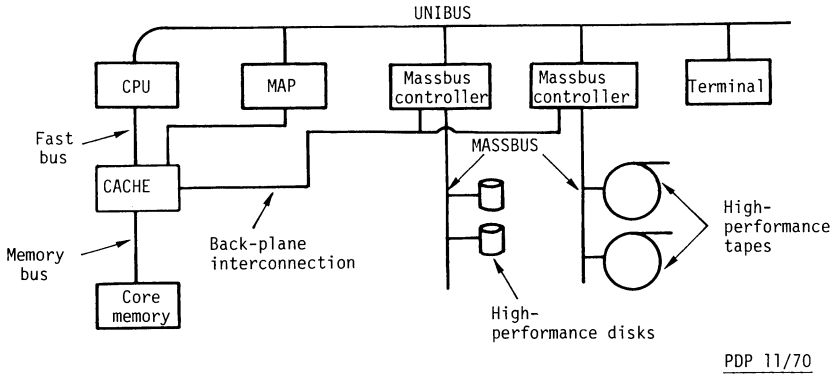


Fig. 2 High-performance DEC minicomputers used more than just a single UNIBUS

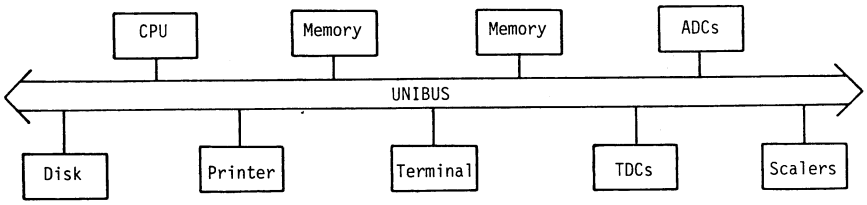


Fig. 3 Interfacing high-energy physics instrumentation to a PDP-11 directly via the UNIBUS

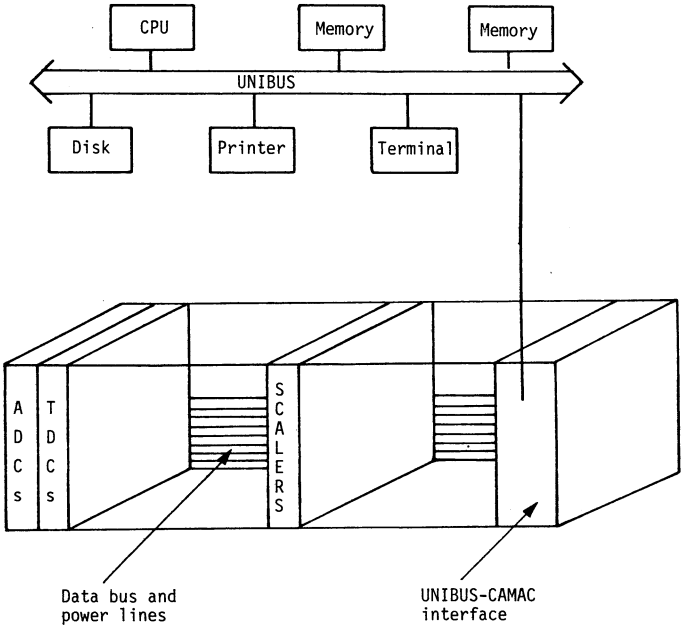


Fig. 4 The CAMAC philosophy

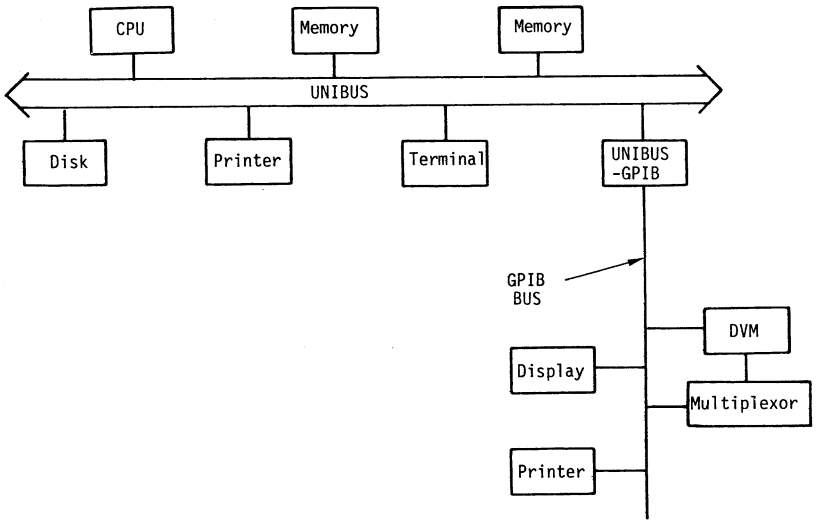


Fig. 5 Use of the Hewlett Packard general-purpose instrumentation bus

It is now over 10 years since CAMAC and the UNIBUS first appeared on the scene. These buses were developed to couple systems over relatively short distances, essentially up to a few tens of metres, although CAMAC has since evolved in order to meet the needs of instrumentation situated hundreds of metres or even kilometres from a controlling computer. Other types of interconnection schemes for computers and peripherals were, however, becoming increasingly widespread in the early 1970's. On the one hand, a vast technology was growing up in response to the need for relatively low-speed remote access to powerful central mainframe computers over distances of 1 to 10^3 km, via PTT telephone lines. On the other hand, communications were increasingly required at medium to high speed between the growing number of computers situated within a typical laboratory, factory, or university campus. Figure 6 shows an example of remote terminal and job entry stations linked to an early 1970's computer centre. This technology evolved into what is commonly known now as wide area networking. Figure 7 illustrates such a network. Figure 8 shows an extremely interesting development made at CERN some 10 years ago, whereby data-acquisition computers situated at different intersection regions of the Intersecting Storage Rings could communicate with the machine control computer. This ring system and its contemporaries were the ancestors of the present-day local area networks, of which Ethernet and Cambridge Ring are two of the most widely known in Europe.

Let me now define a bus. *Buses are shared communication paths, the physical and electrical structures that interconnect processors, peripherals, and electronic instruments.*

These lectures will focus on bus systems used for communications over relatively short distances. However, it should be remembered that most of what are called local area networks also have a shared communications path, and have quite a lot in common with the bus system I will discuss. CAMAC is interesting in that it is a single system that spans the areas of applicability of restricted distance computer buses and local area networks. You may find in Fig. 9 a useful summary of the relative capabilities of different types of interconnection schemes as a function of speed and distance.

2. BUS BASICS

2.1 General concepts

Transfers of information over a bus are activated by *bus masters*. There may be several masters present on a bus and there is therefore a need to resolve *contention* between them via some *arbitration* process. Only one master, the current bus master or *commander,* is allowed to be active on the bus at any time.

A commander can exchange information with one or more *slaves*. A slave is selected to take part in an information exchange by the following sequence of actions. A master places *address* information on the bus; all slaves compare this information with their internally known valid address set and, if a correspondance is found, they become *connected.* Connected slaves are sometimes called *responders.* During the time a master is connected to one or more slaves, *data transfers* can take place between the parties concerned, the object of the exercise being to transfer data from one group of data cells (processor registers, memory locations, registers in an electronic instrument) to another by means of the bus. After the transfer of data has been completed, the master implicitly or explicitly breaks its connection with the responders, which then become *disconnected.* This sequence of operations — from making a connection, transferring data, to breaking the connection — can be called a *transaction* (or message). A commander may relinquish bus mastership after a transaction or continue with other transactions.

The generic term for masters and slaves connected to a bus is *device.* A particular device may act as a master and as a slave, but usually not both at the same time (although this is possible).

Transactions between masters and slaves involve address and data information being transferred over the common bus, but along with this there is also what is called *timing* information, used to synchronize operations taking place on the bus. Stated quite simply, all parties concerned must know when information is valid. Additional *control* information, for example to specify the direction of data flow, may be sent by a master, and *responses* may be returned by slaves.

Almost all bus systems possess mechanisms for generating *interrupts.* An interrupt is a request for attention, made by one device on the bus to another device; the precise form this takes varies widely. Later we shall describe several different ways of implementing such a feature.

Figure 10 illustrates some of the general concepts about buses introduced so far. The set of rules governing the use of the bus by masters and slaves is called *bus protocol.*

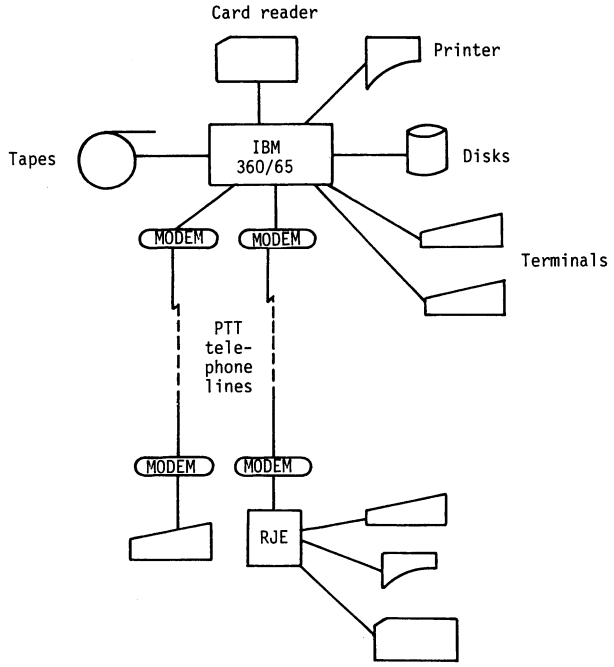


Fig. 6 Computer centre, early 1970s

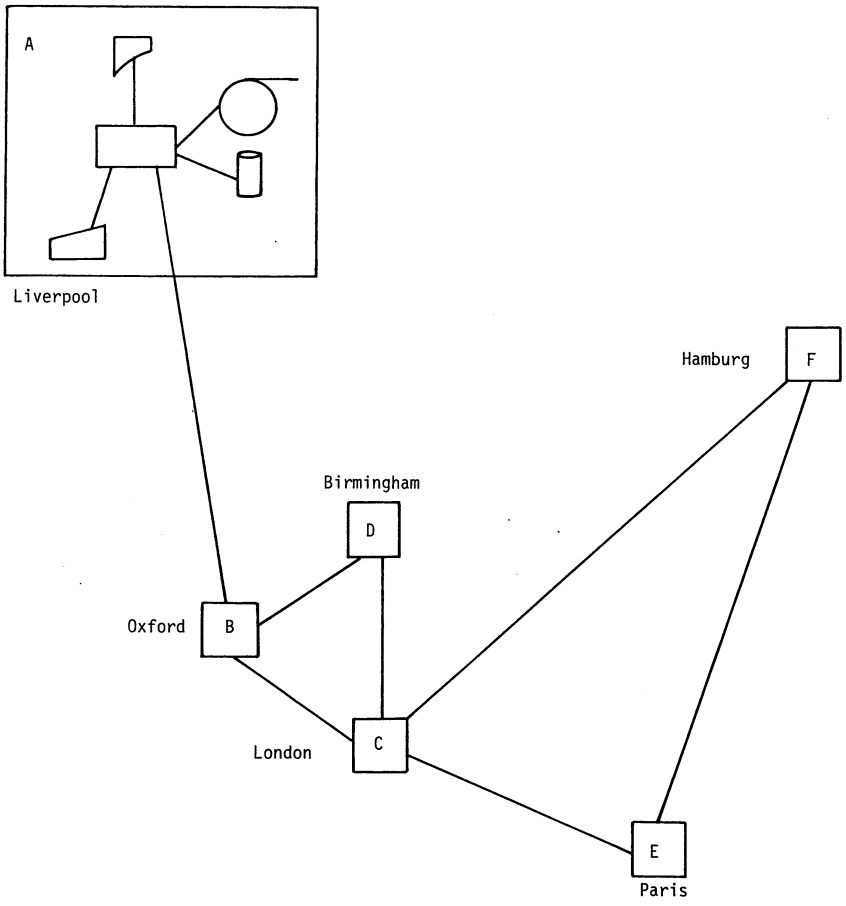


Fig. 7 A wide area network interconnecting several European computer centres

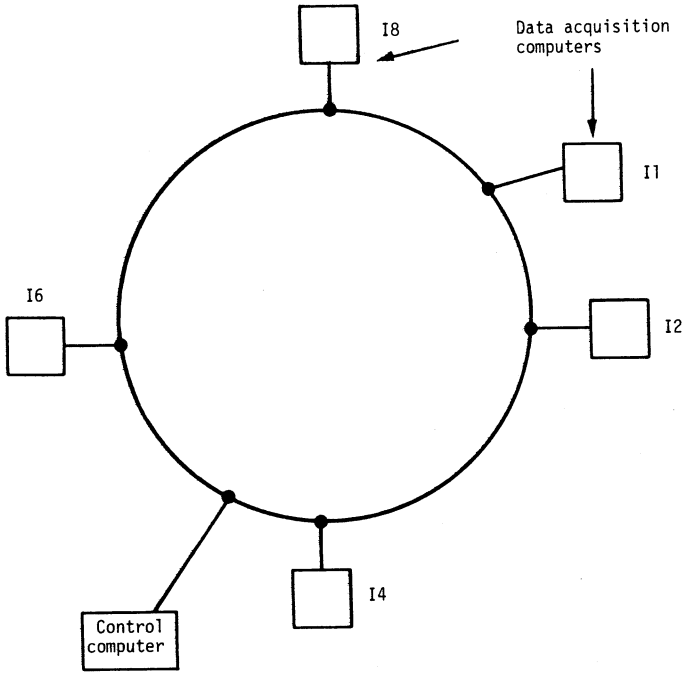


Fig. 8 Experimenters computers at the CERN ISR connected to a machine control computer via a ring network

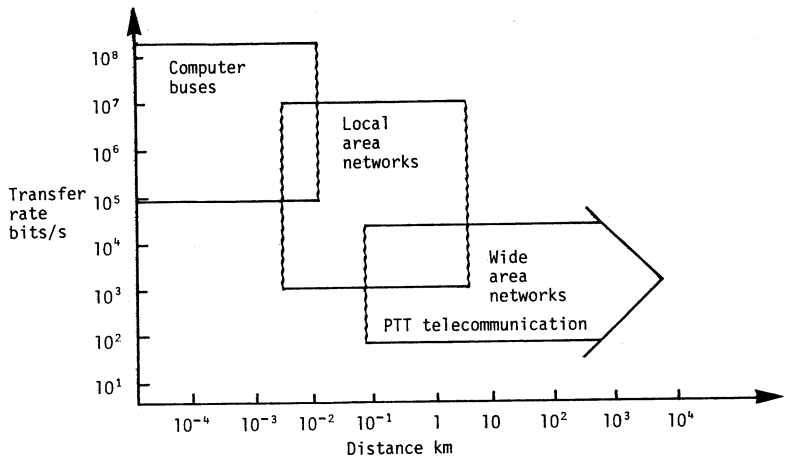


Fig. 9 The spectrum of interconnection technologies

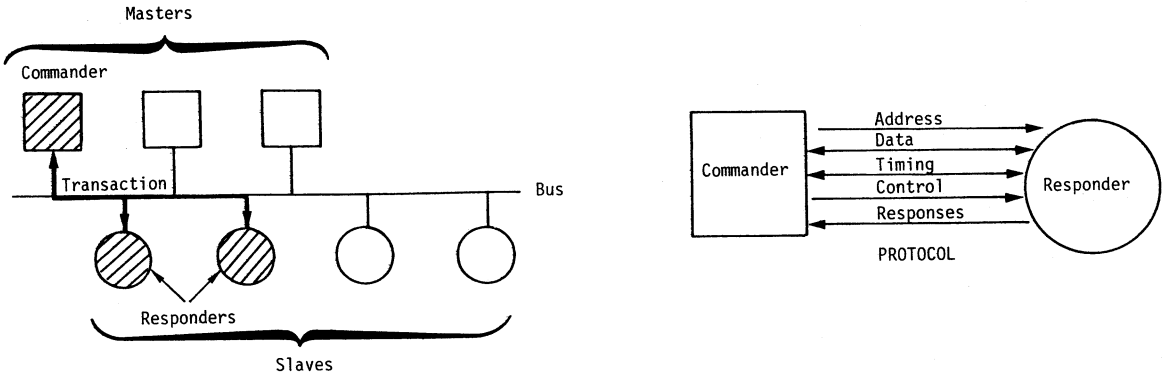


Fig. 10 Bus basics

Until now I have attempted to preserve some generality in my discussion of buses by stressing both their variety and some of their common concepts. I will now mention some of the basic implementation choices that can be made, before going into more detail about the bus concepts described above.

2.2 Serial and parallel buses; one wire or many

We have seen that information flows between master and slave(s) via a transaction. A transaction normally comprises the following:

- address
- data
- control
- timing
- responses.

One of the most fundamental distinguishing features between different buses is the number of information channels — for example wires — present on the bus. This allows buses to be divided into two general types: *parallel* and *serial*. Let us consider two extreme situations:

- in a completely parallel bus, all information flows between master and slave in parallel via separate channels (Fig. 11);
- in a completely serial bus, information is sent bit by bit down a single channel (Fig. 12).

In the latter case, all information is *multiplexed* down a single channel; in the former there is no multiplexing.

Of course, life is nowhere near so cut and dried in practice; one cannot just specify whether there is, or is not, multiplexing. It is more the question of the degree to which multiplexing occurs. A so-called parallel bus may first send address, then data information, over the same set of lines (Fig. 13). "Serial buses" such as the serial CAMAC highway offer the option between what is called bit serial or byte serial operation (Figs. 14a,b). Even what is called bit serial is not one single channel but two, the second being used to transmit timing information.

The arguments over whether to use serial or parallel transmission techniques are in principle easy to understand. Clearly, the more bits that are exchanged in parallel between master and slave, the higher is the over-all transfer rate of information between them. On the other hand, the cost of cable, connectors, and bus transceivers increases almost linearly with the number of separate lines. As the length of the bus increases, more sophisticated and expensive electronics will be needed to take care of noise, cross-talk, and ohmic losses. Cable costs (material plus installation) start to become expensive. In order to reduce the large number of addresses and data lines on a completely parallel system, many buses use multiplexing of addresses and data on the same lines (see Fig. 13). However, at a certain point it will become more cost effective and convenient to accept the overhead involved in serialization and de-serialization, and multiplex everything down a single well-controlled communications channel. A very rough rule of thumb seems to be that any transmission over a distance of more than $\sim 10^2$ m tends to be basically serial in its nature. Almost without exception, local area networks are serial buses; minicomputer and microprocessor buses are always parallel buses. In these lectures we shall concentrate on parallel buses, with a brief detour to describe some of the features of serial CAMAC.

One final point should be made in the serial versus parallel discussion. Transmission taking place within a laboratory or a university campus can — and normally will — be made over transmission media which are provided by and are under the control of that organization. The organization can itself evaluate, for any particular application, whether serial or parallel transmission should be used. However, once transmission is required outside the boundaries of that organization, the transmission medium is provided by a third party (e.g. the PTTs). This medium — whether it is a telephone line, a coaxial cable, or a satellite — will use totally serial transmission.

2.3 Bus arbitration

As described previously, several masters may be competing for the use of a bus. There must therefore be a means for deciding who may use the bus at any time. This process is called arbitration.

Very many different arbitration schemes have been used, and various more or less formal classification systems introduced to describe them. I will not attempt any in-depth coverage, but will restrict myself to answering in a general way, and with some examples, three questions about arbitration:

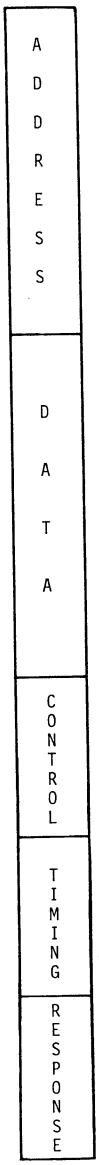


Fig. 11 Parallel bus many information channels

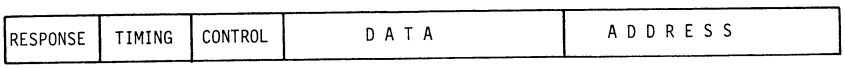


Fig. 12 Serial bus a single information channel

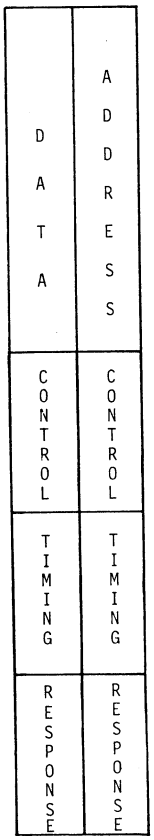


Fig. 13 A bus with multiplexed address and data

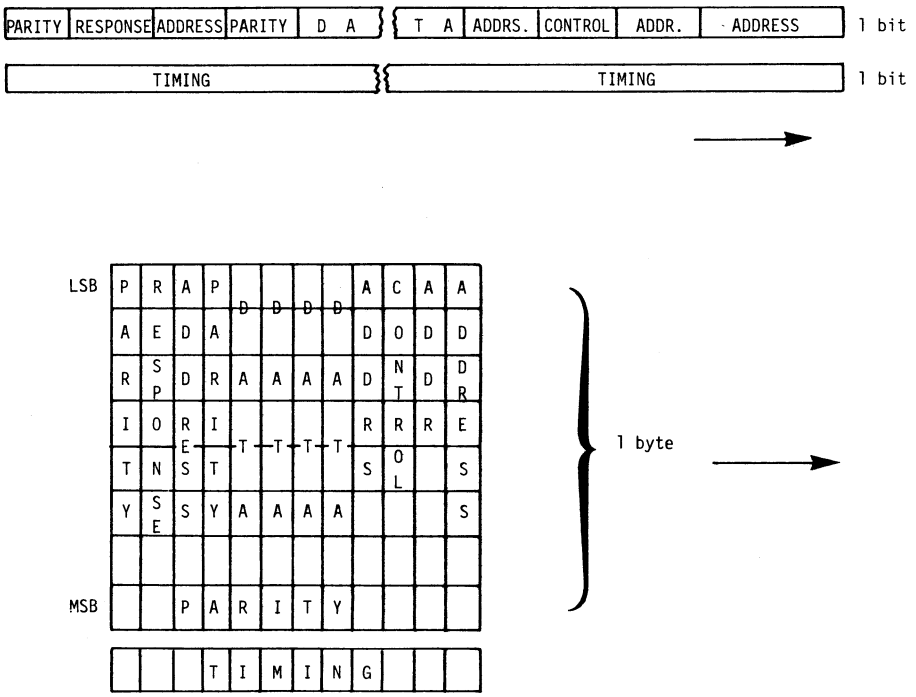


Fig. 14 a) Bit serial CAMAC
b) Byte serial CAMAC

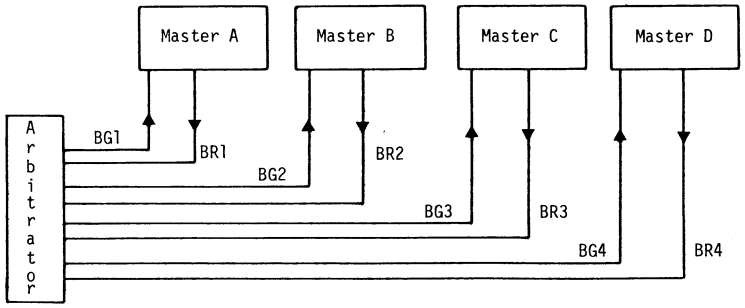


Fig. 15 Centralized arbitration

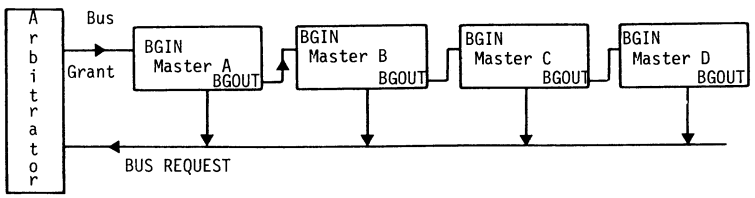


Fig. 16 Daisy-chain arbitration

Where? – Is the location of the arbitration logic centralized or distributed?
How? – What are the rules by which mastership of a bus is allocated?
How long? – What is the time a master has to wait to use the bus?

Centralized arbitration means that a single hardware unit is used to recognize and grant requests to use the bus. Distributed arbitration uses control logic, which is primarily in the competing bus masters in order to allocate the use of the bus. Figure 15 shows an example of a centralized arbitration scheme; a very similar picture appears in the specification for the MULTIBUS, and EUROBUS is another bus with this type of arbitration. Any master requiring access to the bus raises its bus request. The arbitrator returns a bus grant to the master to indicate that it is allowed to use the bus next. This sounds all too easy. However, let us consider what happens if two masters request at the same time. We need rules to decide who goes first. One widely used allocation rule is to use *priority arbitration;* if BR1 and BR4 are both present, then the higher number request takes precedence over the lower one and is honoured first. No lower priority grant will occur as long as there is a higher priority request present. Of course, it is possible for a high-priority master to lock out a low-priority one completely. An alternative would be to implement a *democratic arbitration* scheme whereby a "fairness rule" is introduced to prevent such a monopoly. One example of such a scheme would be a so-called round-robin allocation, whereby bus mastership is allocated in a cyclic fashion to all competing masters. Another fairness rule would be that masters "losing" during a bus allocation procedure would be allocated a higher priority next time round. It should be stressed there is considerable latitude in how the arbitrator allocates bus mastership, and almost any algorithm tailored to a particular need can be implemented.

The centralized arbitration scheme described above has an independent bus request/grant pair for every master. Whilst this allows a rather precise control over the use of the bus, it does however have the disadvantage of increasing the number of lines required on the bus. An alternative, shown in Fig. 16, is for masters to use a common bus request line and to daisy-chain the grant line through all the masters. The scheme works in the following way. If a master wants to use the bus, it asserts a bus request; the arbitrator, on seeing the request, issues a grant. Note that the arbitrator sees the logic OR of all requests; it does not know and cannot even distinguish how many different masters are competing. A master, seeing a "grant in", delays this until it has checked whether it has requested mastership. If it has, it blocks the grant from going any further down the line; if it has not, it passes "grant in" to "grant out" and thus propagates it to the next master on the daisy-chain. This form of arbitration gives highest priority to the device nearest the arbitrator. However, whilst relatively simple and requiring few lines, this method can be somewhat slow, since the bus grant must ripple through all the masters. It is also physically awkward to add, remove, or displace devices, and there is no possibility of implementing flexible allocation algorithms. Nevertheless, this sort of arbitration scheme is used in allocating bus mastership in multicontroller CAMAC crates. Daisy-chain arbitration is a near relation of the token passing schemes used in ring local area networks. Strictly speaking, it is not really a completely centralized arbitration scheme despite the fact that there is a centralized arbitrator. It becomes even less centralized if you do away with the arbitrator altogether and simply connect the "grant in" of the highest priority master to the logically asserted state (Fig. 17). A compromise is very often adopted by combining priority arbitration with daisy-chaining. At least one rather mature bus, the UNIBUS, and one rather recent bus, the VME bus, use such a scheme, which is illustrated in Fig. 18. There are several bus grant and request lines present on the bus; however, each grant line can be daisy-chained through several masters. This scheme is basically centralized arbitration, but with some small degree of distributed decision-making on each bus grant line.

A very interesting distributed arbitration scheme is used in FASTBUS, in the S–100 bus, and also in the proposed P896 standard. Each master is allocated an internal priority vector to be used during the arbitration process. All masters requiring the bus use a common request line. The (simple) arbitrator, upon sensing this request, asserts an arbitration grant. All masters wishing to compete for access assert their vector on a set of common bus lines, which then take up a state in which each line is the logical OR of the logic level being asserted by all contending masters. Masters then perform a bitwise comparison between their internal vector and the composite vector. If a master sees that any bit of its internal vector differs from the corresponding bit of the composite vector it removes all its lower order bits from the bus. After a certain time there is a stable vector on the bus corresponding to that of the winning master. Each master then knows whether it has won or lost; whether or not it is to be the next commander. One simplified implementation of this type of arbitration is shown in Fig. 19. You should verify that, for two masters A and B having priority vectors 100 and 011, the final vector on the bus is 100, and that master A has ISMINE asserted. Use of the

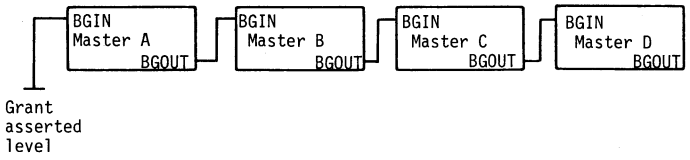


Fig. 17 Daisy-chain arbitration, no arbitrator

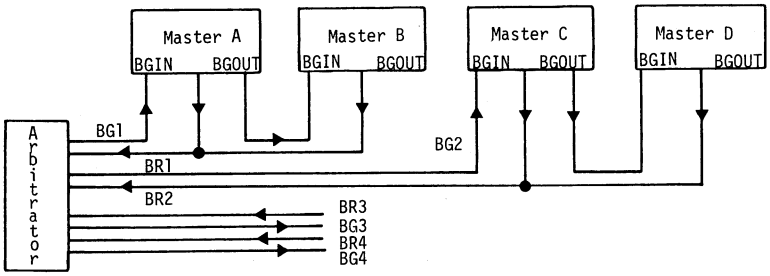
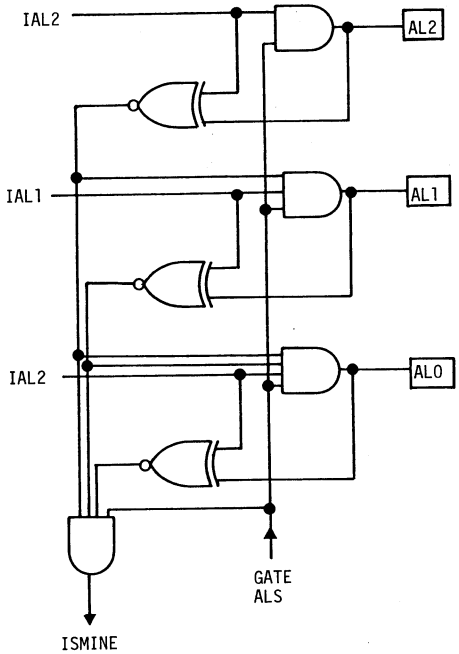


Fig. 18 Priority arbitration plus daisy chain



	Master A	Master B
IAL2	1	0
IAL1	0	1
IAL0	0	1
AL2	1	0
AL1	0	0
AL0	0	0
ISMINE	1	0

Fig. 19 Distributed priority arbitration scheme (simplified version of that used in FASTBUS)

bus is thus allocated according to a self-selecting priority arbitration scheme. However, some form of democracy can be introduced to avoid bus hogging by high-priority devices. For example, FASTBUS provides a mechanism for ensuring that masters that lose an arbitration cycle can have their outstanding requests honoured before any new requests are allowed. A clear advantage of the self-selection mechanism described here is that it is very easy to add, remove, or displace masters in a system. For example, there are none of the problems associated with reconfiguring daisy-chain schemes.

We have seen that in general a distinct set of lines will be used in arbitration mechanisms. This set can be considered as forming an independent *arbitration bus,* which is used to allocate access to the main bus over which transactions occur.

Turning now to the third of our original questions about arbitration: What is the time between a master requesting the use of a bus and the time it starts to perform a transaction over it? This time is called the *latency of access.* Bus latency is a very important and often neglected consideration in multi-master systems. Failure to take it into account can result in errors in data transfers to rotating devices such as disks and tapes, and even to CPU timeouts in certain types of processors.

Bus systems differ in their ability to overlap operations on the arbitration and transaction buses. There is a potentially significant time gain in allocating the next master in parallel with a transaction being carried out by the current master. Where this is possible, the new master can take over control of the bus immediately after the old master has relinquished it (see Fig. 20). When this overlapping is not possible, allocation cannot take place until the current master has given up the bus, and there will be an additional delay, for example whilst a bus grant propagates through a daisy-chain (see Fig. 21).

Another consideration in estimating a total bus latency time is how long a current master holds onto the bus. A pending master may well be allocated, but it still has to wait until the bus is free before it may use it. This waiting time can be minimized by simply requiring a master to keep all transactions short and to relinquish the bus after every transaction. However, bus arbitration may often take some time, and if there are no other requests for the use of the bus, or if such requests have a low priority, it could be better for the current master to retain the bus. A way out of this dilemma is to provide a mechanism to force the current master to give up the bus if someone else wants to use it; the current mastership is pre-empted. There are various ways this can be done. One simple method would be for the current master itself to sense whether any bus request lines were being asserted, and if so to give up the bus. A variation (Fig. 22) would be to give it up if a higher priority master was requiring access. Alternatively, the arbitrator could decide to pre-empt the current master and send a release request on a special bus line, as suggested in the VME bus specification; or a pending master itself could activate a request to release, as can be done in the CAMAC and MULTIBUS systems (Fig. 23).

There are, however, situations where a master must retain mastership: if a processor is performing a Read–Modify–Write instruction via the bus, for example testing and setting a semaphore associated with a shared resource.

2.4 Synchronization of bus cycles; bus timing

We have seen in some detail how mastership of a bus is obtained. It is now time to consider the transactions that can occur between the current master (or commander) and slave(s). A transaction involves a commander connecting to one or more slaves via address information sent over the bus. During the time the connection is present, data can be transferred between master and slave(s), and in some systems, such as FASTBUS, supplementary *secondary address* information can be specified. Every time information is transferred on the bus between a commander and slave(s) we say that a bus cycle takes place. A transaction can be one bus cycle or many; only one cycle may be used to make the connection and transfer data, or address information may occur in one cycle, followed by one or more data transfers over the same lines on subsequent cycles (Fig. 24). However, for each cycle there will always be a need to synchronize the transfer of information. There are two general ways in which bus cycle timing can be carried out: either *synchronously* (periodically) or *asynchronously* (aperiodically).

Synchronous transfers (Fig. 25) require some fixed timing agreement between master and slave. A commander in a synchronous system assumes that a slave can accept or provide information within a certain time. There is no timing handshake from the slave to acknowledge the cycle. The length of the cycle (normally constant) is determined solely by the commander.

Asynchronous transfers involve a timing handshake between master and slave. The master tells the slave that there is valid information on the bus for it to look at; the master will then wait before continuing

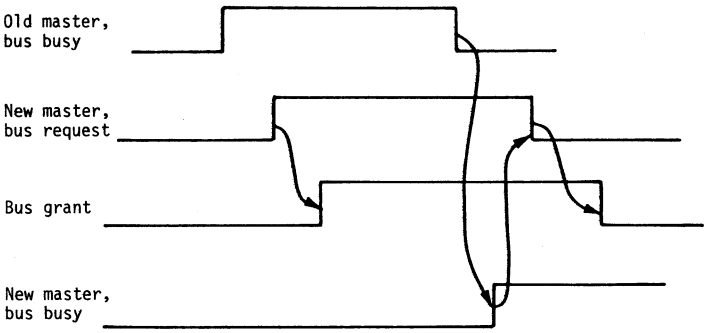


Fig. 20 Overlapping of operations on the arbitration and transaction buses

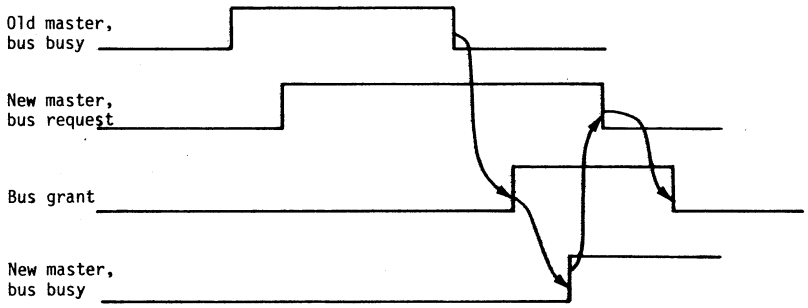


Fig. 21 Arbitration and transaction bus operations not overlapped.

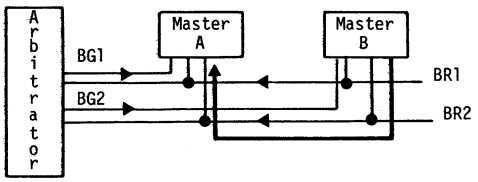


Fig. 22 Master sees higher priority requester and gives up bus. A has become current master at priority 1; B asserts BR2. BR2 priority > BR1 priority, hence A relinquishes bus.

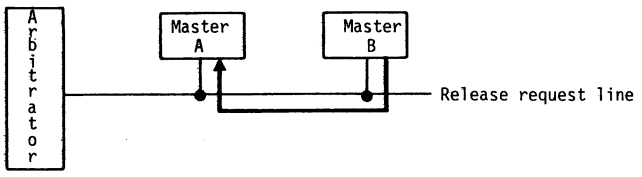


Fig. 23 Current master A, pre-empted by another master B, using release request line

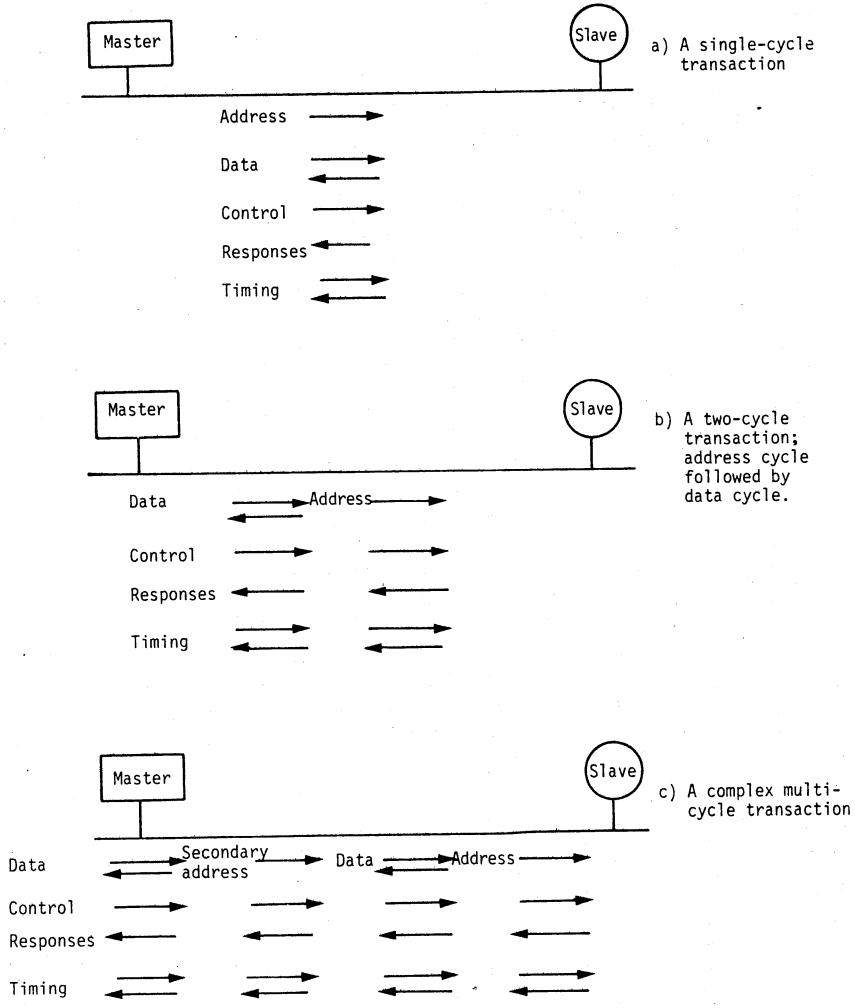


Fig. 24 Some examples of bus cycles and transactions

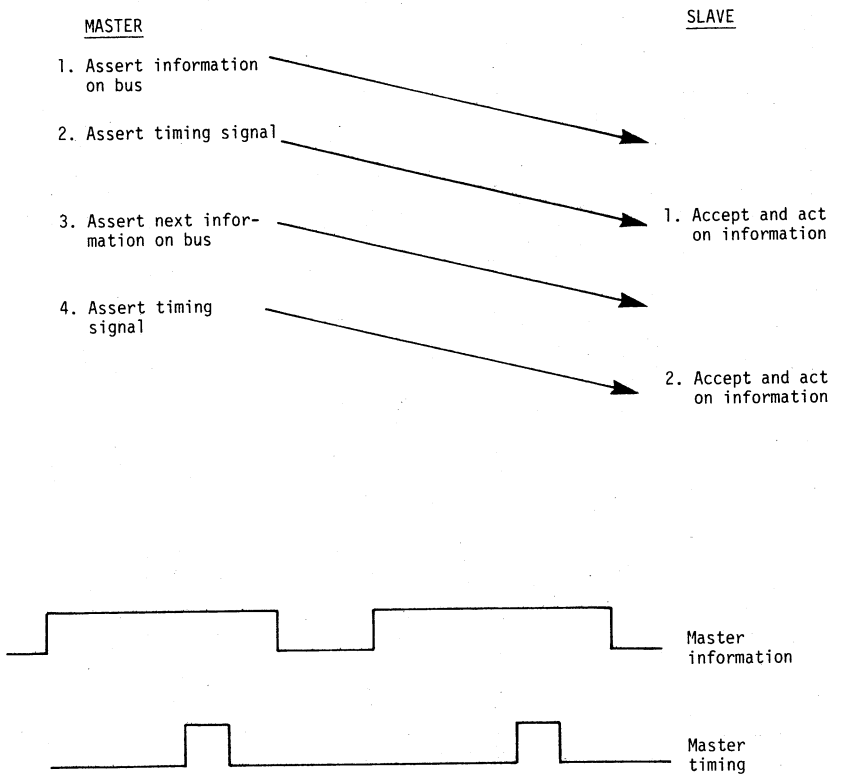


Fig. 25 Basics of synchronous bus timing; an example of information transferred from master to slave

until the slave sends an acknowledgement. The slave may provide data to be transferred to the master along with the acknowledgement (Fig. 26). Note that there is no fixed cycle time. The timing of operations between master and slave is adjusted according to the needs of that particular pair by using the handshake mechanism between them.

Most intrumentation and microprocessor buses are asynchronous in nature. The flexibility provided by a variable master/slave timing can be used to accommodate combinations of processors, memories, and peripherals with different speed capabilities. CAMAC is an exception in that bus cycles in a crate are synchronous. However, there are advantages in using non-handshaken bus cycles, because the handshaking mechanism involves a time overhead. To start with there is the propagation time between master and slave on the bus; and then there are additional electronic logic delays in generating the handshake mechanism. This effect can be typically 4 × the time for signals to travel between master and slave. For relatively slow master-to-slave communication this may not be a significant effect. However, where very high speed is required or where long buses have to be used, it is often preferable to use synchronous timing. Thus synchronous timing is used for high-performance computer buses. FASTBUS, which is a basically asynchronous system, has an optionally non-handshaken bus cycle. To put handshake times into perspective, let me comment that the typical propagation time from one end of a FASTBUS crate to the other is 15 ns, and that the cable used to interconnect CAMAC crates has a delay of ~ 5 ns/m.

2.5 Addressing

Address information is put onto the bus by a current master or commander in order to establish a connection with one or more slaves. This is the first part of any transaction, and it is the part which is concerned with selecting which slaves will take part in subsequent bus operations; it is the allocation of the responders or listeners. Address information sent by a master is used by all slaves to determine whether they should become connected. They become connected as a result of matching the address information on the bus with an internally known address — or in some cases, range of addresses. There are essentially three addressing modes to which a slave may respond:

- geographical (or physical) addressing,
- logical addressing,
- broadcast or multi-listener addressing.

Geographical addressing is most commonly used for establishing a connection, between a master and slaves, in the form of modules plugged into fixed slots of a 19-inch wide crate. A CAMAC crate, for example, has 25 slots or stations to which are allocated the addresses 1 to 25. Modules in a FASTBUS crate can also be selected via a position dependent address, although, as we shall see, the actual mechanism is different. In both these examples, the modules are essentially "told" their geographical address via their connection to the bus at the rear of the crate, i.e. via their back-plane connector. In the case of CAMAC (Fig. 27), an individual selection line runs from a central control station to every other station. Slave modules become connected when this line is asserted. In FASTBUS (Fig. 28), every slot receives from its back-plane connector a unique 5-bit coded station number. Slaves compare this address with address information put on the bus by a master; they will become connected, or attached, if a match is found. Even where slaves are not contained in a crate with well-defined slot numbers, they can still be allocated a single, unique, physical address to which they can respond. For example, a slave on a FASTBUS cable segment can have its physical address assigned via a switch-setting on the device.

We have seen how a connection between master and slave can be made using geographic addressing. The connection results in the selection of some particular module in a crate or of a device attached to a cable bus. However, nothing has been specified about the internal addressing within a slave. A slave will in general contain more than a single register — for example, it may contain a large memory whose different locations need to be accessed by a master. Geographic addressing is therefore almost invariably accompanied by an *internal or secondary address*. The original CAMAC standard supported internal addressing in the form of 16 subaddresses per module (Fig. 29). It became clear that such a small internal address space was in many cases a disadvantage — for example for large memories which have become ever cheaper and larger. Recently proposed up-grades to CAMAC now allow a 24-bit internal address field. FASTBUS allows a full 32-bit internal address.

Geographic addressing is an essentially position-dependent form of addressing. In many applications it is very convenient to refer to slaves by their physical location. For example, one can look at a CAMAC crate and know immediately that in a particular slot you have a number of analog signals going into a

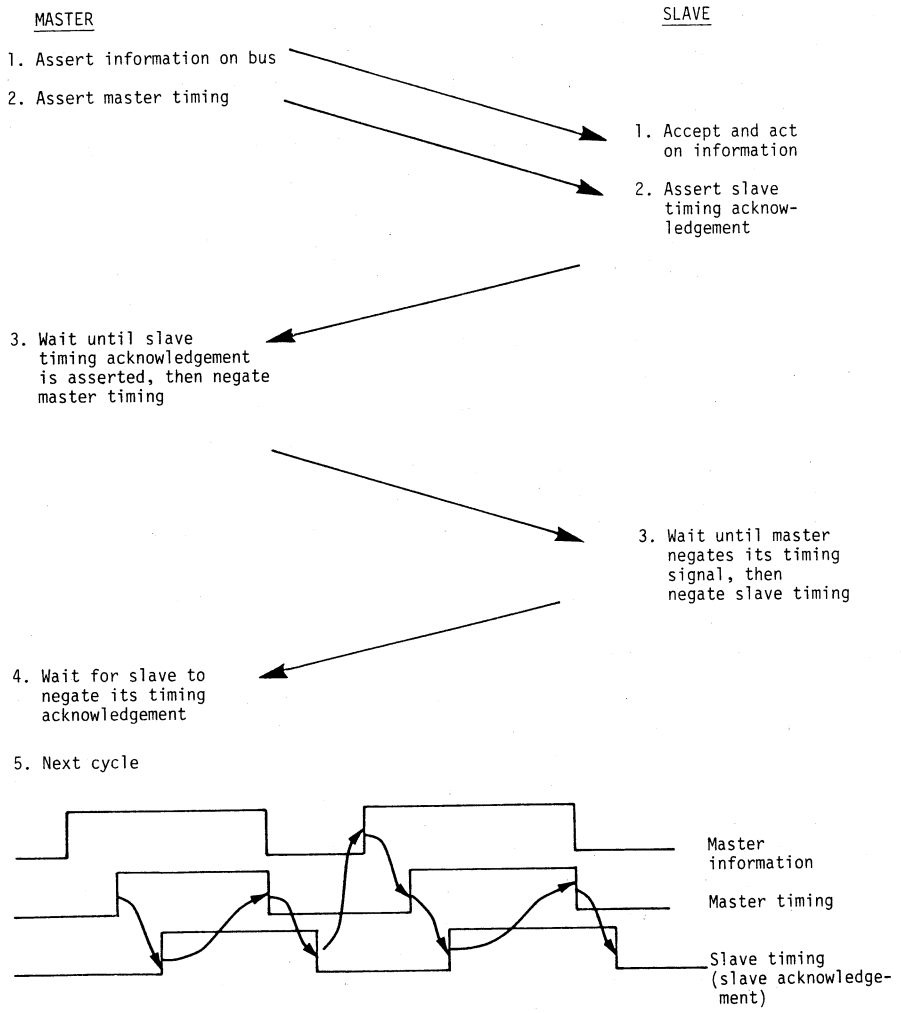


Fig. 26 Basics of asynchronous bus timing; an example of information transferred from master to slave

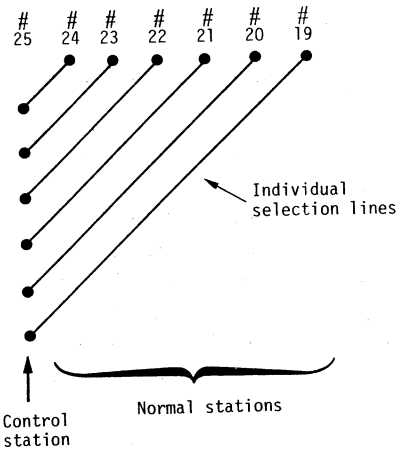


Fig. 27 CAMAC geographical or physical addressing mechanism

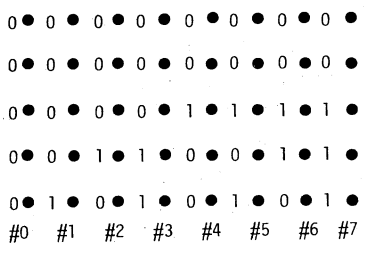


Fig. 28 FASTBUS geographic addressing. Every slot is allocated address via coded pins

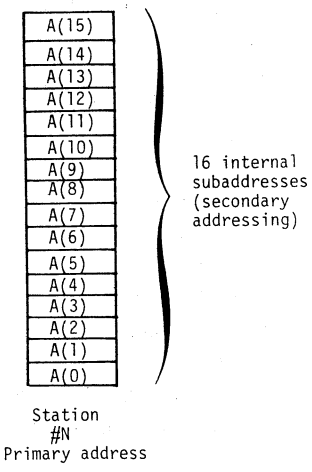


Fig. 29 The concept of CAMAC subaddresses

multiplexor module whose output goes to a digital voltmeter housed in some other well-defined station. In detector readout systems, all modules in a crate may be interrogated sequentially for valid data, using a simple scanning algorithm which runs over all subaddresses in all modules. It may not even be necessary to know precisely how many modules and how many subaddresses there are. Consider a different application where a microprocessor CPU is addressing memory packaged as plug-in modules in a crate. The CPU does not want to be bothered with details of which memory locations are in which modules. It is cumbersome to have to know implicitly that an 8 kword total memory space is comprised of several different physical modules; perhaps the modules may not even all contain the same amount of memory. So we see that, whilst position-dependent (physical) addressing is an attractive feature in some applications, it is a distinct disadvantage in others.

Logical or position-independent addressing is widely used in minicomputer and microcomputer bus systems. We introduce the principles used in this form of addressing by reference to the UNIBUS, a bus which has been employed by DEC for over 10 years and which has become, during this time, an almost *de facto* industry standard of continuing importance. UNIBUS addressing also has the merit of being relatively simple to understand. Address information on the UNIBUS takes the form of 18 signal lines which can be used to select 2^{18} different locations, each location corresponding to a particular number from 0 to $2^{19}-1$ and containing one byte of information. The total address space is thus 256 kbytes (1 kbyte = 1024 bytes) or 128 k 16-bit words. Most of this space is allocated to computer memory, i.e. storage elements for program and data. However, the top 8192 addresses are reserved for internal CPU registers and to registers associated with peripherals (Fig. 30a). For example, the addresses 777560_8 to 777567_8 could be used by a terminal interface to transfer characters between the computer and a VDU (Fig. 30b).

One of the difficulties and drawbacks in using logical addressing is the allocation of addresses to devices attached to the bus. In very many cases, devices have their logical address assigments made by using switches or jumpers. A UNIBUS memory board, containing 16 kbytes of memory corresponding to locations 16384 to 32767, would be hard-wired to respond to a base, or device address, at 16384. Keeping track of logical address allocations can be a major headache, especially where many different devices may be attached to the bus. In addition, reconfiguration always entails considerable disruption and the use of a soldering iron. A far more elegant solution can be found in FASTBUS, where geographic addressing is used to assign logical addresses to those devices requiring them.

Logical addressing has attractive features for certain applications, in particular for addressing memory. When interfacing peripherals or electronic instruments such as those used in high-energy physics, it is often more convenient to use position-dependent physical addressing. The real answer seems to be to allow both. Thus CAMAC, which started out allowing only geographic addressing, has had its scope expanded to allow logical addressing, whilst more modern buses such as FASTBUS contained both options from the start.

FASTBUS, which has a 32-bit wide logical address field, divides it into two parts (Fig. 31). The higher-order bits define a device or base address which is used by master and slave to establish a connection between them, and the lower bits make up the internal address of the slave. The device address can be considered the analog of the geographical address we have considered earlier. In one case a slave connects by recognizing a position-dependent geographic address; in the other, the connection procedure takes place using a position-independent device address.

So far, we have considered how a master (the commander) may establish a connection with a single slave, or responder, using geographical or logical addressing. However, in many cases it may be useful for a master to be able to connect with several slaves. For example, a master may wish to synchronize a bank of devices or send the same data to several slave processors for simultaneous computation. The selection of multiple responders is accomplished using *broadcast addressing*. We shall see that broadcast addressing may, in practice, use very similar mechanisms to those of geographic and logical addressing; nevertheless, we shall treat it as if it were a different address mode. One very simple example of broadcast addressing occurs in CAMAC: each module can be geographically addressed via an individual selection line; thus broadcasting may be very easily achieved by simply asserting several selection lines simultaneously (Fig. 32). The well-known GPIB bus also allows a form of broadcasting whereby information can be transmitted from a single "talker" to multiple "listeners". FASTBUS supports a special broadcast mode of addressing which allows a variety of multi-responder transactions. Multiple slaves can be selected on the basis of their physical position or according to whether they correspond to a particular class of device.

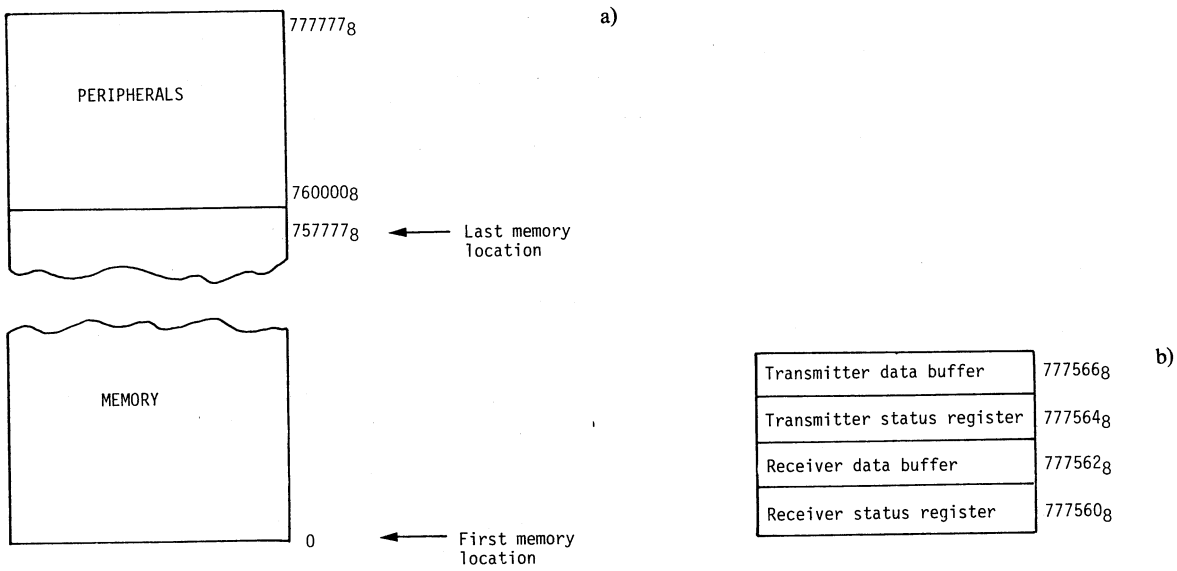


Fig. 30 a) UNIBUS address map
b) UNIBUS addresses used by VDU interface

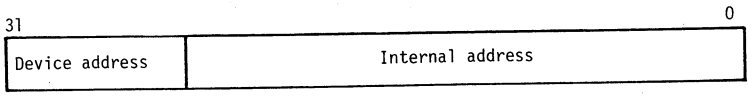


Fig. 31 Format of FASTBUS logical addressing

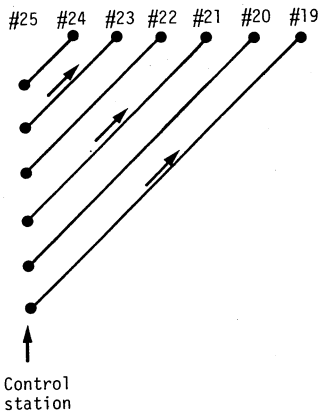


Fig. 32 A CAMAC broadcast, stations 23, 21, and 19 selected

Broadcast addressing is very useful for locating sources of interrupts on a bus, and also for the efficient readout of detector data in high-energy physics.

The type of broadcast addressing described to date allows the selection of multiple slaves. There exists a radically different type of broadcast which is sent unconditionally to all devices attached to a bus (normally via dedicated control lines, see later). Examples of this type of non-addressed broadcast are global requests to clear, inhibit, power up, initialize, etc.

2.6 Data transfers

The fundamental object of most transactions is to transfer data between storage locations in commanders and responders. Following the selection of a particular slave or slaves, data are either written from the commander to responders, or are read by the commander from responders. We shall shortly see how control lines can be used to expand the scope of both address and data transfers, but basically most buses simply allow reading and writing from particular slave addresses. Therefore at this stage I will limit my remarks on data transfers to comments on how to achieve speed and efficiency, and this will involve the concept of what are called *data block transfers*.

Very often a transaction involves the transfer of several data words to and from a slave. The initial addressing to set up a connection is followed by multiple Reads or Writes to that address. For example, the slave may be a disk controller or a data link which works on blocks of data. Especially where address and data are sent sequentially on the same bus lines, it is much quicker to send the initial connection address only once and follow it by a stream or block of data (Fig. 33). A variation on this theme allows still higher transfer speeds, employing a slightly modified handshake between master and slave from that shown in Fig. 34, which shows how synchronization between master and slave is maintained using both the rising and the falling edges of the handshake timing signals. A still more efficient data transfer may be obtained by dispensing with asynchronous timing and using a totally synchronous system (Fig. 35). FASTBUS, in particular, makes use of such block transfer mechanisms to achieve very high data throughput. It should be noted that a slave may choose to modify its current internal address automatically after each data transfer. For example, an address cycle might select a particular slave using geographic addressing and, in addition, specify an internal address of zero. A subsequent six-word block transfer write of data could fill the locations O to 5 inclusive, the slave maintaining an internal address pointer — a next transfer address — which it increments after every write (Fig. 36).

2.7 Control/Command information

Control and command information is sent from masters to slaves to qualify, in some way, the address and data operations. One very simple example could be a control line asserted to specify whether information was flowing from the master, a Write operation, or to the master, a Read operation. There are many other uses made of control information; we shall examine just a few of them. It is worth remarking that many buses are very similar in their address and data paths, but where they do differ is in the number and type of control and command signals.

Let us first focus on control signals associated with addressing. One common feature used in different bus systems is the ability to select more than one type of address space. Thus both FASTBUS and P896 use a control line to distinguish between data space and control and status space (Fig. 37). The idea behind having more than one address space is to allow protection from accidental overwriting. For example, a programming error may cause an array to go out of bounds and corrupt critical memory or control registers. Such corruption is less likely if data and control information are stored in distinctly different address spaces. Another use for control signals is to tell the slave how it should decode or treat address information. For example, control information could be used to specify a logical address of different widths — 16, 24, and 32 bits; this is done in the case of the VME bus. Control lines can also be used to distinguish logical from geographic addressing and to signal a broadcast operation to multiple responders.

As we have seen previously, the most common control signal associated with data transfer is a Read/Write line. Another very common role for control lines is to be able to choose to manipulate a subunit of data in a particular storage location, e.g. to read or write one byte of a 16-bit word (see Fig. 38), or to specify data paths of varying widths. FASTBUS allows some particularly interesting data cycle options via four different combinations of two mode select lines (Fig. 39). Along with normal data cycles, two types of block transfer, handshaken (asynchronous) and non-handshaken (synchronous), are supported. In addition, it is possible to specify a new internal address in the slave via what is called a secondary address

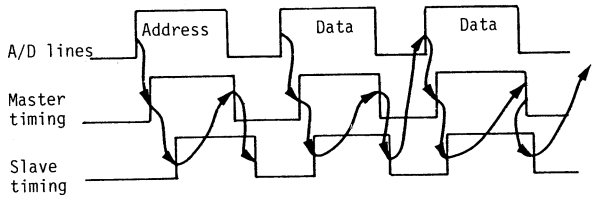


Fig. 33 A two-word block transfer write, data transferred on rising edge of master timing signal to slave. Asynchronous timing.

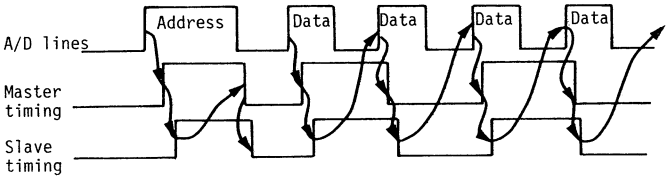


Fig. 34 A four-word block transfer write, data transferred on both edges of master timing signal to slave. Asynchronous timing.

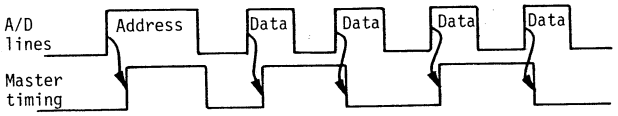


Fig. 35 A four-word block transfer with synchronous timing. No handshake from slave; transfer of data on rising and falling edges of master timing.

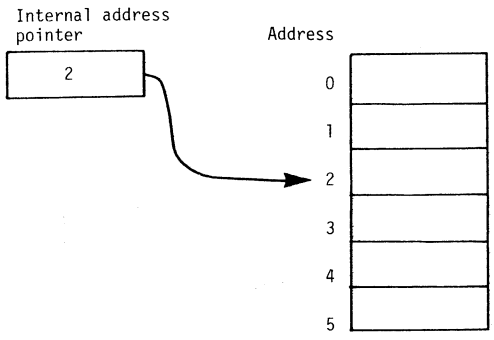


Fig. 36 Use of internal address pointer (next transfer address) during a block transfer

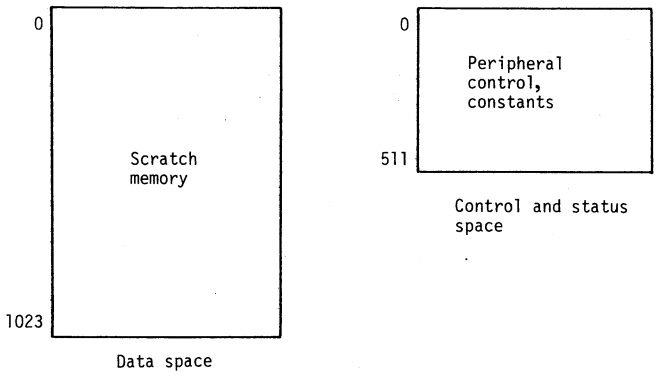


Fig. 37 Use of two independent address spaces

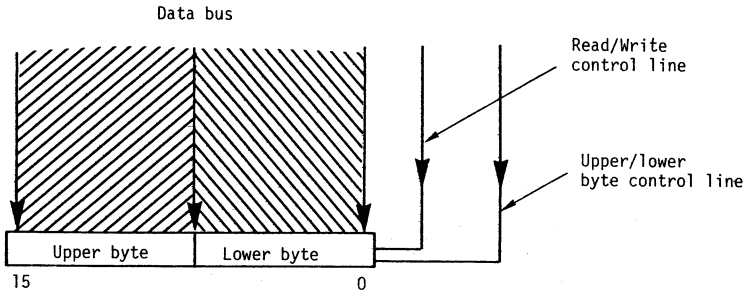


Fig. 38 Use of byte control line; data can be written into the upper or the lower byte, leaving the other unchanged

MS1	MS0		
0	0	random data	Asynchronous cycles
0	1	block transfer handshaken	
1	0	secondary address	
1	1	block transfer non handshaken	Synchronous cycle

Fig. 39 FASTBUS mode select codes used during data cycles

cycle. This feature means that once the initial connection between a FASTBUS master and slave is made, a transaction of almost unlimited complexity is possible. Figure 40 illustrates such a complex FASTBUS transaction.

CAMAC is probably unique in its ability to specify a wide range of different commands to be executed during the course of a bus cycle (Fig. 41). Not merely simple Reads and Writes may be specified, but Read and Clear, selective bit manipulation, Test, Enable, Clear—and more! Of course such operations can be performed using other buses, but normally at the cost of multiple bus cycles. For example, a bit set operation in a slave would require a bus cycle to read the data and, after the necessary bit manipulation, another to write them back. Much of the power and appeal of CAMAC comes from its rich set of standard function codes.

2.8 Responses

Responses are information given by a slave to a master during the course of a bus cycle. They can be viewed as slave status information. Buses differ widely in their use of responses, some having no response lines defined in their protocols, others having the ability for slaves to signal several status conditions in response to address and data cycles. Some of the status information a slave might want to return to a master are listed below:

- An illegal address has been sent; an invalid internal address has been specified by the master.
- An illegal operation has been requested, for example writing to a Read Only memory location.
- The slave cannot provide or accept data. Here two situations can be distinguished: the slave is temporarily in such a state—it is momentarily "busy"; or the state is more permanent. The first case could arise, for example, during a read from a FIFO, where emptying at the output port overtook filling at the input port. The second case would occur when all data had finally been read from the FIFO (Fig. 42).
- The slave detected a parity error on the bus (see Section 2.12).
- The slave wants to inform the master that it needs more time to complete the current bus cycle. This is particularly important for modifying a bus timing which is fundamentally synchronous in order to allow slow slaves to react.

The role of responses is more important in the case of instrumentation buses such as CAMAC and FASTBUS than it is for microcomputer and minicomputer buses such as the UNIBUS, MULTIBUS, etc. One reason is that in the former case slaves attached to the bus act as a source of variable and *a priori* unknown amounts of data which can be delivered at a varying rate. A slave containing wire chamber readout electronics will, for typical physics events, produce a variable number of data words (hits), whereas a disk controller transfers well-defined data blocks to and from disk (for example, a certain number of sectors). Responses permit the implementation of "simple scanning algorithms" for reading data from arrays of modules. Let us put some of these ideas about responses together in an illustrative example. Figure 43 shows a controller reading data from an array of slave modules in a CAMAC crate. Each slave is responsible for the digitization of drift-time information. It takes time to perform the digitization, and an unknown number of times have to be digitized. The controller addresses each module, performing a sequence of Read cycles to collect the data. Two response lines are used by the slave, HOLD and Q. If in response to a Read cycle the slave has valid data to transfer, it returns a Q = 1 response. The master accepts what is on the data lines and performs another data Read operation to get the next word. If there are no (more) words to read, the slave gives a Q = 0 response, and the master ignores information on the data lines and addresses the next module. If a slave is still busy with a digitization when a Read cycle comes along, it asserts HOLD. This prevents the master from continuing with the normal fixed cycle timing. When the slave has data ready, it negates HOLD and the cycle continues. Figure 44 is a flow chart of how the master executes the scanning algorithm. Figure 45 illustrates the use of HOLD.

Responses have proved very important in the efficient readout of data sparsely distributed over a large number of crates and modules. You will often have the term "sparse data scan" to describe such a process.

2.9 Requests for attention (interrupts)

In many situations devices attached to a bus require the attention of other devices—they may have data to transfer, they may have completed some action. A few examples are given below, and are further illustrated in Fig. 46:

Cycle No.	
1	Connect to slave using geographic addressing in data space
2	Write secondary address No. 3
3	Read contents (save)
4	Write secondary address No. 4
5	Write previous saved contents, then disconnect

Fig. 40 Example of a complex but single, multi-cycle, FASTBUS transaction

Function-Code Mnemonic Symbols

Symbol	Value	Definition	Symbol	Value	Definition
RD1	0	Read group 1 register	WT2	17	Write group 2 register
RD2	1	Read group 2 register	SS1	18	Selective set group 1 register
RC1	1	Read and clear group 1 register	SS2	19	Selective set group 2 register
RCM	3	Read complement of group 1 register	SC1	21	Selective clear group 1 register
TLM	8	Test LAM	SC2	23	Selective clear group 2 register
CL1	9	Clear group 1 register	DIS	24	Disable
CLM	10	Clear LAM	XEQ	25	Execute
CL2	11	Clear group 2 register	ENB	26	Enable
WT1	16	Write group 1 register	TST	27	Test

Fig. 41 CAMAC functions

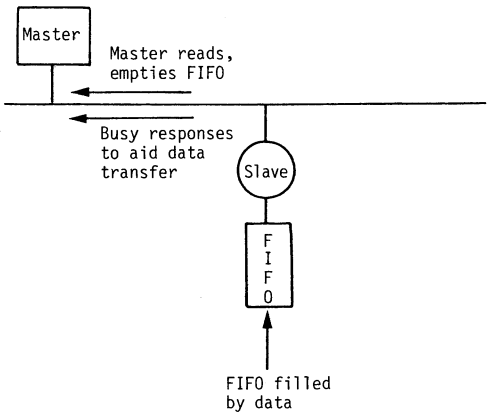


Fig. 42 Example of the use of busy response when reading data from a FIFO

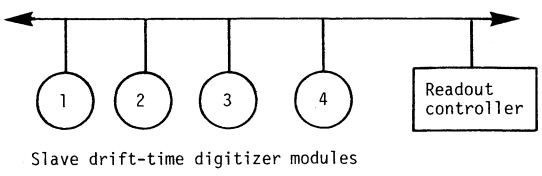


Fig. 43 Readout of digitized drift times from an array of slave modules

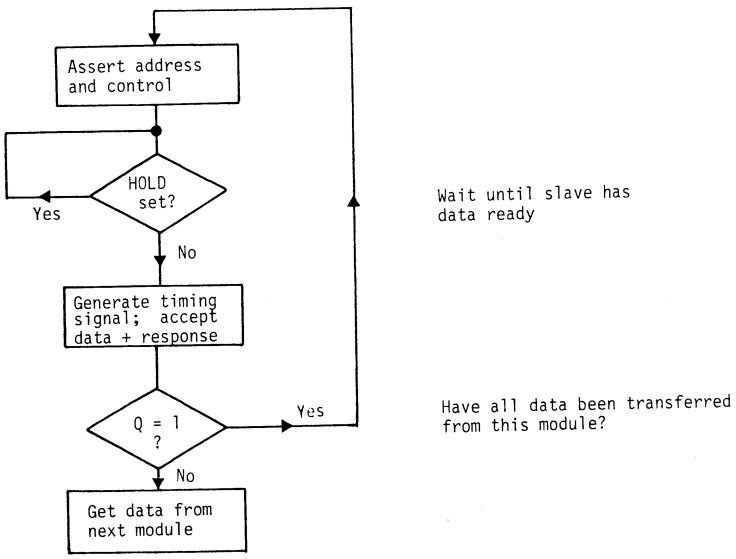


Fig. 44 Scanning algorithm in master

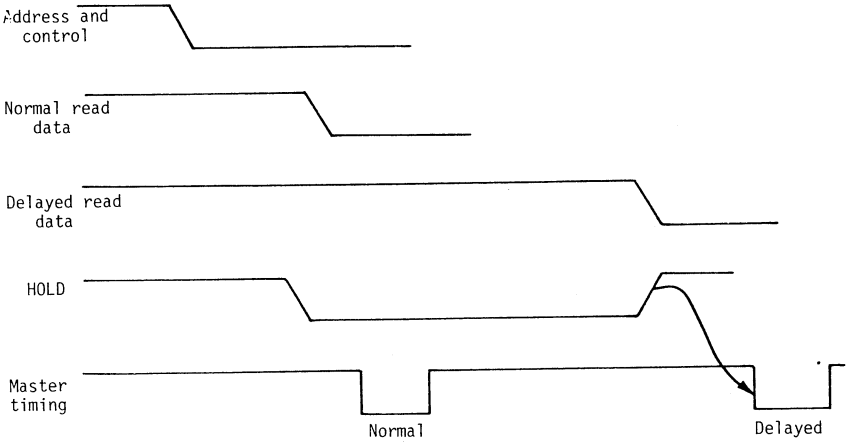


Fig. 45 Use of HOLD line

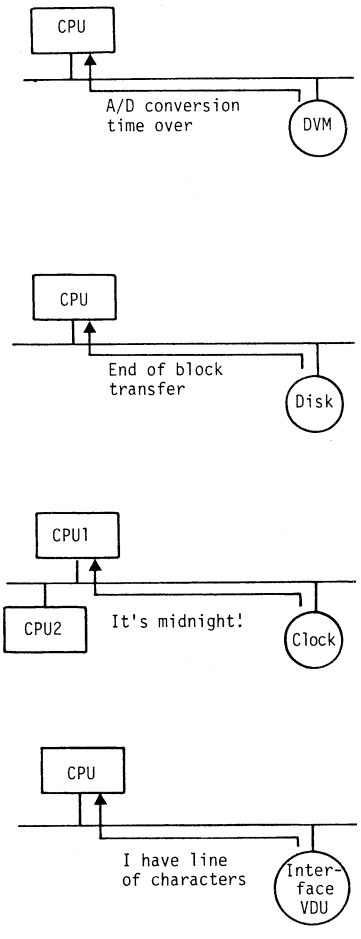


Fig. 46 Examples of requests for attention

- a slave containing a clock may be used to prompt a master to perform some action every hour;
- a teletype interface needs to signal to a processor that it has a character in its input buffer;
- an I/O processor may need to inform the main processor that it has completed a transfer from memory to a peripheral;
- a bus error may occur.

Before proceeding further, let us see what is meant by the term *interrupt*. The word is often used in a loose and misleading way when discussing buses, and it is worth trying to clear up a little of the confusion. Historically, interrupts arose from the realization that input and output operations involving CPU, the memory and peripherals could easily waste large amounts of CPU time. Time used in polling loops for I/O devices to accept or transmit information, was time lost for computation. This sharing of the CPU between the two tasks could be much more efficiently accomplished through a program interrupt facility. The value of interruption lies in the ability of the CPU to respond automatically to certain conditions, either internal or external (coming from peripherals). Conditions occurring at unknown times (i.e. asynchronously) can generate an interrupt and force the CPU to execute some prespecified routine. Basically, then, an interrupt is a subroutine jump executed by hardware. The interrupt occurs after the current instruction has been completed, and afterwards the program flow resumes where it left off. The important point to stress is that requests for attention from devices may very well cause a true interrupt in some processor. However, this is not always the case, and it can sometimes be very misleading to refer to the process of requesting attention as "interrupting". This is particularly true when a bus has been designed to be independent of any type of processor, and is the case with CAMAC and FASTBUS. For this reason we would like to minimize the use of the word "interrupt" in this section, although inevitably it will creep in!

We are going to consider how requests for attention may be transmitted over buses. There are two general ways of doing this:

- a requesting device acts throughout as a slave — it makes a passive request for attention;
- the device becomes bus master in order to transmit its request — it makes an active request for attention.

The mechanism used by CAMAC modules to request attention at a crate level is very simple. Each station has an individual attention line which goes to the controller station. When a device needs attention, it simply asserts its individual Look at Me (LAM) line (Fig. 47). The master occupying the control station knows immediately which station is requiring service. Of course, there may be multiple service requests associated with a single station; in this case, further interrogation of the device is necessary. Despite some of the additional complications that can occur in multi-crate and multi-master systems, at the level of the crate dataway the CAMAC demand-handling is rather straightforward.

Let us now turn to the rather different scheme shown in Fig. 48. Here there is a single attention-request line which is shared by all devices on the bus. Requests from all devices are serviced by a single master, the attention-request handler. There may be other masters, but there is normally only one request handler. When the handler sees a request, it becomes bus commander and polls all slaves to identify which one has an outstanding request for attention. Alternatively, the master can send an acknowledgement along a special bus line which is daisy-chained through all devices. When a device sees an acknowledgement, it looks to see whether it has an outstanding request for attention. If it has, propagation down the daisy chain is blocked; if it has not, the acknowledgement is passed on. A device requiring attention will, after receiving the acknowledgement, place an identifier — sometimes called a vector — on the bus, which is then read by the handler. The handler may thus acquire a unique identification of the device requiring attention. Such a scheme can be generalized further by allowing multiple request lines (seven seem a typical number), each used by several devices and each capable of going to different handlers (Fig. 49). There may be either a separate acknowledgement for each request or a single acknowledgement line which is accompanied by addressing information to identify which request is being acknowledged. MULTIBUS and VME bus use this type of scheme for handling requests for attention.

A distinctly different approach for devices requiring attention is for them to become bus commander and then transmit, to the required destination, an attention request message. Such an approach has been adopted for three of the newest buses to appear on the scene: FASTBUS and P896. Whilst previous examples given have necessitated special extra lines (if you like a separate *"interrupt" bus*), these three buses use standard bus protocols; messages are written as normal data to special addresses within handlers. Figure 50 shows the use of "interrupt" messages in FASTBUS. However, despite the adoption of this phrase in the formal FASTBUS specifications, the reader is warned of the misnomer of the term

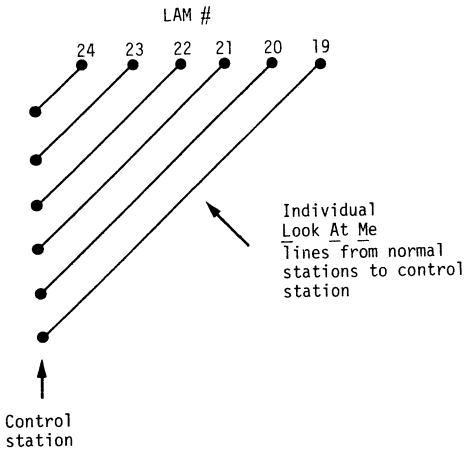


Fig. 47 CAMAC LAMs

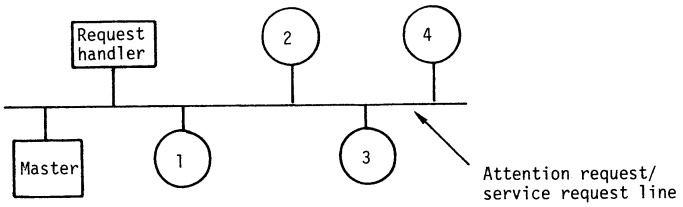


Fig. 48 A simple scheme by which several slaves can request attention from the handler via a single service request line

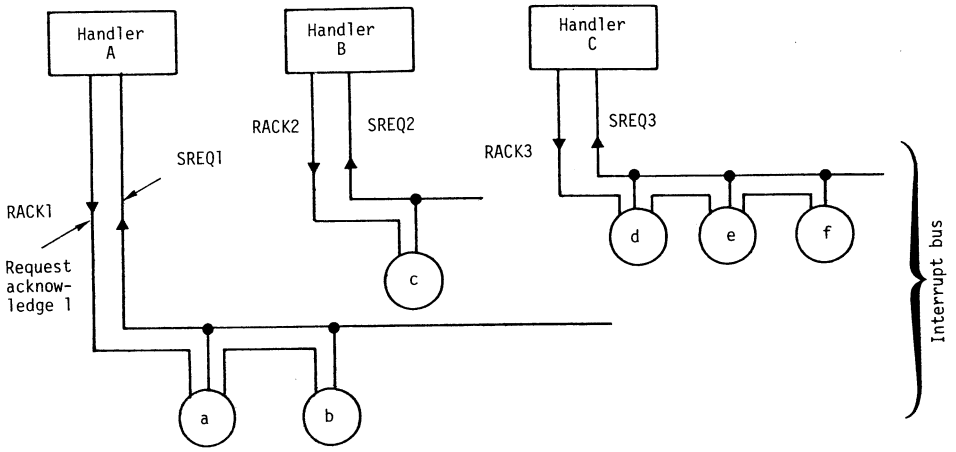


Fig. 49 Multiple service request and acknowledgement lines

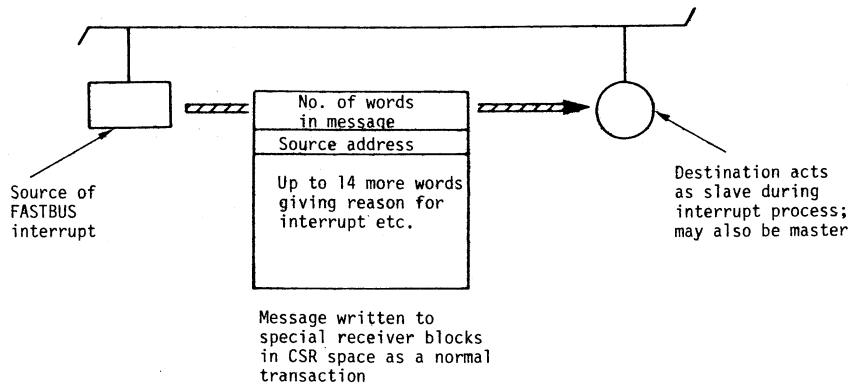


Fig. 50 Use of attention request messages; FASTBUS "interrupt" message

"interrupt" in this context. A message may never cause a true interrupt. There is a clear advantage in handling requests for attention in this way instead of using special lines, daisy-chaining, and so on. The procedure is position-independent, there are no special control stations, and it is very easy to change and reconfigure a system, dynamically if necessary. In contrast, buses that utilize hard-wired interrupt schemes are tedious to change and often quite difficult to debug. One disadvantage of systems where the attention requesters must themselves be able to become bus masters is the extra electronics that may have to be incorporated in a device. However, for systems with multiple attention requesters and multiple request handlers this way of working is much more viable. FASTBUS tries to make the best of both worlds by allowing the use of a single service request line, for simple attention-requesting slaves, to coexist along with the transmission of quite complex interrupt messages.

2.10 Handling errors

I want to mention very briefly some of the error conditions that may arise when operating buses. These can be divided into two general classes:

1) errors introduced in the transmission of information along the bus by noise and pick-up;
2) some sort of master- or slave-related faults, e.g. non-existent internal address, non-permissible operation.

Type (1) errors may be detected by the use of parity generation and checking. Thus for information flowing from master to slave, the master generates one or more parity bits which the slave checks. For information flowing in the opposite sense, the parity bits are generated by the slave and checked by the master. In the former case it is up to the slave to signal that it has found something wrong, for example via a special response. In the latter, it is up to the master to take corrective action, for example by repeating the operation. Very few parallel buses use parity checking for detecting bus transmission errors. This is in contrast to serial buses (especially where communications are going over very long distances or via poor quality lines). Parallel buses are normally assumed to be relatively short and to operate in a non-hostile electrical environment.

Type (2) errors can be actively detected by either master or slaves. Thus a CAMAC module, when presented with a non-valid internal address, can signal, via a special response line, that it cannot perform the required operation. Alternatively, for handshake buses, if a master receives no timing handshake back from a slave within a certain time (because for example no one recognized the address on the bus) it flags this bus cycle as being in error and terminates it (the master "times out").

3. THE INTERCONNECTION OF BUSES

3.1 General principles and problems

A bus, as we have seen previously, consists of a shared communications path that is used by a master to perform transactions with one or more slaves. If there is more than one potential master on the bus, an arbitration mechanism is required to avoid clashes between them. It should be noted, however, that there can only be a single commander at any time.

A single such *bus segment* may be inadequate for many applications:

- the bus segment may become overloaded physically (a full crate) or electrically (limitations in the bus tranceivers); in addition, its capacity for carrying traffic may saturate at some point;
- the maximum permitted bus length may be exceeded;
- the single commander permitted excludes parallel processing.

For these reasons it is important to be able to link bus segments together. Think of trying to join two CAMAC-type crates. The most general interconnection scheme for two buses is shown in Fig. 51. Buses A and B are brought into close proximity and joined by a *bus or segment interconnect.* A and B can act completely independently, M1 dialogues with S1 and M2 dialogues with S2, and there is parallel processing. However, when required, the two buses can be linked: M1 can talk to S2, and likewise M2 and S1. But things are not quite so simple. The mechanism for such inter-segment transactions would involve M1 obtaining mastership of bus A and, after passing through the segment, interconnect to obtain also mastership of B. Transactions would then take place as usual between M1 and S2. There is, however, at least one major problem to be solved. Suppose M1 and M2 are attempting inter-segment transactions at the same time. Both obtain mastership of the segment to which they are attached—the nearside segment—but cannot gain

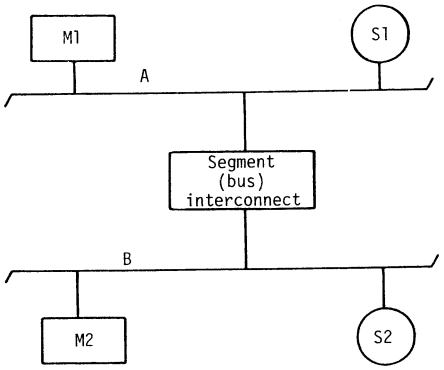


Fig. 51 The interconnection of buses

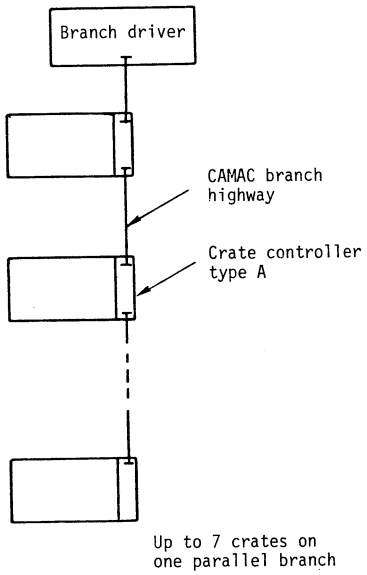


Fig. 52 The CAMAC parallel branch highway

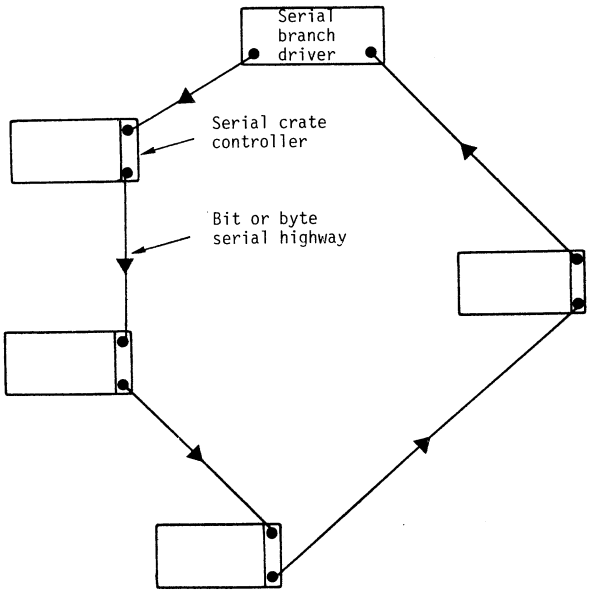


Fig. 53 The CAMAC serial highway

mastership of the far side segment because it is already claimed by the other master. There is thus a *"deadly embrace"* situation which requires some *"tie breaking"* mechanism to resolve it. A subsidiary problem is how to identify or address devices on different segments.

Multi-segment interconnected systems are one of the most interesting and challenging aspects of bus technology. They are also comparatively rare. I will illustrate possible implementations by reference to CAMAC multi-crate systems, interconnected UNIBUSes, and FASTBUS.

3.2 CAMAC multi-crate systems

3.2.1 The CAMAC parallel branch highway

CAMAC, as we have seen, is based on bus segments housed in crates to which plug-in modules can be attached. Multi-crate systems can be synthesized using a so-called branch highway (Fig. 52). The system is controlled in a hierarchical manner by a single master, the branch driver normally placed at the head of the cable bus which links up all crates. The branch highway has an immediate heavy restriction — there is no provision for arbitration. This means that no transactions are possible between a master in one crate and a slave in another crate.

Each crate is attached to the branch highway via a standard crate controller (or bus interconnect), which is allocated a unique crate number from 1 to 7. Addressing information sent on the branch comprises a crate number plus a station number and an internal subaddress. A branch driver generates address information required to perform a transaction at a crate level, plus a crate number. A crate controller acting as a slave reacts to address information sent by the driver; it recognizes its particular address. It accepts further address and control information, plus any data from the master, performs a CAMAC operation within the crate with a module (acting at this time as a master), and returns responses and any data to the branch driver. One can say that a crate controller performs a crate dataway transaction at the request of the branch driver.

The information carried over the branch highway is very similar to that in a crate. However, the bus lines and protocol do differ — in particular, the timing on the branch, which is asynchronous instead of basically synchronous in the crates. The minimum time to perform a CAMAC operation in a module via a branch driver is theoretically $\sim$ 1.5 μs; in practice it is normally over 2 μs. This time will increase by 20 ns for every 1 m increase in the length of the branch highway because of master/slave handshake times. Normally, parallel CAMAC branch highways are operated up to a length of about 25 m. With special arrangements to ensure noise immunity and to overcome ohmic losses, this distance can be stretched to several hundred metres. However, the cycle time for a branch operation would then be typically 2 μs + 20 ns $\times$ 200 m = 2 μ + 4 μs = 6 μs for a 200 m long branch. In addition, the cost of the cable would be high and its installation troublesome and time-consuming. At this point it becomes interesting to consider a different type of inter-crate bus, a serial branch highway.

3.2.2 The CAMAC serial highway

The CAMAC serial highway interconnects a master device — the serial driver — and up to 62 crates. A unidirectional loop passes from the output port of the serial driver, through a standard serial crate controller in every crate, and back to the input port of the serial driver (Fig. 53). All transactions on the bus take place via messages. Such messages are comprised of a sequence of 8-bit bytes (Fig. 54). Bit 7 of every byte is a delimiter bit which allows receiving devices to identify the first and last bytes of a message. Each message contains a header byte which includes a device (crate) address. In messages from the serial driver, the address is the destination message. In messages to the serial driver, it contains the source address. A command message (Fig. 55 shows a Write) is sent by the serial driver in order to execute a CAMAC operation in one of the crates attached to the loop. When a crate controller recognizes its address, it assembles the message and executes the required CAMAC crate cycle. It then sends a reply message (Fig. 56 shows the result of a Read) to the serial driver. The serial crate controller thus acts as a slave at its serial highway port and performs bus cycles, as a master, within the crate receiver. Requests for attention can be sent from a crate controller, in response to LAMs occurring in modules in the crate, to the serial driver using a special 3-byte demand message.

Physically a CAMAC serial highway can transfer information in either byte serial or bit serial form. The transfer of bits or bytes is synchronized via a clock line driven by the serial driver and running up to speeds of 5 MHz. Serial crate controllers can be interconnected by private twisted-pair cables up to distances of several hundred metres. It is possible to extend this distance using commercially available

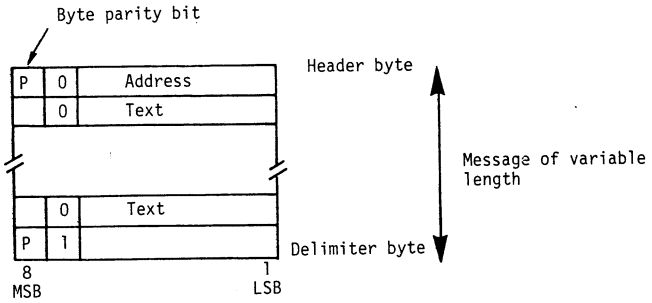


Fig. 54 Basic serial CAMAC message format

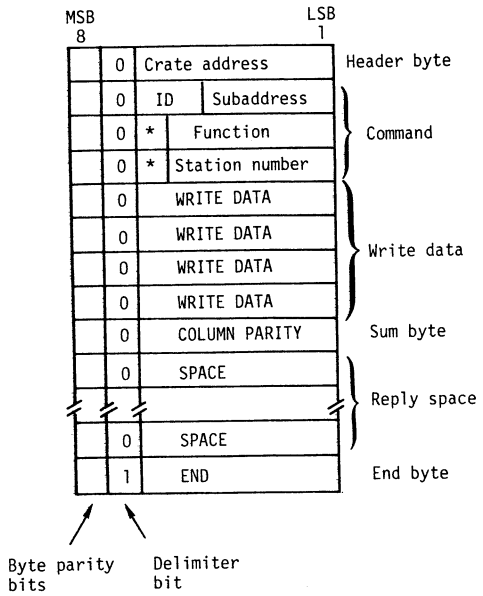


Fig. 55 Write command message

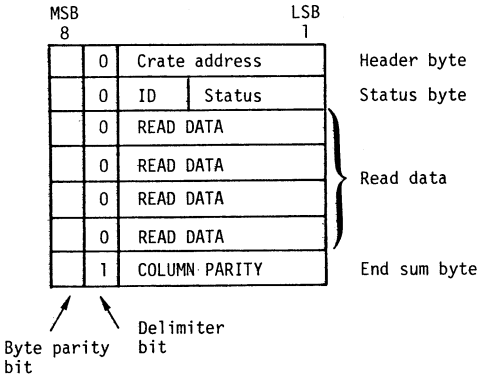


Fig. 56 Read reply message

communications equipment. For example, Fig. 57 shows a MODEM used to extend an inter-crate distance over several kilometres, perhaps using telephone lines or fibre optics. As serial CAMAC has been foreseen for this type of communications and for environments where there may be electrical noise and pick-up induced onto the highway, parity generation and checking at a byte and message level have been foreseen, as shown in Figs. 55 and 56.

Transfer rates obtainable with serial CAMAC highway cross over rapidly with those of the parallel branch once the length of the latter goes over a few hundred metres. A CAMAC operation involves the transmission of typically 20 bytes around the loop. At a clock rate of 5 MHz this takes 4 μs. The delay due to propagation of this information on the loop is 5 ns times the total loop length in metres. A 200 m loop would imply a propagation delay of 1 μs. The total time for the transaction would then be 5 μs, which would compare favourably with the performance of a 200 m long parallel branch. The disadvantage of byte-by-byte transmission is compensated for by the absence of the handshaking required on the parallel branch.

This has been a very rapid outline of the CAMAC serial highway for interconnecting CAMAC crates over long distances. Notice that the possibilities for generalized communication over the loop are really rather restricted. There is normally only a single serial branch driver, although a few applications have operated with more. There is no crate-to-crate communication — crate controllers are rather passive animals obeying messages from a master, and from time to time requesting attention from him. Notice also the similarity to a local area networks (LAN) ring: the fact that serial transmission is used; the use of messages sent by the branch driver to evoke CAMAC crate cycles; the inclusion now of parity; the distances that may be spanned. I also recently realized with interest that the mechanism used to send demand messages on the CAMAC serial loop is almost identical to the one used to send messages on some types of LAN ring — register insertion. So you can see that serial CAMAC really is a bridge between parallel and serial buses. Later we shall briefly return to this point.

3.2.3 The CAMAC system crate principle

The system crate is interesting because it overcomes some of the limitations and restrictions of conventional CAMAC. It is widely used throughout Europe and in particular in the nuclear field.

A system crate is a CAMAC crate which contains four basic types of unit (see Fig. 58):

- sources of CAMAC commands,
- an executive controller,
- branch couplers,
- normal CAMAC modules.

System crate sources are normally driven by minicomputers or microcomputers. These may comprise units for performing simple program control transfers between computer and CAMAC, a high-speed direct memory access controller, and modules for interrupting the computer in response to CAMAC LAMs. Sources use a special (external) arbitration bus in order to become master and dialogue with the executive controller. This, in turn, generates CAMAC cycles to normal slave modules housed either in the system crate itself or in crates attached to parallel or serial branch highways driven by so-called branch couplers. The system crate can multiplex many sources of CAMAC commands (i.e. many masters). It thus partly overcomes the restriction of a single branch highway driver per branch, implicitly present in standard CAMAC. It also allows an extension of the number of crates a computer can drive — on a single branch highway, only seven. This comes about because seven parallel or serial couplers can be accommodated by the scheme. However, it should be stressed that the basic architectural restrictions of multi-crate CAMAC remain.

3.3 The UNIBUS window

A very interesting insight into linking two bus segments together can be obtained by studying the UNIBUS window principle introduced by DEC to interconnect two UNIBUSes (Fig. 59). The UNIBUS has an 18-bit-wide logical address field; this is normally used to identify memory locations, with the top 8 kbytes being allocated for peripheral control and transfers. The trick used by DEC is to allocate "unused" addresses on the near-side bus which are to be passed through the window to generate transactions on the far-side bus. Any unused block of addresses, from 1024 to 64 kbytes, may be designated as the window. Normally this is placed directly above the last memory module (Fig. 60). Thus on a system with 128 kbytes of memory, a 16 k window would be placed from 128 k to 144 k. Once this window is initialized, any near-side address within this range is automatically translated to generate a far-side address within another

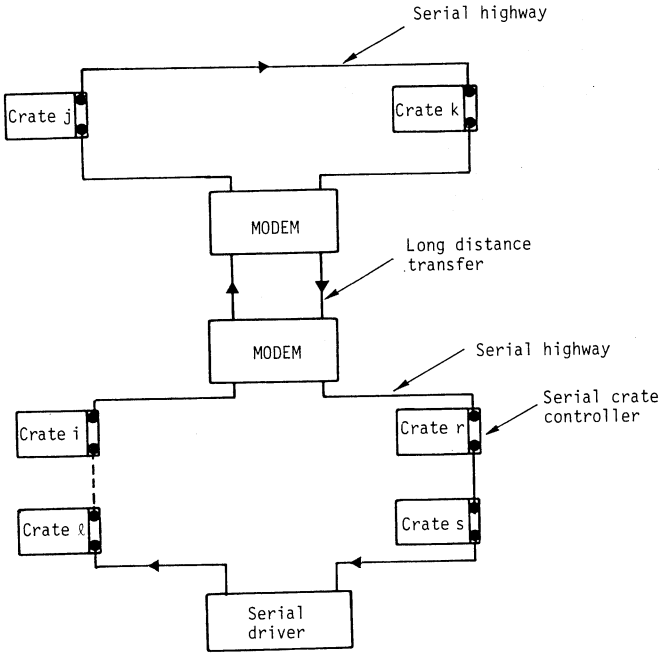


Fig. 57 Long-distance communication using MODEMs

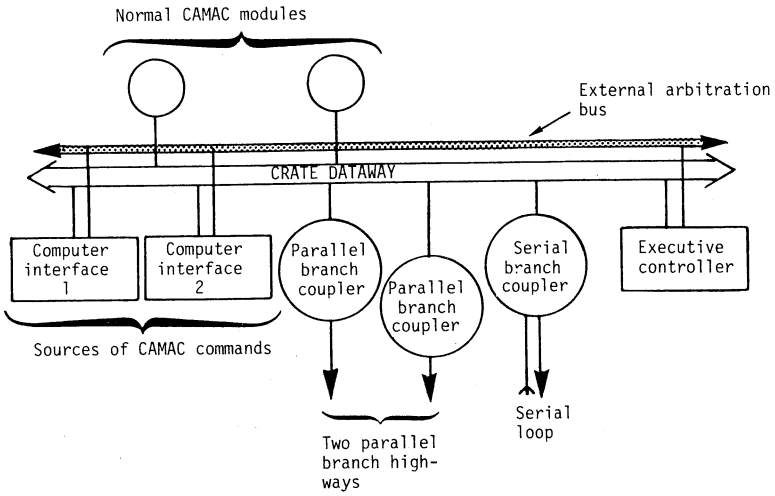


Fig. 58 The system crate principle

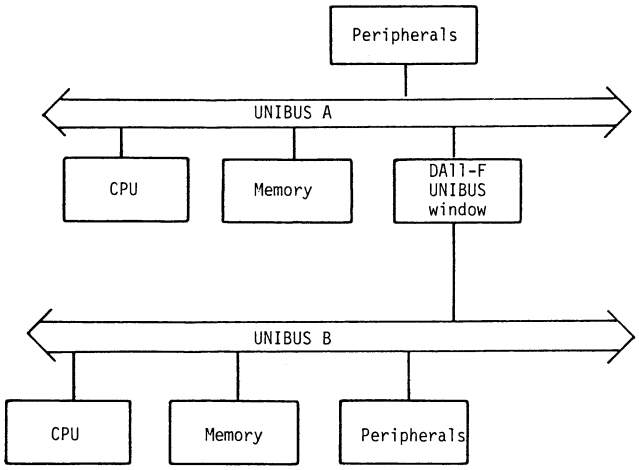


Fig. 59 UNIBUS window

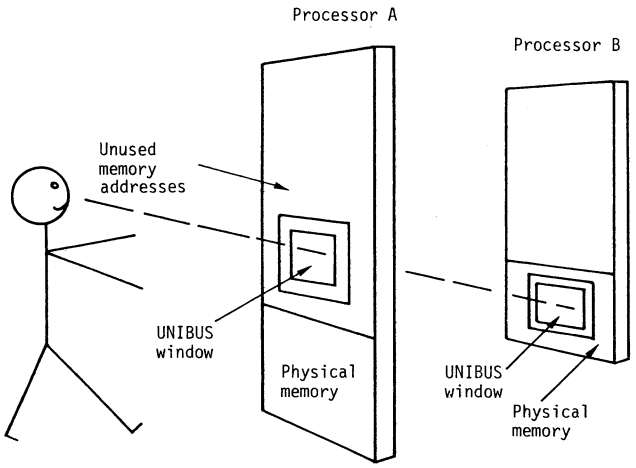


Fig. 60 Principles of the UNIBUS window

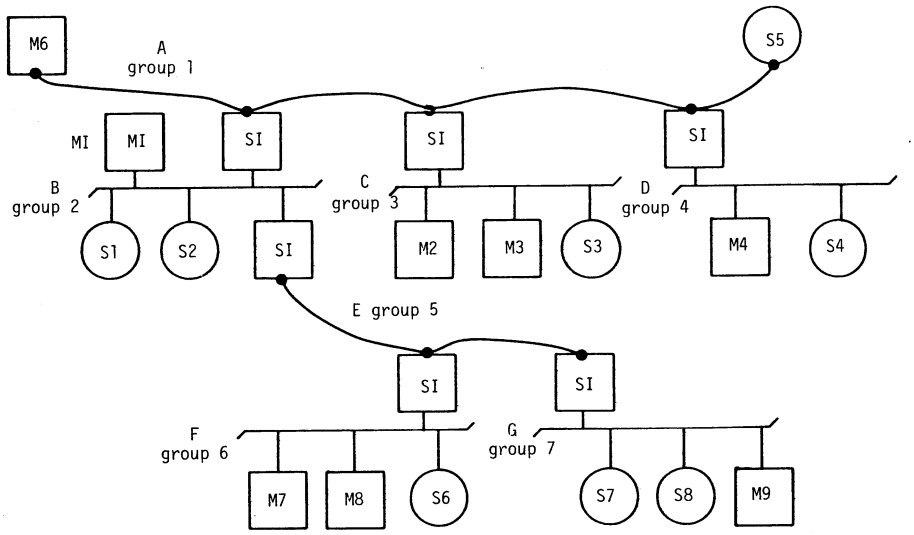


Fig. 61 The interconnection of FASTBUS segments using segment interconnects (SIs)

preselected 16 k region. The range of addresses may be different on the two buses, and address transformation may occur. A window operation involves both the UNIBUSes, the bus interconnect acting as a slave on its near side and a master on its far side.

Let us now consider how data may be written and read. For Write operations, the near-side master first causes data to be written to a buffer in the interconnect module. Since the data are stored, the near-side cycle can be concluded. The interconnect now obtains control of the far bus and transfers data from the buffer to its final destination. For a Read operation, the near-side bus cycle is linked to the far-side bus cycle. This is necessary because data must be obtained from the slave and transferred to the master before the near-side bus cycle can be terminated.

Suppose now that two transactions clash whilst passing through the interconnect. There are three cases to consider:

1) If both ports receive a Write, then, because data are buffered and the two bus cycles not linked, there is no interference.
2) If there is a Read and a Write, the Write waits until the Read is completed; again there is no interference.
3) However, if the two ports receive a Read, neither can be completed, a deadly embrace occurs, and both originating masters will eventually time out.

Two basic principles have been introduced:

- the passing of a range of address out of one bus segment and onto another in order to generate a far-side transaction;
- a possible address transformation during this process.

We will meet them again in the next section.

3.4 The interconnection of FASTBUS segments

FASTBUS systems are normally comprised of two types of bus segments: crate back-plane segments, used for interconnecting plug-in modules; and cable segments, which can be used not only to link crate segments, but also to interconnect devices attached directly to the cable (Fig. 61). The electrical properties of the two types of segments differ. However, the protocol used in each case is identical, and logically speaking there is no difference between them. Figure 61 shows the use of FASTBUS segment interconnects (SIs) to link together different bus segments. All segments can operate completely independently if masters are accessing slaves on the same local segment. However, if a master wishes to perform a transaction with a remote slave, two or more segments must be interlocked to act as one. For example, referring to Fig. 61, if M4 wants to talk to S7, then segments D, A, B, E, and G are involved. Let us see in a little more detail how a connection to a remote slave is established. (Always refer to Fig. 61.)

FASTBUS has a 32-bit primary address. Normally the top 8 bits are used to specify a group field. Each segment is allocated a unique group number. Transactions by a master using a local group number remain on segment. An SI can be set up to pass addresses involving non-local groups from its near side to its far side. It will obtain far-side mastership and repeat the address cycle. The SI acts in a very similar way to the UNIBUS window described in the previous section. This procedure can be continued through successive SIs until the final destination segment is reached and a connection made with the required slave. SIs thus "route through" the request for a connection between the originating segment and the final destination remote segment. Once the connection is established, all segments along the way are "transparent" to further bus cycles but are effectively tied up for the duration of the transaction. The route can even be held established for several transactions. Suppose, now, that M1 wants to talk to S8: M1 generates an address with a group number = 7; the lower SI on segment B passes the address onto segment E, from whence the appropriate SI transfers it through to segment G. Suppose that whilst the route from M1 to S8 is being established, M9 decides to address S1, and that the two transactions collide in one of the intervening SIs. What happens? In this case, FASTBUS responses are used to signal to one of the masters, the one with the lower priority, that it should give up mastership of all buses and try again later.

4. SOFTWARE ASPECTS

Most users of buses are blissfully unaware of all of the details presented so far in these lectures! That this is so is almost totally due to the existence of successive layers of software that shield users from such details and allow them to concentrate on their applications. For example, a programmer performing input or output to microcomputer or minicomputer peripherals will normally use calls, directly or indirectly, to a

device driver from a high-level language (Fig. 62). This device driver, which performs operations directly on the I/O bus to transfer data and control information to and from the peripheral, will most likely be provided by the computer manufacturer. However, this is not normally the case when the peripherals concerned are CAMAC and FASTBUS systems. This type of driver is normally written "in house".

One of the most significant achievements in nuclear instrumentation over the last decade has been the widespread standardization of electronic hardware using CAMAC. Equally important has been the standardization in software which has allowed a high degree of computer independence and transportability of applications software. I want to give two examples of high-level language facilities that are widely available.

In a FORTRAN environment, access to CAMAC has been provided via subroutine calls. A standard set of calls was defined by the ESONE/NIM committees in 1978, and these have been implemented for a very wide variety of computers of all types. In addition to a basic set of calls to perform individual CAMAC functions, there are facilities to suspend and reactivate tasks in response to LAMs and calls to perform CAMAC block transfers. Figure 63 shows some examples of ESONE subroutine calls.

Many applications, in particular those associated with the checkout and debugging of electronic equipment or with the support of relatively small experiments, will greatly benefit from the use of interpretive, BASIC-like languages. CATY is an example of such a language which is very widely used throughout Europe. It has the advantages of having facilities to access CAMAC built into the syntax of the language. Figure 64 shows an example of a simple CATY program. Note that there are mnemonics for all CAMAC function codes (e.g. RD1 for F0), and that CAMAC address can be equated with symbols (e.g. DISPLAY = branch, crate, station, sub-address). This makes it very easy for the complete beginner to write code for quite sophisticated use of the CAMAC buses without having more than a rudimentary knowledge of their detailed workings.

The dual approach — using CAMAC from FORTRAN and "BASIC-like" environments is being continued with FASTBUS. A set of standard subroutine calls has been defined (Fig. 65), and an interpretive FASTBUS diagnostic language has been developed at the University of Illinois. In Europe we have made extensive use of the standard subroutine calls in the CERN FASTBUS pilot project, embedding them in both FORTRAN and CATY programs.

5. SUMMING UP

I want to start by presenting a list of questions that can be asked about buses. This will highlight some of their more important characteristics and give you some idea of how you might choose a bus for a particular application.

1) Is it a widely supported standard?
2) Is there good software support?
3) How easy is it to interface to the bus? Are protocol integrated circuits or other interfacing aids available?
4) Is it manufacturer-independent and processor-independent?
5) What is its addressing capability?
6) What is the ability to handle variable-width data fields (bytes, words, long words)?
7) How many bus lines? Serial or parallel? Multiplexed address and data?
8) Does it have an advanced architecture?
   - multi-master capability?
   - flexible arbitration?
   - good "interrupt" scheme?
   - multiple segment linking?
9) Performance (a few Mbytes/s, a few tens of Mbytes/s, data transfer rates)?
10) IC technology supported (TTL, ECL)?
11) Standard crate, back-plane, and card mechanics?
12) Well thought out power supplies and cooling?
13) Cost? The price per cm^2 of available board space, including interfacing to the bus, board, mechanics, power, and cooling?

In my view, whenever choosing a bus for any application one should, if at all possible, go for a well-supported standard with widely available hardware and software. It is almost always illusionary to think you can produce your own bus with better performance or a lower cost. Remember also that a true price must

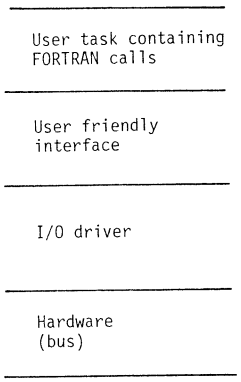


Fig. 62 User shielded from details of bus by successive layers of increasingly friendly software

```
C       SIMPLE FORTRAN PROGRAM RWD 28/7/82
C       FIRST DECLARE CAMAC REGISTERS
        CALL CDREG(SWITCH,1,2,1,0)
        CALL CDREG(DISPLAY,1,2,3,0)
C       CLEAR DISPLAY
        CALL CFSA(9,DISPLAY,DATA,Q)
C       READ SWITCH REG STOP IF = 0
10      CALL CFSA(0,SWITCH,DATA,Q)
        IF(DATA.EQ.0) GO TO 100
C       WRITE TO DISPLAY, THEN LOOP
        CALL CFSA(IG,DISPLAY,DATA,Q)
        GO TO 10
100     WRITE (6,200)
200     FORMAT("BYEE, HAVE A NICE DAY")
        END
```

Fig. 63 "FORTRAN" program using ESONE/NIM standard subroutines

```
10      REM SIMPLE CATY PROGRAM. RWD 28/7/82
20      SWITCH = 1,2,1,0      ; B,C,N,A OF SWITCH REGISTER
30      DISPLAY = 1,2,3,0     ; B,C,N,A OF VISUAL DISPLAY
40      CL1 DISPLAY           ; CLEAR DISPLAY
50      RD1 SWITCH, D         ; READ SWITCH REGISTER
60      IF D = 0 GO TO 90     ; EXIT LOOP IF SWITCH VALUE = 0
70      WT1 DISPLAY, D        ; WRITE VALUE TO DISPLAY
80      GO TO 50              ; CONTINUE LOOP
90      PRINT "BYEE, HAVE A NICE DAY"
100     STOP                  ; FINISHED
```

Fig. 64 A CATY program

```
10      DIM D(256), B(256)
20      DIM S(2) ; STATUS ARRAY
30      CALL INIFBS ; INITIALIZE FASTBUS
40      FOR I = 1 TO 256 ; FILL ARRAY
50      LET D(I) = I
60      NEXT I
70      CALL FWDB(D(1),10,0,256,'10,S(1)) ; BLOCK WRITE
80      PRINT "STATUS",S(1),S(2)
90      CALL FRDB(B(1),10,0,256,'10,S(1)) ; BLOCK READ
100     FOR I = 1 TO 256
110     IF D(I) ≠ B(I) PRINT "MEMORY ERROR"
120     NEXT I
```

Fig. 65 FASTBUS subroutine calls, a memory test using block transfer Write and Read

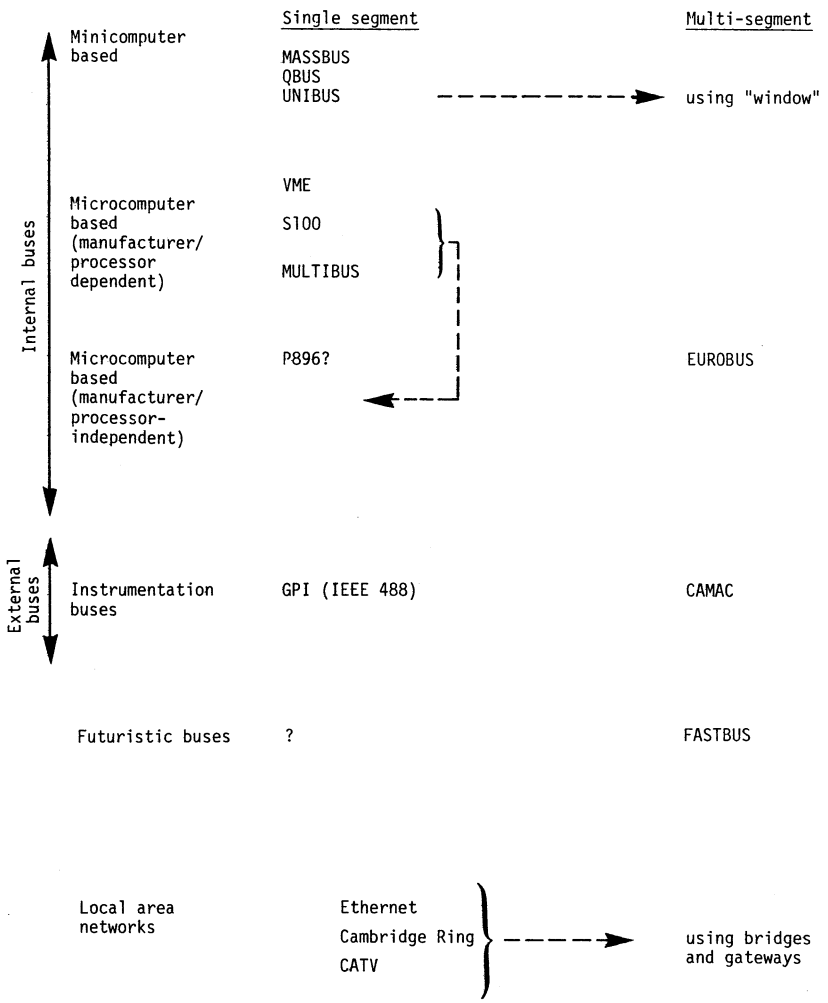


Fig. 66 A classification system for buses

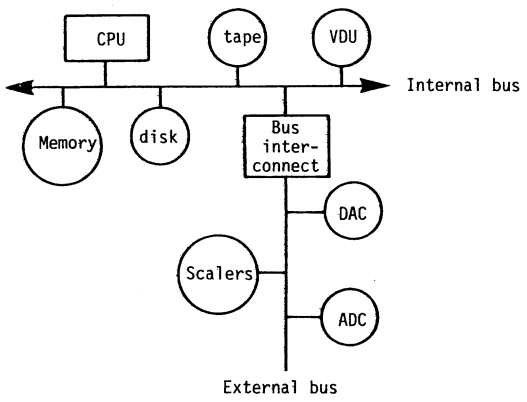


Fig. 67 Internal and external buses

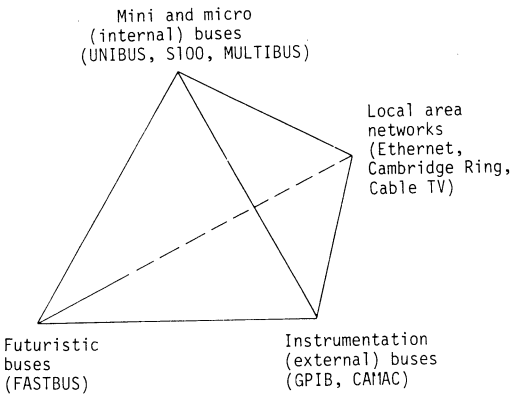


Fig. 68 Classification and interrelation of different types of bus

take into account development, maintenance, and software costs. Note that whilst hardware costs have fallen dramatically over the last decade, software costs, being labour intensive, have risen, are rising, and will continue to rise.

Figure 66 shows a classification system for buses. Notice that I have stressed the difference between buses that support the interconnection of multiple segments in a standard way and those that do not. I think this is an important distinction. Minicomputer buses are nearly always associated with one type of processor and one manufacture, although plug-compatible bus peripherals are often made by other manufacturers. Microprocessor buses may also be tied to one processor and one manufacturer. However, there has been a trend for some buses to evolve into *de facto* standards which can support a wide range of processors from a variety of manufacturers. The S100 and MULTIBUS are examples of this type of bus. There are also microprocessor buses that are designed from the start to be truly processor- and manufacturer-independent. The proposed P896 bus is such a bus, as is EUROBUS.

Minicomputer and microcomputer buses can be described as internal. They tend to be part of the basic building blocks of a computer system, hooking together CPU, memory, and peripherals such as disks, tapes, and terminals. CAMAC and the GPIB, I classify as instrumentation buses, i.e. buses to allow the attachment of ADCs, DACs, scalers, etc., to a computing system. They are external buses — external in the sense that they are one step further away from the CPU and memory (Fig. 67).

The FASTBUS standard is still in its infancy; yet it is, in my view, one of the most interesting buses around by virtue of its multi-segment flexible architecture, sophisticated protocols, and speed. I have therefore classified it on its own as a "Futuristic Bus", capable of fulfilling a very wide range of applications over the coming years.

Finally, I have included local area networks in my classification — Ethernet and Cambridge Ring are after all part of the global bus family.

The problem for any systems engineer is to know which bus or buses to use in his particular application and how to interconnect different buses; remember there is no shortage of choice. This dilemma can be summarized by Fig. 68 and my closing words: "Buses are plentiful, but be sure to catch the one(s) you need."

Acknowledgements

My thanks to Mrs. Wakley and co-workers for their usual and much appreciated improvements to my spelling, grammar, and style. The help of Emily is gratefully acknowledged.

References and Notes

The following is a list by subject.

CAMAC Hardware

1) CAMAC tutorial articles, published by the U.S. NIM Committee, TID-26618.
2) H.J. Stuckenberg, CAMAC for Newcomers, CAMAC Bulletin No. 13, supplement A, 1975.
3) F. Bal et al., What is CAMAC? CERN-NP CAMAC Note 45-00 (1973).
4) P. Clout, A CAMAC Primer, Los Alamos Report LA-UR-82-2718, 1982.

CAMAC Software

1) Subroutines for CAMAC, published by the ESONE Committee, ESONE/SR/01.
2) ESONE/NIM standard CAMAC subroutines, CERN DD/OC/80-4 (1980).

CAMAC Miscellaneous

1) Compatible Extended Use of the CAMAC Dataway. Draft Proposal (COMPEX). For latest information contact C.R.C.B. Parker, ISR Division, CERN.
2) System Crate Catalogue. Available from Fisher Controls, New Parks, Leicester, U.K.

FASTBUS

1) FASTBUS Modular High Speed Data Acquisition System for High-Energy Physics and other Applications. U.S. NIM Committee, 7 June 1982. For the latest version of the specification, contact: Louis Costrell, Chairman NIM Committee, United States Dept. of Commerce, National Bureau of Standards, Washington D.C. 20234, U.S.A.
2) E.M. Rimmer, An Introduction to the FASTBUS system, Internal Report CERN DD/80/27 (1980). (Updated several times over the last two years.)
3) P. Ponting, Old and New Standards for Data Acquisition: CAMAC and FASTBUS. CERN EP Report (1982).
4) R.W. Downing and M. Haney, IEEE Transactions on Nuclear Science, Vol. NS-29, No. 1, February 1982. This volume contains several other interesting articles on FASTBUS.

UNIBUS, QBUS and other Digital Equipment Corporation Bus Systems

1) PDP-11 Bus Handbook, available from your friendly DEC sales person.
2) Computer Engineering, A DEC View of Hardware Systems Design. Again available from DEC but quite expensive if they make you pay. An excellent book for anyone interested in computer hardware. See chapter 11 on Buses. Chapter 20 on Multi-Microprocessors is also relevant.
3) Introduction to Local Area Networks. A recent offering from DEC and LANs. Probably they will give you one free.

Other Bus Systems and General References

1) Standard Specification for S-100 Bus Interface Devices. IEEE Computers, July 1979.
2) Proposed Microcomputer System 796 Bus Standard. IEEE Computer, October 1980 (MULTIBUS).
3) Proposal for a Small System Standard, ES3, submitted by Small System Study Group to ESONE AGA, Zurich, 1981. SS/DOC/40.
4) Tutorial Description of the Hewlett-Packard Interface Bus. Available from Hewlett-Packard (GPIB, IEEE 488).
5) IEEE-P896, A Proposed Standard Backplane Bus for Advanced Microcomputer Systems Draft 4.1.2, February 2, 1982. Latest information from: Professor J.D. Nicoud, LAMI-de-EPFL, Bellerive 16, CH-1007 Lausanne, Switzerland.
6) The VME bus, Specification Manual. I got my copy from Motorola. You can also ask your Mostek or Signetics/Philips salesman.
7) Understanding bus basics helps resolve design conflicts. Articles in EDN, May 27 till June 24, 1981. Read this to get an idea of how many different buses there are around.
8) Bus standards enrich microcomputer board variety. Electronic Design. April 15, 1982.
9) Sol Libes and Mark Garetz, Interfacing to S-100/IEEE 696, published by Osborne.
10) Harold Stone, Microcomputer Interfacing, published by Addison-Wesley.