SLAC-PUB-3326
April 1984
(N)

COMPUTER BUSES: A TUTORIAL

David B. Gustavson*

Stanford Linear Accelerator Center

Address:

Stanford Linear Accelerator Center

P. O. Box 4349, Mail Stop 88

Stanford, CA 94305

*Work supported by the Department of Energy under contract number DE-AC03-76SF00515.

(Submitted to IEEE Micro)

INTRODUCTION

Computer buses are the communication paths between the various parts of a computer system. Most computers, even small ones, contain several different buses, each optimized for a particular kind of communication. Most buses are private, hidden within an integrated circuit or confined to a single circuit board. Some buses appear in accessible places and are described in the computer's documentation so that optional equipment or features can be easily added to the system. Some buses are not only easily accessible but follow public standards; this allows easy communication among a variety of devices made by different manufacturers.

Though there are many different buses in common use, they just represent different choices in the solution of a few basic design problems. We will not present any one bus design ("architecture") in detail but rather will describe the general problems which each bus must solve, mentioning a few common solutions for each. This should provide the reader with the necessary perspective for making sense out of the detailed documentation for any particular bus of interest. Because many readers have entered the computer field without training in electrical engineering, we will review the elementary principles as needed.

The term "bus" generally implies at least the possibility of communication among more than two devices. Connections limited to two devices are usually just called wires or traces or signals. Furthermore, "bus" usually implies parallel related connections, so that several related signals travel together along approximately the same route and at about the same time. Such parallel structures can be simulated by a sequence of signals on a single connection, which is then called a serial bus. This tutorial will emphasize parallel buses.

Today's buses are made of electrical conductors, usually copper line patterns etched on printed circuit boards, copper wires, or fine aluminum line patterns on integrated circuits. Optical communication is not yet economical for bus structures; optical signals are difficult to distribute to multiple destinations and converting between electrical and optical signals is expensive.

HOW BUSES WORK

When one device on a bus (say M3 in figure 1) wishes to communicate with another (say S5), M3 sends signals on the bus which cause S5 to respond. The particular signals which select S5 rather than some other device, say S7, are called the "address." If M3 then sends data to S5, we say M3 "writes to" S5. On the other hand, if S5 sends data to M3 we say M3 "reads from" S5. M3 tells S5 whether this is a read or a write by sending a control signal in addition to the address.

The device which initiates and controls the communication is called the "master" (hence the label M), and the responding device is called the "slave" (label S). Some devices (such as M4/S2) can act as either masters or slaves, but usually not at the same time.

What if there is another device (say M6) which wishes to use the bus? If M6 starts to put signals on the bus while M3 is using it, everyone will get confused because bus lines can only carry one signal at a time. So any device which wants to use the bus must make certain that the bus is free before putting any signals on it.

What if M3 and M6 both decide at the same time that they want to use the bus? They would each think the bus was free and would try to send their signals, interfering with each other. Any bus which can be used by more than one device has to have some mechanism which prevents more than one master from using it at a

time. This mechanism is called "arbitration." There are several good ways to handle arbitration; the choice is one of the reasons buses differ.

We have skipped over some other major problems, too. How are signals sent and received electrically? How does the receiver know when there is data to be received? How did S5 know to look for its address? How did S5 know what its address was? How are connections to the bus made?

Let us back up a bit and start at the beginning.

BUS MECHANICS

Figure 2 shows a typical arrangement for a publicly accessible bus. The "motherboard," or backplane, has copper printed circuit traces in more or less parallel lines, with connectors located at convenient intervals. The bus devices are printed circuit cards, sometimes called modules or daughter boards, which plug into the connectors in order to make contact with the backplane. In addition to signal lines, the backplane usually has heavy conductors providing the electrical power needed by the daughter boards. Several connector pins are also connected to an electrical common point, or ground, which is sometimes a broad inner copper layer (ground plane) in the motherboard, and sometimes a broad trace on the back side of the motherboard.

The connector shown is a card-edge connector made by plating gold on a finger pattern on the edge of the daughter board, so that contact springs on the motherboard make a sliding contact as the daughter board is inserted. The contact springs usually make separate connections to signal traces on both sides of the daughter board, and they are protected by a plastic housing (not shown) which guides the daughter board into proper alignment.

Two-piece connectors are becoming more popular now; in these a plastic shell holding machined contact pins is attached to the daughter board instead of relying on plated printed traces for contact. A corresponding plastic shell with machined sockets is mounted at each access point on the motherboard. Many variations

of this configuration are used, with different contact sizes and connector sizes and with sockets and pins reversed.

The primary problem of connector design is providing good reliable contact even after hundreds of uses or after sitting in a dirty and mildly corrosive environment for years, without requiring large insertion and removal forces, even if the mating pieces are not perfectly aligned or the circuit boards are not perfectly flat.

Usually the system requires the electrical power to be turned off before inserting or removing daughter boards. In multiple-processor systems, this may result in significant delay due to the need to reload and reinitialize every processor. Some systems are therefore designed to tolerate power-on insertion and removal, which is generally called "live insertion" capability. This saves the state and contents of other boards, but usually still requires the bus activity to be stopped temporarily, because it is difficult to design boards which never cause interference as they power on or off. A board which has been removed and replaced is no longer in its original state, of course, so it must be initialized and programs communicating with it may need to be restarted. In some cases this is so complex that live insertion provides no advantage.

Bus specifications normally include details of the mechanical aspects, such as board sizes, card guide size and location, permitted cable connector locations, and maximum component height on the boards. A few also specify maximum power dissipation and

provide for cooling air flow.

BUS SIGNAL TRANSMISSION

Signal transmission is very simple. All the devices using the bus are electrically connected to each signal line, and the signal lines are electrical conductors; when the master puts a voltage on a signal line, the same voltage appears everywhere along the bus and can be sensed by all receivers. The master just changes the voltage from time to time in order to send signals.

Well, that's almost correct. There are many buses which assume it is correct. But, this really isn't adequate any more, so we'll come back to it later, after discussing drivers and receivers a bit.

The circuit that changes the voltage on the signal line is called a bus driver. Any ordinary digital integrated circuit would almost do, since a digital output is always at one of two possible voltage levels. But since bus signal lines have to be shared, driven by different devices at various times, there has to be some way of disconnecting an ordinary digital output when it is not using the bus.

One way of doing that is to use a three-state driver, which has output states "high", "low", and "off". A special input to the driver is used to turn it off except when it is using the bus. The "off" state lets the output go to whatever voltage another driver determines without interference. This must be done with care, so that two devices never try to drive the line

at the same time; otherwise they will fight each other, resulting in high-current spikes and indeterminate voltage levels on the bus, electrical noise, and possibly premature component failures.

Another way is to tie the signal line to a voltage source through a resistor and use drivers which either let the line float or force it to a particular voltage level. If no driver is turned on, the line will float to the signal level set by the voltage source. If two drivers turn on at once, they both force the line to the same level so they share the load and there is no conflict. Drivers like this are called "open collector" drivers in the TTL families of integrated circuits, or "open drain" drivers in MOS circuits. All ECL circuits have this kind of output, called "open emitter."

This scheme not only eliminates the possibility of electrical conflict on the bus, but makes possible a very useful kind of logic, called "wire-OR" (or "wire-AND", depending on the assignment of voltage levels to logic values). If any driver or combination of drivers turns on, a signal appears which is the logical OR of all the driving signals. This is useful in solving the arbitration problem, as we shall see later. Most buses use wire-OR for at least a few lines; some allow its use on all signal lines.

Receivers are circuits which compare the signal voltage at their input with a standard value, which is usually set by internal circuitry, and then generate a logic level output suitable for use on the rest of the daughter board. Transceivers

contain a receiver and a driver internally connected to a common pin.

Now let us return to the description of signal propagation given in the first paragraph of this section. The fundamental problem with this description is that it takes no account of the time it takes for a signal to travel. No signal can travel faster than the speed of light in empty space, about 300 millimeters per nanosecond; on a typical bus made of practical materials the signal speed will be a fraction of this.

A more realistic description would start with the bus line held at one signal voltage level by a resistor tied to a voltage source at the end of the line. When a driver turns on somewhere along the line, it pulls the line to the other signal level at the point where the driver is attached. The resulting voltage change travels in both directions away from the driver, moving a bit slower than the speed of light, until the whole line is at the new signal level.

But this is still not quite realistic enough to explain what happens in real buses. Electrical charges are distributed along the signal line in proportion to the voltage. They repel one another, but are attracted to charges of opposite polarity which may be nearby. More voltage helps them crowd together, but the precise number and distribution depends on the detailed shape of the line and on the distribution of electrical charges in the neighborhood of the line (i.e., the dielectric constant of the circuit board, and the configuration of neighboring conductors).

The charge present per voltage applied is the capacitance of the line, and the capacitance per unit length varies from point to point depending on the local shape and environment of the line.

When a driver turns on and changes the voltage, the charges move, making a current. The moving charges create a magnetic field which affects the motion of nearby charges. The net effect, called inductance, tries to keep the current from changing. The inductance depends on the precise shape of the signal line and nearby materials, and the inductance per unit length varies from point to point in a different way than the capacitance does.

The ratio of the signal voltage to the resulting signal current, called the characteristic impedance (Z_o), depends on the inductance per unit length (L_o) and the capacitance per unit length (C_o):

$$Z_o = \sqrt{L_o / C_o} \qquad (1)$$

How much current flows when the driver turns on? This depends on the characteristic impedance and varies from point to point. The speed of the signal travelling along the bus depends mainly on the effective dielectric constant of the bus; the propagation delay per unit length is:

$$T_{pd} = \sqrt{L_o * C_o} \qquad (2)$$

Both answers are simple for an ideal signal line

(transmission line) like the coaxial cable shown in Figure 3A. A real line, however, looks more like Figure 3B, with strange projections here and there due to connectors, circuit traces, and components on connected daughter boards.

As the signal travels along a real line, it encounters regions with different propagation velocities and impedance. Wherever the impedance changes, the signal cannot proceed unchanged because the relationship of current and voltage required by the line no longer matches the signal. Part of the signal continues onward, but the rest is reflected back the way it came; part of that may be subsequently re-reflected and so on, leading to a very complex signal on any real bus. When the signal reaches the end of the line, it will all reflect back unless it is absorbed by proper terminating resistors. If a resistor is placed at the end of the line, with a value equal to the line impedance, the signal will be absorbed without reflection. For a bus these termination resistors have to be located at each end of the line, and they determine how much current flows to the driver after the reflections have died out. Because real line impedances are never known precisely and depend on which boards are connected where, the resistors never perfectly match the line and some reflection always occurs.

Furthermore, the electric and magnetic fields produced by signals changing on any one signal line create signals in neighboring lines, causing crosstalk. These problems all become more manageable if the bus is designed to have a low impedance and to have a grounded conductive plane near all signal lines,

usually a buried layer inside the motherboard.

Because of imperfections in the lines, signal edges change shape as they propagate, which implies that pulses change shape as well. Any bus has a minimum pulse width which can survive propagation from end to end. This puts an ultimate limit on the number of pulses per second, or bandwidth, which the bus can support. Normally other factors limit the bus throughput before the bandwidth limit is approached, but one possible exception will be mentioned later.

Note that the driver in effect sees two lines at once, one going in each direction, so it has to supply twice the current needed by one line. The impedance seen by a driver on a practical bus is often less than 20 ohms, which results in currents of 150 milliamperes for 3-volt signals. This is well beyond the data sheet ratings for any bus driver generally available today, though real devices happily exceed their rated output if shorter lifetime, lower reliability, and degraded signal levels can be tolerated.

Current does not flow only in the signal lines. Signal currents complete their circuit through the ground pin of the driver. When all the signal lines are active at once, this ground current can be quite large. Worse yet, it is not constant, but has very-high-frequency components as the drivers turn on and off rapidly. The resistance and inductance of the ground plane allows the ground pins of the circuits on the daughter board to have different voltages from each other and

from those of other boards. This in turn may cause receivers to evaluate signals incorrectly and can cause logic circuitry on the boards to malfunction. This ground noise is much easier to avoid by design than to fix later. The solution requires more than just good ground planes on the daughter board and mother board. There must also be many well-distributed good connections between the mother board and daughter board ground systems.

For high-speed buses, there should be about one ground pin per four signal pins. Furthermore, the daughter board should be designed so that the ground current from a given driver flows to a connector pin near the corresponding signal pins. The mother board ground is usually a buried copper layer in a multi-layer printed circuit; clearance holes around the connector signal pins prevent short circuits to this ground. It is important to control the sizes of those holes, so that the ground on one side of the connector is connected to the ground on the other side in many places along the length of the connector and not separated by a row of merging holes. The connector should be wide enough so there will be room on the daughter board to allow the transceivers to be located near the connector, minimizing trace lengths which disturb the bus.

Until recently, most bus designers have handled these problems rather badly, aiming at high-impedance lines on the motherboard because they seem easier to drive with available circuits, ignoring the effects of connector pins and circuit traces on connected daughter boards, neglecting ground planes, and failing to provide the necessary distribution of ground pins

across the connector.

If buses have been so badly designed, how do they work so well? Generally they are saved by slowing them down until they work. If the signals could be made to change slowly from one level to the other, the reflections would become small and the crosstalk insignificant. Unfortunately, most drivers are too fast for the buses they drive, so the signals look terrible. These buses are usually saved by introducing delay in the system, often referred to as the "bus settling time," so that the signals become stable before they are used. Sometimes there is enough delay inherent in the kind of circuitry used, but explicit delays are often added for this purpose. Synchronous buses, which use a central clock to time every transition on the bus, can add delay easily by slowing the clock: all signals can be made to change at one clock edge and not be looked at until the other edge, providing enough delay for propagation and settling. Asynchronous buses must either solve these problems at their source or add artificial delays to nullify their effect.

Slowing the receiver circuits is another help. Sometimes low-pass filters are introduced to make the receivers insensitive to reflections and other high-frequency noise. The MITS Altair bus, for example, used high-power drivers and low-power receivers because the data sheet numbers implied an enormous fan-out, so the bus should have allowed a large number of boards to be plugged in. The signals looked terrible as a result of the fast drivers and awful backplane design, but the low-power receivers were slow enough to reject much of the junk so the bus usually

worked, especially in short-length versions. Unfortunately, there was no complete specification for the bus (before IEEE 696 appeared), and many boards that were made for it worked badly, sometimes because they used receivers that were too fast.

As microprocessor speeds have increased, these slowed buses have become less and less acceptable. As speeds increase, the bus has to be more carefully designed in order to solve these problems. Modern bus designs are pushing fundamental limits. Fast buses have to limit the length of traces on daughter boards, reduce the capacitance of transceivers, connectors, etc., as much as possible, provide good ground planes with plenty of ground pins, and specify transceivers which can handle the real problems of imperfect transmission lines.

The TTL bus drivers and receivers which have been used to date are not very satisfactory. Fastbus, a high-performance bus recently developed under the auspices of the United States Department of Energy, changed from TTL to ECL to solve this problem, and IEEE P896 has specified a new transceiver which reduces capacitance, reduces signal voltages and edge speeds, and rejects noise in the receiver, while using TTL power supplies and signals on the daughter board side. These transceivers and the bus-driving problem are described in more detail in a companion article by R. V. Balakrishnan.

BUS ARBITRATION

Now that we understand how signals are sent, let us look again at the arbitration problem. Somehow, any device which wants to use the bus must get permission first, to avoid conflicts between two or more devices trying to talk at once.

Perhaps the simplest method uses special wiring on the backplane to form a "star" connection, as shown in the lower part of Figure 4. A bus request signal is connected from each device to a central arbiter. A second star connection carries a bus grant signal back to each device. Thus each device has a private two-way connection with the arbiter. The arbiter may use any method it likes to decide who gets the bus. This method is very versatile, allows any conceivable allocation scheme to be implemented, and is very fast and efficient, but it also has some serious disadvantages. The special wiring on the backplane is expensive. Information about the arbitration is not present on the bus, so bus monitoring for diagnostic purposes is made more difficult. Access to the arbiter for changing the algorithm or initializing it via software can be difficult unless the arbiter is accessible from the bus, requiring an expensive connection to a daughter card.

The next method also uses special wiring, but a much cheaper kind, called the "daisy chain," which is shown in the central part of Figure 4. A daisy chain is a pair of pins in each connector, wired so that a signal enters the daughter board on one and returns to the bus on the other. This allows a series

connection of logic from each daughter board along the backplane. This connection is the basis for a very common kind of arbitration. It is used in conjunction with a wire-OR line, which is connected to one end of the daisy chain. When any device wants to use the bus, it drives the wire-OR "bus request" line, and looks for a signal on its daisy-in pin. Each device passes the daisy-in signal to the daisy-out pin, unless it wishes to use the bus. Thus the device nearest the end of the daisy chain connected to the bus request line has the highest priority, and always gets the daisy-in (bus grant) signal when it asserts bus request. If it does not want the bus, it passes the grant along, and so does each other device in turn, until a requestor sees it and refuses to pass it further. Some further rules are needed to prevent a high-priority device from taking the bus away from a lower-priority device in mid-cycle. This can be achieved by synchronizing request assertions with other bus activity.

The daisy chain is very economical, but has several disadvantages. It may be slow, because signals have to travel through logic on each daughter board. Every connector has to have a daughter board or a dummy board plugged in to connect the daisy chain pins, or the grant signal is blocked and the system fails. And, as for the star arbiter connection, there is very little information about arbitration on the bus so diagnosis and monitoring is difficult.

A new scheme which has gained much popularity in recent years was invented by someone at Computing Devices of Canada (UK patent specification 1,099,575, filed in 1966, inventor's name not

listed) and rediscovered by Matthew Taub of IBM in 1975 (see companion article in this issue). It was rediscovered again by Leo Paffrath at SLAC for the Fastbus design project, moved from there to IEEE 696 and IEEE P896, and now also appears in the TI NuBus and Intel's Multibus-II. Taub has recently developed an enhancement to this scheme for P896 which provides totally distributed control (requires no central timer) and is independent of the speeds of the competing daughter boards. The other buses using this method all rely on timing generated by one particular device on the bus.

Taub's method uses only bussed signal lines, so all information about arbitration is present everywhere on the bus; it has no position dependence, requires no special backplane wiring, and is relatively efficient. Depending on the goals of the implementation, it uses two to four wire-OR signals for timing and control and four to seven wire-OR lines for the actual arbitration bus. The basic idea is that every device which wants the bus tries to put its own priority number on the arbitration bus, removing its less-significant bits if it sees a higher number present; after some delay only the highest priority number remains. The device which sees its own priority then controls the bus. After it removes its number, the next-highest wins, and so forth. Adding a simple rule (sometimes called "fairness") which prevents new requests while other requestors are competing, produces a system which guarantees every applicant a turn and prevents the highest priority device from winning the bus all the time.

A degree of fault detection can also be easily added in this scheme, by making the priority number one bit wider and assigning only odd-parity numbers, i.e., numbers with an odd number of bits asserted. The winning priority will always appear with its own parity bit and simple logic can then check for valid parity. This improves the chance of detecting a failed driver or bad connector before it causes too much chaos in the system.

The real purpose of "priority" in modern multiprocessor backplane buses is to break ties between simultaneous requests for use of the bus. In a system with the "fairness" scheme just described, bus priority has little to do with which device gets the most access to the bus, and nothing to do with job or task priority. Confusion between bus priority, which is important for nanoseconds or microseconds, and task priority, which is important for milliseconds or seconds, has been common in bus design. In single-processor buses, however, the processor may be given the lowest priority so that it takes only the bus cycles left over after the needs of disk transfers or other I/O have been satisfied.

BUS SIGNAL ALLOCATION

Now that we have a way to determine which of several devices may use the bus, we need to look more carefully at what happens while the bus is being used.

The first step the master takes is to assert an address on the bus, which selects one of the slave devices and establishes a connection between master and slave. The address usually contains additional information which specifies a particular part of the slave. For example, the more significant bits of the address might determine which of several memory boards is to respond, while the less significant bits determine which word in that board's memory is sought.

There may be more than one kind of address, with additional control signals to specify the address type. Frequently there is a memory address and an I/O port address, similar to the scheme used in Intel's 8086 family of microprocessors. Processors which do not make such a distinction can still use such a bus by adding hardware to translate a certain range of processor memory addresses into bus I/O addresses. Some buses have other kinds of addresses as well. Devices may have multiple address ranges as well as multiple kinds of addresses. The primary requirement is that addresses must be uniquely assigned, so that only one slave device responds to a given address of any kind.

Some systems extend this to allow broadcast addresses, either special address values or special address kinds, which select

multiple or all slave devices. Generally this kind of addressing is used for broadcasting information from the master to all the slaves, but a few systems permit the corresponding read operation as well, sometimes called "broadcall." Broadcall results in the bit-wise OR-ing of the information from all addressed slaves.

The maximum size of the address is determined by the number of signal lines allocated for the purpose and is one of the most fundamental properties of any bus. The address size often limits the amount of memory which can be installed in a system, because each memory word usually requires a unique address in order to be useful. The address size is usually given as the number of bits or signal lines; these lines are often called the address bus.

The width of the data path, or data bus, is the next most important parameter of the bus. Most buses now use some multiple of eight lines (an integral number of bytes) for the data width. A data item which uses the full bus width is usually called a "word," but sometimes the architecture of a family of processors defines the size of the word instead.

Most buses use addresses which specify a particular byte, so a series of transfers on a bus with multi-byte width will have successive addresses incremented by the number of bytes in each transfer. Such buses usually provide a way to transfer information on a subset of the full bus width. This is especially useful for writes, where it may be convenient to change a particular byte in memory without affecting its neighbors.

Other buses address items of full bus width only, so each transferable item has one address. Any transfers of less width are taken care of by the master, by first reading the whole word into the master, modifying the appropriate part, and then writing the whole word back to the slave.

When a bus transfers partial-width items, it may either leave them on the same signal lines they would have occupied if they were part of a full-width transfer ("unjustified") or it may move them so that they occupy the least significant signal lines ("justified"). Justified buses make it less expensive to start with a narrow subset of a wide bus, adding extra justification hardware to the wide boards when they are added in the future. The disadvantage is that future systems with no narrow subset boards still need the justification hardware on every board, though it no longer serves a useful purpose. A more serious disadvantage is that a justified bus does not work well with all computer architectures, reducing its usefulness as a general-purpose interface. Compare the NuBus (unjustified) and Multibus-II (justified) specifications to understand this better.

Some buses include extra lines for error checking. One extra line for each byte of data allows byte parity checking along with simple partial-word transfers. A few more lines would permit using an error detection and correction code, so that badly received data could be repaired by the receiver. This complicates partial-word transfers, however, because such a code becomes more efficient as it applies to more bits and so it usually would be applied to the full word rather than byte by

byte. Error checking also requires a mechanism for telling the sender that the data arrived in bad condition, so it can be corrected by being sent again. Error checking may also be applied to addresses and other parts of the bus.

The address width and data width need not be related. Common address-data combinations are 16-8, 16-16, 20-8, 20-16, 24-16, 24-32, and 32-32.

The bus may have separate signal lines for address and data, or it may use the same lines at different times, which is called multiplexing. Multiplexing slows a bus less than one might expect because data is not useful until after addressing is complete, especially in case of a read, which requires an additional wait for the access time of the slave. Multiplexing is especially attractive for wide buses because it saves so many lines, drivers and receivers, which reduces system power consumption and noise and frees circuit board space as well.

In addition, control signals are needed to specify whether a read or a write is to occur, how wide the transfer is and which bytes are valid, which type of address is being used, and perhaps which protocol is to be used. Two to eight signals, sometimes called the control bus, are usually used for these purposes.

From one to four lines are often used to allow the slave to respond with error codes or status information, on a status bus.

The arbitration lines, described in the previous section, may

add three to 11 more signals. Some buses include another set of lines for interrupts, which are signals from slave devices requesting service from a particular processor. Interrupts can be handled similarly to arbitration, since the problem is deciding which of several interrupters should receive service first. Daisy chains and central arbitration circuits are common solutions. Buses which are designed to handle multiple processors, however, tend to eliminate special "single-processor" interrupt mechanisms from the bus, and require any device which requests service from another to write a request to it through the normal bus protocols. This simplifies the bus, and eliminates the need for dedicated mechanisms to specify which processor is to handle the interrupt service.

It is becoming common practice to include one to four lines for connection to a serial local network. Because serial communication is necessarily slower than parallel bus communication, and networks can be implemented either way, there has been disagreement about the usefulness of such a serial link in a fast parallel backplane. Some think it should serve as a full-function path providing redundancy in case of failure of the parallel bus. Others would assign it special functions such as interrupt handling or task priority sorting. Still others consider it an independent resource which might be used for communication with local-network peripheral devices. Some implementations limit the serial connection to the backplane, while others allow it to link multiple backplanes or even extend over kilometers. Fastbus introduced the concept because of a clear need for communication among

diagnostic devices on different backplane bus segments, so that broken interconnect devices could be detected and diagnosed; once implemented, however, nothing prevents its use for other purposes.

Several buses now include four or five position-encoded pins on the connector, so each connector presents a unique code or slot number to the daughter board. This code can be used for initializing each board with unique addresses or priority codes after a system power-on or reset.

The final group of lines is often the most important, requiring the greatest care in bus design. These are the timing lines: strobes, syncs, and clocks. This group usually accounts for two to six lines, depending on the bus protocol.

When power supply pins and ground pins are added to the above list, the need for big connectors is clear; it is barely possible to fit a 32-bit bus on a 64-pin connector. Some buses use connectors with hundreds of pins, including special-purpose sub-buses, free lines for private communication among parts of multi-board subsystems, many paralleled pins for passing heavy power-supply current, etc. Table I summarizes the connector pin allocation.

Line Names	Typical Number of Pins	
Address	16-32	} May be combined in
Data	8-32	} multiplexed buses.
Arbitration	3-11	
Control	2-8	
Status	1-4	
Clocks, strobes	2-6	
Serial Network	1-4	
Position code	4-5	
Ground	2-20	
Power	2-20	

Table I: Typical Allocations of Pins to Bus Functions

BUS PROTOCOL

We have talked about addressing and transferring data, but have not really explained how those operations work. When a master puts an address on the bus, it is likely that not all the bits will arrive at the same time. Some may travel longer paths on the board; some may travel through mapping hardware which translates processor addresses to bus addresses; some lines, drivers, and receivers are faster than others, which causes some bits to arrive before others, producing an effect called "skew." All slave devices need to know when the address becomes valid, so that they can check whether to respond to it.

The situation for data transfers is more complicated, because data can flow in both directions in most buses. In the case of reads, there is a delay while the slave searches for the requested data, so the slave must be the one to signal when the data is valid. The system must allow for bus skew in data transfers as well.

The method the bus designer chooses for signaling the validity of address, data, commands and status is called the "bus protocol."

There are two major classes of protocols, synchronous and asynchronous. Synchronous protocols time all signals relative to a system clock, while asynchronous protocols provide separate validity signals for each sub-bus. In actuality, every protocol includes some synchronous aspects and some asynchronous

ones, but nevertheless the style of the two types is quite different.

Synchronous protocols tend to use fewer bus lines, and are simpler to understand, implement, and test. But they are not as flexible: locked to a particular maximum clock rate, and thus tied to a particular level of technology, they cannot take advantage of advances in performance which occur after the design is frozen. Now that buses are approaching their ultimate physical speed limits, however, that disadvantage is less serious than it used to be.

Asynchronous protocols are self-timing, so that a mixture of fast and slow devices, using old and new technology, can share a bus. Bus speed adapts automatically to the requirements of the particular devices which are communicating at the moment. Thus, as technology improves, faster devices can be added to the system and the user will benefit from the resulting performance improvements as they occur. It is not necessary to replace all the old devices to speed up the system, as it is with a synchronous system. The price for this is some increase in complexity.

Fashion has favored asynchronous buses in recent years, but some of the latest high-performance designs are synchronous. With present technology, synchronous designs run a bit faster than asynchronous designs, and these buses are now so near their ultimate speed limits that any future speed increase which technology might give to asynchronous systems can make them only

slightly faster than present synchronous ones.

Synchronous buses have a central clock oscillator which drives a bus signal line to distribute timing information throughout the system. Figure 5 shows a read operation using a simple protocol, essentially that of the TI NuBus. The rising edge of the clock is the time when bus signals make their changes, and signals are assumed to be valid, i.e., to have successfully propagated throughout the system, just before the next rising clock edge. NuBus uses an asymmetric clock, so that the falling edge serves as the time reference for valid signals, but that is a convenience rather than an essential feature. Other systems use delay from a clock edge instead.

A start signal marks the presence of the address and control information on the multiplexed bus lines. When the slave recognizes its address and finds the requested data, it puts the data and status on the bus and marks their presence with an acknowledge signal.

A write operation would look similar, the only difference being that the data would be supplied by the master, starting the next clock cycle after the address, and would remain on the bus until the acknowledge and status are sent by the slave.

In synchronous systems, the speed of travel of signals does not appear explicitly in the protocol, but must be considered in the bus design. The clock usually propagates along the bus at normal signal speed, though it is possible with some cost and

effort to deliver simultaneous clock signals at every bus connector. The clock frequency must be chosen so that there is time for signals to flow from any starting point to every other point well before the end of the clock cycle, allowing for differences in clock arrival time as well. Thus, shorter buses can be designed to run faster, and simultaneous clock distribution allows higher speed than a centrally located clock source, which in turn is faster than a bus with a clock propagating from one end.

Note that all operations have both read and write aspects, and there is a signal of validity for each direction. Control and address always flow from master to slave, and status flows from slave to master. Data can flow either way.

In asynchronous protocols, every set of signals which is put on the bus is accompanied by a corresponding timing signal, called a strobe or sync signal. Timing signals generated by the slave are often called handshakes or acknowledges.

Figure 6 shows an asynchronous read operation, using a protocol similar to that of Fastbus or P896. First, the master asserts the address and control information on the bus, waits for a skew time and then asserts the address sync to signal validity. The slaves look at the address and check whether to respond. The one which was addressed then responds with status followed by an address acknowledge. When the master sees the address acknowledge, it knows that a connection has been established and it can check the status. The address is no longer needed on the

bus, though the slave may have saved a copy of all or part of it.

The master then changes the control information, waits a skew time, and asserts the data sync. If this were a write operation, the master would assert the data at the same time as the new control information. However, in the case shown, the control information tells the slave that this is a read, and when the slave finds the data it puts it on the bus with new status information, and asserts the data acknowledge. When the master sees the data acknowledge, it reads the data from the bus and removes data sync to indicate that it is finished with the data. In this simple example, it also removes address sync; however, in more complex examples, address sync could remain in order to maintain the connection across several data cycles. When the slave sees data sync removed, it removes data and status information and removes data acknowledge. Address acknowledge is also removed, in response to the removal of address sync, restoring the bus to an idle condition.

The timing diagram of Figure 6 shows the effects of bus signal speed and device response speed explicitly, since they are fundamental parts of the operation of an asynchronous system. As in the synchronous case, every operation has aspects of both read and write. In effect, control is written and status is read; data is timed and driven like control or status as appropriate.

Part of the protocol has to inform the arbitration circuitry when the bus is in use. In the synchronous example, the bus is

busy from start to ack, and no special busy signal is needed. In the asynchronous example, assertion of either the address sync or address acknowledge indicates that the bus is busy.

Note that the asynchronous system is fully handshaken, i.e. in every case both parties agree before any information is removed from the bus. Thus, one of them might be built with very fast circuitry and the other with very slow, yet they could communicate successfully. This is one of the nice features of asynchronous systems, because it allows gradual replacement of parts of a system with newer and faster boards, with a resulting gradual improvement in performance.

The synchronous system is partially handshaken, in that the slave can take as many complete clock cycles as it needs to find the requested data before it responds with ack. On the other hand, there is an implicit requirement that the slave either finish with the address and control information in a fraction of a clock cycle, or quickly copy it before it disappears. Similarly, the master had better be able to accept the read data within one clock, or it will be lost. This could be a problem if the master used dynamic memory and happened to be refreshing it when the data arrived. Extra buffer memories are normally used to handle problems of this sort. Notice also that if a synchronous slave is just a bit too slow to be able to respond in one clock cycle, the operation is lengthened by a full clock cycle. This can be a very significant disadvantage compared to asynchronous systems, which use only as much time as they need.

In fact, asynchronous systems are not really completely handshaken: in order to prevent the system from waiting forever for a response which will never come, due to programming error or hardware failure, timeouts are always provided which cause the operation to abort after a reasonable wait with no response. This possibility introduces new problems--for example, what if the master times out just as the slave sends the acknowledge? Some circuit has to decide whether the acknowledge arrived in time and, if not, how to get the bus cleaned up again. Data cycle timeouts, which usually mean broken hardware, are rare, so the timeout can be set ridiculously long without hurting the system. Address timeouts are more frequent, however. They occur when a program is initializing the system and is searching to find what devices are present, for example, and often when a program error results in a bad address. Therefore, the bus specification usually sets a fairly short timeout for addresses, requiring all slave address decoding to be fast enough to meet that fixed time. Thus, practical asynchronous systems have certain synchronous aspects as well.

Bus skew, which is the worst-case difference in propagation time between the fastest and the slowest signal line, has to be provided for in both kinds of systems. It is included in the determination of the fastest safe clock rate for synchronous systems and can be ignored elsewhere in the design. Asynchronous systems, however, must allow for skew in every handshake in every device. When the master asserts a sync signal, it waits a skew time after the data is on the bus, so that when the slave sees the sync it can assume that the data is good. The slave may also

have internal data path skew; this can be compensated for by adding additional delay before using the sync signal it received. When the slave returns data to the master, it must wait a skew delay after asserting the data before it asserts the acknowledge.

Actually, the skew delay can be accounted for by either master or slave or a combination, as long as enough delay is provided in total. The P896 bus provides for skew as described above, but Fastbus puts all skew responsibility in the master; the slave asserts the acknowledge at the same time as the returned data, and the master waits a skew time before looking at the data. Skew delay is system dependent, technology dependent, and a property of the type and length of the physical bus implementation. Fastbus expects to have various implementations, including cable buses at least 10 meters long as well as short backplanes; since Fastbus emphasizes data acquisition, there should be many more slaves than masters. By putting all skew responsibility in the master, Fastbus reduces the number of devices which have to be modified when the bus properties change. Slaves are still responsible for taking care of their own internal skew problems. P896, however, is solely concerned with a standard backplane bus with known properties and expects to have mostly master devices in a multiple-processor system. Thus P896 uses a symmetric approach in which each sender of information accounts for the bus skew itself.

These simple protocol examples do not exhaust the possibilities, especially for multiplexed buses, which pay a time

penalty for every address cycle. For example, it may be useful to have a read-modify-write to the same address. The details of these more complex operations follow the same principles we have discussed above, but vary from bus to bus. One protocol deserves special mention, however, because it has broad implications for system design.

Block transfers are a single address followed by multiple data cycles (either read or write, but normally not intermixed.) Usually the address is presumed to start from the given initial value and increment after each data cycle. Transfers to I/O devices or FIFOs, however, may not have any increment implied. The difference mainly affects slave internal design, but it also has implications for the master if error recovery is needed--what address does the master use if it wants to repeat the transfer of the tenth word in a block because of a parity error? Both synchronous and asynchronous systems may implement block transfers. Asynchronous systems can make yet another improvement, however.

The data transfer of Figure 6 shows that the final edge of the data sync and of the data acknowledge are not really transferring data but provide time to turn off drivers and let the signals disappear, cleaning up the bus. This is necessary if the bus direction is to change from read to write, for example, but not if a block of data is to be sent in the same direction. Therefore, some protocols allow block transfers to go at twice the rate of single transfers by using both the leading and the trailing edges of the data sync and acknowledge for data

transfer. The bus still has to be cleaned up at the end of the block, however, which may require an extra cycle. The handshake is still complete, as either party can slow the transfer whenever necessary. Only the unneeded bus cleanups have been removed.

There is still another possibility which asynchronous systems can use. Observe that the time it takes to transfer a word includes the time it takes the word to travel from sender to receiver, and then for the acknowledging handshake to return to the sender. In addition, there are internal delays in both master and slave, and there are extra handshake edges for bus cleanup. During a block transfer, it may be possible to avoid all these delays and run at the maximum throughput which the bus transmission line bandwidth and skew can support, by allowing the sender to proceed to the next data cycle without waiting for the handshake. Cycle-by-cycle error recovery is impossible and the transfer rate has to be carefully tailored to the needs of the particular transfer, but ultimate blinding speed is available. The bus becomes a pipeline carrying data in one direction and handshakes in the other. The only system we know of which provides this mode is Fastbus; it is probably only worth the trouble in a system which permits long cable buses.

There is one more problem which should be discussed here, which affects both kinds of buses and all interfaces to the real world: the synchronization or metastable-state problem. Consider a microprocessor which is interfaced to a keyboard. From time to time, the micro reads a status register to see whether a key has been struck. The status register has to

"decide" and return a one or a zero. What if the key is struck just as the status register is being read? Did it happen in time for this read, or not? It does not matter which way the decision goes; if the key is not sensed this time it will be sensed a few milliseconds later when the micro checks again. The decision is usually made by a clocked register, a bistable flip-flop with a data input, a clock input, and a data output. When the clock occurs, the flip-flop decides whether the data input was a one or a zero, and remembers it at the data output until the next clock.

This trivial problem turns out to be fundamentally difficult. The specification for any real flip-flop gives a time interval near the clock signal during which the input is not permitted to change. But if the data is not already synchronized to the clock, and comes from some independent source like a keyboard or another microprocessor or other system, there is no way to prevent it from changing during the forbidden interval. When this happens, the flip-flop may go into a metastable state and simply refuse to decide for a while. Its output may be an ambiguous intermediate voltage level for an uncertain length of time before some random noise pushes it one way or the other. Meanwhile, other circuits (possibly internal to the package) have proceeded to take action based on their interpretation of the ambiguous output of the flip-flop. When the flip-flop finally decides, the interpretation may prove wrong, but it is too late to undo the action. The system may end up in a logically inconsistent state, causing a serious error.

The problem is not just that the flip-flop is badly designed;

it is a fundamental property of nature like the uncertainty principle of quantum physics: the nearer the criterion, the slower the decision.

Although the problem cannot be solved in principle, careful design can reduce the possibility of error during the lifetime of the system. The method is to use proper components, because some flip-flops do decide faster than others, and to allow more time for the decision. Using a second flip-flop to decide about the output of the first one, clocking it with a delayed clock, can reduce the probability of error to insignificance. Unfortunately, many designers have ignored the problem and it has caused occasional errors.

Both asynchronous and synchronous systems will encounter this problem when they interface with the external world, or even with various subsystems. For example, a synchronous microprocessor may interface to a bus which uses a clock rate different than its own. Decision delays in such interfaces must be minimized to maintain high system speed.

Asynchronous systems do have another option, however: circuitry can be added to detect metastable states, and an asynchronous system can simply wait until a decision is made before proceeding.

EFFICIENCY

Any bus system has a limit on its capacity, or throughput, which depends on the bus width, speed, and protocol. There may also be overheads, such as arbitration unless it takes place while the previous master is still making good use of the bus. Even a single fast microprocessor may be able to use up the entire capacity of a backplane bus, especially if it is fetching instructions as well as data and not using block transfers.

For multiple-processor systems, therefore, it is a good idea to think of the backplane bus as a communication path between the various processors and a few I/O controllers, and provide each processor with its own private memory for instructions and most of its data. This greatly reduces the load on the backplane bus. If the processors use the bus primarily for I/O and message passing, most of the traffic can use block transfers, gaining nearly a factor of two in throughput. However, depending on the number of processors and the nature of the application, the bus may still become a bottleneck. In fact, if the bus is not idle a reasonable fraction of the time, processors will spend an unacceptable amount of time just waiting for it. A message-passing system begins to act more like a network than like a simple I/O bus.

One solution to the throughput problem is to use more buses with fewer processors on each. Fastbus uses this approach, using a single address space shared by a number of separate buses, called segments. These operate independently but link together

automatically as required whenever a master on one segment addresses a slave on another. This automatic linking results in interference with traffic on all intermediate segments, however, so it must be used sparingly or bottlenecks will occur. If high-traffic paths are provided with their own shortcut cable segments this problem can be controlled. Judicious use of store-and-forward nodes along with a network message protocol can further reduce congestion by smoothing the load and allowing as few as two segments to be linked at a time.

RELIABILITY AND FAULT TOLERANCE

Fault tolerance is a popular topic in bus design discussions. The hope is to use error-correcting codes on the bus so that any single component failure or temporary noise burst can be detected and the problem automatically corrected--a common practice in large memory systems.

Unfortunately, this is not as simple as it seems. There are several sets of signals which would need protection independently, such as control, data, status, and arbitration. Error-correcting codes tend to be inefficient for small numbers of signals, so a heavy burden of extra correcting signals is required. Furthermore, it takes time to compute the checks and corrections and this slows the bus. Complexity may also reduce reliability, so the gain may be less than desired, and external interference may cause errors in more than a few signals at once. It is not clear how to protect timing signals with such schemes. The resulting cost in complexity, performance, and additional hardware makes other solutions more attractive for most purposes.

Higher-level solutions, which check and correct whole blocks of data or whole actions of programs rather than individual cycles on the bus, are more practical. Redundant processors and buses can check one another; software can make intelligent changes to the system configuration when problems are detected and notify the operator when units need replacing. Even if the bus itself implemented error correction, some level of higher intelligence would still be needed for these functions to prevent

a system from gradually deteriorating until the correction mechanisms could no longer compensate.

There are certain common hardware implementations which should be avoided in systems which are concerned with fault tolerance. If an error is detected, it should be possible to retransmit the data in order to correct it. This implies that the original transmission can have no irreversible side effects. For example, if reading data from a peripheral device erases the original data or clears status flags as a side effect, the read cannot be repeated successfully. FIFOs have similar problems, since bad data stored inside cannot be retrieved and corrected. Queues and buffers should be in addressible memory instead, and clears or resets should be explicit commands rather than side effects. FIFOs can be made to work if extra addressible buffers are added to hold data until successful transmission is complete.

SOFTWARE ASPECTS

Bus and bus interface design has significant software implications. We will raise some of the relevant topics but not attempt to analyze the wide variety of solutions possible for the software problems. These topics should be kept in mind when evaluating any bus system to ensure that it handles the problems adequately. Several questions arise: How is a system to be initialized? How are addresses and arbitration priorities allocated to the various devices? Does the bus provide enough start-up support and programmable flexibility to allow automatic reconfiguration when new devices are added? If the bus supports live insertion or removal, how does the system reconfigure while running?

Present processors do not automatically support the kind of block transfers modern buses offer. How can the software interface cause block transfers to occur? How do processors interface to the bus interrupt mechanism, and how do devices find out where to address their interrupts? If the bus address types do not correspond directly with the processor address types, how does the software access them?

Software often needs to arrange for exclusive access to certain system tables. In a multiple-processor system, the bus often provides a locking mechanism to prevent access by other processors until the table is available again. If the system has multiple-port memories, the lock mechanism on one bus port may need to affect access from the other port as well. Processors

with multiple bus interfaces have similar problems.

How do processors of various kinds communicate on the bus? There may be a need for a mailbox facility at a well-known address on the bus with some agreed-on format for messages. The two common byte-addressing schemes (left to right versus right to left within a word) often have problems in this area.

It is becoming common to define certain special addresses as part of a bus specification, so that registers needed for system initialization or housekeeping functions can be easily found. The registers which set the arbitration priority or logical address of a device, contain its model and serial number, or contain control bits which reset or restart it are all more useful if easily found by general system support software. The special address space or section of regular address space which contains these registers is often called Control and Status Register space, or CSR space.

CSR space is usually accessible via special addresses which depend only on a device's physical position on the bus, using the position-encoding pins in the connector. This makes it possible to address devices by their position in order to initialize them, to assign logical addresses in the normal address space, and to assign priority codes. If sufficient information and control is provided, automatic configuration of systems becomes possible, eliminating troublesome mechanical switches.

SUMMARY

We have looked briefly at all the main aspects of buses. The many choices which must be made in any bus design are the reason why there are so many different buses.

Modern designs are approaching physical limits, which makes successful design more difficult. The resulting high cost of a proven design may reduce the rate of creation of new buses.

ACKNOWLEDGEMENTS, AND A BIT OF HISTORY

The problems of the MITS Altair bus got me interested in this subject. I bought serial number nine of that first S-100 machine, and got actual blueprints in late 1974, so that I could start building interface boards and a useful memory in advance of hardware delivery in February 1975. The computer came with a 256-byte memory board, expandable to 1024 bytes! Still, it was a bargain at $495, assembled and tested, at a time when the 8080 CPU chip was selling for $350 by itself. Len Shustek, who was then a graduate student at Stanford, and is now vice president of Nestar Systems, was helping me design a similar system when we learned about the Altair and abandoned our own design. He had built an earlier machine using the Intel 8008, with graphics display, three-voice music generator, and a lot of nice software. We noticed from the blueprints that there was no way to turn the CPU's bus drivers off, so it would be impossible to have multiple bus controllers for DMA or multiprocessor systems. MITS responded to my anguished call by adding driver-disabling signals on the bus, further delaying hardware shipment. Unfortunately, none of us recognized the problems which would be caused by the inverted write signal--when bus masters were changing and no one was driving it, it caused writes to memory--and we did not understand the need for better grounding.

Bob Stewart, now vice president of the Computer Society, was annoyed by these problems in his Altair, and had the audacity to suggest that the situation could be improved if the IEEE would get involved in standardizing the Altair bus, then called the

S-100 because of its 100 pin connector. MITS got a good deal on those connectors, so used them even though they couldn't think what to do with all the pins. Every other manufacturer thought of lots of different things to do with them, though, and incompatibility and chaos reigned.

Bob's incredible energy and perseverence got the project rolling, and I joined up partly because I felt guilty for missing those bus problems which could have been fixed so easily in 1974. George Morrow (Thinker Toys), Howard Fullmer (Parasitic Engineering), Kells Elmquist (InterSystems), Tim Paterson (Seattle Computer, now of MS-DOS fame), Mark Garetz (CompuPro), Gary Feierbach (Inner Access), and numerous others made real contributions to that project, called IEEE-P696. One offshoot from this project was P896, an effort to start fresh and do it right for 32-bit buses, in the hope of heading off another round of chaos in the industry. Now addicted to this punishment, I joined that project too and both the technical and sociological parts of the project were a real education for me. I benefited from the efforts of many contributors to that project, but I am particularly grateful to Paul Borrill, whose understanding of buses and many other things is both broad and deep, and to Matthew Taub, for his analysis of the performance of his arbitration scheme.

During the P696 work, I was invited to join the Fastbus design team, an effort sponsored by the Department of Energy. Fastbus was to deliver the maximum performance possible, while still being simple and inexpensive, to help handle the enormous

data acquisition and computing needs of high-energy physics experiments. It had to be very flexible and general, because no one knew what future requirements might be. Though reality always requires compromises, Fastbus did a good job of meeting those goals, and I gained a lot from that design effort. My simultaneous work on 696 and 896 proved to be synergistic, as I served as a conduit for ideas and experiences among the projects.

I am particularly glad that the Department of Energy enabled Bob Downing, of the University of Illinois at Urbana/Champaign, to spend a year at SLAC working on the system design. Discussions with Bob and with Leo Paffrath at SLAC were very stimulating and productive, and I think that year had a lot to do with the coherence of Fastbus. I do not mean to minimize the important contributions made by many other members of the design team, but good design requires more sustained interaction than is possible in committee. Ray Larson of SLAC was responsible for the design project, and was very helpful and generous in his support. I recall with special pleasure many discussions with Ed Barsotti of Fermilab, John Biggerstaff of Oak Ridge, Ken Dawson (Fastbus editor) of TRIUMF, and Don Machen of Los Alamos (now with Scientific Systems International).

Special mention is also due Louis Costrell of the National Bureau of Standards, chairman of the NIM committee. The NIM committee brought us the NIM module standard, which had no databus at all but is still widely used, and collaborated with the European ESONE committee to bring us CAMAC (IEEE 583), which has a 24-bit data bus. The NIM committee then sponsored Fastbus

(IEEE P960) with support from the Department of Energy, responding to pressure from the user community. Lou got things started and kept them moving, finding trouble spots and maneuvering around them, handling the sociological problems as well as participating in the mechanical and thermal design, and handling the distribution of information in this multilaboratory, multinational project.

The list of references and suggestions for further reading was greatly enhanced by Bob Dobinson of CERN (Geneva, Switzerland), presently at the University of Illinois at Urbana/Champaign, who recently taught a course on buses and who is a collaborator in the Fastbus project. Many useful comments were also provided by the Computer Society reviewers, and by Bill Ash of SLAC.

FOR FURTHER READING

There is a great deal of useful information and many helpful references in the other articles in this special issue. Of particular relevance to specific topics mentioned in this paper are:

R. V. Balakrishnan, "The P896 Future Bus Solves the Bus Driving Problem." Also see the references there to related articles covering noise, crosstalk, and reflections.

{Editor please supply title} by Matthew Taub, describing the arbitration mechanism for P896 in detail.

See also Taub's early paper on the arbitration scheme, "Contention-resolving Circuits for Computer Interrupt Systems," D. M. Taub, Proc. IEE, Vol. 123, No. 9, September 1976.

There is an excellent discussion of the synchronization problem in: Introduction to VLSI Systems, by Carver Mead and Lynn Conway, Addison Wesley, 1980. See especially p 220 and pps 236-242. References to early work on the problem are also given.

The problems and features of the wire-OR bus connection are discussed more fully in: "Wire-OR Logic on Transmission Lines," by D. B. Gustavson and John Theus, IEEE MICRO Vol. 3 No. 3, June 1983.

The IEEE bus standards are especially relevant to this subject. They are available from: IEEE Service Center, 445 Hoes Lane, Piscataway, New Jersey 08854, USA. See especially:

IEEE Standard 696-1983, Interface Devices. This is the S-100 bus, widened and improved. It is presently finding wide use as a 16-bit bus, but will probably not be stretched to 32 bits.

IEEE Standard 796-1983, Microcomputer System Bus. This is essentially the Multibus, improved.

IEEE Standard 488-1978, General Purpose Interface Bus.

IEEE Standard 728-1982, Code and Format Conventions for Use with ANSI/IEEE Std 488-1978. This makes 488 more useful by defining the format of information to be transmitted.

Frederick A. Kirsten, "A Standard Data Busing System for Use with NIM Modules," IEEE Transactions on Nuclear Science, Vol NS-31, No. 1, February 1984, pps 175-177. This paper describes the incorporation of 488 into an existing class of standard modules, the NIM standard, which dates back to 1964 but is still widely used.

Tutorial Description of the Hewlett-Packard Interface Bus, Hewlett-Packard 1980, revised January 1983.

CAMAC Instrumentation and Interface Standards, SH08482, IEEE 1982, The IEEE, Inc., 345 East 47th Street, New York, NY 10017. This volume includes IEEE 583 and several related standards which form a family of standards called CAMAC (Computer Automated Measurement And Control). Additional standards relate to software and other aspects of CAMAC. CAMAC is a 24-bit data bus, optimized for use in a single-processor data acquisition system. It is the predecessor of Fastbus, a faster and more symmetric 32-bit system. Though the CAMAC early 1970's bus technology is pretty old, the system is still very cost-effective, with many manufacturers supplying catalogs full of useful modules containing the latest technology.

A CAMAC Primer, by P. Clout, Los Alamos Report LA-UR-82-2718. This is a good introduction to CAMAC.

FASTBUS, Modular High Speed Data Acquisition and Control System for High Energy Physics and Other Applications, U. S. NIM Committee. DOE/ER-0189, available from the National Technical Information Service, U. S. Department of Commerce, Springfield, Virginia 22161. Fastbus is currently just beginning to appear in manufacturers' catalogs. Prototype systems are operating now, and large systems are being built. Though it was designed for the needs of high-energy physics, it should be of use in many applications once it becomes widely available. It should become IEEE

Standard 960 within a year. It is of some theoretical interest as well, because of its support for independent bus segments which dynamically link together as needed, automatic message routing mechanisms, ideal solution to the wire-OR cable driving problem, and extensible architecture. It has significantly influenced the development of other modern buses. Further information about the current status and latest developments of Fastbus (and also CAMAC) is available from Louis Costrell, Chairman, NIM Committee, National Bureau of Standards, Washington, DC 20234.

IEEE P896/D6.2, Backplane Bus. Draft available from Paul Borrill, Chairman, P896 Working Group, University College London, Mullard Space Science Lab, Holmbury St. Mary, Dorking, Surrey RH5 6NT, England.

IEEE P961/D2, Proposed Standard for a Microprocessor System Bus Based on the STD bus. Copies are available from Matt Biewer, Chairman, IEEE P961 Working Group, Pro-Log Corporation, 2411 Garden Road, Monterey, CA 93940.

STD Bus Technical Manual and Product Catalog, Pro-Log Corp., August 1982

IEEE P1000, STE Bus, draft standard available from the chairman, Mr. W. Shields, 1161 Cushman Ave, San Diego, CA 92110. This bus is similar to the STD bus, but uses Eurocard packaging and has some other differences.

IEEE P970, Advanced Backplane Bus (Versabus), draft standard available from the chairman, John Black, Jr., Motorola Inc., 3407 E. Hubbel, Phoenix, AZ 85003. This is a very wide bus, with large daughter boards.

IEEE P1014, Versatile Backplane Bus (VME), draft standard available from the chairman, Wayne Fischer, 82 Shereen Place, Campbell, CA 95008. See article in this issue.

Digital Equipment Corporation (DEC) has developed a variety of buses which are widely used de facto standards. The following references may be of interest:

C. Gordon Bell, J. Craig Mudge, John A. McNamara; Computer Engineering: A DEC View of Hardware Systems Design, Digital Press 1978. Chapter 11, by John Levy, describes the general properties of DEC buses, Unibus, Q-bus, Massbuss, SBI.

The PDP-11 Bus Handbook, Digital Press 1979. Full

Unibus description, some information on LSI-11 bus (Q-bus), Massbuss, PCL-11 bus (an interesting network-like time-slotted bus)

The PDP-11 Architecture Handbook, Digital Press 1983-84. Short description of Unibus, detailed up-to-date reference to LSI-11 Q-bus, technical specification in Appendix E.

The VAX Hardware Handbook, Digital Press 1983. This describes the VAX SBI bus, a high-speed synchronous design.

Other manufacturers have also specified their own buses, often offering them to the public for standardization through the IEEE. Documents should be available from their local representatives. Also, every microprocessor defines its own local bus. Some buses of particular interest:

Multibus-II Bus Architecture Specification Handbook, Intel #146077-B. Multibus-II is a 32-bit justified synchronous bus.

Intel Multichannel Bus Specification, #142804-rev C.

Intel ILBX Bus Specification, #145695-rev A.

NuBus Specification, Texas Instruments TI-2242825-0001
NuBus is a 32-bit unjustified synchronous bus.

There are a number of bus-specific books available now. Two useful examples are:

Interfacing to S-100/IEEE 696, Sol Libes and Mark Garetz, Osborne/McGraw Hill.

Interfacing to the IBM Personal Computer, Lewis C. Eggebrecht, SAMS. This is highly recommended.

General References:

K. J. Thurber et al., "A Systematic Approach to the Design of Digital Busing Structures," AFIPS, Proceedings of the 1972 Fall Joint Computer Conference, Vol 41, part II, pp 719-740. Classic article, not current nomenclature.

Paul Borrill, "Backplane Bus Standards, Why We Need Them, What We Have Got, Who Makes Them." Introductory article to a special issue of Microprocessors and Microsystems, Volume 6, Number 9, 1982, pp 450-454. Issue includes useful articles on STD, S100, Versabus, VME, Multibus, Eurobus, P896.

Paul Borrill, "Microprocessor Bus Structures and Standards," IEEE Micro, Vol. 1, No. 1, pp 84-95, February 1981.

Approach to Unified Bus Architecture Sidesteps Inherent

Drawbacks, John W. Conway, Computer Design, Vol. 16 No. 1, January 1977, pp 71-76. This is a description of the Honeywell split-cycle bus, a nice example of a write-only system.

Harold S. Stone, Microcomputer Interfacing, Addison Wesley, 1982. Chapter three has a useful discussion of bus protocols and arbitration. Other chapters cover transmission lines, shielding, and use of specific integrated circuits in interfacing.

FIGURE CAPTIONS

1. Several Master and Slave Devices Attached to a Bus.

2. A Typical Backplane and Daughter Card. Springy connector contacts connect printed circuit traces on the daughter cards to the bus lines, connecting drivers and receivers to the bus and also providing power and ground connections.

3. Transmission Lines. Part A shows an ideal shielded uniform line, a section of coaxial cable. Part B shows a typical real bus line, with branches, neighbors, loads, and antennae.

4. Typical Bus Features. The upper part shows simple bussed signal lines. The center part shows a daisy chain tied to a bussed grant line, for arbitration purposes. The lower part shows un-bussed connections to a central arbiter.

5. Synchronous Bus Read Protocol. The clock marks off time for the whole system. The start signal shows when to look for an address and the ack signal marks the end of the transfer.

6. Asynchronous Bus Read Protocol. Each direction of signal flow has its own timing signal: sync and acknowledge. Both edges of such signals are used. The actual duration of each signal depends on the distance between master and slave as well as the observation point along the bus. These signals are shown at the master.

Biographical Sketch and photo:

Please use the one which appears in: "Wire-OR Logic on Transmission Lines," by D. B. Gustavson and John Theus, IEEE MICRO Vol. 3 No. 3, June 1983.

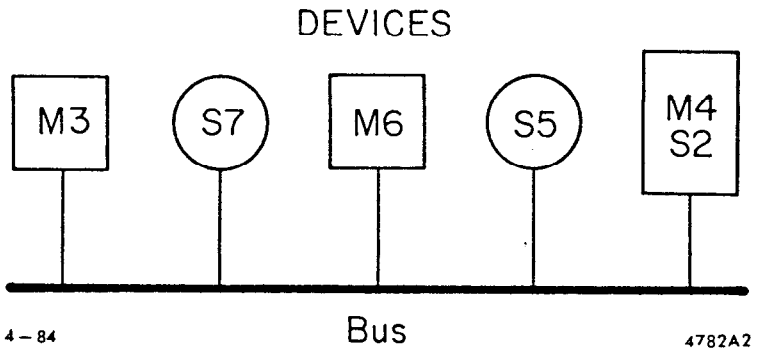


FIGURE 1

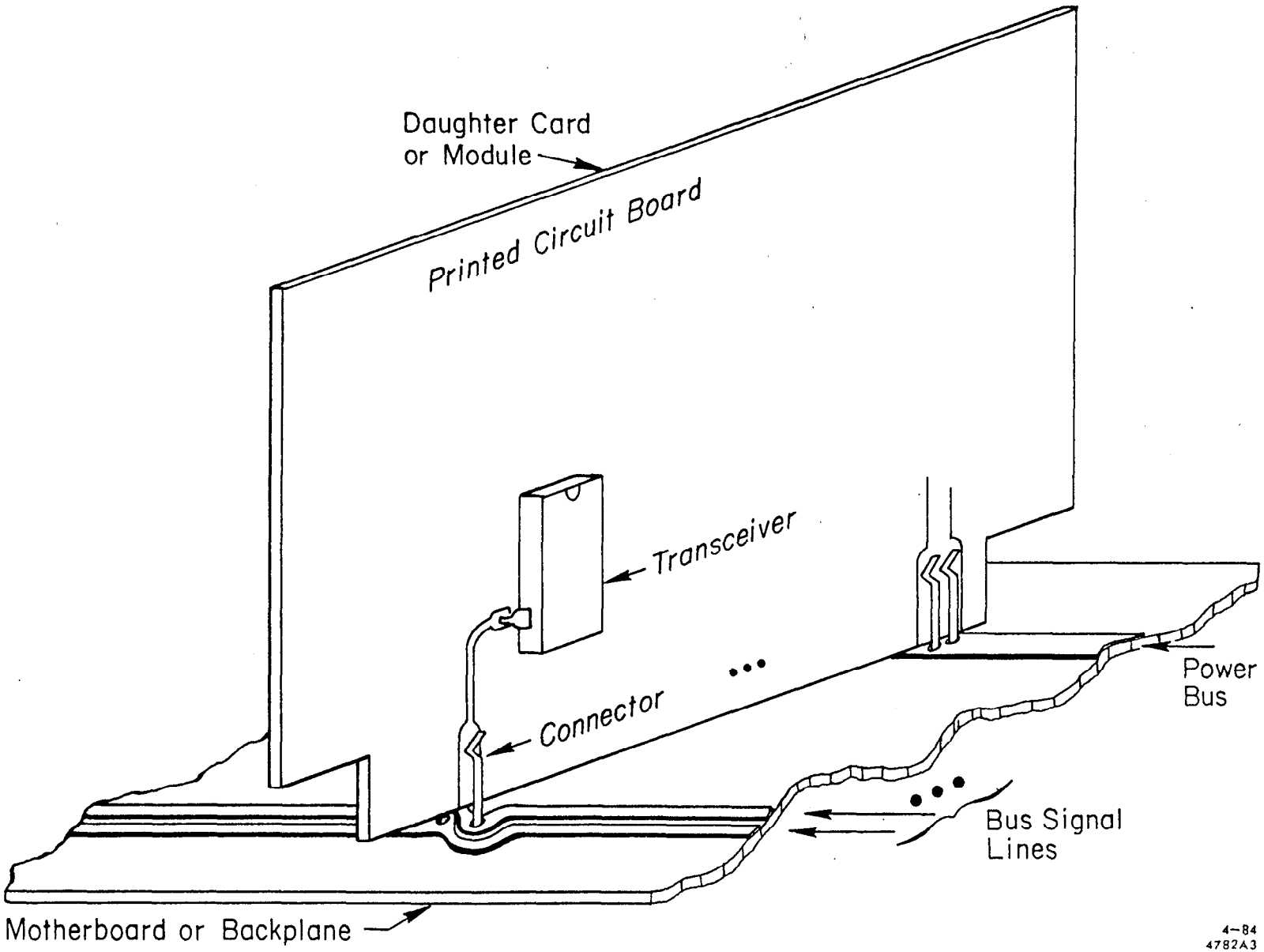
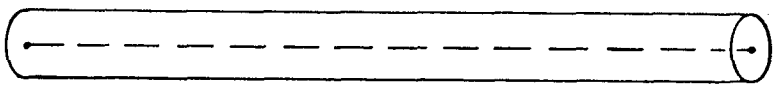
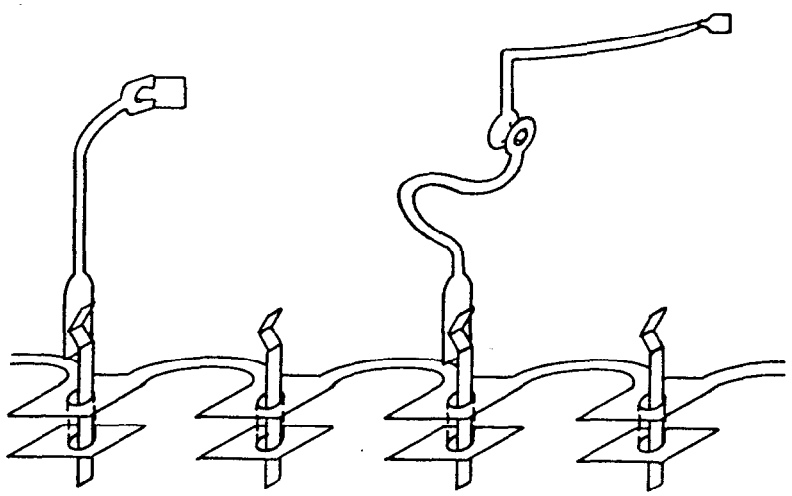

Daughter Card
or Module
Printed Circuit Board
Transceiver
Connector
Power
Bus
Bus Signal
Lines
Motherboard or Backplane
4-84
4782A3

FIGURE 2



Ideal Transmission Line



4-84 Typical Bus Signal Line 4782A4

FIGURE 3

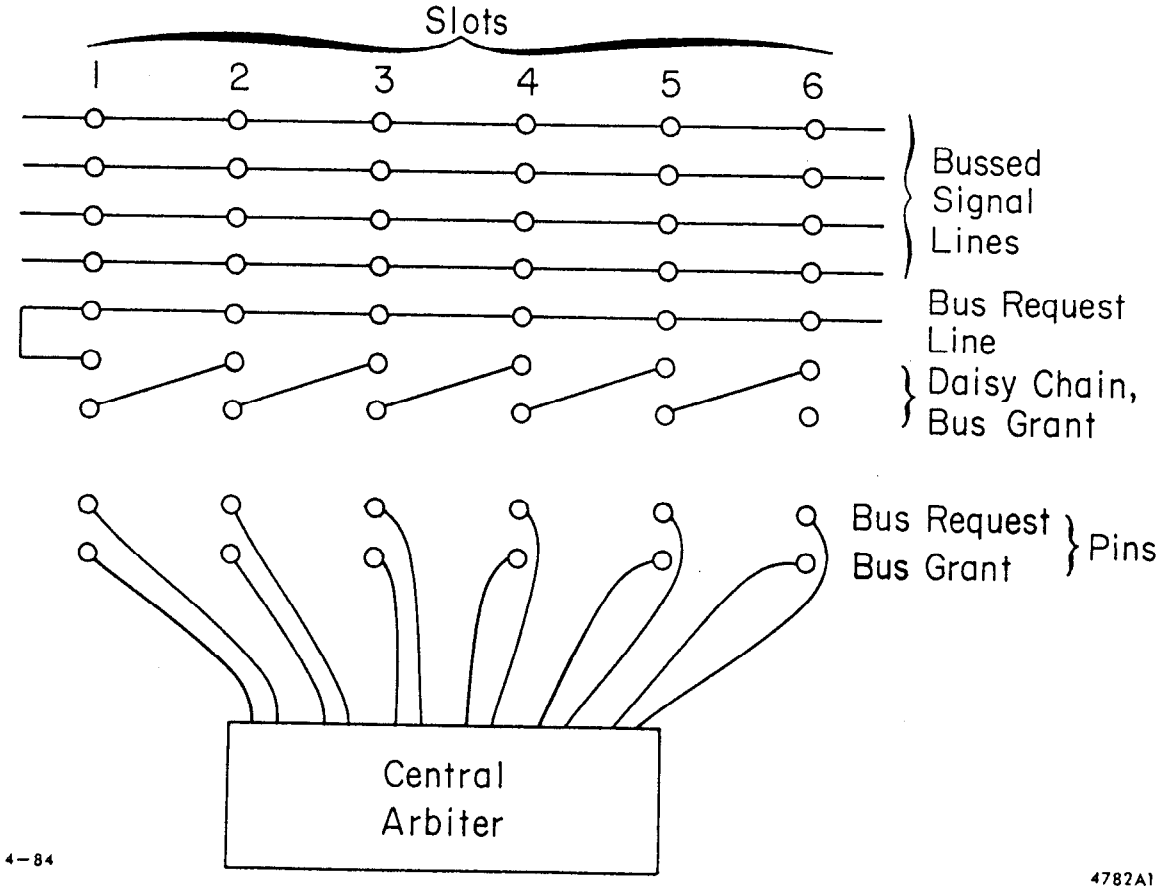
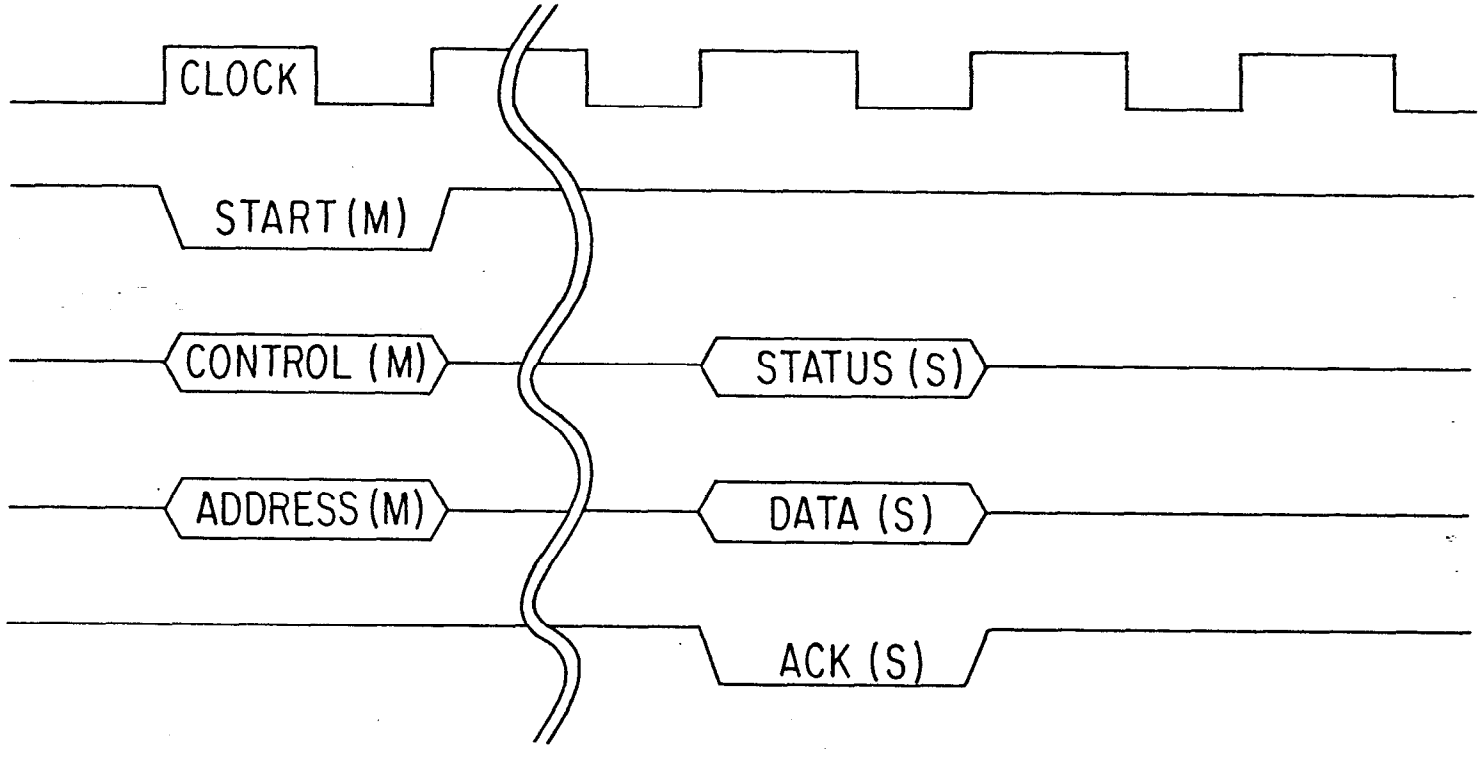
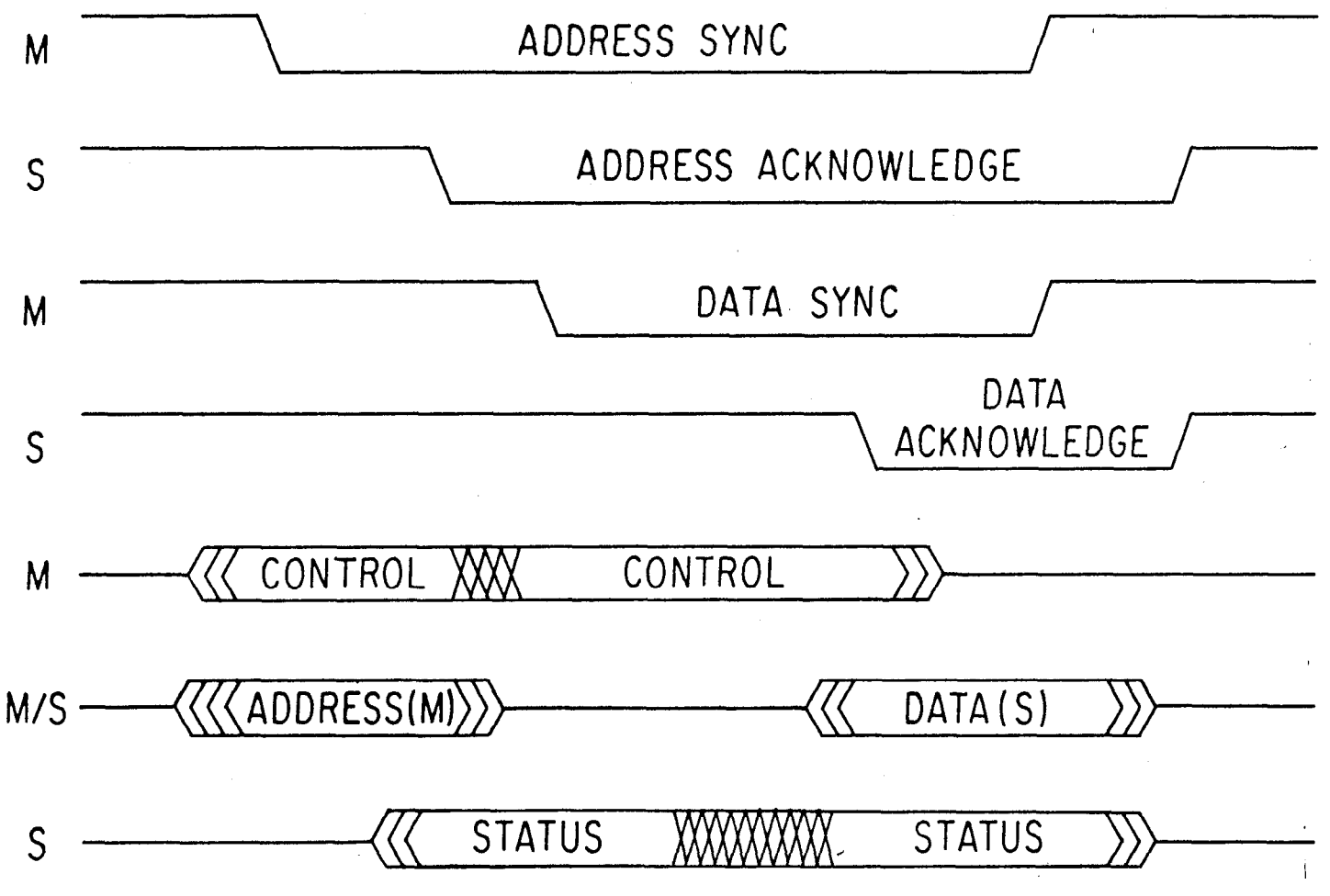

Slots
1
2
3
4
5
6
Bussed
Signal
Lines
Bus Request
Line
Daisy Chain,
Bus Grant
Bus Request
Bus Grant
Pins
Central
Arbiter
4-84
4782A1

FIGURE 4



4-84 SYNCHRONOUS BUS READ 4782A6

FIGURE 5



4-84 ASYNCHRONOUS BUS READ 4782A5

FIGURE 6