THE TRIMOSBUS

by

Ivan E. Sutherland[1], Charles E. Molnar[2]
Robert F. Sproull[3] and J. Craig Mudge[1,4]

[1]California Institute of Technology
Pasadena, CA 91125

[2]Washington University
St. Louis, MO 63110

[3]Carnegie-Mellon University
Pittsburgh, PA 15213

[4]Digital Equipment Corporation
Maynard, MA 01754

The research leading to this paper was supported in part by the Defense Advanced Research Projects Agency, contract no. N00039-77-C-0185; in part by the NIH Division of Research Resources, grant no. RR00396; and in part by Digital Equipment Corporation.

CALTECH CONFERENCE ON VLSI, January 1979

ABSTRACT

This paper describes a family of communication buses that permit individual senders to communicate with an arbitrary number of receivers and to wait for the last receiver to respond. TRI in the name signifies the use of three wires for sequencing. The bus is speed-independent in that no assumptions about the relative or absolute speed with which bus participants respond to bus signals are required to ensure proper sequencing of bus operations.

Data are passed by two separate mechanisms. First, during normal bus operation a number of parallel data wires are used to transmit individual characters or numbers. The MOS part of the name refers to the fact that the high input impedance of MOS circuits permits us to use these data wires themselves as storage nodes. Second, for debugging, testing, and error recovery a slower serial data path is provided.

This paper also describes a TRIMOSBUS message protocol. The protocol uses sequences of bus cycles to transmit messages of arbitrary length. The unique sender can be selected either on a message by message basis by arbitration, or within a message by mutual consent. We plan to use the TRIMOSBUS and this message protocol in a variety of system designs at Caltech, Carnegie-Mellon University and Washington University.

SECTION I: INTRODUCTION

In this paper, we discuss a bus design intended for communication among several bus participants. The design permits any participant to send information to one or more other participants at once and to delay subsequent communications until the last participant has signaled receipt of the data. The TRIMOSBUS design differs from most asynchronous buses, such as the Digital Equipment Corporation UNIBUS, mainly because of its one-to-many communication capability. Any of the receivers in a TRIMOSBUS may arbitrarily delay the completion of any communication. In bus designs such as the UNIBUS, a single receiver is designated for each transmission, and although other receivers may observe the transmission, they may not arbitrarily delay its completion and thus cannot reliably extract data from it in the absence of a speed specification.

We distinguish three levels of specification in a TRIMOSBUS design. 1) The TRIBUS is a bus sequencing scheme which uses three sequencing wires to permit individual senders to communicate with multiple receivers and wait for the last of them to respond. 2) The TRIMOSBUS augments this basic sequencing scheme with two data transmission mechanisms well suited to implementation in MOS technology. 3) A TRIMOSBUS MESSAGE PROTOCOL accommodates a variety of system communication needs. The reader is invited to imagine alternatives at each level of specification. The designs we have chosen favor the properties of MOS circuits.

First Level of Specification: SEQUENCING

The TRIBUS level of specification deals mainly with the use of three sequencing wires. In transmission of a single datum, only two of the three wires convey signals; on successive transmissions, successive pairs of wires are used in rotating order. By providing three sequencing wires instead of the single clock wire used in synchronous buses, or the two wires typically used in point-to-point asynchronous buses, we are able to detect reliably the completion signal from the slowest of multiple bus receivers.

The TRIBUS minimizes the number of sequential transitions required to transmit each datum. At most, three transition times are required, one to establish the correct data value on the data lines, one to signal that the data are valid, and one to signal that all receivers have received the data. It is difficult to conceive of a bus design which uses fewer transitions per communication while still providing for acknowledgement from multiple receivers. Other asynchronous one-to-many buses, such

as IEEE 488 [1], use more signaling transitions to transfer a single datum. The use of a minimum number of transitions is especially advantageous in MOS circuitry because its relatively low output current makes off-chip transitions slow.

The TRIMOSBUS design relies upon the low output current property of MOS circuitry to avoid transmission line problems. Because of this low output current and because of the relatively small physical size of systems built with highly integrated MOS components, signal transition times can be kept long compared to transmission line delays. In effect, one can treat each signaling wire as an equipotential node rather than as a transmission line. In a transmission line environment, the task of communicating from one sender to many receivers is made more difficult by the need to accommodate transmission delays and to avoid reflections in the transmission line. An equipotential environment is free of these difficulties.

How long can a TRIMOSBUS be? In typical MOS circuits today, off-chip transition times are measured in tens of nanoseconds, and so propagation delays of about a nanosecond, which correspond to bus lengths of about 20cm, should generally satisfy the equipotential assumption. Buses with greater physical extent must be operated more slowly. It appears that the equipotential assumption may be satisfied for any bus length if the ratio of the bus drive current to its capacitance per unit length is sufficiently low and the interconnecting path is lossless. For high resistivity interconnects, such as diffused or polycrystaline silicon paths, this simple analysis does not suffice; at the higher circuit densities anticipated in the future the scaling of both distances and interconnect properties needs further examination.

The reliable one-to-many communication of the TRIBUS presents several opportunities to ease debugging and testing. For example, the bus can be "single-stepped" simply by controlling one receiver's response with a single-step switch. Because the bus requires responses from all receivers to operate, it will be stalled until the single-step receiver is released.

Second Level of Specification: DATA FLOW

An important attribute of MOS circuits is the high impedance of inputs, and of outputs that are disabled. This permits data wires to be used easily as temporary storage nodes. In the TRIMOSBUS design, a sender drives the data wires to the desired value, and senses the voltage on these wires. When the data wires reach the correct voltage, the sender turns off the driving current source before signaling that the data are ready.

Thus, a sender disconnects as soon as a data value has been delivered to the bus, rather than having to wait for the slowest receiver to capture it, and the bus itself provides a level of buffering for each transmission.

The TRIMOSBUS design includes several features designed to facilitate error detection, debugging and testing. For error recovery and testing, the TRIMOSBUS design includes a separate serial communication line as part of the bus. We intend that each bus participant include a serial shift register debugging mechanism which can, upon command, report the content of key state registers in the bus participant. Control of this shift register is obtained by highly redundant coding on multiple bus wires, so that it will operate in spite of severe malfunction of the bus. Serial communication is used to minimize the number of pins required for this function.

Third Level of Specification: MESSAGES

We have designed a simple message protocol for use in systems of integrated circuits. This message protocol provides for messages that consist of one or more consecutive bus cycles. The last cycle in each message is marked so that all bus participants may easily distinguish the end of each message and, hence, the beginning of the next message. Our protocol allows senders to transfer the right to use the bus as a part of the message format; arbitration between contending senders need be used at most once per message.

The protocol is defined in terms of some simple codes used to herald various types of messages. The coding space available for such heralds is not nearly filled. We hope to add message types to provide for a rich variety of messages in systems involving multiple processors and memories. For example, messages to report system status and to assist in debugging and testing could be included.

SECTION II: SEQUENCING

Outline of Operation

The TRIMOSBUS contains two kinds of interconnection paths which are terminated differently: three sequencing wires and an arbitrary number of data wires, as shown in Figure 1. The three sequencing wires are used in a wired-or configuration. Each of them is terminated with a pullup resistor, as shown in Figure 1, which, in the absence of drive from any of the bus participants, will cause it to assume a logically inactive state. In N-channel MOS and TTL circuitry, this inactive state for a wired-or signal is the HIGH state. The bus participants can clamp one or more of these sequencing wires to the active state, the LOW state in TTL or N-channel MOS designs.

The data wires are terminated with negative resistance to a voltage source at the switching threshold so that they will remain in either the HIGH or LOW state for an unlimited time after they are driven to that state and the drive is removed. The negative resistance termination is weak enough that it can easily be overpowered by any of the drivers of the bus, but strong enough to maintain logically defined signal levels in spite of noise pickup and charge leakage. Stray capacitance between the bus wires and ground is, of course, an additional stabilizing influence.

A communication on the bus requires three successive transitions on bus wires. In the first transition, a single sender, selected from several which may be ready to send by an arbitration mechanism to be described later, places its data onto the data wires of the bus. This requires one transition time unless all data wires are already in the proper state. The sender senses the state of the data wires and removes its drive when it observes them to be in the correct state. Because of the negative resistance termination, the data wires will retain their state indefinitely long even after the drive is removed.

After removing drive from the data wires, the sender generates the second transition, a sequencing signal indicating that valid data are present on the data wires. Sequencing signals appear on the three sequencing wires in rotation. Before the DATA VALID transition, one of the sequencing wires is clamped in the LOW state and two are unclamped and in the HIGH state, as shown in Figure 2a. This is one of three equivalent idle bus states as shown in Figure 2a, f and h. The sender generates its DATA VALID signal by clamping the "next" signaling wire in rotation order to the LOW state, as shown in Figure 2b. After the DATA VALID transition, two of the three sequencing wires are in the active (LOW) state and one is

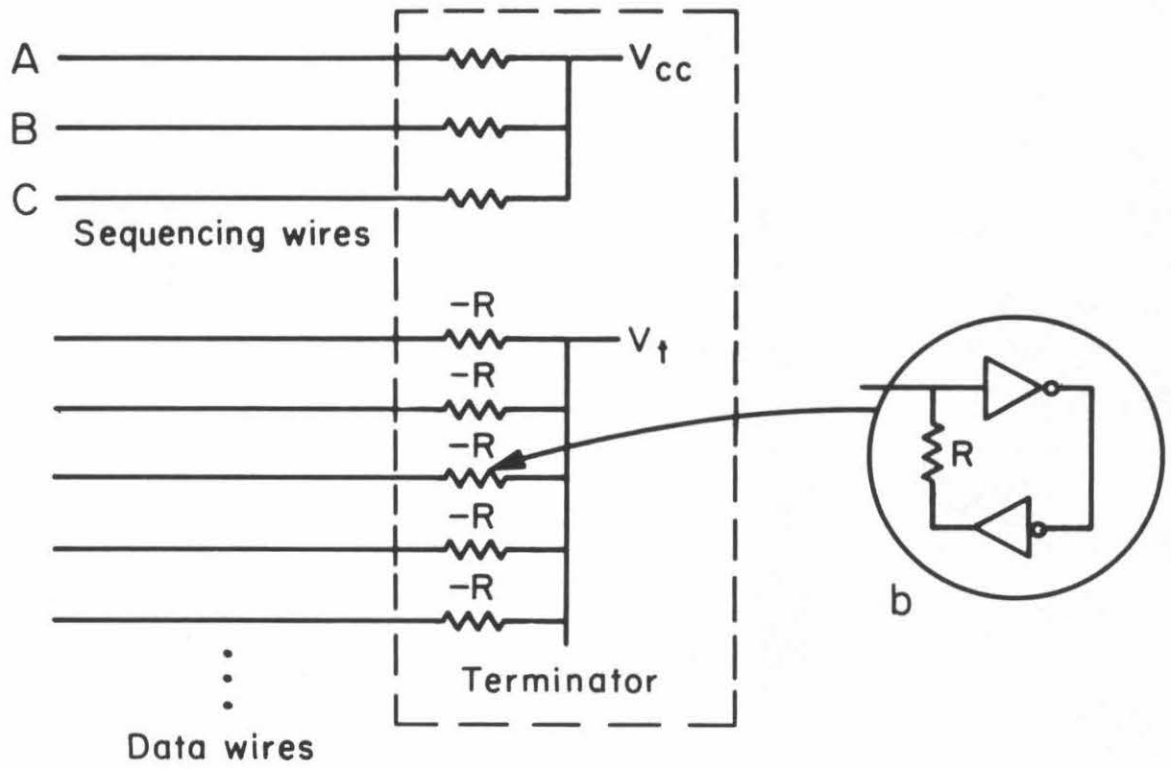

Figure 1: The TRIMOSBUS uses two sorts of wires. Three sequencing wires are terminated to Vcc to permit wired-or operation. An arbitrary number of data wires is provided, each terminated with a negative resistance to the inverter threshold voltage Vt. This negative resistance termination allows the wires to be used reliably as storage nodes. Inset: one implementation for the negative resistance termination.

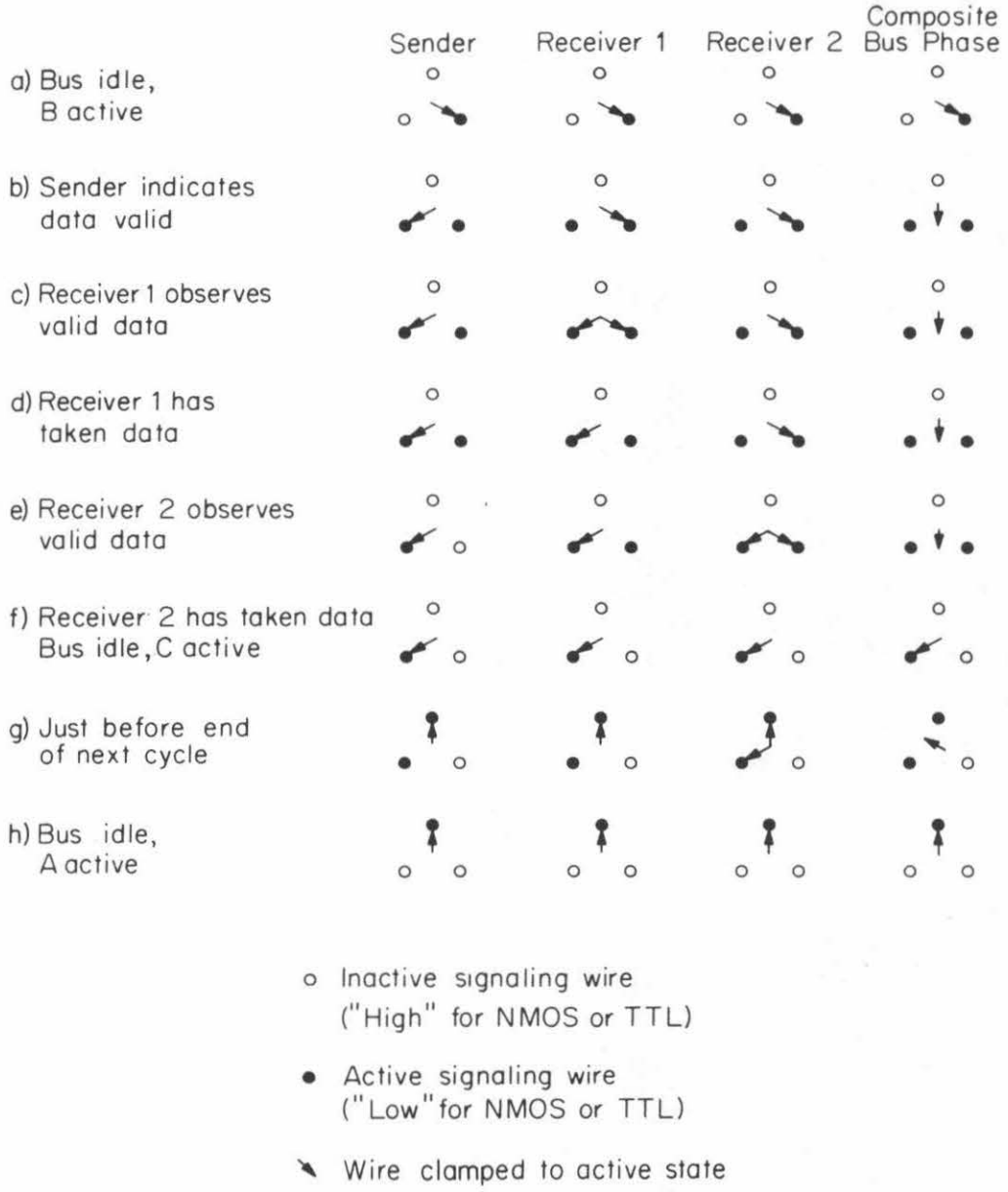

Figure 2: Three-phase bus sequencing is used to indicate valid data and to wait for all receivers to acknowledge receipt of the data. Because sequencing wires are terminated to Vcc, any unclamped sequencing wires assume the "inactive" state. The three idle states of the bus are illustrated in cases a, f and h. We will use the term "bus phase" to distinguish the sequencing states.

inactive (HIGH), which indicates that valid data are available on the bus, but have not yet been accepted by all receivers.

Receivers recognize that data are valid when they observe the DATA VALID transition. While the bus was idle, each receiver was clamping the single active sequencing wire in the active (LOW) state. Upon observing the DATA VALID transition, each receiver must clamp the next sequencing wire in the active (LOW) state as well, as shown in Figure 2c and 2e. Then, when it has satisfactorily received the data, each receiver unclamps the originally clamped sequencing wire, as shown in Figure 2d and 2f. When the final receiver unclamps the originally clamped sequencing wire, the resistive termination will cause the third transition, DATA ACCEPTED, by pulling it into the inactive (HIGH) state, as shown in Figure 2f, thus signaling to the sender and to all other bus participants that all receivers have satisfactorily received the data and that the bus is free to begin the next cycle. Note that the bus has now sequenced to its next idle state. Figure 3 shows how data transitions interleave with sequencing transitions in several successive bus cycles.

Three sequencing wires are required to provide unambiguous indication that all receivers have successfully received the data. To achieve speed-independence, one must ensure that a fast subsequent sender cannot cause confusion in a slow receiver. Thus, one cannot permit two consecutive signaling transitions on the same sequencing wire, because a slow receiver might not observe them. It follows that if only two wires were used, successive transitions would have to occur on alternate wires. This is not feasible in a bus with more than two participants for the following reason. In order to achieve a one-sender multiple-receiver bus with a minimum number of sequential signaling transitions, a transition that signals data validity must be given by a single sender, which may be any of the bus participants; a second transition that signals data acceptance requires agreement by all bus participants. Hence, the former must be accomplished by a "wired-or" connection and the latter by a "wired-and". If these transitions are to alternate and yet not occur consecutively on the same wire, there is no way to use only two wires, since this would require that two adjacent transitions on the same wire be both "wired-or" or both "wired-and".

If a third wire is available, it is possible to satisfy both the condition that two consecutive transitions must not occur on the same wire, and that alternate transitions on the same wire be an alternation of "or" transitions and "and" transitions. Such a three-wire design does require that the functional interpretation of a given signal value on a given

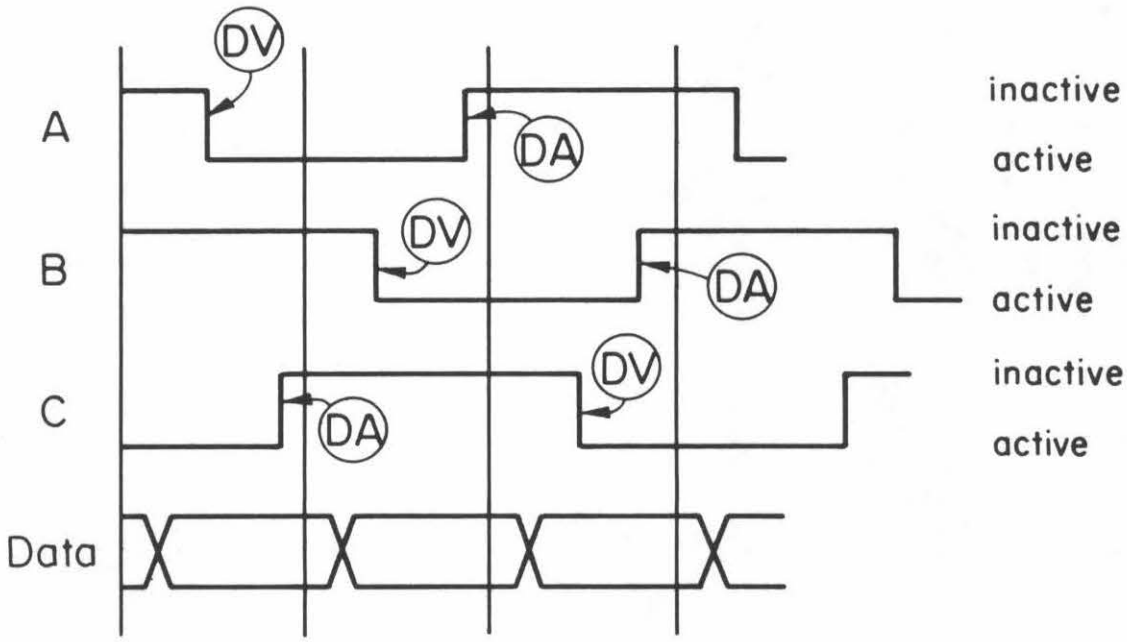

Figure 3: Each bus cycle requires three transitions. The first drives data wires to the correct state. The second indicates data validity (DV) by clamping a sequencing wire. The third indicates data acceptance (DA) by all receivers when a sequencing wire becomes inactive.

wires not be the same for all bus cycles. This property was previously encountered in a two-wire bus design that was considered and rejected for use in Restructured Macromodules [2].

The reader is no doubt concerned, as are we, at the extravagance of three sequencing wires, but they are required for reliable speed-independent operation with more than two participants. As will be seen below, we have shared their cost by using them to provide master clear and maintenance functions as well as sequencing.

The TRIMOSBUS requires that data values be stored on the data pathways as charge on their shunt capacitance. This is a consequence of the sparseness of the sequencing signals that are exchanged on the signalling wires. If timing specifications are to be avoided, a sender must be sure that bus data are valid before sending a DATA VALID transition. Removal of drive from data paths is a more complex matter. If the sender waits to remove drive until after the DATA ACCEPTED signal, it is certain that data validity has been maintained until all receivers have taken the data. However, if the sender is slow in then removing drive from the data paths, it is possible that the next sender will drive the data paths while they are still being driven by the previous sender. This condition appears undesirable and may produce erroneous data values if the next sender removes its data path drive while the previous sender is still driving.

A second alternative is for the sender to remove data path drive after sending the DATA VALID signal without waiting for DATA ACCEPTED. Once again, if the sender is slow in removing drive and all of the receivers and the next sender are fast, overlapping of data path drive by two senders is possible, unless the sender also acts as a receiver and ensures that the DATA ACCEPTED signal on the bus does not occur until after data path drive is removed. In this latter case, however, there is no way to ensure that data path drive will be maintained until all receivers have taken the data. Thus, if conflicting drive of data paths by two senders is to be avoided, while at the same time maintaining assurance that data remain valid on the data path until taken by all receivers, it is essential that data validity, once attained, be maintained by the data path even after data path drive is removed by the sender.

One way of doing this is to equip the bus pathways themselves with storage ability by terminating them with a negative resistance in such a way as to make a bistable circuit. For MOS technology, in which gate inputs and gate outputs in the "OFF" condition can be made to have very high impedance values, it is also possible to achieve "dynamic" storage without such negative resistance terminators. Although this violates our

goal of speed independence, since some upper limit, determined by leakage currents, is then placed on the length of time that data values on the bus path can safely be assumed to remain valid after drive is removed, this may still be a practically useful approach.

A Design for the Sequencer

TRIMOSBUS sequencing has a three-fold circular symmetry; the sequencing wires are used in succession, repeating their function every three cycles as shown in Figure 2. This symmetry permits one to think of the transitions on the sequencing wires as changes in the "phase" of the bus sequence. The three-fold symmetry of sequencing suggests that the sequencing control should also have three-fold circular symmetry. Indeed, we have found that a simple control mechanism can be implemented using tri-flops, the tri-stable analog of the bi-stable flip-flop. This section of the paper will describe a simple control circuit for TRIMOSBUS senders and receivers which we have built and tested. The control is shown in Figure 4.

The control for each sender and receiver contains a tri-stable circuit to keep a record of the most recent bus phase as shown in Figure 2a, 2f and 2h. The control detects phase changes in the bus by comparing the bus phase to the phase of its tri-flop. The control causes changes on the sequencing wires by advancing the phase of its tri-flop.

Senders and receivers can detect transitions on the sequencing wires with simple circularly symmetric logic functions which relate the actual phase of the bus to the recorded phase. Thus, for example, "bus ahead" might be used to describe a logic function A*Sc+B*Sa+C*Sb, in which A, B, and C represent the active states of the three sequencing wires and Sa, Sb, and Sc represent the corresponding internal states of the tri-flop. By thinking of the relationship between the bus phase and the tri-flop phase in terms of such circularly symmetric functions, one is led quickly to simple bus control designs.

In N-MOS or TTL logic, the wired-or arrangement for the bus wires is a LOW-active configuration. Thus when the three sequencing wires are in the idle state, one will be LOW and active and the other two HIGH and inactive. If we call the three wires A, B, and C and assume that sequencing is in that order, then if the bus is idle with B LOW and A and C HIGH, new bus activity will be signaled by C going LOW. The circularly symmetric function "LOW AHEAD" implies that the bus wire ahead of the internal state of the control has changed to the LOW or active state indicating DATA VALID. The circularly symmetric

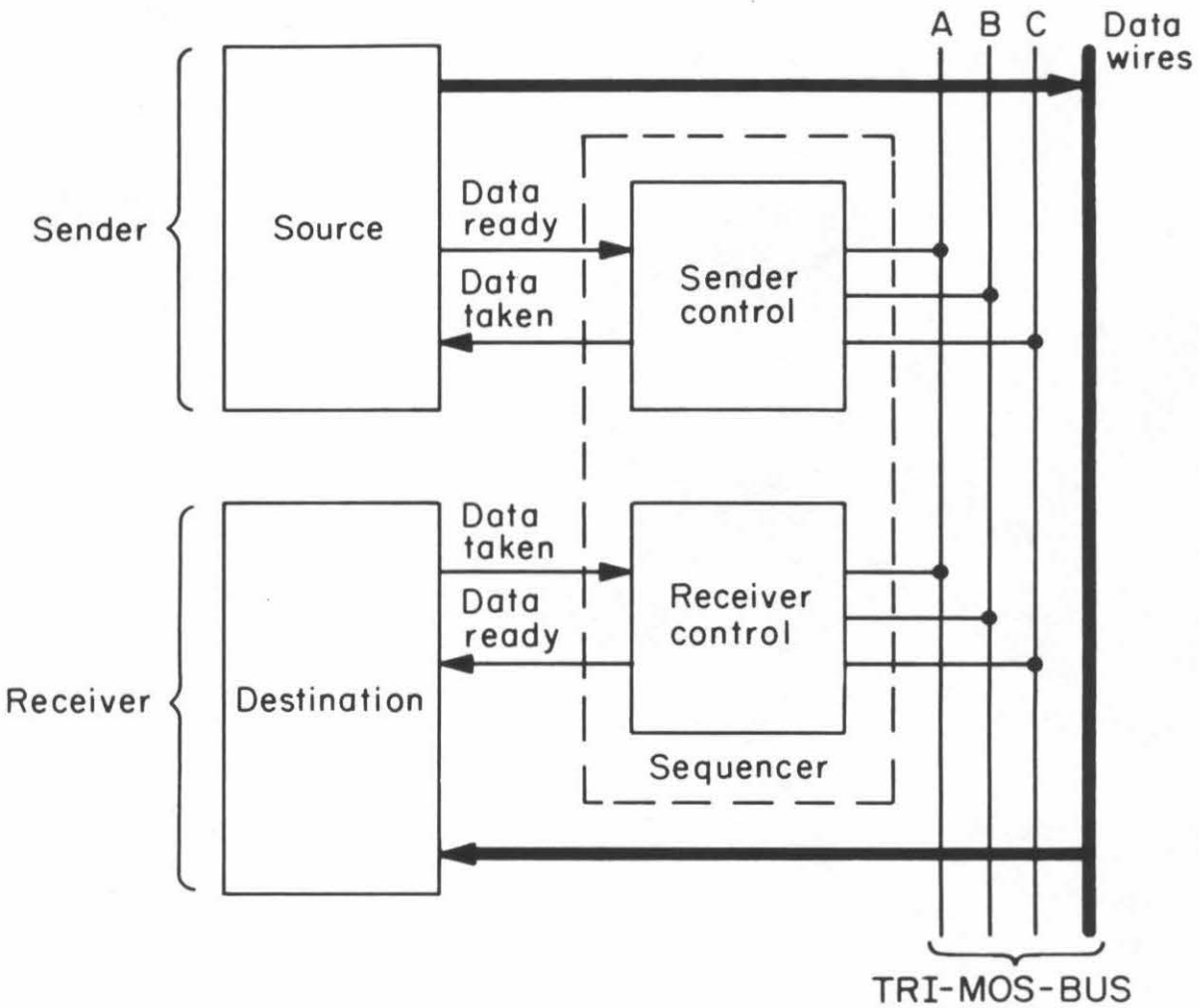

Figure 4: Schematic diagram of a bus participant. The source and destination will drive and sense the data wires respectively. The sequencer converts between TRIMOSBUS sequencing signals and conventional two-wire handshaking signals.

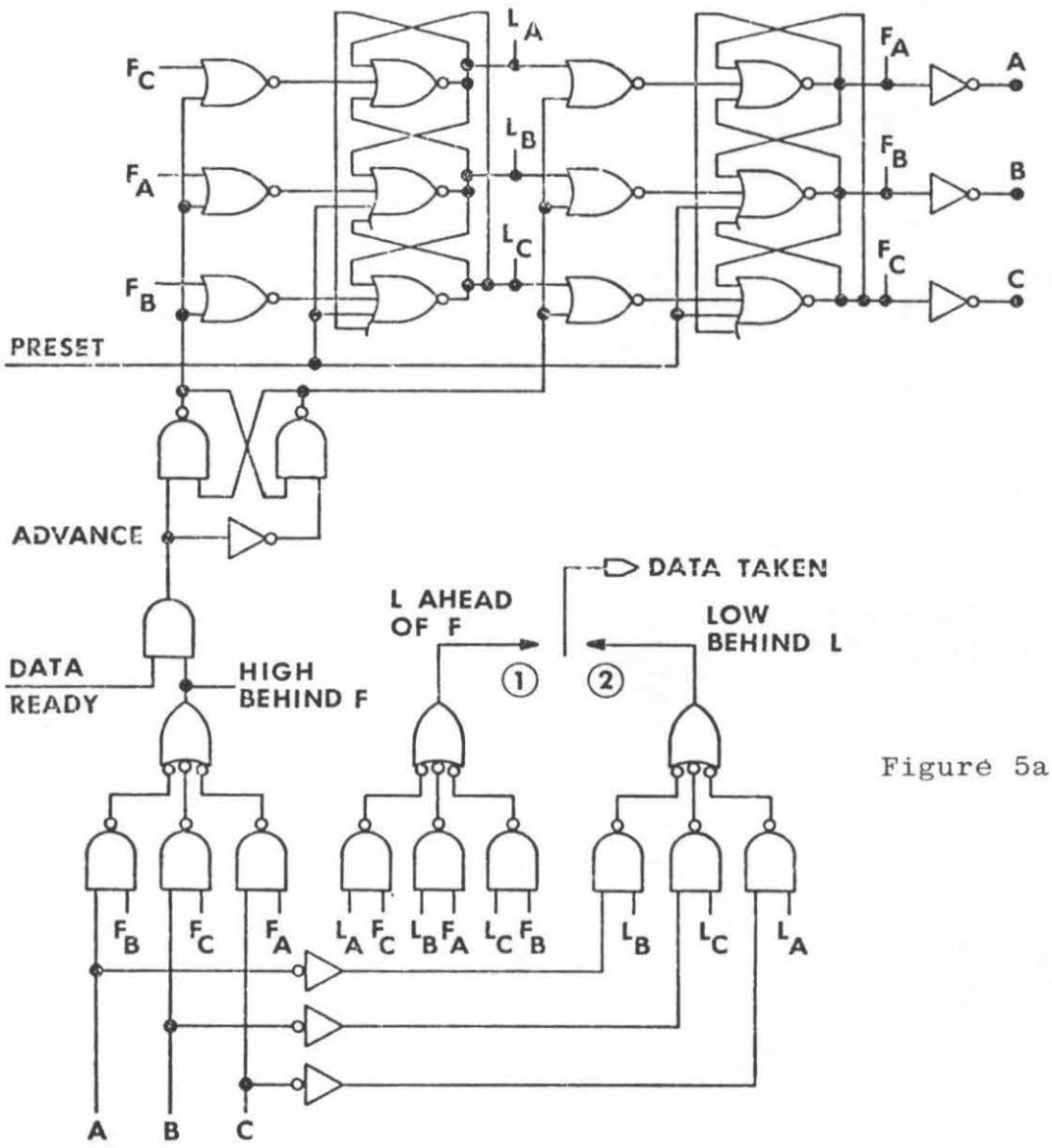

Figure 5a

Figure 5: Logic diagram for sender (5a) and receiver (5b) of experimental TRIMOSBUS control for use as shown in Figures 4 and 6. The DATA READY signal in the sender circuit may be derived from source (1) or source (2), corresponding to the signaling interpretations shown in Figures 6a and 6b. The receiver circuit can be seen to be very similar to that for the sender.

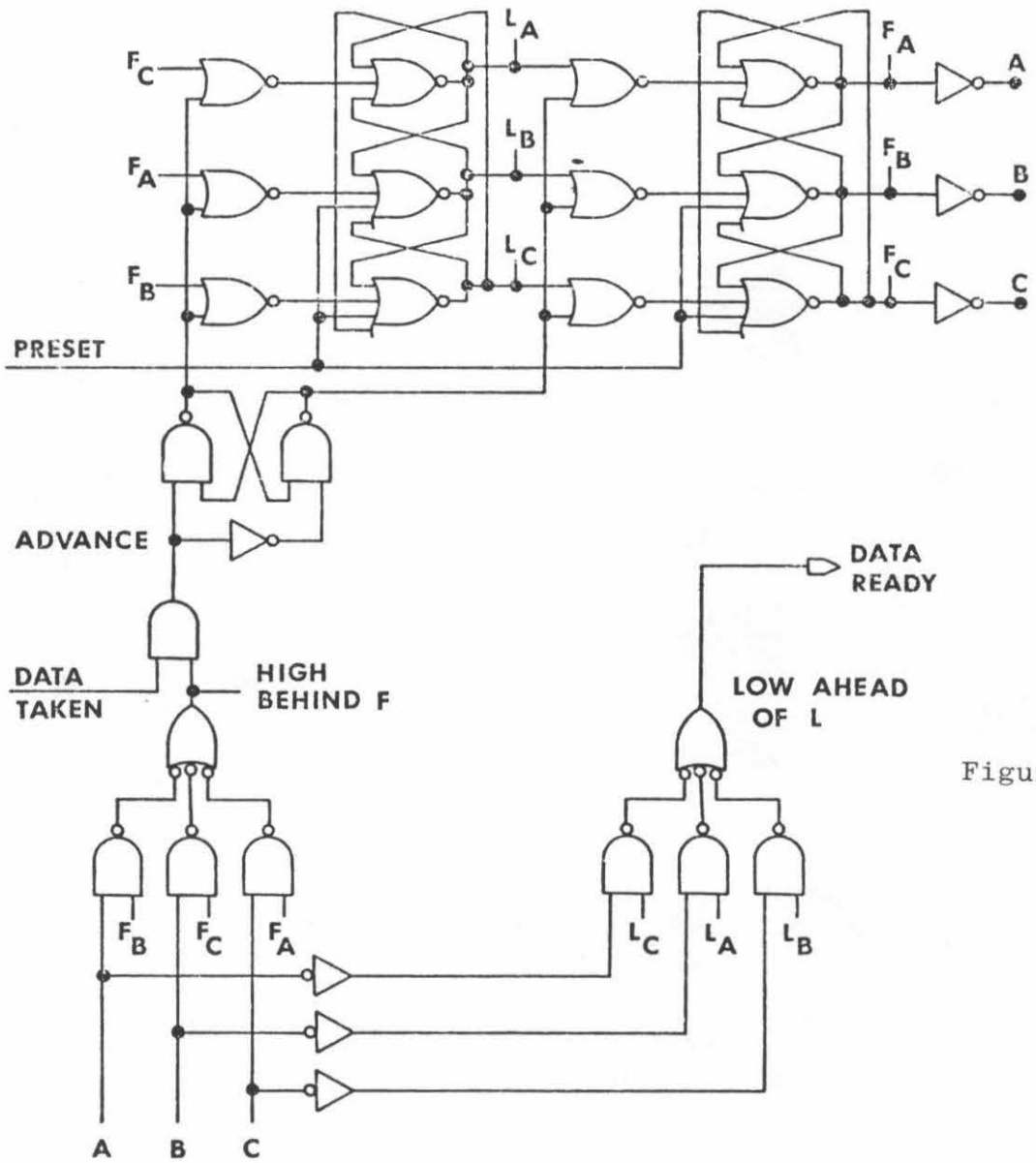
L_A
F_A
A
L_B
F_B
B
L_C
F_C
C
F_C
F_A
F_B
PRESET
ADVANCE
DATA READY
DATA TAKEN
HIGH BEHIND F
LOW AHEAD OF L
F_B
F_C
F_A
L_C
L_A
L_B
A
B
C

Figure 5b

function "HIGH BEHIND" implies that the bus wire behind the current state has returned to the high or inactive state, indicating that the last receiver has accepted the data from the previous cycle. LOW AHEAD thus heralds new activity, while HIGH BEHIND signals completion of former activity. Because three sequencing wires are used, the HIGH BEHIND and LOW AHEAD conditions are not mutually exclusive and may be observed at the same time. Because three rather than two signaling wires are used, these two conditions can nevertheless be unambiguously signalled even if they should appear to overlap in time.

A simple experimental bus control circuit is shown in Figure 5. This control circuit is intended to convert two-wire four-phase handshake signals as shown in Figure 6 to the signalling conventions of the TRIBUS. This design was intended as an experiment to demonstrate a simple bus example with one sender and two receivers. It does not include provision for multiple senders or for arbitration among competing senders, although any number of receivers up to the limits of distance and circuit constraints may be used. The control circuits for sender (5a) and for receiver (5b) are quite similar in that both use a dual-rank tristable circuit that is connected as a ring counter. When the ADVANCE line is asserted, the contents of the following (F) tri-flop are copied into the leading (L) tri-flop. When the ADVANCE line is not asserted, the contents of the leading tri-flop are copied into the following tri-flop. The outputs of the following tri-flop provide drive to the three transistors that can clamp bus signalling paths A, B, and C to the low state. Since only one of the three outputs of a tri-flop is high when the tri-flop is in a stable condition, the sender and receiver controls shown here always clamp exactly one bus signaling wire while in a given stable state.

The combinational circuits at the bottom of Figures 5a and 5b generate the ADVANCE signal and signals to the SOURCE component of the sender and the DESTINATION component of the receiver. In the sender, a SOURCE that has data to send asserts DATA READY. When the bus becomes inactive, as indicated by the HIGH BEHIND F condition, ADVANCE is asserted and the leading tri-flop is loaded with the shifted contents of the following tri-flop. This does not change the clamping of the bus signaling paths, but the change in the leading tri-flop causes the condition L AHEAD OF F to be satisfied. This generates the assertion of DATA TAKEN to the SOURCE, signaling that the bus data paths are free and the sender control is primed to advance the bus. Upon receipt of this signal, the SOURCE places its data values on the bus data wires. When they have reached a valid condition, the SOURCE removes its data path drive and then deasserts DATA

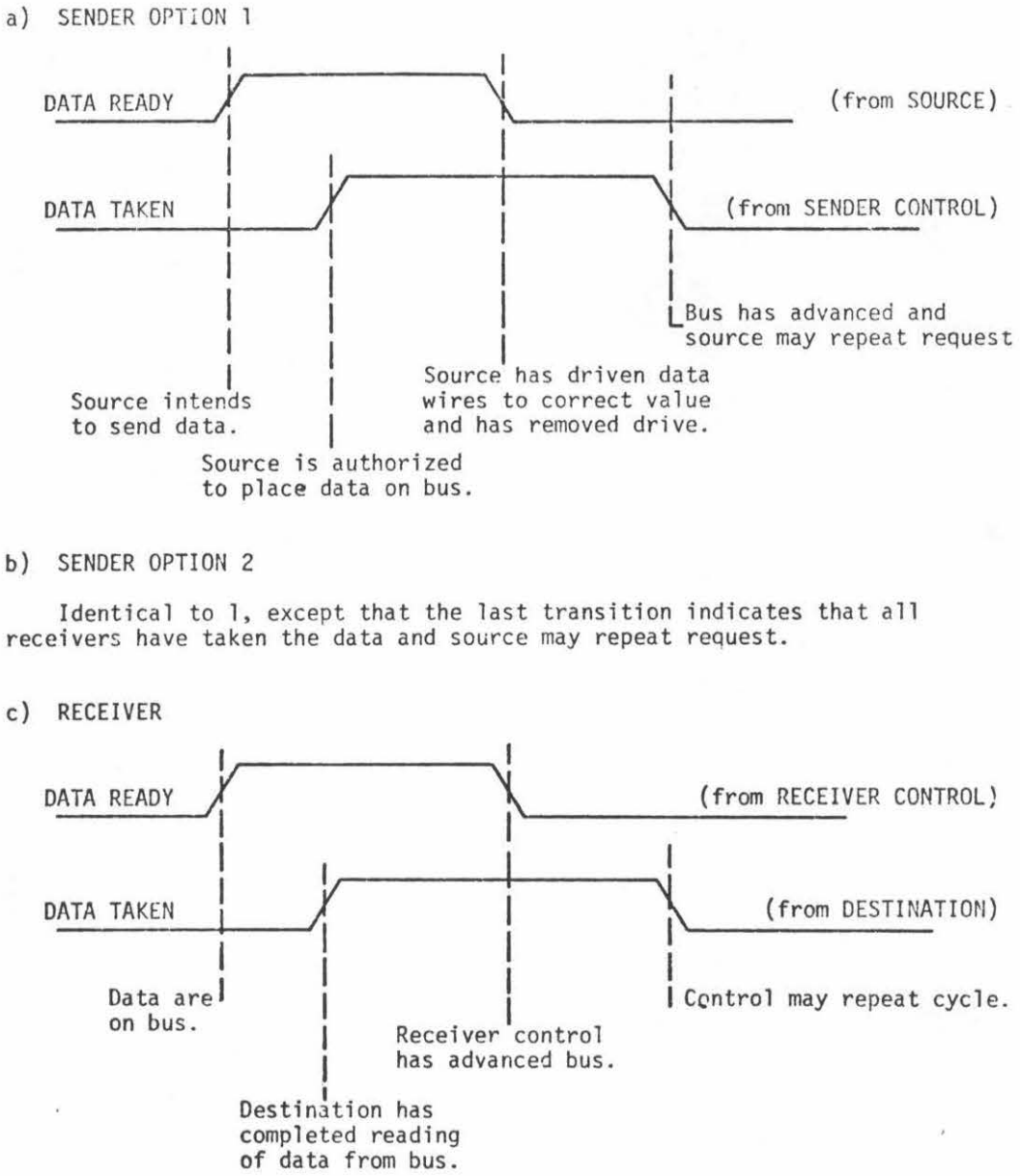

Figure 6: Two-wire handshaking signals used by the sequencer. Two variations of the sender control are shown. The first allows the data wires to provide a single level of buffering between the sender and the receivers.

READY. This in turn causes the deassertion of ADVANCE, which causes the following tri-flop to be loaded with the contents of the leading tri-flop. This advances the bus by clamping the next signaling wire in rotation.

The advancing of the following tri-flop also causes the HIGH BEHIND F and L AHEAD OF F conditions to be no longer satisfied. This removes the assertion of DATA TAKEN to the SOURCE, enabling the SOURCE to begin preparation of its next request to send on the bus. The HIGH BEHIND F condition does not hold until the bus completes the cycle and all receivers have taken the data from the bus data wires. Response of the sender control circuit to the next assertion of DATA READY by the SOURCE cannot begin until HIGH BEHIND F holds.

Alternatively, the DATA TAKEN signal to the SOURCE may be generated from the LOW BEHIND L condition, as indicated by option 2 in Figures 5a and 6. In this case, the assertion of DATA TAKEN follows the loading of the leading tri-flop as before; the deassertion of DATA TAKEN is however held up until the LOW BEHIND L condition is removed by the signal from all bus receivers that they have taken the data.

The receiver control operates in a similar manner. The advancing of the bus by a sender causes the LOW AHEAD OF L condition to hold, generating a DATA READY signal to the DESTINATION. The DATA TAKEN signal that follows causes ADVANCE to be asserted, which advances the leading tri-flop and removes the LOW AHEAD OF L condition. This in turn removes DATA READY. Following the receipt of the deassertion of DATA READY, and after it has completed the taking of data from the data paths, the DESTINATION deasserts DATA TAKEN, thereby allowing the deassertion of ADVANCE, which allows the bus trailing signaling wire to be unclamped and a DATA ACCEPTED signal to be generated on the bus signaling wires when the last receiver has accepted the data.

An experimental TRIBUS has been built and tested using a TTL implementation of the circuits of Figure 5a and 5b. It was operated successfully over a wide variety of internal delay conditions. We have observed timing asymmetry introduced by loading one sequencing wire heavily with shunt capacitance; correct sequencing was maintained in spite of loading.

In a detailed and complete design for the bus control circuit, careful attention must be given to avoid race conditions within the circuit. We do not view such a requirement as compromising our intention that the bus design be speed independent; any circuit design that satisfies the signaling sequence conditions at the sequencer terminal is acceptable.

SECTION III: DATA FLOW

The data communication mechanism in the TRIMOSBUS uses the bus wires themselves as a storage register. The negative resistance termination on the bus wires has already been described in Figure 1. One can, of course, use as many data wires as one chooses.

Speed independence in the presence of variations in the electrical characteristics of individual data wires and drivers can be guaranteed by source checking. The source detects the state of the data wires and signals DATA VALID only when it senses all data wires to be correct. Source checking eliminates the data transition time entirely if data values for two successive bus cycles are the same. Source checking also can detect certain transmission errors, such as those caused by short circuits on the data wires, leaving the bus stopped in the offending state.

Extenders

The equipotential assumption may limit the practical length of a TRIMOSBUS to dimensions of a few feet in MOS and no more than a few inches in faster technologies. Point-to-point extension of the bus, however, is relatively straightforward. The idea is that two TRIMOSBUSs remote from each other might be connected by a cable of arbitrary length and delay. A suitable controller connects each end of the cable to its TRIMOSBUS. These controllers serve as senders or receivers on their respective TRIMOSBUSs and communicate with each other through the cable with a traditional point-to-point asynchronous signaling scheme.

Two interconnections possible in such a network of TRIMOSBUSs are: 1) all of the buses in the network are forced to operate in rigid sequence, since each of the point-to-point controllers will delay completion of any transmission until its point-to-point companion reports completion; 2) each extender may provide a store-and-forward mechanism, allowing the separate TRIMOSBUSs to sequence concurrently. In the latter case, of course, one must provide means to avoid choking the extenders with data and thus causing some form of deadlock.

Arbitration

Although there can be any number of receivers, the TRIMOSBUS design assumes a unique sender for each bus cycle. The task of selecting a single sender from many contenders that may asynchronously request permission to send on the bus is called arbitration. Arbitration must be attended to carefully, since there are a number of pitfalls which can cause low probability system failures that are very difficult to find and correct [3].

In a practical TRIMOSBUS design there are two ways in which sender selection may be done. First, arbitration will not be required if the system design calls for fully sequential operation. This is the case, for example, in systems in which a single CPU sends read requests to multiple memories. Each memory must receive all memory requests and address values, so that it may decide whether or not the request is addressed to it, but the TRIMOSBUS design requires an address assignment scheme that ensures that only a single memory will respond.

On the other hand, there are systems that must have many independent senders: for example, systems with collections of processors operating together, or with channel controllers which communicate independently with memory. In this case asynchronous arbitration using any of a variety of techniques [4] can be used to select a single unique sender. It remains to be shown here only how to adapt such schemes to the protocol of the TRIMOSBUS.

An important and somewhat subtle question is what is the earliest time that the arbiter may safely signal to the next sender that it is authorized to initiate a message. If a message consists of only a single bus cycle, then the arbiter may not designate the next sender until the arbiter has seen the HIGH BEHIND condition that signals the end of the current bus cycle. This is necessary because otherwise there is no way for the arbiter to be sure that the next sender has already seen the beginning of the current cycle (indicated by LOW AHEAD). Thus, if the arbiter sends the designated next sender a signal before seeing the HIGH BEHIND of the current bus cycle, it is possible that the next sender will attempt to initiate a bus cycle concurrently with the current bus cycle. Because the next sender (like all other bus participants) must have recognized the current bus cycle and accepted it before HIGH BEHIND could occur, it is sufficient that the arbiter observe HIGH BEHIND to ensure that the next sender has already recognized the current cycle.

If a message consists of more than one bus cycle, another method will allow the next sender to be designated earlier than the HIGH BEHIND transition of the last cycle of the current

message, thus allowing overlap of data transmission with designation of the next sender. All that is necessary is: 1) that the arbiter observes the HIGH BEHIND condition following the first cycle of the current message before designating the next sender; and 2) that the next sender has a means of identifying the last cycle of the current message. In this way, there can be no ambiguity, since the next sender must recognize the beginning of the current message before it receives from the arbiter the signal designating it as the next sender. This, plus the ability to identify the last cycle of a message, removes all ambiguity.

SECTION IV: DEBUGGING, TESTING AND ERROR CONTROL

The TRIMOSBUS design makes explicit provisions for debugging, testing, and error control. Although a bus terminator is required primarily to provide proper electrical termination for the three sequencing wires and for the data-transmission wires, it is a convenient place to house debugging aids as well. The terminator contains a bus receiver and a short shift register which records a history of recent bus data values. The receiver also provides the ability to "single-step" the bus by refusing to release its clamp on the "previous" sequencing wire until an external signal to the terminator indicates that the bus may proceed. Both the history and single-step functions can be controlled by connecting the terminator to a computer system that provides debugging functions.

The terminator also detects certain types of errors on the bus and reports them to the debugging computer. We have chosen to devote one data wire of the TRIMOSBUS to data parity; the terminator constantly monitors bus cycles to detect and report parity errors. Many kinds of errors on the sequencing wires can also be detected, such as an individual wire going high and then low again without any activity on the other wires. The terminator can also detect prolonged inactivity on the bus and notify the debugging computer that the bus has "timed out". It is necessary to introduce the notion of "time" into the TRIMOSBUS only to detect inactivity; the timeout interval can, however, be made arbitrarily long.

Other bus participants can also help to detect errors. Each one can independently check the parity of the bus, a test which serves to uncover bad sockets or inoperative bus receiver circuits. Also, a participant may discover errors within itself that compromise any further operation. In either case, the error may be reported simply by failing to let the bus sequencing wires advance, thereby causing the terminator to detect a bus timeout. Alternatively, a failed bus participant may remove itself from participation in bus sequencing by ceasing to clamp or drive any wires.

Serial Communication

If the occurrence of an error halts normal bus operation, either because the error has rendered the bus inoperative or because a bus participant has deliberately stalled bus operation, we need to be able to inspect the state of individual system participants by a means independent of normal bus operation. For this purpose, the TRIMOSBUS links all participants

together in a single serial connection shown in Figure 7. This connection is used to shift state information into and out of the participants to which it is connected. A debugging computer can examine this state and modify it if necessary.

The notion of making a chip's state available by a serial connection is not new. IBM has incorporated such an idea into standard integrated circuit design practice [5]. Several manufacturers link printed circuit boards with a shift register to provide a debugging computer access to vital system state [6,7]. What we have done in the TRIMOSBUS design is to define this facility as part of the bus itself.

Although the serial connection itself requires only two additional pins for each bus participant, some mechanism must be provided to sequence the shift registers. Additional controls are also desirable: for example, to read the participant's state into the shift register, or to write the participant's state from the shift register. Separate reading and writing controls greatly simplify testing, as they permit arbitrary values to be inserted in registers and flip-flops that otherwise could not be tested exhaustively. These control functions and the shift register clocking signals are encoded in a highly redundant form on the bus data wires; thus, although the bus may not be fully operational, we assume that most failures will allow it to work well enough to transmit the needed codes. This mechanism, called HHH signaling, is taken up in the next section.

HHH Signaling

When normal bus operation is prevented, it is nonetheless essential to transmit a small number of codes to all system elements. One such code, INIT, is needed to initialize the system and put all receivers in a state in which they are clamping the same sequencing wire. Three additional codes are needed to control the serial line: a command to READ the machine state into the shift register, a command to SHIFT it, and a command to WRITE the machine state from the shift register. These four codes, and possibly others, are transmitted over the data wires by the terminator in a redundant way, so that the failure of any one bus wire or its connectors or receivers will not prevent detection of the code. These wires also carry a "code validity" signal, or "clock", in redundant form. The coding scheme we have chosen requires that the bus have at least nine data wires.

To commandeer the data wires for this debugging function, the terminator drives all three sequencing wires to the inactive

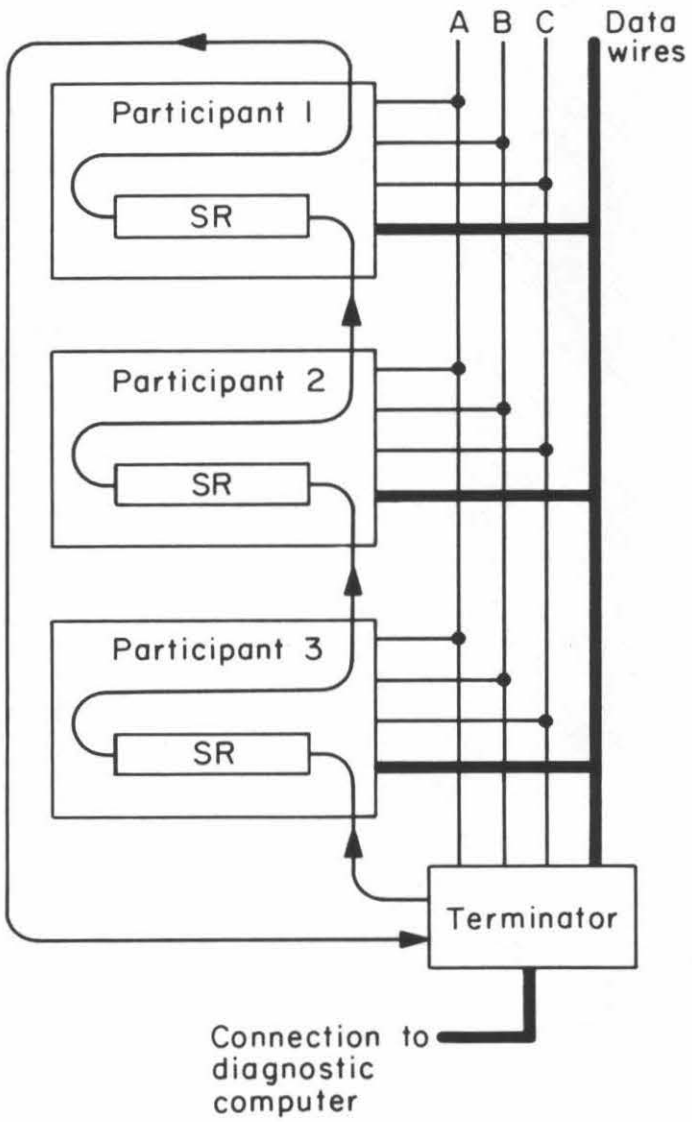

Figure 7: A serial communication line is used to read and write the state of the bus participants for debugging.

state, HIGH in NMOS implementations. This requires substantial drive capability, because various participants in the system may be clamping one or more of these wires to the active state. However, as soon as a participant detects the "all high" (HHH) condition, it removes all drive on sequencing and data wires. Then it looks for codes and clock on the data wires which will control recovery or debugging.

The HHH signaling mechanism is not speed-independent. No acknowledgement is provided by any of the bus participants. Consequently, information must be transmitted slowly enough to ensure that even the slowest receiver responds properly.

SECTION V: MESSAGES

Although the basic signaling conventions of the TRIMOSBUS are sufficient to transmit information from one bus participant to another, these conventions establish no interpretation for these signals. Communicating among elements in a computer system linked by the TRIMOSBUS requires another and higher level of protocol specification. Our choices for this level of design are motivated by our desire to use a TRIMOSBUS to construct systems containing experimental MOS integrated circuits. Although some of these systems may use the bus in a conventional way to link processors and memories, some may wish to experiment with more exotic communication protocols.

Bus Width

Choosing the width of buses in a computer system is a delicate process, for it inevitably constrains the performance that can be achieved by the communication system. In addition to the three sequencing wires, we have chosen to provide ten wires for transmitting information from the sender to all receivers: eight data wires, a "tag" wire, and a parity wire that is set to guarantee odd parity of all ten wires. This choice is primarily driven by pin limitations: we want a design that allows small integrated circuits to connect easily to the bus.

Although HHH signaling requires a minimum bus width of 9, nothing prevents expanding the bus to arbitrary widths. The only difficulty in such expansion is to devise a scheme that allows chips with differing bus widths to be connected together. The store-and-forward bus extenders described in an earlier section could, for example, be used to link buses of different width.

Messages

Bus participants communicate with one another by sending messages, each of which requires a sequence of consecutive bus cycles. Because these messages may be arbitrarily long, the TRIMOSBUS can be used to transmit objects that exceed eight bits in size. Each of the receivers in the system must decode, or "parse" the messages it receives on the bus. Not all messages will be valuable to a particular bus participant, but it must nevertheless inspect each one to determine its relevance.

Although the message mechanism can be put to several uses in a computer system, it will be illustrated by considering the conventional communication between a processor and its memories. A processor will construct a message that contains all the information required for a memory to "write" a new value at a given address, and transmit the message over the TRIMOSBUS. Of all the system components that decipher the message, only one particular memory should recognize the address as its responsibility, and perform the requested write operation.

The message generated by the processor contains three parts: a "herald" that indicates that the message is a "write" command, an address, and a data value. Generally, a transmission of all of these parts would require more than one bus cycle because more than eight bits are needed. A message to write a 16-bit value in a 24-bit address space would typically use a single bus cycle for the message herald, three for the address, and two for the data value.

Correct operation of the system requires that all system elements parse messages according to the same conventions. This requirement does not impose a rigid structure on messages. Rather, the decoder can be viewed as a finite-state machine that takes as inputs the successive data values transmitted, beginning with the message herald. After one or more cycles, the state machine may determine that the current message is of no interest, and wait for a new message to begin. The state machine may also "accept" the message, and cause its bus participant to take action. This acceptance is not to be confused with acknowledging each bus cycle, which must be done in every case.

Proper parsing of messages requires a synchronizing mechanism to ensure that all receivers begin parsing when a message herald is transmitted. This synchronization is achieved by marking the last bus cycle of a message with the "tag" bit. All bus cycles except the last set the tag wire to zero; the last cycle of a message sets it to one. This synchronizes the receivers in a way that makes it easy to construct variable-length messages.

Transferring Sendership

The TRIMOSBUS message conventions do not require that every bus cycle of a message be transmitted by the same sender. Instead, the initial sender can hand control of the bus to the next sender, which in turn may hand control to a third party,

or back to the original sender, etc. The conventions for transferring sendership must be rigidly enforced, for there must always be exactly one next sender.

The transfer of sendership is best illustrated with a "memory read" operation. The processor transmits a message herald that specifies a "read", followed by the address of the memory location to be read. Then the processor falls silent, and relies on the specific memory element that recognized the address to become the sender, so that it may transmit the data value requested as soon as it is available. The memory tags the last bus cycle, in order to terminate the message, and, by convention, sendership reverts to the processor.

The ability to transfer sendership allows very simple systems to be built that require no bus arbitration mechanism. A single bus participant starts all messages, which may be completed by memories or input/output devices that respond with requested values.

Message Arbitration

If the TRIMOSBUS provides communication among a number of asynchronous processors, an arbitration mechanism is required to decide which may use the bus. The arbitration mechanism operates at the message level, i.e., it determines which bus participant may use the bus to transmit the next message, which may consume several separate bus cycles. This approach allows arbitration decisions to proceed rather slowly compared to the speed of bus transmission without limiting bus performance.

It may be necessary to permit a sender to retain control of the bus in order to transmit several consecutive messages without interruption. For example, if multiple processors share a TRIMOSBUS to communicate with memory, the familiar "test and set" operation might require a read message and a write message to occur without relinquishing the bus. Alternatively, the entire operation might be defined as a single message.

The Message Repertoire

Message protocols in the TRIMOSBUS can be designed to meet special needs faced by particular systems. The examples we have used that illustrate a processor communicating with several memories are only simple cases. Generally, the messages will deal with objects and operations that are implemented in

the chips or boards connected to the bus. Communication with functional units such as floating-point arithmetic modules, or "execution contexts" will lead to more sophisticated message formats than we have illustrated. Object-oriented systems such as Smalltalk [8] or SIMULA [9] may use messages that specify an object name and an operation to perform on that object.

The message protocols we have designed all require each receiver to "associate" on some part of the message to decide whether it must act. The processor-memory example requires each memory to decode a memory address and to decide whether it contains the data associated with that address. More generally, a message contains a "name" of an object on which to operate. The object may be the responsibility of more than one bus participant; a memory cache is a simple example in which a data value is stored both in a cache module and in a primary memory module. As another example, one could imagine an aircraft collision-detection system in which the TRIMOSBUS broadcasts positions of aircraft when they change. One of the participants will use this information to update its understanding of the aircraft's new position. Other bus participants will receive the information and compare it to positions of aircraft for which they are responsible, to see if a collision is possible.

The associative nature of the message-parsing process is more suited to the TRIMOSBUS than are explicit physical origin and destination addresses. Specific addresses would fail to exploit the one-to-many transmission offered by the TRIMOSBUS by requiring an explicit destination address. Association also allows dynamic configuration by altering which bus particpants are responsible for which names. Although association may require somewhat more circuitry in a bus participant, the highly integrated circuitry used to implement these modules may render such cost negligible.

It is interesting to note that TRIMOSBUS messages can assume several different roles in conventional computer systems. The bus peforms well enough to be used as a processor-memory interconnection. However, it can connect objects of enough processing ability to be used the way computer networks are now. Thus message repertoires may occasionally mix low-level communication protocols such as memory fetches with others that resemble high-level network protocols [10]. As the level of integration of TRIMOSBUS participants increases, message protocols will look less like a processor-memory connections and more like high-level network protocols.

SECTION VI: A FAMILY OF BUS DESIGNS

In the preceding description of the TRIMOSBUS, we have intertwined our discussion of the TRI and MOS aspects of the design. The basic three-wire sequencing mechanism, the TRIBUS, is applicable to a variety of technologies. Indeed, our prototype controller is implemented in TTL. The discussions of the equipotential assumption, arbitration, and bus extenders are likewise associated with the TRIBUS.

The TRIMOSBUS adapts these ideas for MOS implementation. The most important observation is that MOS implementation makes it easy to allow the bus data wires to be storage nodes. Terminating these wires with negative resistance increases noise immunity and reduces power consumption in all chips except the terminator.

Methods for debugging, testing, and recovering from errors are integrated into the TRIMOSBUS design. The integration is desirable in part because components such as the terminator and the data bus wires can be used both for normal bus operation and for the less frequent interventions. The integration is also desirable to encourage a style of system design that recognizes from the outset the need to deal with errors, debugging, and testing. Clearly, a similar philosophy can be implemented in technologies other than MOS.

The TRIBUS and TRIMOSBUS designs, as discussed here, represent "bus families", rather than completely specified buses. Different circuit designs, different bus widths, different communication distances, different arbitration schemes, and different higher level message protocols can be found that are consistent with the basic ideas expressed here.

SECTION VII: CONCLUSIONS

We have described three levels of specification for a one-to-many self-timed communication bus. These levels provide a framework within which one might describe a variety of specific bus systems. Readers are invited to design their own favorite communication system within the framework.

One-to-many communication in systems which contain parts with different and perhaps unknown response times seems entirely feasible, and in retrospect, fairly simple. The sequence in which all participants in such a communication detect the elements of communication must be consistent. This consistency requirement seems to imply a direct relationship between the physical size of such a bus and its speed. We have satisfied this requirement in the TRIMOSBUS design with the "equipotential assumption"; we assume that the rise time of all signals is slow compared to the maximum propagation delay. We speculate that a rigorous proof of the necessary conditions for safe operation may be found.

Although large systems can be built with point-to-point interconnections between separate TRIMOSBUSs, such systems must either run relatively slowly or be organized so as to localize most communication within the individual TRIMOSBUSs, and to use the extensions relatively infrequently. Thus, we find that digital systems can obtain high speed performance only by careful system organization. This seems to us to be a system level version of the speed requirement handled in individual TRIMOSBUSs with the equipotential assumption. As we have expressed elsewhere [11], the difficulties in contemporary system design stem mainly from communication problems and not from logic design issues. The limitations on TRIMOSBUS performance are similarly related to communication.

ACKNOWLEDGEMENTS

The ideas in this paper came from many sources. To the best of our recollection, recognition of the need for three-wire signaling (TRI) originated in the Washington University group, following an unsuccessful attempt to design a two-wire scheme. The idea of using the bus wires themselves for storage (MOS) arose at Caltech from a suggestion by Chuck Seitz. Fred Rosenberger of Washington University made a number of valuable suggestions.

BIBLIOGRAPHY

1) Institute of Electrical and Electronics Engineers, "IEEE Standard Digital Interface for Programmable Instrumentation," IEEE 488-1975.

2) "Restructured Macromodules," Technical Report 49, Computer Systems Laboratory, Washington University, St. Louis, p. 25, Feb. 13, 1974.

3) T. J. Chaney and C.E. Molnar, "Anomalous Behavior of Synchronizer and Arbiter Circuits," IEEE Trans. TC-22, 4, pp. 421-422, April 1973.

4) C. L. Seitz, "System Timing," Chapter 7 in C. A. Mead and L. A. Conway, Introduction to VLSI Systems, Computer Science Dept., Caltech, in manuscript, 1979.

5) E. B. Eichelberger and T. W. Williams, "A Logic Design Structure for LSI Testability." Proc. 14th Design Automation Conference, June 20-22, 1977.

6) Used by Digital Equipment Corporation in the VAX-11/780 central processor, 1977.

7) Used by Evans and Sutherland in CAORF digital image generator, 1975.

8) A. Goldberg, A. Kay (eds), "SmallTalk-72 Instruction Manual," Xerox Palo Alto Research Center, SSL76-6, March 1976.

9) O. J. Dahl and K. Nygaard, "SIMULA--An ALGOL-Based Simulation Language," Comm. ACM, 9, 9, p. 671, September 1966.

10) R. F. Sproull and D. Cohen, "High Level Protocols," IEEE Proc., 66, 11, p. 1371, November 1978.

11) I. E. Sutherland and C. A. Mead, "Microelectronics and Computer Science," Scientific American, September 1977, p. 210.