THE MULTIBUS DESIGN FRAME

Gaetano Borriello
University of California, Berkeley
Xerox Palo Alto Research Center

Randy H. Katz
University of California, Berkeley

Alan G. Bell
Xerox Palo Alto Research Center

This report is also available from the Xerox Palo Alto Research Center as Technical Report No. ISL-85-7 or P85-00063.

Research supported by the Defense Advanced Research Projects Agency (DoD) under Contract No. N00034-K-0251 and the Xerox Corporation.

THE MULTIBUS DESIGN FRAME

ABSTRACT

This document has been prepared to serve as an introduction to ***design frames***, a new system integration methodology for custom VLSI circuits. Design frames give an applications designer the ability to rapidly embed a prototype custom chip into an actual computer system for debugging and evaluation. They are the hardware equivalent of an operating system, giving a custom chip access to the other components of a computer system.

The report is broken up into three principal sections. The first is a reprint of a paper presented at the 1985 Chapel Hill Conference on VLSI that describes the design frame concepts and the experiments conducted to validate the ideas. The second section of the report is a detailed user's guide to a specific instance of a design frame, the Multibus Design Frame. The last section contains some details of how a Multibus Design Frame chip can be placed in Multibus-based SUN workstation running UNIX 4.2bsd. A designer wishing to use the Multibus Design Frame should find all the information required in these last two sections.

While we have chosen the Multibus due to its versatility and wide availability, the design frame concept can be applied to any system architecture. Therefore, another function of this report is to serve as a description of what is required in generating such a system building infrastructure.

DESIGN FRAMES

A New System Integration Methodology

Gaetano Borriello
Randy H. Katz

Computer Science Division
Electrical Engineering and Computer Science Department
University of California, Berkeley
Berkeley, California 94720

ABSTRACT

Design frames support a new methodology for the integration of custom VLSI chips into computer systems. They give an applications designer the ability to rapidly embed a prototype custom chip into an actual computer system for debugging and evaluation. Design frames are the hardware equivalent of an operating system, giving a custom chip access to the other components of a computer system. With the standard interfaces provided by design frames, a unified approach to chip integration and testing is made possible.

The Multibus Design Frame is an example of such an integration aid for computer systems based on the popular Intel Multibus. Experiments to validate the concepts have been completed at the University of California at Berkeley. Within two semesters, students in an introductory level graduate course in VLSI design were able to complete the design of a microprocessor and have it execute programs while residing in a Multibus-based workstation running UNIX.

1. Introduction

One result of the integrated circuit design methodology of Mead and Conway [MEAD80] is to make VLSI technology available to the system architect - the person with expertise in a specific applications area. It is unreasonable to require that the architect of a speech recognition chip be both an expert in signal processing and chip manufacturing. The details of how to realize a physical chip from its mask specification represent an implementation obstacle for the system designer. To make VLSI more available as a system implementation technology, such details must be encapsulated and the mechanics of fabrication presented as an abstract service.

Implementation services such as USC's Information Sciences Institute [LEWI84], reduce the complexity of system construction by encapsulating implementation details in the services they provide. The designer provides the mask specification; the implementation service understands the details of the chip manufacturing process, including how to have masks made, dies produced, and packaged chips delivered to the designers [COHE81, MOSI84].

A functional hardware system typically requires more than a single isolated chip. It must be surrounded by interface and other support circuitry to enable access to system components, such as memory, input/output devices, network controllers, etc. The chip must first be mounted on a printed circuit board, which in turn is inserted into the system backplane. To assist designers in prototyping their own systems, USC-ISI now provides a printed circuit board implementation service [LEWI85], and to support large quantity production, a facility for the screening of fabricated chips.

While implementation services help the designer with the manufacturing aspects of the system design task, they provide no assistance when it comes to embedding a custom chip within a system. An environment similar to that provided to programmers by an operating system is needed for the VLSI system designer. In the early days of programming, applications and systems programmers were one in the same. Each programmer provided all the support routines required for his/her application, including any special functions (e.g., sine and cosine) and system support services (e.g., I/O device handlers). Obviously, effort was duplicated and there was little sharing of developed software.

By providing standard routines for accessing system services, operating systems have evolved to provide programmers with the necessary tools for rapid system prototyping. Changes to hardware or software components are localized to the relevant portion of the operating system, and do not affect user programs. While the lack of operating systems and run-time environments did not prevent the development of programs. Their development allowed more people to write better software more efficiently.

2. Design Frames

Design frames support a new methodology for VLSI system construction. They are like operating systems in that they provide standard interfaces to system components. By adhering to standard interfaces, experimental hardware can be prototyped more rapidly and be made accessable to a larger community [CONW83]. A chip-level system may need access to some of the other devices that comprise the computer system. We would like the designer to avoid having to understand every detail of the hardware system into which the chip is to be embedded.

The design frame, at the conceptual level, is a system structure that provides an interfacing service between the designer's circuit and the system bus. It implements the details of how to interface to the bus: the bus protocols and the signal semantics, timing requirements, electrical characteristics, and locations within the backplane. The design frame consists of elements at the chip, board, and software levels. On-chip circuitry implements a simple interface protocol for use by the designer's circuit to communicate with other system components to which it interfaces through the system bus. This is merged together with the designer's circuit to form a single chip specification. The fabricated chip is inserted into a printed circuit board that has been designed to accept the chip's footprint and has the appropriate formfactor for the target system bus (see Figure 1). Software drivers are used to access the chip from the computing environment of the system.

The conventional method for integrating custom chips into systems is to create new interface circuitry in each instance. The integration task can require an effort comparable to the design of the chip. Usually, the resulting system interface cannot be reused for other custom chips. The RISC processor is a typical case [PATT84]. To embed it in an environment and provide it the memory and I/O subsystems it needs, requires a custom designed processor board and circuitry consisting of 40 TTL chips. Each new integration attempt would require a new interface design.

Design frames enhance the reusability of the design by supporting standard interfaces. The printed circuit board component of the design frame is designed once, and is used for all circuits residing within that frame. Design frames for different system busses can support the same internal protocol, enabling designs to be moved to a new system context without redesign. They encourage sharing of printed circuit boards, circuit designs, and entire subsystems throughout a much larger design community than previously possible.

Design frames enable a larger community of designers to create hardware systems incorporating VLSI components. By using design frames, designers can quickly experiment with their ideas realized as physical devices. An ability to rapidly construct VLSI systems supports an experimental method much more akin to software system design than the current methods employed for hardware systems. Design frames provide a framework within which VLSI modules can be prototyped, debugged, and improved while functioning in a realistic environment. The large amount of effort needed to integrate individual pieces into the system is replaced by a single, more modest, and reusable effort of generating the appropriate design frame for the target system environment.

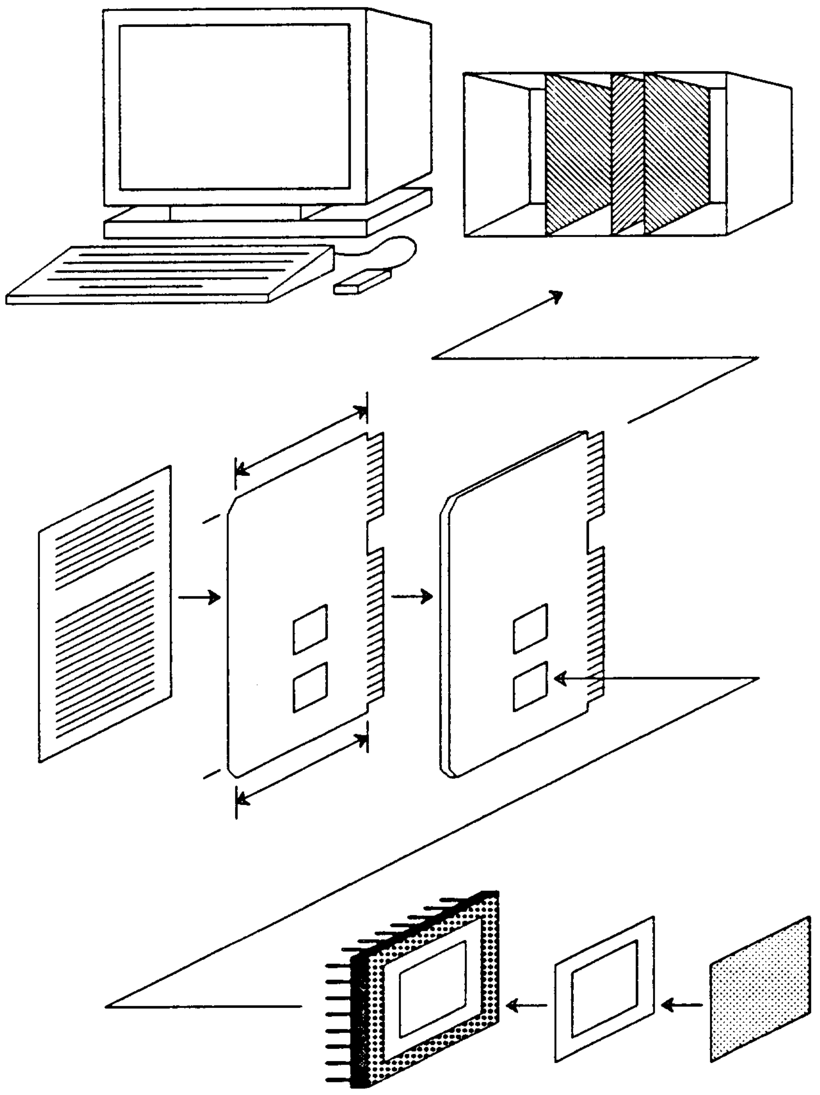


Figure 1. The Multibus Design Frame has two hardware components, a chip-level frame and a board-level frame. A user circuit specification is merged with the chip-level frame and fabricated as a single chip. Another specification for user board circuitry is merged with the board-level frame an a printed circuit board produced. The finished board and chip are then inserted into a Multibus-based computer system.

3. Multibus Design Frame

An operational design frame has been developed for the Multibus [BORR85]. It provides a direct interface between the designer's circuit and the Multibus. It consists of on-chip interface circuitry, a standard pad frame, a printed circuit board designed for the Multibus that accepts the chip-level design frame, and software to make the chip's functions accessible to the rest of the computer system. If an experimental processor or input/output device is designed within the context of the Multibus Design Frame, it can be integrated into a Multibus-based system easily and begin to request or provide services immediately. Custom circuits now have access to a wide variety of system services implemented as boards for standard system busses: single-board computers, memory subsystems, and a large number of input/output device controllers.

The Multibus was chosen because it is a popular system bus for microprocessor based systems readily available throughout the research community and industry. Hundreds of off-the-shelf system building blocks are available for the Multibus [IRON84]: processors, memory, disk controllers, display controllers, etc. By using the Multibus Design Frame the designer's circuit itself becomes a system building block.

3.1. Multibus Overview

The Multibus is an asynchronous bus that uses four-cycle handshaking. There is a byte-addressed memory space addressed by a 20-bit address bus and a separate I/O address space using only 16 bits of the address bus. There are two types of devices on the Multibus: masters and slaves. One master at a time can control the bus. Masters issue request for reads or writes to be performed by either I/O or memory slaves. Slaves never control the bus but simply service the requests of the masters. The Multibus provides a protocol for the exchange of bus control between masters and a choice of priority schemes for arbitrating any conflicts. The use of an asynchronous protocol allows many different types of subsystems to coexist comfortably in the same environment.

The Multibus specification is a 100-page document detailing interrelated mechanical, electrical, logical, and timing constraints that must be satisfied in designing a Multibus subsystem [INTE82].

3.2. Chip Level Design Frame

The Multibus Design Frame is implemented using nMOS technology with a feature size of four microns. The design uses no butting or buried contacts for enhanced fabrication line independence. There are no dynamic storage nodes that would cause a designer to be concerned with clocking limitations imposed by the design frame circuitry.

The external interface at the chip-level is directly compatible with the Multibus specification. The input and output pads of the design frame have the proper electrical characteristics to drive the Multibus backplane directly. This eliminates the need for any "glue" chips to provide special drive capability.

The internal interface of the chip-level frame is considerably different than the Multibus, and has a high degree of uniformity. It was designed and optimized for MOS digital design, unlike the Multibus which is optimized for TTL circuitry and subsystems built on boards rather than chips.

The internal interface is completely synchronous. A two-phase non-overlapping clock is provided by the frame for the use of the user's circuitry. The frame provides configuration control for the clock with the use of a novel semi-digital circuit developed at Xerox PARC [BELL83]. By the use of two external frequency inputs, the period and duty-cycle of the clock can be controlled over a wide range. Synchronizers are provided by the design frame to make all internal signals synchronous to the two-phase clock.

The input and output pads provide automatic latching of the data and address busses. The user circuitry does not need to provide any special circuitry for these functions which typically consume five to fifteen chips on most Multibus boards. The busses are uni-directional; the decision is left to the designer whether to make them bi-directional.

A circuit built inside the Multibus design frame can act as either a master or slave and change this behavior dynamically. The circuit can be placed in either memory or I/O space or be made to issue commands to devices in either space. Address comparison logic built into the frame allows the user circuitry to respond to addresses in an arbitrary segment of the address spaces.

The frame also provides all the bus arbitration logic needed in order to negotiate for control of the bus with other masters which may be present. When the user circuit issues a read or write command the request for the bus is automatically generated and arbitration for control commenced. The user circuitry simply sees an acknowledge signal when the entire operation has been completed. Therefore, instead of a Multibus-like four-cycle asynchronous handshake, the user circuit uses a simple synchronous request/acknowledge exchange (see Figure 2). Special mechanisms are also provided for issuing many commands atomically (i.e. without relinquishing control of the bus in between) and being notified of requests for bus use by other masters.

Circuitry to handle non-vectored interrupts is present in the design frame. A slave built inside the frame has its interrupt status register implemented by the design frame. All polling and interrupt clear requests are handled directly by the frame, the user circuitry just signals the interrupt.

To provide a uniform interface, all the internal frame signals are on the same layer (polysilicon). All input signals to the frame present a uniform load to the user circuitry and all frame outputs have the same drive capability. This uniformity decreases the likelihood of errors due to interface idiosyncracies of particular signals.

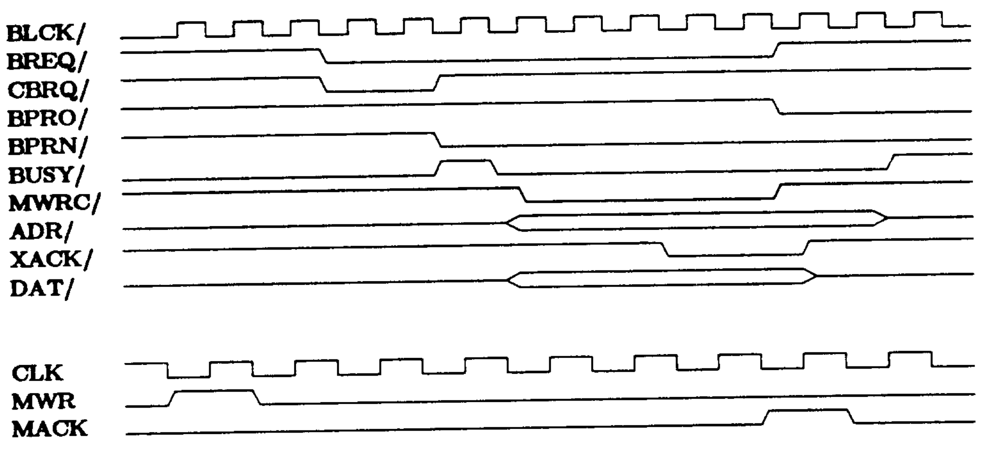


Figure 2. Shown above are the traces for a master write request. The bottom three traces represent the internal interface of the design frame to the user circuitry. The others are the Multibus signals affected by the request. The internal interface is much simpler, especially when one considers the timing and electrical constraints not shown here.

3.3. Board Level Design Frame

The Multibus Design Frame board is designed with the proper formfactor for the Multibus and with a footprint to accept the packaged 84-pin chip. The design frame uses 58 of the 84 pins and leaves the rest for the designer's use. A standard board is available which has a large wire-wrap prototyping area for any special user circuitry which could not be integrated into the chip. However, a designer is free to layout his own printed circuit board patterns and place these within the CIF of the board starting frame which specifies the board formfactor, connector positions, and technology to the manufacturer.

The board has sockets for the crystal oscillators that control the internal two-phase clock. There are jumpers to be placed so that one of the eight interrupt lines of the Multibus can be connected to the chip. A board reset switch allows reinitialization of on-chip circuitry without having to reset the entire system.

The Multibus Design Frame board is sparse, with substantial prototyping area for the designer's use. Most commercially available prototyping boards use more board area while providing only limited slave capabilities. They do not provide any master capabilities, nor do they provide the possibility of manufacturing the prototype board in large quantities.

3.4. Software Drivers for the Design Frame

Software drivers are provided to access Multibus locations in memory and I/O address spaces from the UNIX operating system. Application programs, written by the designers of the chips, use these drivers as low-level subroutines for accessing the chip from the operating system of the workstation. The system used for prototyping is a Multibus-based SUN Microsystems workstation that runs UNIX.

4. OPUS: a Microprocessor for the Multibus Design Frame

The methodology was validated in the Fall 1983 offering of the Mead/Conway design course "CS 250: Introduction to VLSI Systems" at the University of California at Berkeley. The students were given the opportunity to design, verify, have fabricated, and test a real system component within a two semester course sequence. Thirteen projects were designed for the frame. Ten were implementations of the same microprocessor architecture, based on a hybrid of the HP2100 and the PDP-8. Students had the valuable opportunity to compare their design against those of other students. Additional projects included a pipelined multiplier, a special "configuration controller" for setting up the decode addresses for a board with several Multibus Design Frame chips, and a general purpose block memory mover that is to serve as a portion of the SOAR (SmallTalk on a RISC) processor board.

The course ran from August to December 1983. While the first half concentrated on teaching the Mead/Conway methodology, the remainder was devoted to the projects. The most successful design completed during the semester course was a 16-bit microprocessor (dubbed OPUS by its designers and shown in Figure 3). Of the four students involved in the OPUS design, David Wood and Richard Rudell are computer science students, only one, Joe Pierret, is an IC student, and Tim Mills is working in bioengineering. A fully simulated version of the OPUS processor, embedded in the design frame, was submitted for fabrication in early January 1984. In parallel with the fabrication of the OPUS chip, the design frame printed circuit board was fabricated and received in early March. A test structure was used to debug the chip and board level design frames in the SUN workstation. Packaged OPUS chips were received in April, placed on the Multibus Design Frame board (see Figure 4), and integrated into the SUN workstation. They were found to be fully functional (executing many sample programs) at the expected speed. Thus, within two semesters, the student designers had a working microprocessor executing programs within a real hardware environment.

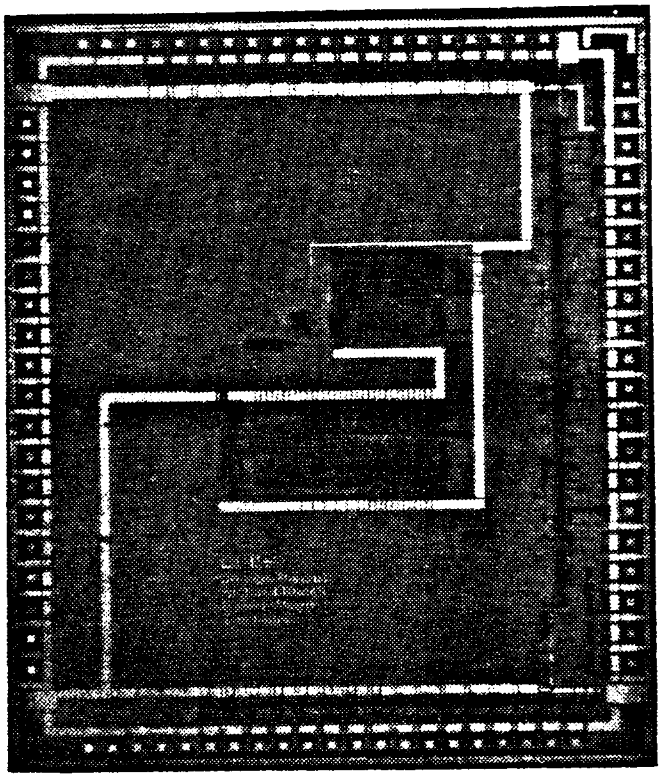


Figure 3. A photograph of the OPUS microprocessor chip. The design frame circuitry is visible around the periphery of the chip. The designers' names are in the bottom center of the chip.

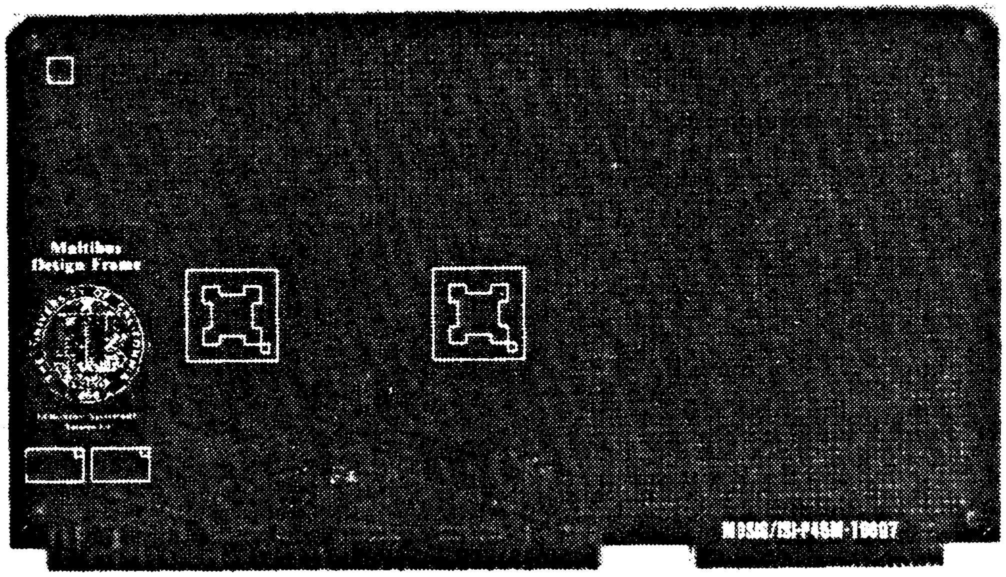


Figure 4. The Multibus Design Frame printed circuit board allows for two chips to be placed on the same board. Most of the board area is for additional user circuitry.

5. Design Integration and Testing

The standard interfaces of design frames allow fast system integration and evaluation of a design within its operating context. However, this is only one of the advantages of being able to place a chip within a computing environment quickly. Since the chip is integrated into an actual computer system the full power of that system can be used to not only exercise and evaluate the design, but also for testing purposes.

Scan path techniques can be utilized within the design frame to provide a designer with access to the internal interface of the design frame [WILL83]. Testing procedures can then use the same test vectors used in simulation of the designer's circuit, bypassing the frame circuitry entirely. Cells for adding elements to the scan path permit the designer to access other internal signals.

A special purpose testing chip, also built inside the design frame, provides software support for designer access to these signals [BROW84]. This chip resides on the same board as the chip under test and acts as a scan-path controller. Commands to the controller chip sent over the system bus exercise the scan path and resulting values are read back by the test software.

In the case of the Multibus Design Frame this scan controller chip finds an additional use. There are many frame configuration signals for the design frame that need to be set at start-up time. In addition, there is the matter of setting the address range to which the chip will respond. A straight-forward approach would waste valuable pins to bring in this configuration information from external switches. Using scan-path techniques this task is greatly simplified. All the configuration signals are placed along the scan-path and are loaded by the scan-controller chip through software control. This gives the designer increased flexibility in the design of the circuitry, as now software configuration control is also possible.

The scan-controller chip will be utilized by all the users of the design frame. This chip can afford to have pins dedicated to external switches for address configuration control. Through the use of design frames the designer is also supplied with a complete test fixture for all the chips that may be prototyped. The scan-controller continues to be useful after a design is completed as a general-purpose initialization and diagnostic device. A unified approach to system integration and testing is now possible where only ad-hoc methods were previously used.

6. Design Frame Evaluation

There are many important criteria for the evaluation of a design frame. One of these is the flexibility of the design frame interface in dealing with a wide variety of circuits. The frame must not only have a general-purpose interface but it must also be easy to understand. Our experience in using the Multibus Design Frame in two VLSI courses has caused us to make some changes to our original specification. For example, initially the handshaking protocol used a four-cycle synchronous handshake. This was found to be wasteful in the number of states required in a finite-state machine that would implement the protocol and changed to our current two-cycle synchronous scheme.

If design frames are to be used in production environments the overhead associated with using the design frame must be small. The design frame circuitry lies between the system bus and the designer's circuit, introducing additional delays through the frame's logic circuitry. Obviously a custom tailored interface to the system bus could have better performance. Operating systems are very similar, if the cost of using the services is not kept small, then designers will not make use of them for their final product.

There are two overhead parameters of interest, the time required to take control of the bus and the time to complete a read or write transaction. There is negligible overhead associated with obtaining control of the Multibus. The design frame circuitry, as regards this function, is completely asynchronous to the clock of the chip and any delay is due strictly to arbitration among bus masters.

According to the Multibus specification, the minimum theoretical transaction time is equal to the sum of the access time of the device, the time to latch the data being read, plus a minimum of 100ns for bus arbitration and address set-up and hold times (see Figure 5). Although the Multibus is an asynchronous bus, the Multibus Design Frame, assumes that the circuit residing within it will be synchronous. The frame has built-in synchronizers and a finite-state-machine-like internal interface. This circuitry causes a minimum transaction time to be larger than the theoretical limit of the Multibus. The difference is approximately three chip clock cycles. Although this may seem substantial, any synchronous circuit that interfaces to an asynchronous bus must perform some synchronization functions. Also, some delay is involved in the internal control that generates the commands for the design frame to execute. The actual cost of using the design frame is thus reduced to only one or two cycles above the minimum that can be achieved in practice. With a typical access time to off-the-shelf memory subsystem on the order of 300 to 500ns this translates to an overhead of less than 20% (assuming that a clock cycle is 100ns).

For an asynchronous circuit, the current design frame would clearly be inappropriate. An asynchronous device could communicate on the Multibus at a rate close to the maximum bandwidth (if the circuit is fast enough). A design frame with clocking circuitry tailored for this type of device could achieve a negligible overhead. The clocking subsystem used in the Multibus Design Frame can be easily modified to yield a clock that can be idled while the circuit is awaiting an acknowledgement of its transaction over the system bus.

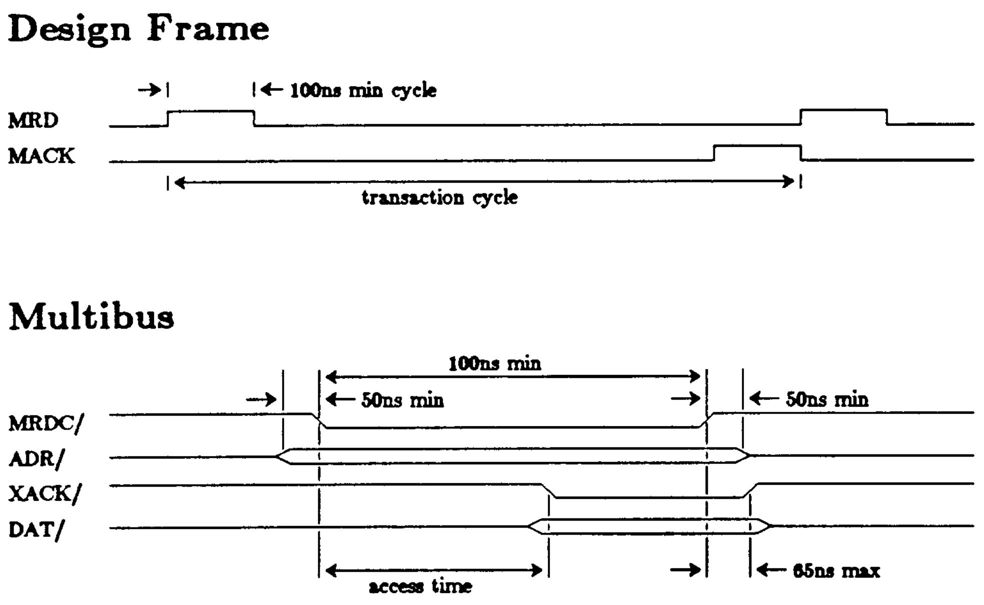


Figure 5. The design frame imposes an overhead cost on a Multibus transaction. This is due the circuitry required to transform the asynchronous interface of the bus to the synchronous interface to the custom circuitry. For a chip clock cycle time of 100ns this cost is less than 20% of the practical limit.

7. Future Work and Conclusions

Work is underway to make the scan controller chip and testing mechanism a reality. A scan-controller chip is currently being designed along with a wide variety of scan-path cells for a designer to place around and within the custom circuitry to be debugged. Software for utilizing this subsystem also needs to be developed.

A CMOS version of the Multibus Design Frame has been designed by a group of four students at Berkeley and will soon be fabricated. Design frames for a wider variety of system environments are also needed. For example, for high performance system busses, for interconnecting highly parallel machines, and for pipelined, serial interconnections for signal processing applications. The design of each of these frames should be general enough to satisfy the needs of that environment. In this way, a small number of standard frames can evolve that provide designers with a wide choice of system integration alternatives.

Design frame are a new methodology for the rapid development of systems and system integrable custom parts. They package the many idiosyncratic details that must be understood before a system integrable design can be realized as abstract services. A design frame for the Multibus has been designed and validated within a university environment. Documentation and design files for the Multibus Design Frame are available through the authors.

8. Acknowledgements

The Multibus Design Frame is not the only example of the system integration methodology we described. Alan Paeth at the Xerox Palo Alto Research Center designed a Basic Design Frame that allowed students to design projects that used a set of three switches as inputs and seven LEDs as outputs [PAET83]. Henry Fuchs at the University of North Carolina used this design frame in VLSI courses and demonstrated many successful student projects. The Basic Design Frame is now available through MOSIS [NEWK83]. A microprocessor design frame was designed at the University of Wisconsin at Madison [KATZ82, KATZ83]. Designers built chips that were pin compatible replacements for an Intel 8086 processor on a single board computer system.

We gratefully acknowledge the help of the students of the VLSI design class offered at U.C. Berkeley in the fall semester of 1983 and 1984, who acted as test subjects for our ideas, and the support of the Xerox Palo Alto Research Center and the Defense Advanced Research Projects Agency, for making this research possible, and Lynn Conway and Alan Bell for many contributions to the ideas presented in this paper.

Research supported by Defense Advanced Research Projects Agency (DoD) under Contract No. N00034-K-0251 and the Xerox Corporation.

9. References

[BELL83] Bell, A. and G. Borriello, "A Single Chip nMOS Ethernet Controller," ISSCC Digest of Technical Papers, 1983.

[BORR85] Borriello, G., R. Katz, and A. Bell, "The Multibus Design Frame User's Guide and Specification," Xerox Palo Alto Research Center and University of California at Berkeley joint technical report, to appear.

[BROW84] Brown, R., "Implementation of a One-Chip Scan-Based Test System for the Multibus Design Frames," Master's Report, Computer Science Division, University of California at Berkeley, December 1984.

[COHE81] Cohen, D., G. Lewicki, "MOSIS: The ARPA Silicon Broker," Proceedings of Second Caltech Conference on VLSI, Pasadena, CA, January 1981.

[CONW83] Conway, L., A. Bell, "System Kits, Design Frames, and Network Services for the Rapid Prototyping of Advanced Computer Systems," Xerox PARC Working Paper, January 1983; also presented at the Australian Microelectronics Conference March 1983.

[INTE82] Intel Corporation, Intel Multibus Specification, 1982.

[IRON84] Ironoak, The Multibus Buyer's Guide, Winter 1984.

[KATZ82] Katz, R. H., S. Weiss, R. Kelkar, "An Experimental Design Frame for VLSI Circuit Prototyping," VLSI Design, May/June 1982.

[KATZ83] Katz, R. H., S. Weiss, "A Standard Design Frame for VLSI Circuit Prototyping," J. of VLSI and Computer Systems, Computer Science Press, V 1 N 1, Spring 1983.

[LEWI84] Lewicki, G., D. Cohen, P. Losleben, D. Trotter, "MOSIS: Present and Future," Proceedings of Conference on Advanced Research in VLSI, Cambridge, MA, January 1984.

[LEWI85] Lewicki, G. and D. Cohen, "Chips and Boards for Systems Through MOSIS," CompCon, San Francisco, CA, February 1985.

[MEAD80] Mead, C., L. Conway, Introduction to VLSI Systems, Addison-Wesley, Reading, MA, 1980.

[MOSI84] The MOSIS Project, "The MOSIS System (what it is and how to use it)," ISI Technical Report, ISI/TM-84-128.

[NEWK83] Newkirk, J., R. Matthews, eds., The VLSI Designer's Library, Addison-Wesley, Reading, MA, 1983.

[PAET83] Paeth, A., "The Design of a Basic Design Frame," Xerox PARC Working Paper, May 1983.

[PATT84] Patterson, D., "VLSI System Building: A Berkeley Perspective," Proceedings of Conference on Advanced Research in VLSI, Cambridge, MA, January 1984.

[WILL83] Williams, T., K. Parker, "Design for Testability - A Survey," Proceedings of the IEEE, Vol. 71, No. 1, January 1983.

Multibus Design Frame Specification

and

Users' Guide

Table of Contents

1. INTRODUCTION

The Multibus Design Frame consists of two hardware components: the chip-level frame and the board-level frame. The chip-level design frame concerns itself with the electrical, timing, and logical requirements of the Multibus. The board-level design frame deals with the electrical and mechanical constraints of the Multibus.

The Multibus Design Frame provides the integrated circuit designer with a greatly simplified interface to the Multibus. Rather than dealing with all the details and idiosyncracies of this system bus, the designer has these interfacing functions performed by the design frame.

The chip-level frame contains:
- pads with the appropriate electrical characteristics to interface to the Multibus,
- a standard finite-state-machine-like clocking convention,
- circuitry to synchronize all signals to the user's system clock,
- latches for the data and address busses,
- logic circuitry to handle data transactions between system components,
- and a large area into which the designer can place his circuit.

The board-level design frame provides the designer with:
- a Multibus board with the proper formfactor,
- two zero-insertion-force sockets to accept chip-level frames,
- a capability for clock customization,
- a board reset switch,
- and a prototyping area for user circuitry.

Design files for the nMOS circuitry of the chip and board level design frames are available on the March 1985 Berkeley VLSI Tools Distribution tape. This is a complete suite of VLSI design tools, developed at the University of California at Berkeley and elsewhere, that run under UNIX 4.2bsd. A copy of this tape and complete distribution information can be obtained from:

Professor John Ousterhout
Computer Science Division
University of California, Berkeley
Berkeley, California 94720

The design file for the chip-level design frame is compatible with the Magic VLSI design system. The board-level frame is described in CIF format and is provided only to serve as an example of a printed circuit board specification.

Multibus Design Frame have been fabricated through DARPA's MOSIS silicon brokerage and the printed circuit boards through PCBIS. Both of these services are implemented by the University of Southern California's Information Science Institute. The chip circuitry and boards have all been fully tested. Anyone interested in obtaining a printed circuit board for the Multibus Design Frame should contact:

Gaetano Borriello
Computer Science Division
University of California, Berkeley
Berkeley, California 94720

2. FUNCTIONAL DESCRIPTION

2.1. Introduction

This section provides an overview of the operation of the Multibus Design Frame. It is concerned with the interface between the user circuit and the chip-level design frame. A brief overview of the types of Multibus components is provided. This is followed by a description of the capabilities of the design frame and a detailed description of the interface signals presented to the designer. This section ends with a description of typical behavior of the design frame during various bus functions.

2.2. Notation and Terminology

We use a consistent notation for signals throughout this document. An active low signal is indicated by a "/" appended to the signal name. For example, a signal name such as MRD is active high, a signal name such as INIT/ is active low.

In the standard use of the Multibus, a bus module can either be a *master* or a *slave.* Only a bus master can initiate bus transactions, while a bus slave can only service a transaction initiated by a master. The Multibus Design Frame does not impose any choice between these two modes of operation on the user circuit. At a particular moment in time the user circuit can behave as a slave and, at some later time, behave as a master. Signals which are meant to be used when the circuit is a bus master are prefixed by "M" and those for slave operation by "S".

The Multibus has two separate address spaces, one for memory and one for input/output devices. The memory address space is 1Mbyte (or 16Mbyte if 24 address bits are used). The I/O address space is 64Kbytes and only uses the low-order 16 bits of the address bus. At any given time, a device in the Multibus Design Frame can be considered to be a slave within memory space, a slave within I/O space, a master addressing memory space, or a master addressing I/O space. The range of address values within a given address space to which a slave will respond is called its address range. A slave responds to a read or write request generated by a master. Only slaves responding to an address range in I/O space may interrupt a master.

2.3. Multibus Specification Compliance

Multibus transactions can take place in one of two address spaces, one for memory and one for input/output devices. The Multibus Design Frame allows the use of either as well as the ability to change from one to the other dynamically. The memory address space is 20 bits wide with support for expansion to 24 bits, the I/O address space is 16 bits. Although the Multibus is designed for both 8-bit and 16-bit data transfers we have chosen to support only 16-bit data transactions in both address spaces. Recently designed Multibus-based systems universally support 16-bit transactions. The Multibus is byte-addressed, but due to the choice of 16-bit transactions the design frame uses one less address bit and addresses only 16-bit quantities. Interrupt handling is assumed to be non-bus-vectored. While they may differ greatly on the handling of the more complex bus-vectored interrupt schemes, all Multibus systems support the non-bus-vectored interrupt capability.

MULTIBUS COMPLIANCE: Master/Slave D16 M20 I16 V0 L (only 16-bit/ no 8-bit)

2.4. Design Frame Signal Classes

The internal signals of the chip-level design frame can be grouped into several classes based on their functions. These classes are:

- Clocks and Initialization Lines,
  PHI1, PHI2, INIT
- Slave Operation Lines,
  SSETM, SSETIO, SRD, SWR, SACK, SEN, SDIS
- Master Operation Lines,
  MSETM, MSETIO, MRD, MWR, MACK
- Extended Bus Control Lines,
  GETB, ORQST, RELB
- Interrupt Lines,
  SI, MI
- Data Lines,
  DATI0-DATI15, DATO0-DATO15
- Address Lines,
  ADRI1-ADRI19, ADRO1-ADRO19
- and Address Comparison Lines.
  ADRM0-ADRM3, ADRDM0-ADRDM3, ADRDIO0-ADRDIO3

NOTE: All unused signals should be tied low unless otherwise stated.

2.4.1. Clocks and Initialization Lines

2.4.1.1. System Clocks (PHI1 and PHI2). A two-phase non-overlapping clock to be used in the user circuit. All interface signals are synchronous to this clock when used in a typical finite-state-machine-like fashion. Inputs are expected to be valid on PHI1 and outputs should only be changed on PHI2. This convention hold for all design frame signals. Provisions for configurability at the board-level allow for specification of the duty-cycle and period of these clocks (see Section 3.2).

2.4.1.2. Initialization (INIT). A chip initialization signal asserted for one cycle. It is used as a system wide signal at power-up to insure that the system starts in the proper state or whenever system initialization is required. A push-button switch is available on the Multibus Design Frame board to generate a local board INIT signal to initialize the board but not the entire system.

2.4.2. Slave Operation Lines

2.4.2.1. Slave Address Space Set-up (SSETM, SSETIO). These signals are used to set-up slave operation in either the memory or I/O address space. They are asserted for one cycle (min) by the user circuit. If the capability to switch dynamically is not required, one can be tied high and the other low.

2.4.2.2. Slave Read (SRD). Asserted by the frame for one cycle when a bus read command is issued to an address in the range and space (memory or I/O) recognized by the chip. It follows the transaction protocol defined below with SACK.

2.4.2.3. Slave Write (SWR). Asserted by the frame for one cycle when a bus write command is issued to an address in the range and space (memory or I/O) recognized by the chip. It follows the transaction protocol defined below with SACK.

2.4.2.4. Slave Acknowledge (SACK). Asserted by the user circuit for one cycle (min) to signal the completion of either an external master initiated read or write request. It follows the transaction protocol defined below with SRD and SWR.

2.4.2.5. Slave Operation Enable and Disable (SEN, SDIS). These signals are used to enable or disable slave operation. They are asserted for one cycle (min) by the user circuit. SEN is used to set-up normal slave operation, SDIS is used to prevent the frame from responding to any Multibus commands. If the capability to switch dynamically is not required, one *must* be tied high and the other low.

2.4.3. Master Operation Lines

2.4.3.1. Master Address Space Set-up (MSETM, MSETIO). These signals are used to set-up master operation in either the memory or I/O address space. They are asserted for one cycle (min) by the user circuit. If the capability to switch dynamically is not required, one can be tied high and the other low.

2.4.3.2. Master Read (MRD). Asserted by the user circuit for one cycle (min) when it wants to issue a single-word bus read command to the address space set-up by MSETM and MSETIO. It follows the transaction protocol defined below with MACK.

2.4.3.3. Master Write (MWR). Asserted by the user circuit for one cycle (min) when it wants to issue a single-word bus write command to the address space set-up by MSETM and MSETIO. It follows the transaction protocol defined below with MACK.

2.4.3.4. Master Acknowledge (MACK). Asserted by the frame for one cycle to signal the completion of a user circuit initiated read or write request. It follows the transaction protocol defined below with MRD and MWR.

2.4.4. Extended Bus Control Lines

When performing a single-word bus transaction using MRD/MWR, bus control is relinquished at the end of the transaction. These extended bus control lines are used by a user circuit that may need to perform a series of bus transactions atomically (i.e. not interrupted by any other bus masters). Such a capability is used, for example, multi-word data transfers.

2.4.4.1. Get Bus Control (GETB). Asserted by the user circuit for one cycle (min) to have the frame initiate a bus request sequence.

2.4.4.2. Have Bus Control (HAVEB). Asserted by the frame for as long as the user circuit is the master in control of the bus. This is true whether the bus was acquired due to a standard transaction request (MRD or MWR) or by GETB.

2.4.4.3. Other Bus Request (ORQST). Asserted by the frame for as long as another master is requesting control of the bus while the user circuit is in control. This is true whether the bus was acquired by a standard transaction request (MRD or MWR) or by GETB. This signal can be used to decide to relinquish control of the bus before originally planned.

2.4.4.4. Release Bus Control (RELB). Asserted by the user circuit for one cycle (min) to have the frame release control of the bus after the current transaction is completed.

NOTE: A bus acquisition sequence is initiated whenever the user circuit asserts either MRD/MWR or GETB. Unlike MRD/MWR, a GETB request does not immediately generate a bus transaction request. After the bus transaction initiated by MRD/MWR is completed bus control is relinquished, unless GETB was asserted some time between the MRD/MWR and the completion of the transaction. One should assert GETB either before or at the same time as the first MRD/MWR.

2.4.5. Interrupt Lines

Only I/O devices can interrupt a bus master. The Multibus provides for eight interrupt lines, with any number of devices tied to the same interrupt line. The board-level design frame provides a way to select one of these eight lines to be the board interrupt line. A method is provided for polling the devices (tied to the same line) to determine which caused the interrupt. The master that receives the interrupt performs a read to that device to determine the contents of its interrupt status register. Once it finds the slave responsible for the interrupt, it executes the appropriate actions for that interrupt. The master then clears the interrupt by writing into the interrupt status register of the device. The frame provides an interrupt status register whose low-order address bits are all ones (i.e. the high-order location of the slave's address range, this cannot be used by the user circuit) (see Address Configuration Lines below).

2.4.5.1. Slave Interrupt (SI). Asserted by the user circuit for one cycle (min) to signal an interrupt to the current bus master. The user circuit must be operating as a slave in I/O space. The frame has logic circuitry to handle the interrupt status requests generated by the master to poll the possible interrupting devices. The frame will also clear the interrupt when requested to do so by the master. The user circuitry need only generate the interrupt. No other action on the part of the user circuit is required.

2.4.5.2. Master Interrupt (MI). Asserted by the frame to signal that another device wishes to interrupt the user circuit. Only asserted while the user circuit is master of the bus. The user circuit is then responsible for polling the devices to determine which was interrupting and for clearing the interrupt. This signal is a level rather than a one-cycle pulse so as to allow the handling of multiple devices using the same interrupt line.

2.4.6. Data Lines

Data lines are unidirectional. Although the Multibus specification allows for devices that are byte-oriented the Multibus Design Frame allows only for word-oriented accesses.

2.4.6.1. Data In (DATI0-DATI15). Lines to get data into the user circuit during a slave write or a master read. DATI0 is the least significant bit.

2.4.6.2. Data Out (DATO0-DATO15). Lines to get data off the user circuit during a slave read or a master write. DATO0 is the least significant bit.

NOTE: Designs that require a single set of bidirectional data lines can easily adapt by providing a bidirectional buffer cell controlled by an enable line, that has two unidirectional inputs and a bidirectional output.

2.4.7. Address Lines

Address lines are unidirectional. Note that inside the frame Multibus addresses lose the low-order bit since it is used for byte addressing (which is not supported). If a user-circuit is dealing with memory space then all (19) of the address lines should be used. If, however, the user circuit operates in I/O space then only the low-order 15 of the 19 lines are required.

2.4.7.1. Address In (ADRI1-ADRI19). Lines to get the address into the user circuit during slave operations. ADRI1 is the least significant bit.

2.4.7.2. Address Out (ADRO1-ADRO19). Lines to get the address off the user circuit during master operations. ADRO1 is the least significant bit.

NOTE: Designs that require a single set of bidirectional address lines can easily adapt by providing a bidirectional buffer cell controlled by an enable line, with two unidirectional inputs and a bidirectional output.

2.4.8. Address Configuration Lines

Since the Multibus has two distinct address spaces, one for memory and one for I/O, there are two sets of address configuration lines. These lines determine what address range in each of the two address spaces the chip will respond to while operating as a slave. The memory address configuration lines are compared against the four high-order address bits on the Multibus. The I/O address configuration lines are compared against the four high-order bits of the low-order 16 bits of the address on the Multibus. Individual bit comparisons can be disabled by asserting the appropriate disable comparison line. This allows for extensions to a general mechanism for address range specification. The interrupt status register is located at the address that is the concatenation of the four I/O address configuration lines and a string of all ones. Note that inside the frame, Multibus addresses lose the low-order bit since it is used for byte addressing (which is not supported).

2.4.8.1. Address Configuration for Memory (ADRCM0-ADRCM3 and ADRDM0-ADRDM3). Lines used to determine the address range in memory space. An ADRCM line is compared against an address line only if it is not disabled by its corresponding ADRDM line. ADRCM0 is the least significant bit. ADRCM3 is compared with ADRI19 if ADRDM3 is low, ADRCM2 is compared with ADRI18 if ADRDM2 is low, ADRCM1 with ADRI17 if ADRDM1 is low, and ADRCM0 with ADRI16 if ADRDM0 is low.

2.4.8.2. Address Configuration for I/O (ADRCIO0-ADRCIO3 and ADRDIO0-ADRDIO3). Lines used to determine the address range in I/O space. An ADRCIO line is compared against an address line only if it is not disabled by its corresponding ADRDIO line. ADRCIO0 is the least significant bit. ADRCIO3 is compared with ADRI15 if ADRDIO3 is low, ADRCIO2 is compared with ADRI14 if ADRDIO2 is low, ADRCIO1 with ADRI13 if ADRDIO1 is low, and ADRCIO0 with ADRI12 if ADRDIO0 is low.

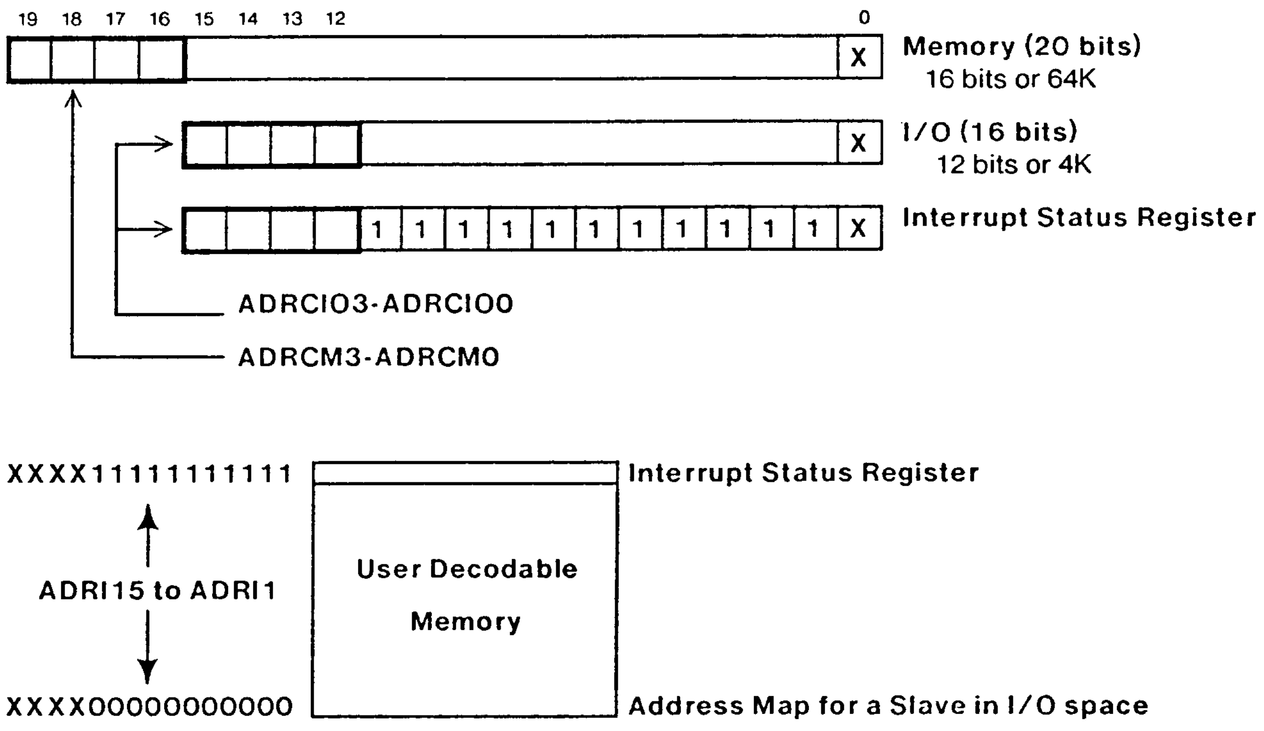


Figure 2-1. Address Space and Range Configuration for Slave

2.5 Design Frame Operation

This section will provide the designer with the information required to logically interface to the design frame. The section is divided into five parts, each explaining design frame operation for that class of functions.

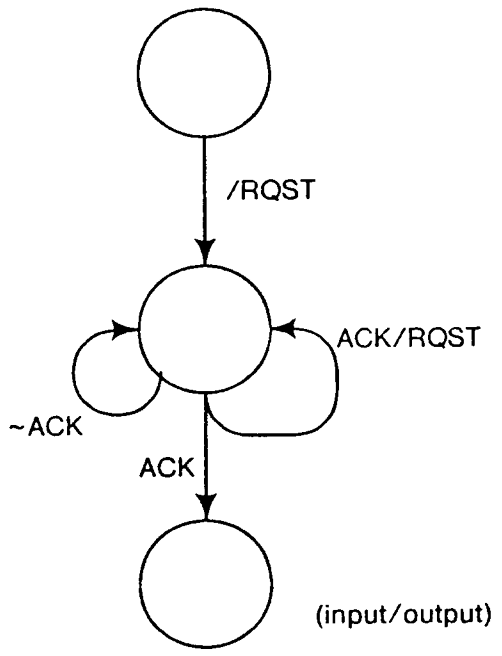
2.5.1. Power-up, Initialization, and Set-up

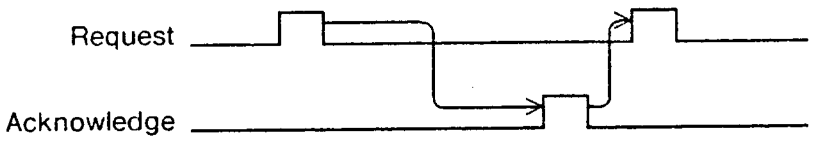
Multibus initialization is done at power-up and whenever the system is reset. A system-wide initialization signal is generated that is to be used by all system components to insure that they enter a known internal state (i.e. no device is stuck and holding on to the bus). On the Multibus Design Frame board, a push-button switch is provided to generate a local initialization signal. This is provided as a debugging aid, allowing the reinitialization of the board under test while avoiding a system-wide reset which might entail restarting the software operating system.

NOTE: There are a collection of frame signals that *must* be specified at initialization time. The designer must decide: whether slave operation will be in memory or I/O space (SSETM, SSETIO), whether master operation will be in memory or I/O space (MSETM, MSETIO), whether slave operation should be enabled or disabled (SEN, SDIS), and what address range should be recognized. Address configuration and master address space selection can be affected by many methods ranging from software programming of internal registers to external switches and dedicated pins. When the chip recieves an INIT signal at power-up, it must initialize to a proper state. If, as is most common, the chip is to begin operation as a slave, the address space and range must be specified and slave operation enabled (SEN). If it is to begin operation as a master, then the master address space must be set at initialization.

2.5.2. Transaction Protocol

There is a standard transaction protocol observed by all signals requiring a two-way handshake. A request signal is asserted for one cycle. Some time later the acknowledge signal will be asserted for one cycle indicating that the transaction has been completed (see Figure 2-2). The function of translating this protocol to one that is compatible with the Multibus is taken care of by the design frame. There is an overhead of 3.5 system clock cycles incurred for this operation. This does not include the actual time to arbitrate for the bus.





The finite-state machine reaches a state where it asserts a request. It then loops in the next state until the request is acknowledged. After the acknowledge is received it can either go on to do other work, or issue another request.

The timing diagram is applicable to both the operation of SRD/SWR and SACK, as well as MRD/MWR and MACK.

Figure 2-2. Transaction Protocol

2.5.3. Data Transactions

Address and data latches are built into the design frame. A general rule is that outgoing addresses and data must be valid during the cycle that the relevant request or acknowledge is asserted. Incoming addresses and data will be valid from when the relevant request or acknowledge is asserted to the beginning of the next command or acknowledge.

Timing diagrams are provided below for slave read and write, and for master read and write.

2.5.4. Extended Bus Use

A user circuit may want to perform a collection of data transactions, and be able to insure that they will be performed atomically. The design frame supports this usage. Not only does the design frame insure that other requesting masters are not granted the bus, it also asserts the LOCK/ signal. This alerts other modules that this is to be an atomic operation. This is a useful feature in systems with dual-ported memory devices.

An additional reason for providing this capability is to reduce the overhead of regaining control of the bus for multi-word transactions. The bus arbitration overhead can be significantly reduced if the bus can be acquired by the first transaction, and held through the rest.

The only limitation to this type of operation is that the bus not be controlled exclusively by a master for a period longer than 12 microseconds. This value is imposed by the Multibus Specification to insure that all devices will be able to perform a bus transaction within a reasonable of time.

2.5.5. Interrupts

The Multibus has eight interrupt lines, each of which can have any number of devices attached to it. The Multibus Design Frame provides for the connection to one of these eight lines.

When a bus master receives an interrupt, it must poll all the devices attached to that line to determine which generated the interrupt. After it determines the origin of the interrupt and performs the appropriate operations, the master must then clear the interrupt. The master can determine the interrupt status of a device by performing a read operation in I/O space to the address of the interrupt status register in that device. The contents of the interrupt status register (a single bit) are placed on the low-order data line. A logical high value on that line signifies that it was, in fact, that device that signalled the interrupt. To clear an interrupt, a write operation is performed to the same location. For a slave inside the Multibus Design Frame that location is the concatenation of the four bits used to set its address range (ADRCIO3 to ADRCIO0) and a string of eleven ones followed by a zero or a one (for the irrelevant low-order bit). The interrupt signal (MI) will not be deasserted until all interrupting devices have been serviced.

When a slave inside of a Multibus Design Frame wants to generate an interrupt all it must do is assert SI for one cycle (min). The design frame concerns itself with making the interrupt status known to inquiring masters and clears the interrupt when requested to so by a master. All the command servicing and acknowledging is performed by the frame. The user circuit will not see commands addressing the interrupt status register.

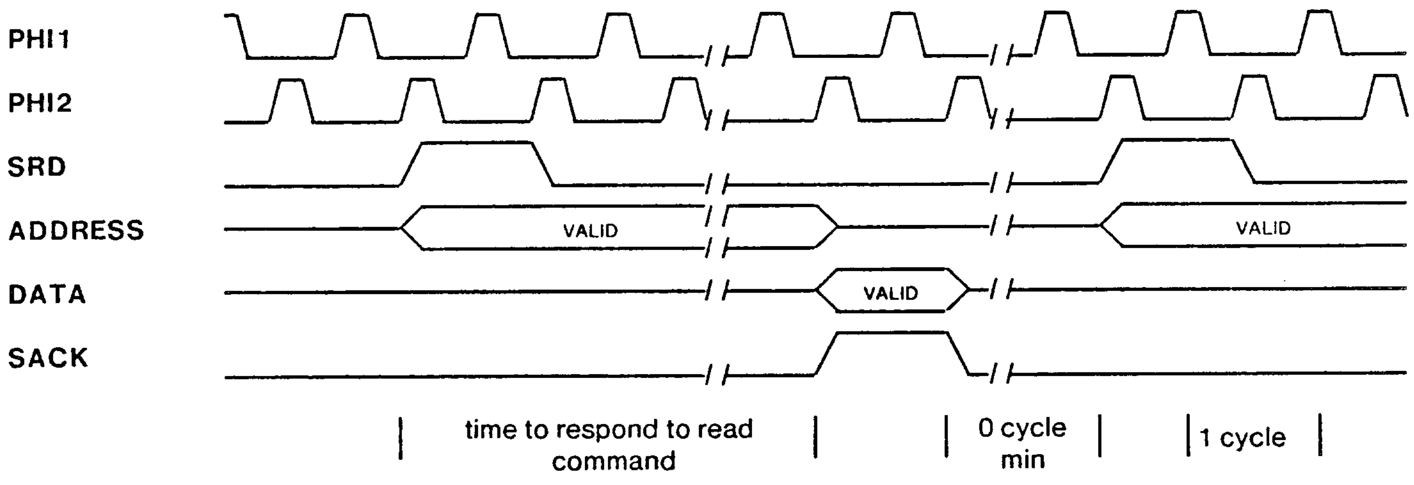


Figure 2-3. Slave Read Timing

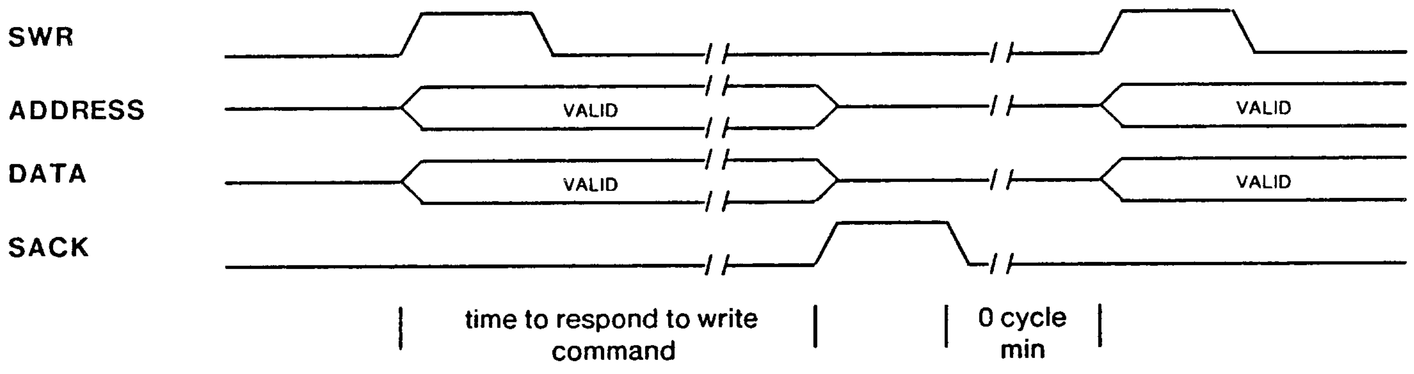


Figure 2-4. Slave Write Timing

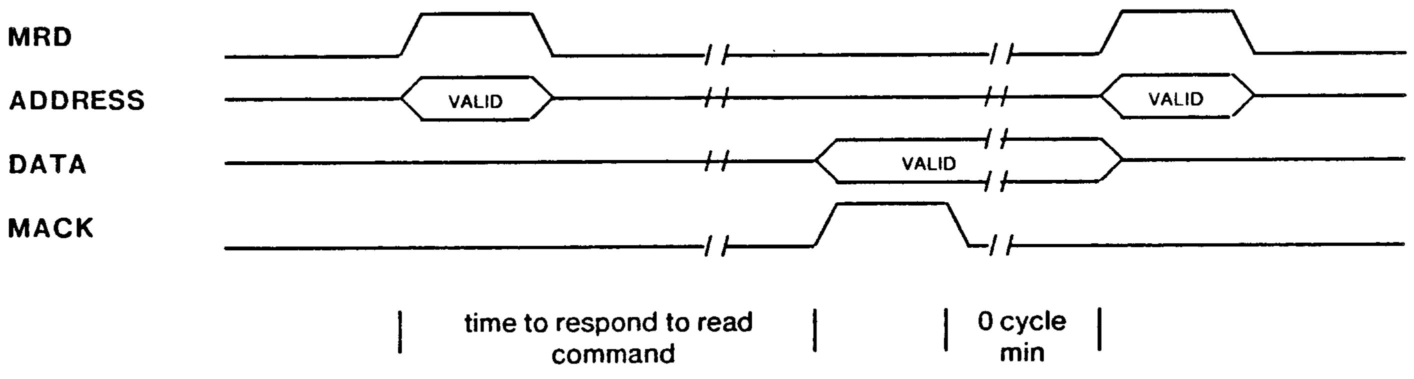


Figure 2-5. Master Read Timing

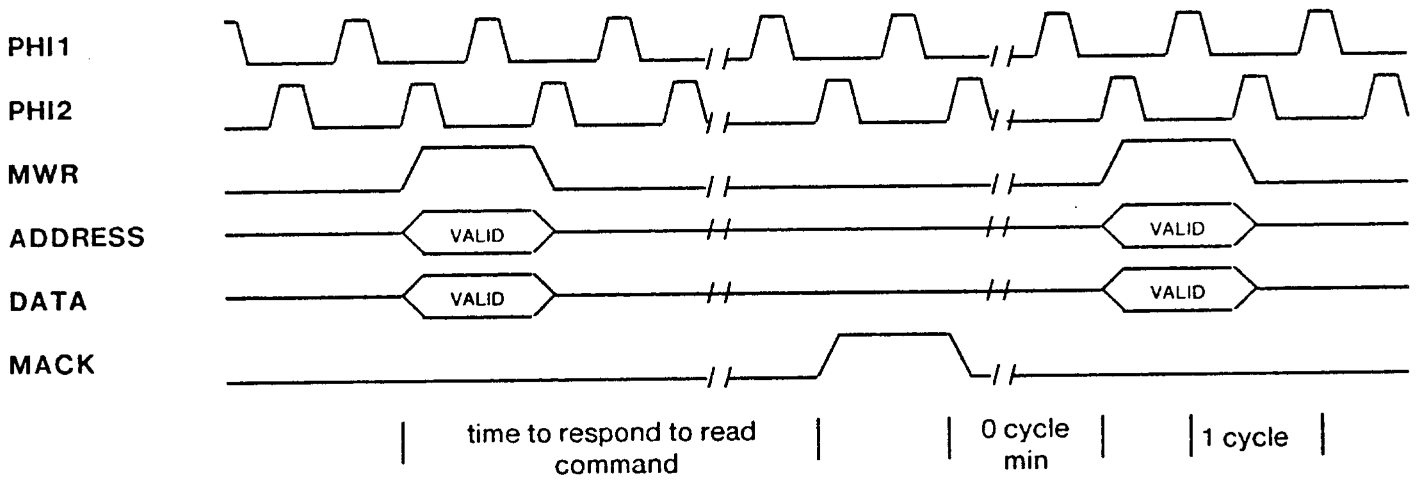


Figure 2-6. Master Write Timing

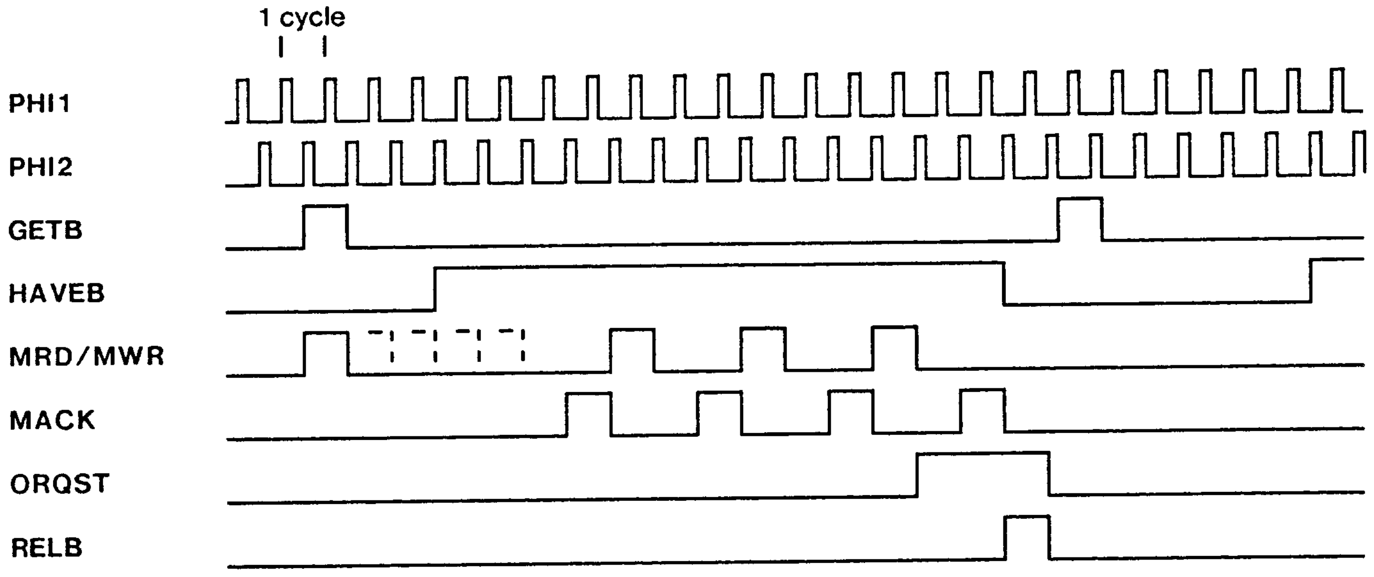


Figure 2-7. Extended Bus Use Timing

3. CHIP-LEVEL SPECIFICATION

The Multibus Design Frame was designed using the Mead/Conway design methodology. It should be fabricated at a λ equal to 2μ. No butting or buried contacts are used in the design. The circuitry is completely static. This should impose no limits on testing methods that may utilize relatively slow testers. The pad frame conforms to MOSIS standard pad frame 84P79X92. The chip should be packaged in the standard 84 pin grid array package provided by MOSIS.

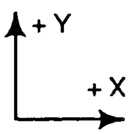
3.1. Electrical and Layout Interface

The Multibus Design Frame presents as uniform an interface as possible to the user circuit. In addition to having uniformity at the logic level (i.e., transaction protocol) and at the timing level (i.e., a standard finite-state-machine-like clocking convention), the frame also provides a uniform electrical interface for inputs and outputs.

All inputs to the frame present a uniform load to the user circuit (64 μ^2 of gate area) and are expected to be level restored. All outputs of the design frame (except PHI1 and PHI2) are driven by super-buffers with 1/2 square pull-ups and 1/8 square pull-downs. At the layout level, all inputs to the frame are 2λ wide and all outputs are 4λ wide. All signals (except VDD, GND, PHI1, and PHI2) are on polysilicon. The cavity available to the designer is 2982 by 3643 λ (or 5964 by 7268 μ when λ is equal to 2μ). There is metal along the entire perimeter so that the effective metal area available is 6λ less in both dimensions.

The Multibus Design Frame has 84 pins. Of these 58 are dedicated to interfacing to the Multibus, providing power and ground connections and clock signals. The remaining 26 pins are available to the user to use as he wishes.

Signal Name	I/O For User	Position (x,y of center in lambda)
INIT	I	2982, 3193
SRD	I	2982, 2527
SWR	I	2982, 2343
SACK	O	2982, 2982
SSETM	O	2982, 2472
SSETIO	O	2982, 2398
SEN	O	2982, 382
SDIS	O	2982, 420
MRD	O	2982, 1724
MWR	O	2982, 1677
MACK	I	2982, 2801
MSETM	O	2982, 1992
MSETIO	O	2982, 1942
GETB	O	2982, 1798
HAVEB	I	2982, 881
ORQST	I	2982, 505
RELB	O	2982, 1848
ADRO1-ADRO19	O	i = 1 2982, 365 i = 2 2982, 319
		i = 3 2982, 192 i = 4 2982, 85
		i = 5 2978, 0
		1014 + 150(19-i), 0 for i = 6 to 19
ADRI1-ADRI19	I	i = 1 2982, 326 i = 2 2982, 280
		i = 3 2982, 92 i = 4 2982, 78
		i = 5 2971, 0
		890 + 150(19-i), 0 for i = 6 to 19
ADRCM0-3*	O	939 + 150(3-i), 0 for i = 0 to 3
ADRDM0-3	O	943 + 150(3-i), 0 for i = 0 to 3
ADRCIO0-3*	O	1539 + 150(3-i), 0 for i = 0 to 3
ADRDIO0-3	O	1543 + 150(3-i), 0 for i = 0 to 3
DATO0-DATO15	O	714 + 150(15-i), 3643 for i = 0 to 6
		564 + 150(15-i), 3643 for i = 7 to 15
DATI0-DATI15	I	599 + 150(15-i), 3643 for i = 0 to 6
		449 + 150(15-i),3643 for i = 7 to 15
SI	O	2982, 3390
MI	I	2982, 3489
USER1-USER26	—	1. 491, 3643 10. 0, 2370 19. 0, 682
		2. 341, 3643 11. 0, 2182 20. 0, 494
		3. 191, 3643 12. 0, 1995 21. 0, 307
		4. 0, 3495 13. 0, 1807 22. 0, 119
		5. 0, 3307 14. 0, 1619 23. 191, 0
		6. 0, 3120 15. 0, 1432 24. 341, 0
		7. 0, 2932 16. 0, 1244 25. 491, 0
		8. 0, 2745 17. 0, 1057 26. 641, 0
		9. 0, 2557 18. 0, 869
VDD	—	Top Edge y = 3643
GND	—	Bottom Edge y = 0
PHI1	I	Rectangle 2982, 3 to 2986, 3643
PHI2	I	Rectangle 2989, 3 to 2993, 3643



* load of these signals is 96 sq. microns of gate area, all others are 64 sq. microns

Table 3-1. Position of Multibus Design Frame Signals

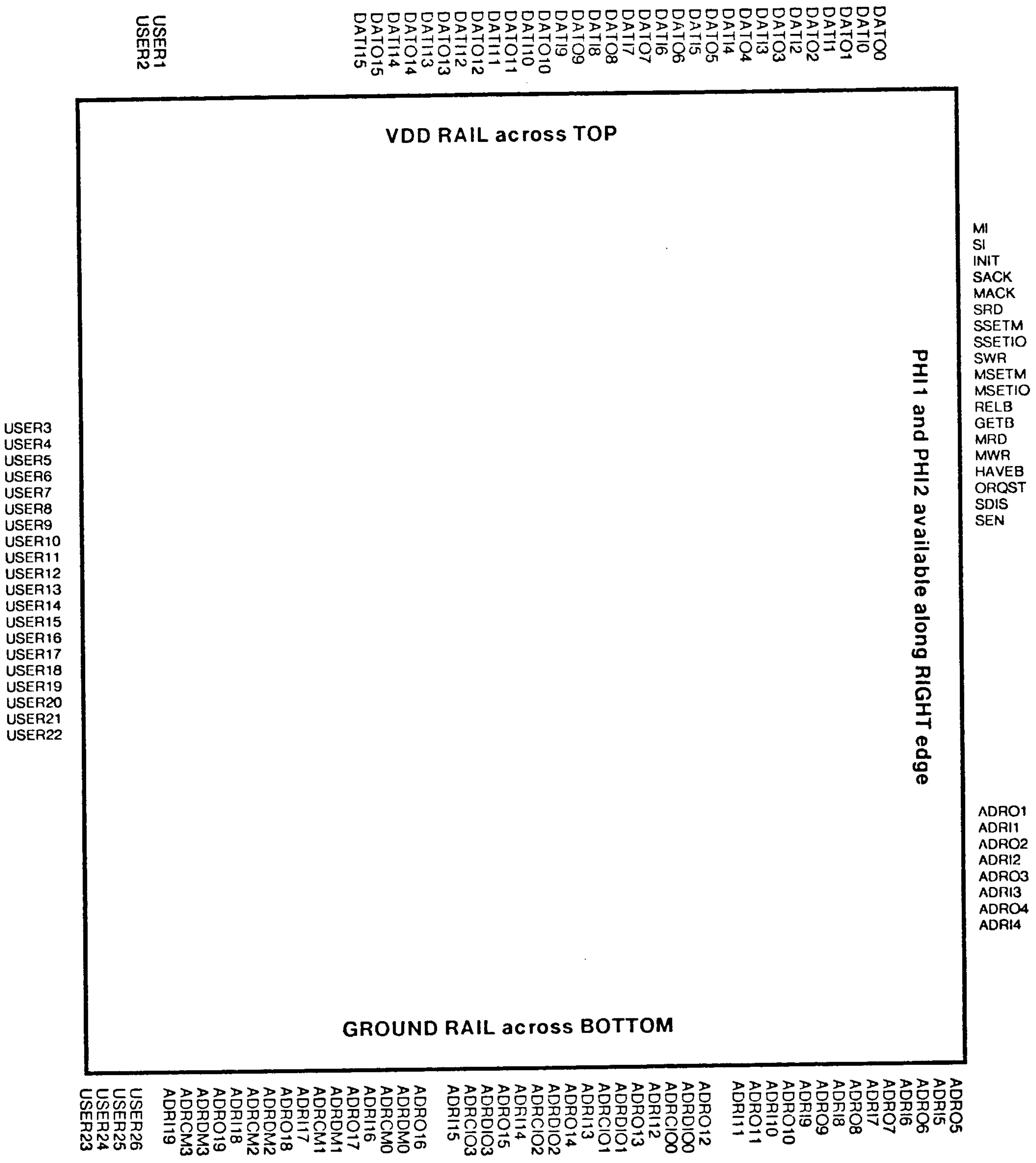


Figure 3-1. Logical Positions of Multibus Design Frame Signals

Mulitbus Signal Classes:

		# of Pins
A. Control Lines		
1. Clock	BCLK/	1
2. Commands		
a. Memory Write	MWTC/	1
b. Memory Read	MRDC/	1
c. I/O Write	IOWC/	1
d. I/O Read	IORC/	1
3. Acknowledge	XACK/	1
4. Initialize	INIT/	1
B. Address Lines		
1. Address Lines	ADR0/ - ADR19/	20
2. Byte High Enable	BHEN/	1
C. Data Lines	DAT0/ - DATF/	16
D. Interrupt Lines		
1. Interrupt Request	INT/	1
E. Bus Exchange Lines		
1. Common Bus Request	CBRQ/	1
2. Bus Priority		
a. Bus Priority In	BPRN/	1
b. Bus Priority Out	BPRO/	1
3. Bus Busy	BUSY/	1
4. Bus Request	BREQ/	1
5. Bus Lock	LOCK/	1
F. Power/Ground/Substrate		
1. Power	VDD	2
2. Ground	GND	2
3. Substrate	SUB	1
G. Others		
1. System Clock	CLK/	1
2. Reference Clock	REF/	1
H. User Definable Pins	USR1-USR26	26
		84 pins

Table 3-2. Multibus Design Frame Pins

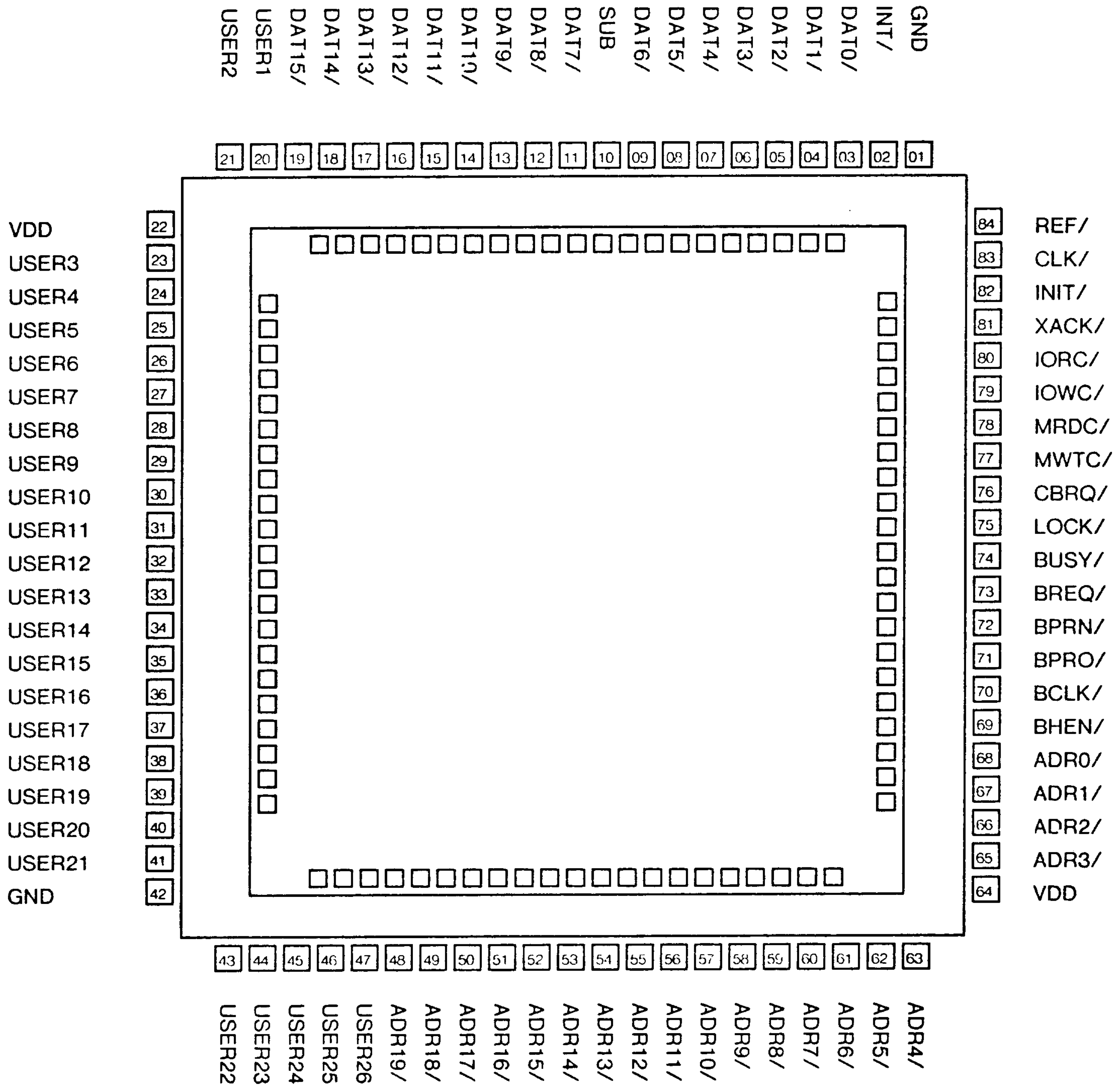


Figure 3-2. Pinout of Multibus Design Frame in MOSIS Standard 84 Pin Grid Array Package

3.2. System Clock Generation

The Multibus Design Frame allows for precise control of the two most important parameters of the system clock, the duty-cycle and the amount of non-overlap. This is accomplished using a self-calibrating delay line circuit developed at the Xerox Palo Alto Research Center. The two parameters are functions of two frequencies that are input from crystal generators on the board or from the Multibus 10MHz clock. The reference frequency for the delay line circuit controls the amount of time a clock line is asserted. PHI1 and PHI2 will each be high for half the period of the reference input. PHI1 will occur on a rising transition on the clock input and PHI2 on a falling transition. In this way, the frequency of the clock input regulates the amount of non-overlap between the two phases. Designers can choose PHI1/PHI2 pulses to be between 50 and 200ns wide and have periods of any value greater than 200ns.

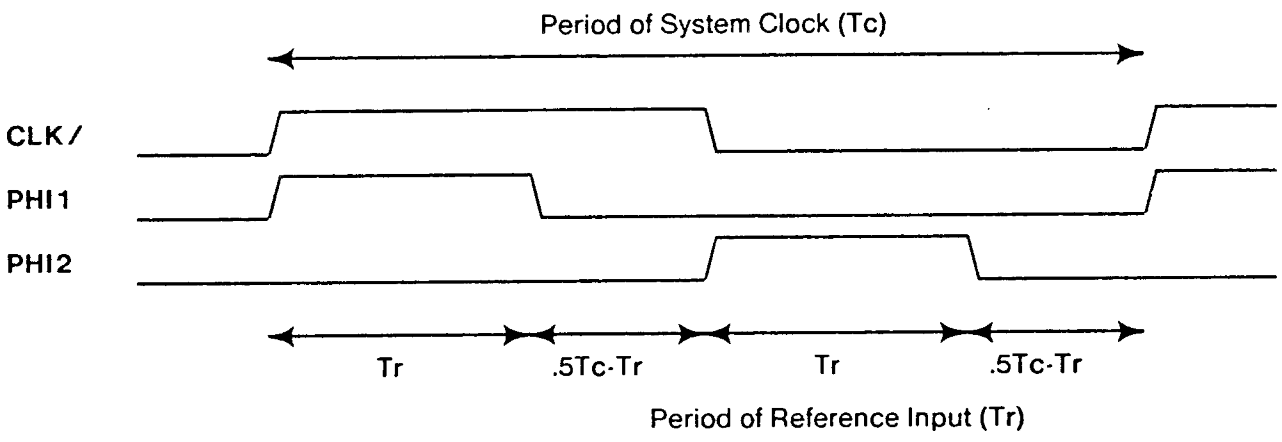


Figure 3-3. System Clock Parameters

4. BOARD-LEVEL SPECIFICATION

The Multibus Design Frame printed circuit board is based on a two signal layer technology. Two additional internal planes provide power and ground distribution. This helps decrease noise and ground loop problems. The boards have been fabricated using the MOSIS PCB implementation service and conform to the design rules distributed by MOSIS. The board is Multibus standard 6.75" by 12.00" and meets Multibus specifications.

4.1. Board Elements

The board provides space for two zero-insertion-force (ZIF) sockets that accept Multibus Design Frame chips. This was done so that two chips could be mounted on the same board, a chip under test and a scan-path controller chip. The socket connections are identical. The ceramic bonding package recommended is the standard MOSIS 84-pin grid array. All the user-definable pins are brought out to vias where wire-wrap pins can be inserted and the signals connected to other user circuitry in the prototyping area.

A large wire-wrap prototyping area is provided for user circuitry and for placing connectors to get signals off the board. The area is bounded on the left and right by vias connected to ground. An equal number of vias connected to power are in the center. The prototyping area and the user pins of the Multibus Design Frame chips have a Cartesian coordinate system so that pin locations can be easily referenced. These coordinates are specified along the right and top edges of the board.

Along the bottom edge of the board are vias to all the Multibus signals used by the Design Frame. These connections are provided for the user's convenience should he require access to these signals.

A reset switch is in the top-left corner of the board. It provides the board with a local reset signal. This switch should be useful in the debugging of a design. A designer testing his circuit can reset it to a known state without having to go through a complete system reset which may involve a long initialization sequence to reload the operating system. The details of the switch are in Figure 4-2.

Directly below the UCB logo are two sockets for crystal oscillator packages. These should be used if other frequencies are needed besides the standard Multibus 10MHz clock signal. One oscillator sets the period of the circuit clock and the other sets the duty-cycle of the two-phase clock (see Section 3.2).

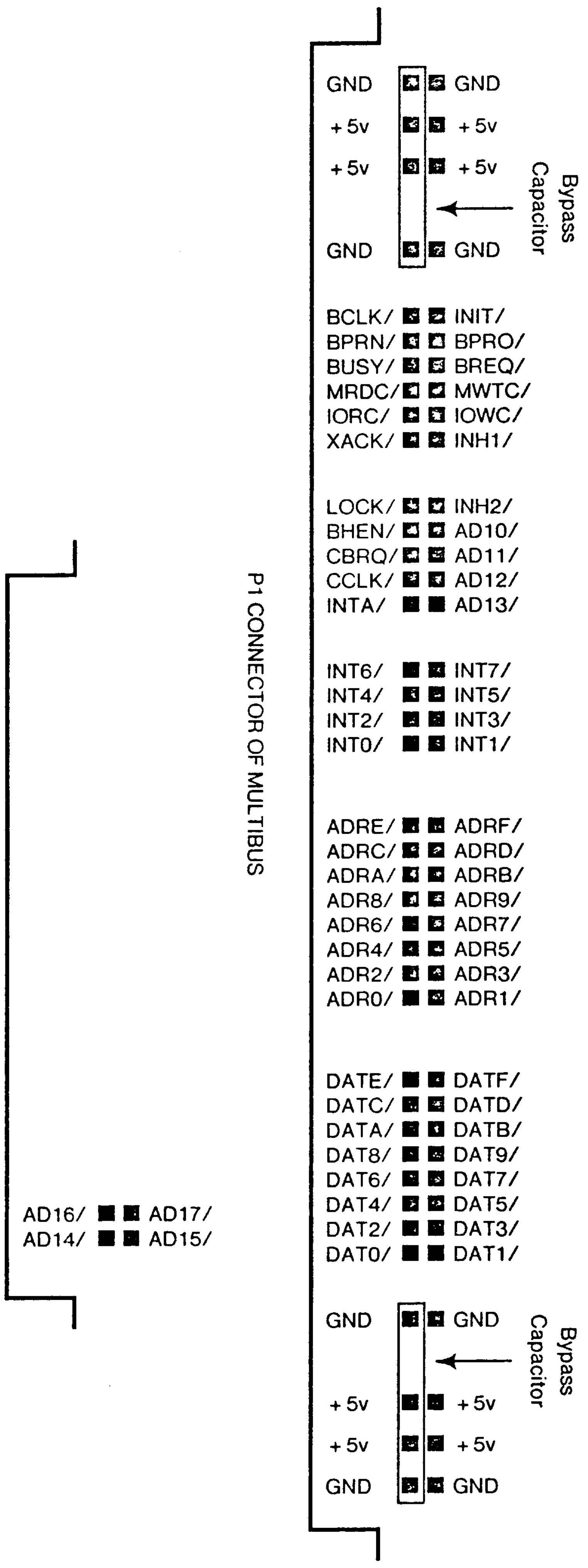


Figure 4-1. Multibus Signal Pins

To completely set up a board the following parts are required:

- 3 14-pin sockets
- 2 84-pin grid zero-insertion force sockets
- 1 push-button switch, 1k ohm resistor, 10k ohm resistor
- 2 bypass capacitors between the power and ground pins of the board
- 2 crystal oscillators if required by the application
- miscellaneous wire-wrap pins to facilitate access to relevant signals

4.2. Board Set-up

4.2.1. Clock Source Set-up

In the lower-left corner of the board (near the left power supply bypass capacitor) is a set of six wire-wrap pins. These pins are used to control the system clock supplied to the chips. The leftmost three are used to control the system clock's duty cycle and the rightmost three are used to control its period (see Section 3.2). The middle pin of the leftmost set connects to the reference input of the chip and middle pin of the rightmost set connects to the clock input. Both bottom pins connect to the 10MHz clock provided by the Multibus backplane. The top leftmost pin is the output of the left crystal oscillator (if present) and the top rightmost pin is the output of the right crystal oscillator (if present). The middle pin of both sets should be tied (or jumpered) to either the 10MHz signal provided or the output of its respective crystal oscillator.

The REF input should be of a frequency between 5 and 20MHz. This corresponds to PHI1/PHI2 pulse-width of 200 to 50ns, respectively. The CLK input can be of any frequency less than 5MHz. For example, if a designer requires PHI1/PHI2 pulses 100ns wide and a period of 400ns then the REF input would be 10MHz and the CLK input would be 2.5MHz.

4.2.2. Interrupt Set-up

Along the bottom edge of the board are two connectors. In the center of the leftmost and largest of these is a group of a eight pins with a single pin centered directly above them. These eight lines are the Multibus interrupt lines. If the user circuits on the board make use of the interrupt capabilities of the design frame, then one of these eight lines should be tied (or jumpered) to the pin directly above them. This will determine the interrupt line to which the entire board will be connected.

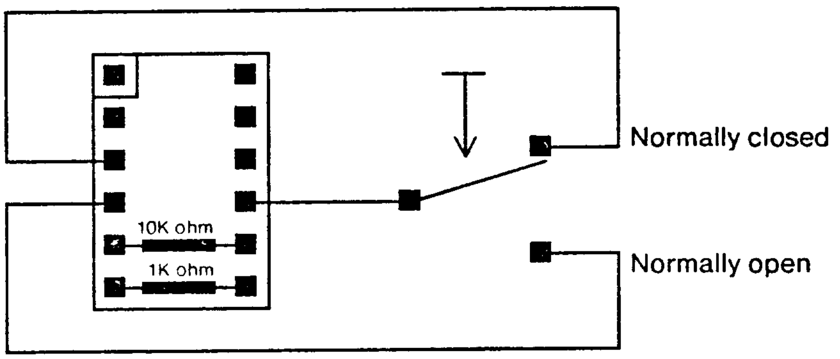


Figure 4-2. Board Reset Switch

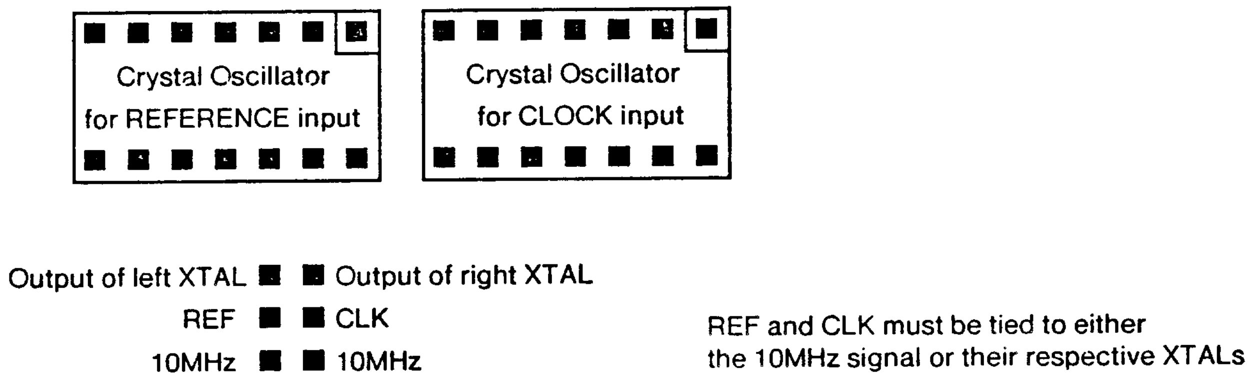

Crystal Oscillator for REFERENCE input
Crystal Oscillator for CLOCK input
Output of left XTAL
Output of right XTAL
REF
CLK
10MHz
10MHz
REF and CLK must be tied to either the 10MHz signal or their respective XTALs

Figure 4-3. Clock Configuration

4.3. Extra Address Lines and User Pins

The four pins in the bottom right hand corner of the board are provided for eventual compatibility with 24-bit address space Multibus systems. If a designer needs to make use of these pins then they can be wire-wrapped to the four user pins closest to the address bus pins and the chip-level Multibus Design Frame can be easily modified to include four additional address pins.

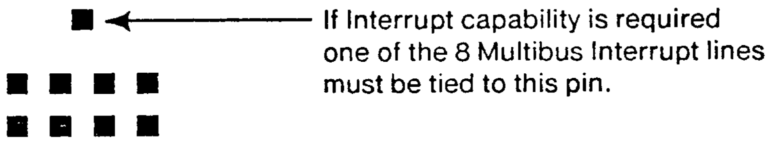

If Interrupt capability is required one of the 8 Multibus Interrupt lines must be tied to this pin.

Figure 4-4. Interrupt Configuration

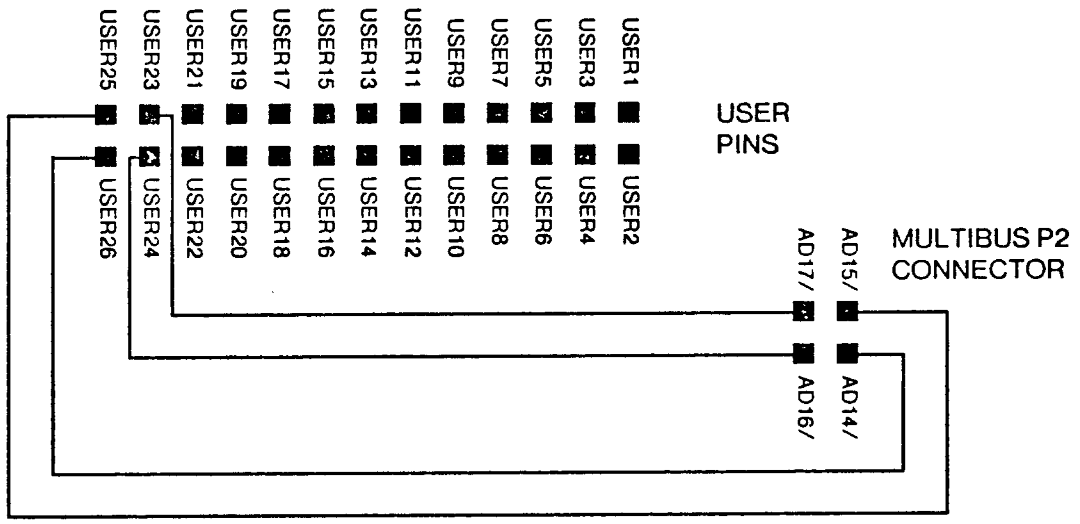


Figure 4-5. Multibus Design Frame Board Connections for 24-bit Address Space

5. DESIGN GUIDELINES AND EXAMPLES

The following is a collection of design choices that may have to be made when using the Multibus Design Frame and some possible alternatives that may help resolve them. The examples range from ways of configuring a chip to recognize a certain set of addresses to how a simple microprocessor was designed to function within the design frame.

5.1. Configuration of Address Space and Range

Initializing a Multibus Design Frame chip to recognize a certain address range within memory or I/O address spaces is like the chicken-and-egg problem. If the designer wants to have software control, he must provide a register inside his circuit to which he may write the bits that will drive the address configuration lines and SEN, SDIS, SSETM, SSETIO, MSETM, and MSETIO. However, how does he get the chip to recognize the address of this register?

One solution is to provide a set of switches (wire-wrapped in the prototyping area of the board) to set some user pins that are directly connected to the relevant signals. This solution is acceptable if those pins are not required for other functions. Here, the register is essentially outside the chip and is not under software control.

A more general solution, one which can be extended to provide many more features, is to place the above register on a scan path. In this manner a special controller (that has its address configured through pins as above) can be designed to control the scan path and load the register with the appropriate values under software control. This method lends itself nicely to general purpose scan-in/scan-out techniques for testing (see Section 5.4).

5.2. Bidirectional Data and Address Busses

The Multibus Design Frame's internal address and data busses are unidirectional. However, they can easily be adapted to be bidirectional. A designer simply includes a cell which takes a direction signal as input and contains two drivers. The schematic for such a cell is given below. The decision to make the busses unidirectional was made to minimize the amount of superfluous circuitry in the design frame. The frame's pads provided unidirectional signals, and since the conversion process is straight-forward, it seemed appropriate to leave the decision to include bidirectional adapters to the designer.

5.3. Modifying the Transaction Protocol

Some designers may find that the one-cycle duration pulses of the Multibus Design Frame's transaction protocol may place too stringent a requirement on their circuits. An implementation decision was made to implement this protocol because it was simpler for use with finite-state-machine control and because it would be easy to modify to other protocols. If a user prefers an n-cycle acknowledge signal, for example, the addition of a set-reset flip-flop (a pair of cross-coupled NOR gates) would be sufficient. This circuit would cause the acknowledge signal to stay asserted until the next request is issued.

5.4. Scan-In/Scan-Out Path for Testability

A possible future enhancement to the Multibus Design Frame may be support for testability using a scan-in/scan-out path. With such a path through the chip, internal states can be observed and controlled. The cost of such a path is a few pins and some added circuitry. Besides the obvious uses in testing the design, such a path would also provide a mechanism for chip configuration at initialization (see Section 5.1). To make this valuable capability an integral part of the Multibus Design Frame concept, a chip is being designed that will reside in the frame and support the use of a scan path that passes through other design frame chips on the board.

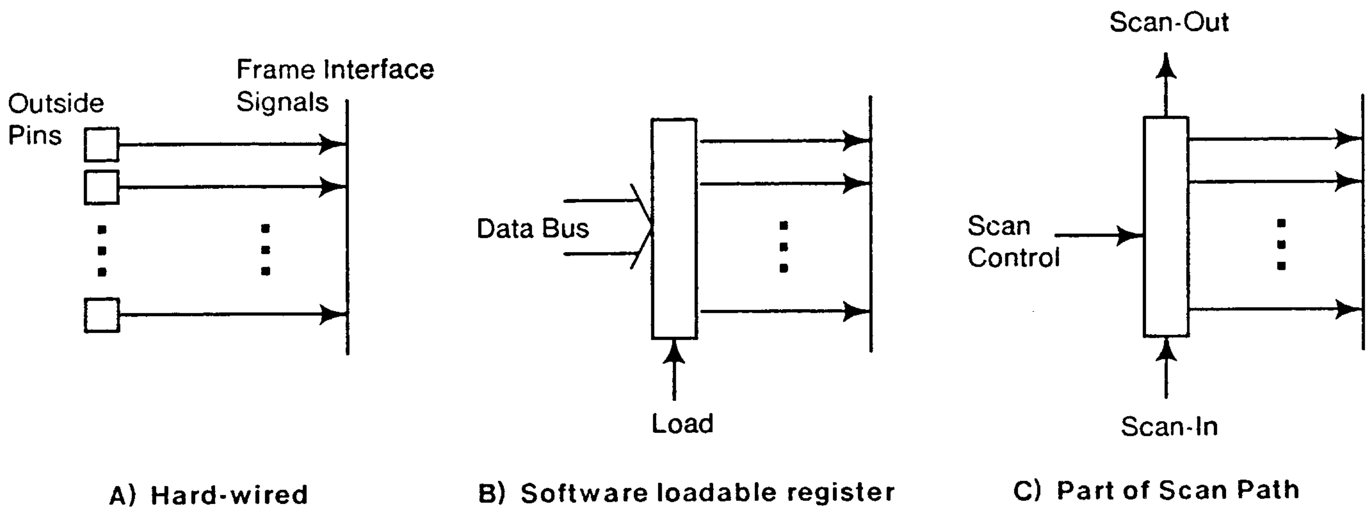


Figure 5-1. Frame Configuration Alternatives

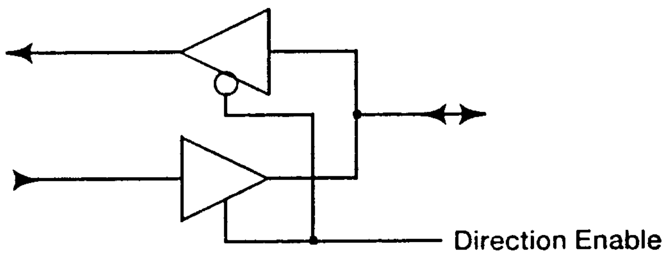


Figure 5-2. Unidirectional to Bidirectional Conversion

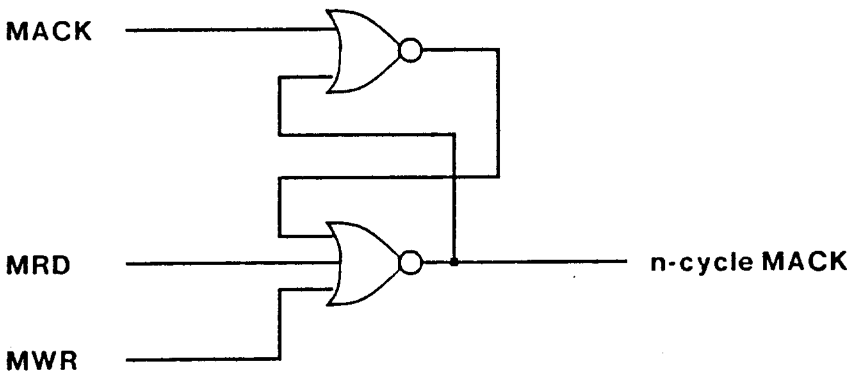


Figure 5-3. Transaction Protocol Conversion to n-Cycle Acknowledge Pulses

NAME

MDF - an nMOS frame for the integration of custom VLSI into Multibus-based systems

SYNOPSIS

magic -T nmos ~cad/lib/mdf/MDF
- starts magic with the entire Multibus Design Frame loaded.

magic -T nmos ~cad/lib/mdf/MDFCONNECTIONS
- starts magic with the cell containing the outline of the user circuit cavity.

DESCRIPTION

The **Multibus Design Frame** is an nMOS frame for the integration of custom VLSI into Multibus-based computer systems. The design frame provides a simplified interface to the backplane of the computer system. A circuit designed within the context of a design frame can be quickly integrated into a computer system upon fabrication. In many ways, a design frame is much like a hardware operating system. It can be used to rapidly prototype custom VLSI circuits and for the evaluation of the design in a real system context.

The **Multibus Design Frame** consists of elements at the chip and board levels. The nMOS circuitry within which the user circuit is placed and then sent to fabrication is provided in *magic*(5) format. The top-level cell is **MDF.mag**. It contains all the circuitry of the **Multibus Design Frame**. For the convenience of the designer a cell containing a 3 lambda wide outline of the user circuit cavity is also available (**MDFCONNECTIONS.mag**). All the connections points are labeled with the name of the signal. This cell is called by **MDF.mag**. If a circuit is designed within **MDFCONNECTIONS.mag**, CIF for the entire design can be generated by simply loading **MDF.mag**.

Designers may also find the need to have inputs and outputs other than those to the frame interface. Pads are available in **PADINUSER.mag** and **PADOUTUSER.mag**. The pads used by the design frame can also be used by the user, however, these pads invert their inputs and outputs.

When the fabricated chips are returned they can be placed on the **Multibus Design Frame** board and placed in a Multibus card-cage. The file MDFPCB.cif is the CIF used to fabricate the **Multibus Design Frame** printed circuit board through the PCBIS service of MOSIS. This file is provided as an example of what a printed circuit board description looks like, users are not expected to fabricate their own boards.

FILES

~cad/lib/mdf/MDF.mag
- The top level cell of the Multibus Design Frame

~cad/lib/mdf/MDFCONNECTIONS.mag
- Cell containing the outline of the user circuit cavity and labels on all the connection points.

~cad/lib/mdf/*.mag
- Magic files for all the cells in the Multibus Design Frame

~cad/lib/mdf/MDFPCB.cif
- CIF description of the Multibus Design Frame printed circuit board.

NOTES

Complete documentation, users' guide, and a detailed specification of the **Multibus Design Frame** chip and board level frames is available by writing:

Gaetano Borriello
Computer Science Division
573 Evans Hall
University of California at Berkeley
Berkeley, California 94720

gaetano%ucbkim@Berkeley.ARPA

SEE ALSO

magic(1), magic(5)

AUTHOR

Gaetano Borriello

The Multibus Design Frame

in a SUN Microsystems Workstation

Gaetano Borriello

Computer Science Division
Electrical Engineering and Computer Science Department
University of California, Berkeley
Berkeley, California 94720

ABSTRACT

This paper is a guide for designers wishing to integrate a custom integrated circuit, fabricated within the context of the Multibus Design Frame, into a SUN Microsystems workstation. The SUN workstation is Multibus based and runs the UNIX 4.2bsd operating system. The first section of this paper gives a brief overview of the hardware configuration required when testing a chip contained within the Multibus Design Frame. Section 2 is a description of the user-level C routines that can be used to exercise Multibus Design Frame chips. Listings of these routines are included in the appendix.

1. Hardware Configuration

There are many models of SUN workstations with a wide variety of possible configurations. The exact number of boards in a SUN is dependent on the devices attached to the workstation and the amount of memory present. Some models are rack-mounted, that is, they are equipped with an easily accesible card cage. These models are recommended for initial testing and debugging of Multibus Design Frame chips.

The rack-mounted SUN workstations provide enough space for 15 Multibus boards and allow the use of extender cards. Extender cards permit a board under test to protrude from the main card-cage. In this manner, signals and chips on the board can be easily accessed without requiring removal and reinsertion of the board. pp Approximately five boards are required in order to sustain an adequate computing environment. Obviously, the processor board and its supporting processor memory boards are essential (this usually implies 3 boards). Another board for either an Ethernet controller or a disk controller is needed in order to provide a place to store and retrieve programs. Workstations can be set-up as disk-less nodes on an Ethernet and use a network file server. This is a preferred configuration since it eliminates the possibility of damage to stored files due to a faulty chip addressing the disk controller (the probability of such damage is extremely small, however). The only other requirement is for a display. Another board is required to support a high-resolution monochrome display that is standard with SUN workstations. In its stead, one could use the RS232 port on the processor board to support a standard terminal.

The Multibus Design Frame board can easily drive up to seven other boards. It should be placed in one of the available controller slots in the card-cage. If Multibus memory is required for a particular application then another board is needed to provide this storage. It is recommended to use a separate memory board so that any conflicts of usage with memory already present in the machine are avoided.

After any board is placed in a Multibus chassis the SUN PROM monitor should be used to make sure that the board is actually in the address space intended and at the appropriate starting address. The manual for the PROM monitor can be found in the standard set of documentation provided by SUN. The monitor is booted automatically on start-up if the workstation cannot access a boot file on disk or across the Ethernet. Therefore, powering up the machine without its Ethernet and/or disk connected will start up the PROM monitor. Once all the boards added have been verified, UNIX can be booted in the normal fashion.

2. Accessing From Software

To properly utilize all the capabilities of the Multibus, a device driver should be written for each type of Multibus Design Frame chip that is to be inserted into the SUN. However, this would require installing the driver into the operating system and possibly decreasing the effectiveness of the UNIX protection mechanisms. Most applications do not require the full capabilities that a device driver would provide. In fact the only capability that cannot be utilized without a device driver is the Multibus Design Frame interrupt mechanism.

In the appendix can be found user-level C routines that use the standard Multibus memory and I/O device drivers already present in SUN UNIX. These routines can access all Multibus locations that are not already used by SUN UNIX. This is another of the reasons for adding an extra memory board if required by the application. The header file in the appendix should be useful in linking these procedures to user-level programs.

There are two classes of procedures. The first open an address range in the device specified (memory or I/O) and map these locations into the virtual address space of the running program. The second type of procedure is then used to read or write from specific locations within the mapped ranges.

The mbmem_map and mbio_map procedures require two arguments. The first specifies the first Multibus address to which the board will respond. The second argument is the number of words, not bytes, that defines the range of the device. The range must be a multiple of the page size. Since the standard configuration of the Multibus Design Frame allows a memory device a 32K word range an I/O device a 2K word range, this number should be a multiple of 2048. These procedures return a pointer to the first location in the virtual address space to which the physical locations specified have been mapped. No checking of arguments or address ranges is done by these procedures.

The read and write procedures require three arguments. The first is the starting address in virtual memory of the device (i.e. the pointer returned by the map procedures). The second is the offset, in words, from that address. The last is the pointer to the location into which to read a 16-bit value on a read or from which to obtain the value to write out on a write. Again, no checking is done in these procedures.

The devices */dev/mbmem* and */dev/mbio* must have their access rights modified to allow access by the user. In standard systems these devices are only accessible by root. Mapping of the address space into virtual memory will not be possible unless there is no other device currently in that Multibus address range. This means that each user must make certain that the installed boards are in an address range that is currently considered free by their operating system. If these conditions are met, these procedures should work properly.

```
#include <stdio.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <sys/file.h>

short int *mbmem_map(mbmem_starting_address, mbmem_address_size)
int mbmem_starting_address;
int mbmem_address_size;

{
        char *valloc();
        caddr_t base;
        int fd;
        int md;

        base = valloc( (unsigned int) mbmem_address_size * 2);
        if( (fd = open("/dev/mbmem", O_RDWR) )  <= 0 ) {
                fprintf(stderr, "Cannot open 'mbmem' %d\n", fd);
                exit(-1);
        }
        if( (md = mmap(base, mbmem_address_size * 2, PROT_WRITE | PROT_READ,
                        MAP_SHARED, fd, (off_t) mbmem_starting_address) ) != 0 ) {
                fprintf(stderr, "Cannot map memory-space into virtual memory %d\n", md);
                exit(-1);
        }
        return((short int *) base );
}
```

mbmem_read

```
mbmem_read(base, offset, data)
short int *base;
int offset;
short int *data;

{
        *data = (short int) ( (*(base + offset * 2)) & 0x0000ffff );
}
```

mbmem_write

```
mbmem_write(base, offset, data)
short int *base;
int offset;
short int *data;

{
        *(base + offset * 2) = (short int) ( *data & 0x0000ffff );
}


short int *mbio_map(mbio_starting_address, mbio_address_size)
int mbio_starting_address;
int mbio_address_size;

{
        char *valloc();
        caddr_t base;
        int fd;
        int md;

        base = valloc( (unsigned int) mbio_address_size * 2);
        if( (fd = open("/dev/mbio", O_RDWR) )  <= 0 ) {
                fprintf(stderr, "Cannot open 'mbio' %d\n", fd);
                exit(-1);
        }
        if( (md = mmap(base, mbio_address_size * 2, PROT_WRITE | PROT_READ,
                        MAP_SHARED, fd, (off_t) mbio_starting_address) ) != 0 ) {
```

```
                fprintf(stderr, "Cannot map io-space into virtual memory %d\n", md);
                exit(-1);
        }
        return((short int*) base );
}
```

mbio_read

```
mbio_read(base, offset, data)
short int *base;
int offset;
short int *data;

{
        *data = (short int) ( (*(base + offset * 2)) & 0x0000ffff );
}
```

mbio_write

```
mbio_write(base, offset, data)
short int *base;
int offset;
short int *data;

{
        *(base + offset * 2) = (short int) ( *data & 0x0000ffff );
}
```

```
extern short int *mbmem_map();
extern mbmem_read();
extern mbmem_write();
extern short int *mbio_map();
extern mbio_read();
extern mbio_write();
```