AN ABSTRACT OF THE THESIS OF

Harish Chandra B. Nayak for the degree of Master of Science in Electrical and Computer Engineering presented on August 7, 1989
Title : Design Strategies for the Interface of Bus Architectures

Abstract Approved : Redacted for Privacy

James H. Herzog

This paper discusses the design strategies for interface of bus architectures. It also describes the design of interface between the two popular bus architectures, the Micro Channel and the NuBus.

The main objective of this thesis is to develop a procedure for bus interface design and use this procedure to design a Micro Channel Interface for the NuBus. The different problems, to be considered while designing an interface between two system buses, are identified and several methods of solving these problems are discussed. Several methods of implementing the interfaces are also discussed.

A Micro Channel slave interface is designed using the procedure developed. A Custom VLSI bus controller approach is chosen for the design. The interface is designed in such a way that it can be easily identified as two blocks and a VLSI component can be designed for each of the blocks. This interface may prove to be useful to design cards for the Micro Channel which can be used on the NuBus systems.

Design Strategies For Interface Of Bus Architectures

by

Harish Chandra B. Nayak

A THESIS
Submitted to
Oregon State University

in partial fulfillment of
the requirements for the
degree of
Master of Science

Completed August 7, 1989
Commencement June, 1990

APPROVED :

Redacted for Privacy

Associate Professor, Electrical and Computer Engineering in charge of major

Redacted for Privacy

Head of Department of Electrical and Computer Engineering

Redacted for Privacy

Dean of Graduate School

Date thesis is presented : 7th August 1989

ACKNOWLEDGEMENT

I would like to express my appreciation to each member of my committee for the time they invested in this thesis. A special note of thanks to Prof. James Herzog, my advisor, who took me on as his student and guided me in the right direction during the course of this thesis. Also I thank him for the many hours he spent reading my work and commenting on it while it was being done.

I thank one and all in the dept. of Electrical and Computer Engineering for their full hearted cooperation throughout my stay here.

Finally I thank my parents, brother and friends for constant support and encouragement.

To all of these people, Thank you.

TABLE OF CONTENTS

LIST OF FIGURES

DESIGN STRATEGIES FOR INTERFACE OF BUS ARCHITECTURES

1. INTRODUCTION

A bus, in the context of a computer system, is a group of signals that communicate among devices and that typically connects multiple devices in parallel. It is a communication channel which interconnects a micro-processor to its support devices, main CPU board to add-on boards and systems to systems. Physically it consists of a number of electrical interconnections realized in a variety of forms depending on the nature and proximity of the communicating devices.

Within a Printed Circuit Board, intercommunication between integrated circuits over short distances (<50 Cm) is accomplished over a bus or strip of micro lines; between Printed Circuit Boards, interconnections over larger distances (<100 Cm) are made via edge or indirect connectors to a backplane bus consisting of printed circuit board tracks or multi-way ribbon cable. Between elements of the computer housed in separate enclosures, the interconnecting bus is a multi-way cable.

The interconnections can be Serial, where information is transferred one bit at a time, or parallel, where information is transferred many bits at a time depending on the width of the bus. RS-232 is a serial bus [2] and IEEE std. 488 (GPIB) [3] is a 8-bit parallel bus. A point-to-point interconnection allows the exchange of information between only two units. RS-232 and Centronics printer cable are the examples of point-to-point interconnection. A multi point interconnection allows the exchange of information among many units on the same physical link. The Micro Channel and the NuBus are multi point links. The data transfer rate is determined largely by the number of interconnections and the transmission characteristics of the bus.

Buses can exist at several levels in microprocessor based systems as shown in Figure 1.1. Most processors define their own local, or CPU, bus structure with address and data paths and control signals required for the peripherals designed for that processor. A local bus is structured to optimize the processor-to-memory bandwidth. It is highly processor dependent, tightly linked to its processor, memory, and specific support peripherals. The consequence of this close association is loss of flexibility. In a single board system, this may be the only bus present. The buses of Intel 80x8x and Motorola 680x0 are the examples of local bus. [3]

In a multiboard system, a higher level bus is present ; the Back plane bus also called the System bus. Unlike local buses, System buses offer a general protocol, or transfer method, for system CPUs or I/O subsystems to interchange data. This is accomplished by treating the bus as a resource. To get control of the resource, a peripheral or processor must request its use formally, in competition with others. The system bus integrates hardware, cards and subsystems into one smoothly running machine. These buses support the information exchange over distances less than 100 Cm [3]. The Micro Channel, NuBus, VMEbus, EISAbus, Multibus are such system buses.

At an even higher level are Inter System Buses. These buses connect independent systems and operate over longer distances than backplane buses. The information transfer over these buses is slower compared to that on the system buses. One common example of such a bus is the General Purpose Interface Bus (GPIB), also known as IEEE - 488. [27]

1.1 Bus Architecture

There are four common classes of signals which are present in all the bus architectures; address, data, control signals and power lines.

The address and data buses are tri-stated signal lines. The width of the address bus determines the addressing capability of the system bus and the width of the data bus is one of the factors which determines the speed of data transfer on the system bus. In the case of multiplexed buses, the same signal lines are used to transmit both the address and the data. The address is put on the address/data bus and after the address is latched in the addressed peripheral, data is put on the bus. The control bus consists of transaction control signals, interrupts, bus request and acknowledge signals and arbitration signals. In the case of Serial buses, where the information is transferred one bit at a time, a single line is used for the transfer of control signals, address, and the data. Each bus has its own requirements to be met before a transaction is made, i.e. certain signals should be in proper status before the address is put or removed from the address bus and data is put or removed from the data bus. Certain sequence of control bits should be transmitted before the transmission of address or data in case of a serial link. Thus each bus has its own protocol of transaction. The bus protocol also defines the arbitration process, interrupt structure, and the mechanical and electrical requirements. There is a considerable difference between different bus protocols.

A peripheral device which initiates a transaction on a system bus by addressing of another card or the processor on the main logic board is a Bus Master. It acts as a commander in a transfer operation. In a typical microprocessor system, the master modules are the CPU and the DMA controllers. In a single master system bus architecture, the master always has the control over the bus. Hence the master can

transfer information whenever required. In case of multiple master system buses, before the master generates any data transfer control signals, it must first gain control of the bus. The master competes in an arbitration contest to own the bus for the required transaction.

A bus slave is a device that responds to being addressed by another card acting as a master. The slave does not compete in the bus ownership contest. The slave cannot initiate a transaction. When the slave wants a transaction to be done, it interrupts a master and the interrupt routine of the master takes care of the requested transaction. It acts as a responder in a transfer operation. The slave modules in a typical computer system are memory and I/O subsystems.

The arbitration protocol [1] is a set of rules used to indicate which masters require the bus and to specify their priority when more than one request is pending. These are the signals provided in the protocol of a multi master system bus for the master devices of the bus to compete for the bus ownership to initiate a transaction. The arbitration contest is the core mechanism to resolve bus ownership between one or more competing masters. There are two basic categories of arbitration control schemes.

- Centralized schemes, which use a special device (the bus controller) to arbitrate between requests and allocate the bus.
- Decentralized schemes, in which all potential bus masters have their own arbitration and allocation logic.

In case of a Centralized scheme the bus controller communicates with the bus masters using two control signals ("bus request" and "bus grant") as shown in Figure 1.2. Bus request is a common wired-OR input to the bus controller which can be read and asserted by any of the bus masters. Bus grant is an output from the controller which propagates along a daisy-chain from the highest

priority master through the other bus masters to the lowest priority master. Masters may take control of the bus only if they have made a request and they sense that their incoming bus grant line has been asserted. Masters not requesting use of the bus pass on the bus grant signal along the daisy chain.

Much faster allocation times are possible with decentralized schemes. On power-up one master is automatically given control of the bus. When it is ready to hand over control to another master, it asserts the bus available control signal (Figure 1.3). Masters that wish to use the bus respond by placing their priority codes on the logically wired-ORed bus priority lines. The master drives the most significant priority line first. The next significant line is driven only if that master's most significant bit of the priority code matches with the most significant bit of the bus priority lines. Thus only the highest priority master ends up driving all the bus priority lines. The priority code is defined so that if a master reads the bus priority lines and finds that the code it reads back is greater than its own priority code, then some higher priority master is also requesting control of the bus. If this is the case, it negates its own request by removing its code from the bus priority lines. The process continues until only the highest priority master is left asserting the priority lines. At the end of this arbitration process, the highest priority master asserts the bus accept control signal until the previous master has released the bus available signal. It then releases the bus accept and takes control of the bus.

The time taken by the arbitration process is known as the arbitration interval. The length of the arbitration interval depends on bus propagation delays and on the settling times of the arbitration logic in each of the competing masters. The bus protocols specify a fixed time period within which all masters must have completed arbitration.

The assignment of priorities is associated with the complex problem of fairness. If a fixed priority scheme is adopted, the higher priority masters may control the bus most of the time, and lower priority masters may never gain control of the bus. A round-robin priority scheme inevitably leads to a situation where the high priority masters must wait for relatively unimportant bus transactions to be completed. A variable priority scheme solves this problem to some extent. The priority of a master might be increased every time it fails in an attempt to gain control of the bus, or the priority of the current master might be set to the lowest possible value once it has gained control of the bus.

An interrupt is an asynchronous signal to indicate to the master that one of the slaves on the bus desires to transfer some information on the bus.

A transaction is the basic bus data transfer operation, which begins with a start cycle and ends with an acknowledge cycle. Start cycle, in general, may be asserting a start signal or putting the address on the bus and activating the status signals of the transaction. During an acknowledge cycle, the addressed module informs the master that it has responded as required. A master device on the bus can initialize the transaction by addressing another device on the bus.

In order to use the bus for a transaction, a master must first have ownership of the bus. It obtains the ownership by requesting the bus and waiting until the arbitration logic determines that the next owner will be itself. Then, the master waits until the bus is idle, and does a start cycle by addressing the desired address. The master which initiated the transaction drives the status lines to indicate the type of the transaction (read, write, block transfer). If it is a read transaction, the addressed module determines that the address refers to itself, then in some subsequent cycle puts the requested data on the data bus and issues an acknowledgement. This completes

a read transaction. In case of write transaction, after the start cycle the master puts the data on the data bus. In the subsequent cycle, the addressed slave module samples the data lines and then issues an acknowledgement, thus completing the transaction. The bus ownership may or may not pass to some other module.

A block transfer transaction is one in which a single address is conveyed by the master and multiple data items from sequential addresses are then communicated between the master and the slave.

Burst data transfer is a operation of read and write of multiple data items done by a DMA controller. It requires special signals given in the bus protocol to indicate the burst data transfer operation and to indicate the final burst cycle.

Two types of system buses are of interest to this work. A host system bus is the system bus for which an interface is to be designed. An External system bus is a system bus to be interfaced to the host system bus.

1.1.2 Characterizing System Buses : System buses can be characterized as follows :

- Width of the address bus : 16-bit, 20-bit, 24-bit, 32-bit. This characteristic determines the addressing capability of the bus.
- By number of the bits in the data bus: 8-bit, 16-bit, 32-bit. This determines the data format and the speed of the bus.
- Type of data transfer control : Synchronous or Asynchronous. If a bus has a synchronous data transfer control, that means the bus control signals change state (asserted or deasserted) only at a specific edge of the clock provided in the bus protocol. In case of asynchronous buses, no relationship exists between the clock provided and the state change of the signals.

- System Address/Data bus may be multiplexed or non multiplexed.
- Number and Type of interrupts : Edge triggered, Level triggered, Fixed priority, Programmable priority etc.
- Ability to handle multiple bus masters.
- Type of bus allocation protocol (Arbitration) : Centralized, Decentralized, etc.
- Direct Memory Access support given in the bus protocol : Burst data transfer capability, number of DMA channels etc.
- Maximum data transfer rate. This is also called as the bus bandwidth.
- The configuration process of the cards on the bus. Some bus architectures like Micro Channel and NuBus have special configuration registers and ROM for the card configuration. This eliminates the jumpers and switches from the system board and cards. Also it permits installation of multiple identical feature cards. [12] and [13]
- Electrical and Mechanical characteristics. Voltage and current specifications, Capacitive Loading of the bus lines, etc. are some of the electrical characteristics specified in the bus protocols. Number, layout, and spacing of lines; type of bus connector; size of the card, etc. are some of the mechanical characteristics specified.

Most computer systems have at least 3 levels of bus protocols, Local bus, System bus and the Inter system bus, for communication with different peripherals and I/O subsystems. Each bus has a different set of rules and signals to be matched for data transfer. The different buses should be interfaced properly to have an efficient communication between the devices on different buses. A good interface design allows extensive utilization of the facilities given in each of the buses, uses a small amount of real estate and power, and gives proper signals for the cards or the other buses to be interfaced.

1.2 Literature Survey

Research work has been done by Thurber and co-workers [24] to provide guide lines for the design of new bus protocols and to provide the frame work for understanding the operation of all types of existing buses. They identify a number of basic functions common to all buses and suggest a systematic classification of the different techniques that can be used to realize these functions. Michael Slater has developed a method of defining and describing bus operation which emphasizes the similarities in structure and control of a wide range of buses [1]. This simplifies interpretation of the manufacturer's bus specifications, which are all too often inadequately explained and difficult to understand. Extending this approach it is possible to develop the guide lines for the systematic design of bus interfaces.

There are several kinds of bus interfaces : Components to Local bus interface, Local bus to System bus interface, System bus to I/O bus interface, System bus interface for the cards, Host system bus to external bus interface. All the systems with more than one integrated circuit need to have a components to local bus interface. In case of single board computers this is the only kind of interface required. Most of the present day computers, to support I/O subsystems, have back plane buses on which I/O subsystems can be installed. So it is necessary to have a local bus to system bus interface. The computer system vendors take care of these two kind of interfaces [14]. The chip sets for system bus interface for some popular system buses have been designed by several vendors [15] and [16]. Present VLSI technology makes it possible to design integrated, lowpower interfaces.

Paul L. Borrill in his paper [26] says two or more buses are necessary in a computer system to interconnect system modules cost effectively. Super computers, for instance, often use VMEbus for I/O

rather than their proprietary system bus. Thus the host system bus to external system bus interface has a good commercial value. It enables the cards and I/O subsystems designed for the external bus to be used on the host system. Its utility is directly proportional to the popularity and acceptance of the external bus in the computer industry. A general block diagram of the interface between the two systems on two different system buses is shown in Figure 1.4

Design work has been done on the interfaces like the GPIB (IEEE 488 std.) interface for the Micro Channel [19], the Multibus interface for the PC bus [4]. Roger Russ in his design paper discusses a interface between the Unibus and the VERSAbus [25]. GPIB (IEEE 488 std.) interface card was designed for the Micro Channel using available bus controller chipsets for both the buses. Due to this interface the devices designed for the GPIB bus could be used on the Micro Channel [19]. Multibus interface for the PC bus was designed to take advantage of the PC environment by the devices designed for the Multibus [4]. It was designed using the off the shelf integrated circuits. Micro channel and the NuBus are the most popular system bus architectures in the micro computer industry. Design done for this thesis gives a Micro Channel slave interface to the NuBus.

1.3 Objective

The definition of the rules or protocols that specify how and in what form the digital information should be transferred and the design of the interfaces that control the flow of data to and from the buses are fundamentally important to the construction and operation of the entire computer. The interpretation of the manufacturer's specifications and the design of logic to implement bus interfaces remain two of the most time consuming and difficult problems facing the Digital Systems Engineer[3]. This thesis addresses the problem of implementation of the bus interfaces.

The objective of this thesis is to develop a design procedure for the bus interface design and to use this procedure to design a Micro Channel interface to the NuBus architecture as shown in Figure 1.4. The design procedure developed for this thesis discusses several problems encountered while designing a interface between two system buses and discusses several techniques for solving these problems. Chapter two includes discussions about level of interface, master and slave interface between the two buses, transparent and non transparent interfaces, address mapping, data format interface, signal synchronization, etc. Chapters three and four give a overview of the Micro Channel and the NuBus protocols. The fifth chapter discusses the differences between the Micro Channel and the NuBus protocols and explains the design strategies used to design this interface. Chapters six and seven have the descriptions of the Micro Channel interface for the NuBus.

Since both, the Micro Channel and the NU bus architectures are new and dissimilar bus architectures, developing an interface would prove to be a challenging task. The interface would prove to be vital in reducing the design time in the development of the I/O subsystems for the NU bus.

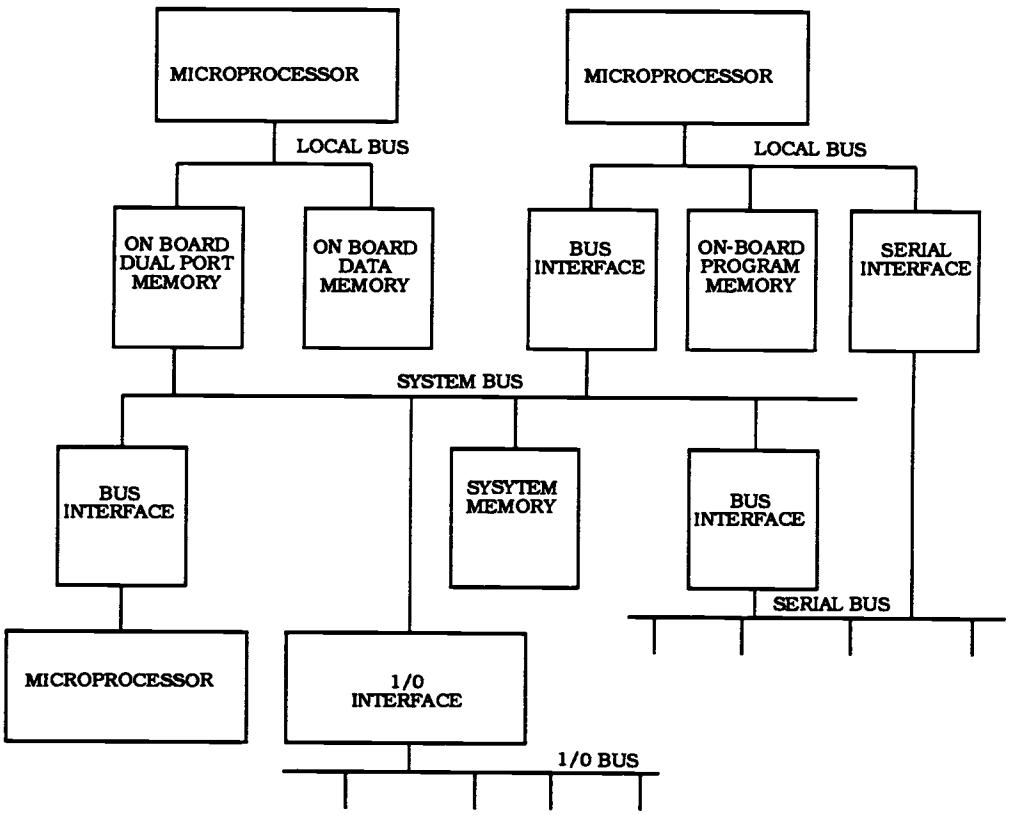


Figure 1.1 : Block Diagram of a Computer System with Four Levels of Buses

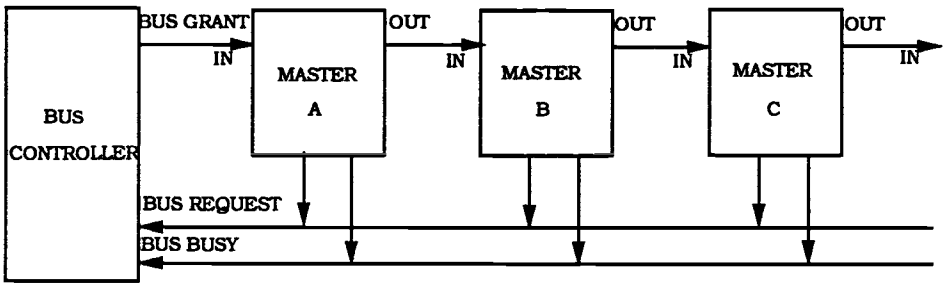


Figure 1.2 : Centralized Arbitration Techniqe

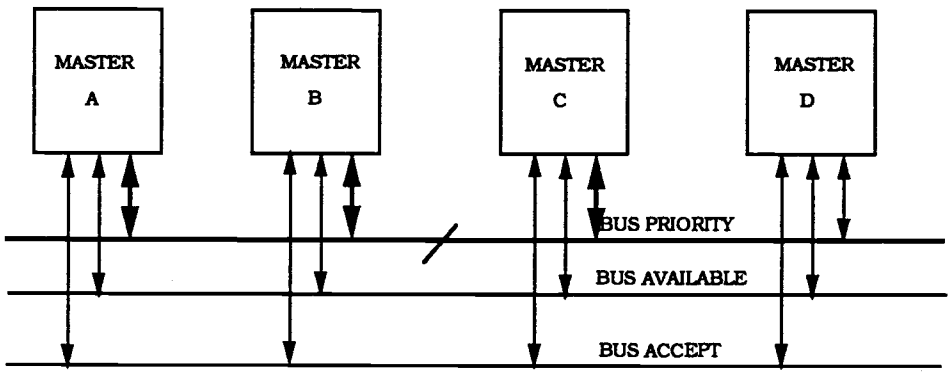


Figure 1.3 : Decentralized Arbitration Technique

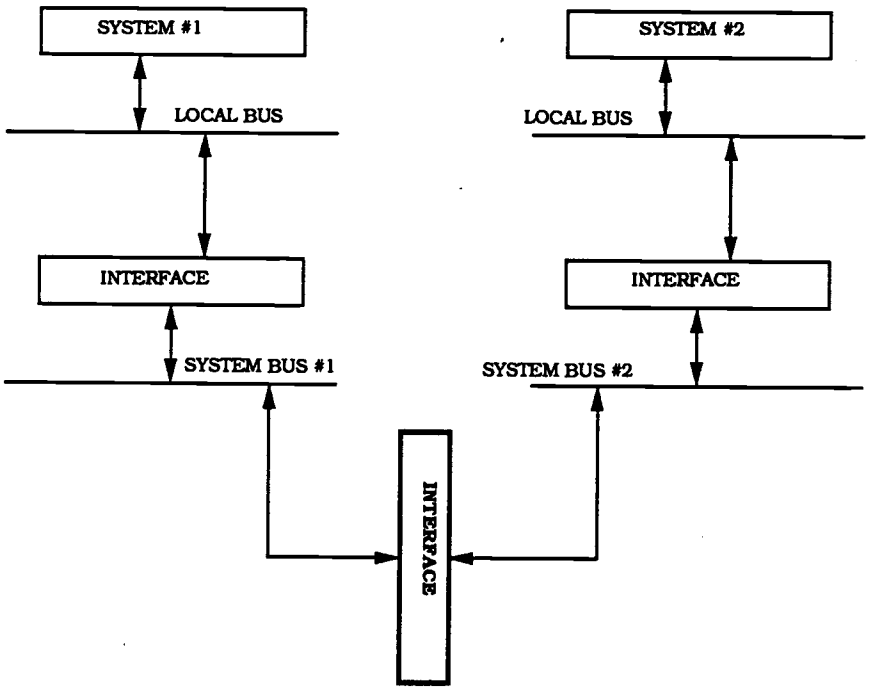


Figure 1.4 : Block Diagram Of a Interface Between Two Systems on Two System Buses

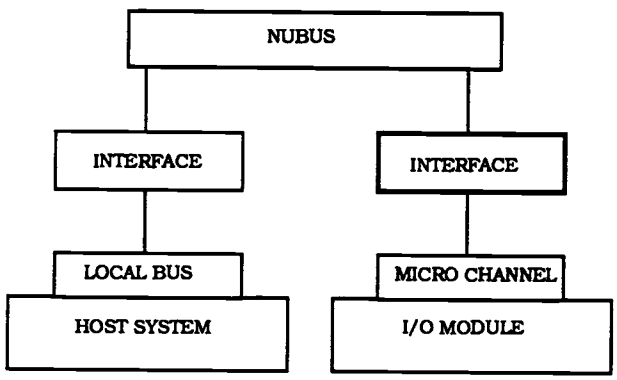


Figure 1.5 : Block Diagram of a NuBus System with Micro Channel Interface

2. BUS INTERFACE

2.1 Introduction

There can be several levels of buses in a computer system. Each may have its own unique protocol. Hence a bus interface between the local bus and the system bus is required for proper communication between the local processor and the devices on the system bus. Also the interface between the system bus and the intersystem bus ensures efficient communication between the devices on the two buses. Another aspect of the bus interface is the need to provide several system bus interfaces to the host system bus. The main advantages of this kind of interface are as follows.

- Devices designed around one system bus can be used on another system bus with little changes in the software and hardware associated with the devices. This drastically reduces the design time and cost.
- Two systems designed around different bus architectures can communicate and share resources

A bus interface is a front-end logic circuit that converts the physical, electrical, and the functional characteristics of a device or another bus into those defined in the host bus specification.

The bus interface can be of two types.

- Master interface
- Slave interface

The Master interface enables a master device designed for the bus to be interfaced to the bus. i.e. the device can compete in the arbitration contest to own the bus. This interface must provide a proper interface to the device with all the signals of the bus as shown Figure 2.1. A Master interface is necessary if two systems on

the two different buses must communicate efficiently. This allows the systems to share the resources like memory.

A Slave interface facilitates the interfacing of a slave device to the system bus (Figure 2.2). As the slaves do not compete in the arbitration contest, the arbitration signals can be neglected. This results in a simple and cost effective design. Most of the cards designed for any bus are designed to be the slaves on that bus. The slave interface given between the two system buses facilitates the use of slave cards designed for the external system bus to be used on the host system bus.

2.2 Strategies Of Bus Interface Design

2.2.1 Transparency

The ideal bus interface is transparent to the host system. The host should perceive no difference between an access to its own internal memory or I/O space, and an access to memory or I/O space that is resident on the external bus.

For every application a completely transparent interface won't be possible. Examples include situations where the host bus is a high- speed bus with no facility for slow data transfers, or when interfacing a 8-bit data bus to a 32- bit data bus that does not provide facilities for byte- wide data transfers. Neither will a transparent interface be possible when the external bus has a memory or I/O space that's larger than the host can support.

Nontransparent operations require some knowledge of the external bus. Special facilities should be included in the design to perform these operations. Such special facilities might consists of software drivers and additional bus - conversion logic to handle all external bus communication. Even if it is possible to develop a

completely transparent interface for an application, the cost of the necessary hardware and the inconvenience of any special handling techniques may be prohibitive. The designer must be aware of the tradeoffs which depend on the application as well as on the two buses used.

2.2.2 Function of the Interface

The designer must understand the function the interface must provide. He must decide whether the interface should be a master interface or a slave interface. If a master interface is provided, the masters of the external system bus can be interfaced to the host system. All the signals of both the buses should be matched. The design may become complex and expensive. In general, a slave interface makes the design simpler and cost effective. The arbitration signals need not be interfaced. The slave cards of the external bus can be used on the host system. A survey has shown that most of the cards designed for any bus are slave cards. Hence the slave interface serves the purpose of using the cards of the external bus on the host system.

2.2.3 Synchronization

In the design of bus interfaces, the major issue of concern is synchronization. In many cases, the two buses are not synchronous with each other. The time at which the signals of one bus become active may not have any relationship with the clock signal of the other. Also the problem of metastability must be considered.

The flipflops which are used for synchronization have certain timing requirements on the inputs. The setup time of a flipflop is the finite time for which a input should be stable before the clock edge arrives. The hold time is the finite time for which a input should not change state after the clock edge. If a signal does not satisfy these

conditions the output of a flipflop may go into damped oscillations, which may be hazardous for the system. This condition of the flipflop is known as metastability.

There can be three combinations of two buses to be interfaced; both synchronous, both asynchronous, and one synchronous and another asynchronous. Several approaches are present to achieve synchronization.

A separate clock signal can be chosen (depending on the speed of the transfer) for the interface and the protocol signals of both the buses can be synchronized with respect to this clock using flipflops. Master-slave flipflops must be used to avoid metastability as shown in Figure 2.3. Figures 2.4 and 2.5 show two conditions, one where the input asynchronous signal satisfies the flipflop timing requirements and the other where the input does not. This method of synchronization can be employed for any kind of bus combination. All the inputs to the interface must be synchronized with respect to the clock of the interface. If both the buses are synchronous all the outputs of the interface must be synchronized with respect to the respective clocks. If both the buses are asynchronous the outputs to the buses need not be synchronized. If one of the buses is synchronous the outputs for that bus must be synchronized with respect to its clock.

The clock of one of the buses can be used for the interface design and the signals of the other bus can be synchronized with respect to this clock using master-slave flipflops. This approach reduces the overhead of synchronizing the signals of both the buses. The input and output signals of the other bus need to be synchronized with respect to its clock. If the other bus is a asynchronous bus, only the inputs from this bus must be synchronized with respect to the clock of the interface. This method cannot be employed if both the buses are asynchronous. Also

it is not the best scheme to be used when a faster transfer rate between the two buses is required.

Address and data on the buses can be synchronized by using tri-state drivers and enabling these drivers by proper control signals which are activated in synchronous with the required clock as shown in Figure 2.6.

2.2.4 Address Mapping

Depending on the application, the interface may have to provide address mapping for memory or I/O addresses on the external bus which are inaccessible to the host or those duplicate addresses needed by the host for its other functions. This means some kind of address mapping has to be provided to access those addresses in the memory map of the external bus which are not present or reserved in the memory map of the host system bus. Depending on the address space there are three cases of interface. The host system bus and external system bus can have the same address bus width. The host system bus address width can be more than that of the external system bus or viceversa. Also one of the buses or both the buses may support a separate I/O space. In each case the address mapping from one bus to the other can be done in several ways.

As one possible solution, the problem of mapping can be left to the card designer. The card designer can be asked to configure the cards designed for the external bus in such a way that they map on to the proper memory space in the memory map of the host system bus. This method can be used to solve any mapping problem. Even if this design requires a small change in the card hardware, it saves the cost of hardware required for mapping in the interface and the software drivers required to perform the mapping function.

If the host system bus and the external system bus, both have the same address width and same memory map then the mapping is not required. If the memory map is not the same then the address interface must provide a mapping function.

As a general solution to address mapping, all of the memory on the external system bus memory can be addressed by adding a address-mapping-function register to the bus interface logic and arranging for software routines to load high order address bits into this register. This provides a means for the host system bus to selectively access the external system bus address space in different windows as shown in Figure 2.7. The example of interfacing a system bus with 24-bit address to a system bus with 32-bit address is shown in this figure. This allows the system bus with 24-bit address to access 4 GBytes of 32-bit address space as 256, 16 MByte windows.

If the host system bus address width is less than that of the external system bus then the address interface logic must provide a means for the host system to access entire memory space of the external system bus. The mapping can be provided as explained above.

If the host system bus address width is more than that of the external system bus then a dedicated block of the host system bus memory space can be committed to the external system bus address space and therefore only address translation required is a decoded select signal for the interface as shown in Figure 2.8.

If both the buses support a separate I/O space then the address interface for the I/O space should be designed in a similar way as the address interface for the memory.

If the external system bus supports a special I/O space and the host system bus does not, then the card designer can be asked to

change the design to be a memory mapped I/O. Another solution for this problem is to dedicate a specific block in the memory map of the host system bus as the I/O block of the external system bus. In the address interface this address block should be decoded and the interface should issue proper I/O read or write commands to the external system bus.

If the host system bus supports an I/O space and the external system bus supports only memory mapped I/O, no interface needs to be provided for I/O addresses.

2.2.5 Data Format

The difference in the data format of the two buses under consideration should be studied carefully and any differences should be taken care of.

If the two buses have the same data width then the two data buses can be interfaced directly. If the external system bus data width is smaller than that of the host system bus the external system bus data lines can be interfaced to part of the data bus of the host system bus. The host system bus configuration process must provide a means to inform the host system about the byte lane usage of this external data bus. If the external system bus data width is more than that of the host system bus, then the configuration process of the external bus must provide a means to inform the device on the external system bus about the byte lane usage of the host system bus. Interface does not do word conversions; it is the responsibility of the devices on the buses.

Big endian (Figure 2.9) and the little endian (Figure 2.10) representations of the words [2] is one of the problems to be solved in case of 16 and 32-bit data bus interfaces. The big endian representation of a word of data has its most significant byte of data

on the data bits 0 to 7 and least significant byte of data on the data bits 24 to 31. A little endian representation has its most significant byte on data bits 24 to 31 and least significant byte on 0 to 7 data bits. This creates problem in storing the data in the memory. The big endian representation stores the most significant byte of data in the lower memory address space, where as a little endian representation stores the least significant byte of data in the in the lower memory address space. If one of the buses is big endian and the other one is the little endian the data lines should be properly configured such that proper bytes are available to both the buses as shown in Figure 2.11.

2.2.6 Multiplexed Buses

In case one of the buses has a multiplexed address/data bus and the other has separate address and data buses then the address/data interface must provide the function of multiplexing and demultiplexing. The address must be put on and removed from the address bus at proper time as explained in the bus specifications. The interface must provide required control signals to achieve this multiplexing and demultiplexing function and timing requirement. Figure 2.12 shows a example of interfacing a multiplexed address/data bus to a non multiplexed address/data bus.

2.2.7 Arbitration

The master interface must provide the interface for the arbitration signals of the two buses under consideration.

If both the buses have a centralized or decentralized arbitration protocols then the interface has to just pass the arbitration signals to or from the host system bus. The logic has to pass the signals in proper sequence and with correct logic levels as specified in the bus protocols of the two buses as shown in Figure 2.13. If one of the bus

protocols does not support the signals required by the arbitration protocol of the other system bus then the arbitration controller must generate those signals in correct sequence.

If the external system bus has a centralized arbitration scheme and the host system bus has a decentralized arbitration scheme then the arbitration control logic of the interface is complex. It has to be a central arbitration control point for the external system bus and a local arbitration control point for the host system bus as shown in Figure 2.14. As a central control point it has to receive bus request signals from the external system bus, pass this request to the host system bus as a local control point, get the ownership of the host system bus or pull-out from the contest, and pass this information to the external system bus. The state machine approach is the standard method used to implement this arbitration control logic of the interface.

If the external system bus has a decentralized arbitration scheme and the host system bus has a centralized scheme then a similar design as explained above must be implemented. The arbitration control logic must receive the bus requests from the external system bus and pass it to the host system bus. The host system bus central arbitration control point makes the decision to give the bus ownership to this request or not. The interface controller has to pass this information to the external system bus.

2.2.8 Bus Protocol

The bus interface between the two systems must provide all protocol handling functions, such as signal timing and additional wait states. The interface must interpret the control signals of the host system bus and generate necessary control signals for the external system bus. A state machine has to be designed to generate control signals and to pass address/data between the external system bus

and the host system bus in a proper sequence as described in the transaction protocols of the two buses. This state machine can be implemented using any one of the methods of implementation as discussed in Section 2.2.10.

2.2.9 Interrupts

The host should be able to do interrupt servicing for both systems in exactly the same manner. The interrupts must be interfaced such that the interrupt active logic levels should be matched. If the external system bus has edge triggered interrupts and the host system has level triggered interrupts then the interface logic has to recognize the edge triggered interrupt from a device on the external system bus and interrupt the host system with a interrupt signal of proper pulse width.

The external bus interrupts must be ORed together in to one or more groups and connected to one or more of the host system bus interrupts. If the external system bus interrupt logic requires an acknowledgement then the controller must drive that signal when it accepts the interrupt from a slave on the external system bus. If the host system bus also supports a interrupt acknowledge signal then this signal can be passed to the external system bus in correct sequence and proper timing. The slave module on the external system bus must be able to interrupt the host system in exactly the same as a slave module on the host system bus. Figure 2.15 shows a general interface for the interrupt structures of the two buses.

2.2.10 Implementation

The method of implementation is a critical decision to be made while designing the bus interface.

Bus support devices: The logic to implement particular parts of the bus protocols is fabricated as one or more integrated circuits. These are designed for a specific interface. The design of interfaces based on bus support devices is relatively inflexible and expensive, but it is the most efficient interface. Under the present requirements where the speed of data transfer is a critical issue and with well developed VLSI technology this solution appears to be the best for the interface design.

Semicustom interface devices: The logic is implemented using an array of uncommitted logic elements which has been fabricated on a master slice. Interface is developed using a CAD package that takes the random logic design, determines the required interconnections pattern, automatically routes the necessary tracks. In this case the faster development of the interface is possible. But this does waste considerable chip area which in turn reduces the speed.

Field-programmable logic arrays: The interface is implemented on one or more user- programmable AND/OR arrays. These universal multi- input/multi-output logic circuits are programmed by blowing on-chip fusible links. Some devices have clocked flipflops connected to the inputs and outputs and can be used to implement entire synchronous interface designs. As the PLA technology is developing at a rapid speed, the development of a bus interface can become faster and inexpensive.

Single-chip microcontrollers: A logic is realized in firmware using a general-purpose, program controlled VLSI device-the single chip microcontroller. The microcontroller has an instruction set designed to facilitate I/O operations, and can be programmed to implement a complete interface in one device. But this kind of design results in slower bus conversion, which results in slower data transfer rates.

2.3 Design Algorithm

The interface between two system buses must have a address/data interface and a interface controller. The address/data interface depends on the address and data formats of the two buses under consideration. It also depends on the strategies explained in Sections 2.2.1, 2.2.2, 2.2.4, 2.2.5 and 2.2.6. The address/data interface may provide some select signals if necessary. It must have proper input control signals to control it's operation. Figure 2.16 shows a general block diagram of an address/data interface for the case of one multiplexed bus and the other non multiplexed bus. The address/data interface can be implemented using off the shelf buffers, multiplexers, decoders and comparators or it can be implemented as a custom VLSI component.

The interface controller must support all the control signals of both the system buses and the control signals required for the correct operation of the address/data interface. It must support read and write transactions on both the buses. The control signals must be driven in a proper sequence as defined in the protocols of the two buses. A state machine must be designed to achieve this function. The control signals for the address/data interface must be generated according to the timing requirements defined in the bus protocols. The controller must provide a proper interface for the interrupt structures of the two buses as explained in the Section 2.2.9. The inputs and the outputs of the controller must be synchronized with respect to the respective clocks as explained in the Section 2.2.3.

Depending on the decision made about the function of the interface (Section 2.2.2) the interface controller may have to provide correct interface for the arbitration signals of the two buses as explained in Section 2.2.7. If a master interface between the two buses is desired then a state machine must be designed to facilitate the interface controller to go through the arbitration sequence of the

two buses. Once this arbitration state machine gets the control of the desired bus it must give the control to the state machine controlling the transactions on the buses. After the transaction is over the transaction control state machine must give the control to the arbitration control state machine. Thus these two state machines must work hand in hand (Figure 2.17) in case of master interface and during the transactions between the masters of the two buses.

A general flow-chart of sequence of operations of a interface controller is shown in Figure 2.18. The interface controller can be implemented using any method discussed in the section 2.2.9. The method to be used depends on the complexity, cost, design time and data transfer rate of the two buses under consideration.

Proper decision should be made for each point listed depending on the requirements and tradeoffs before the design procedure begins. Good planning and proper design decisions result in lesser design iterations and design time and lower cost, which are the basic requirement of any design process.

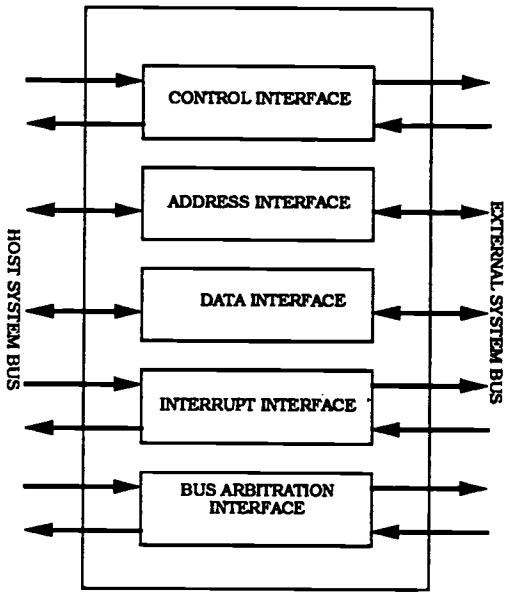


Figure 2.1 : Master Interface Between the Host System Bus and the External System Bus

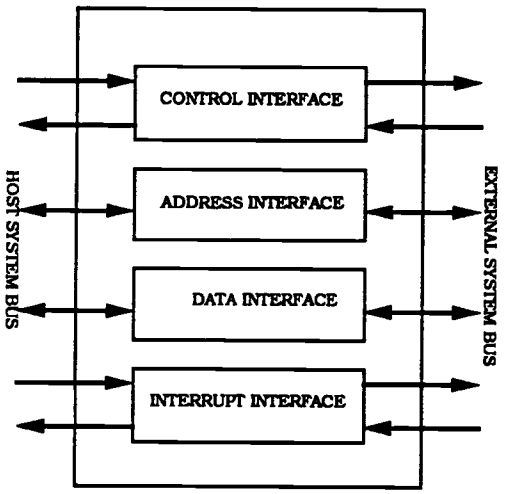


Figure 2.2 : Slave Interface Between the Host System Bus and the External System Bus

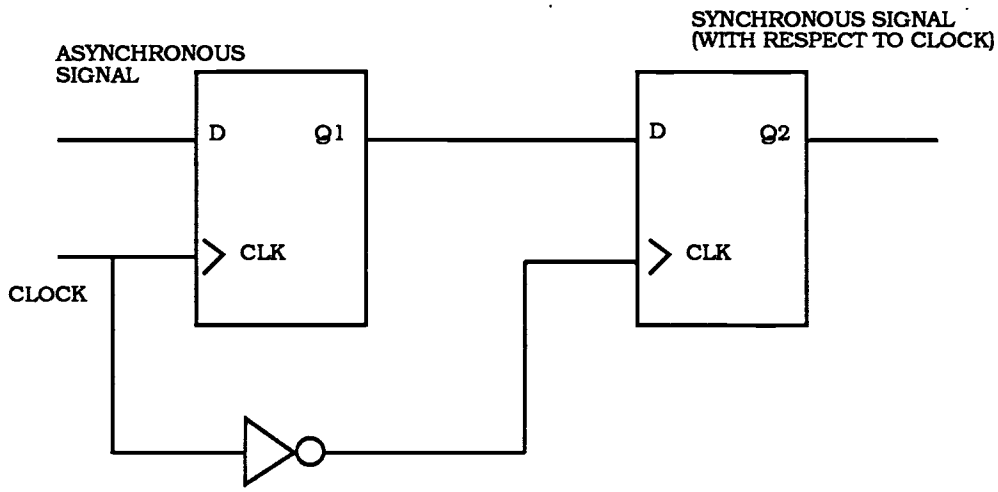


Figure 2.3 : Synchronizer

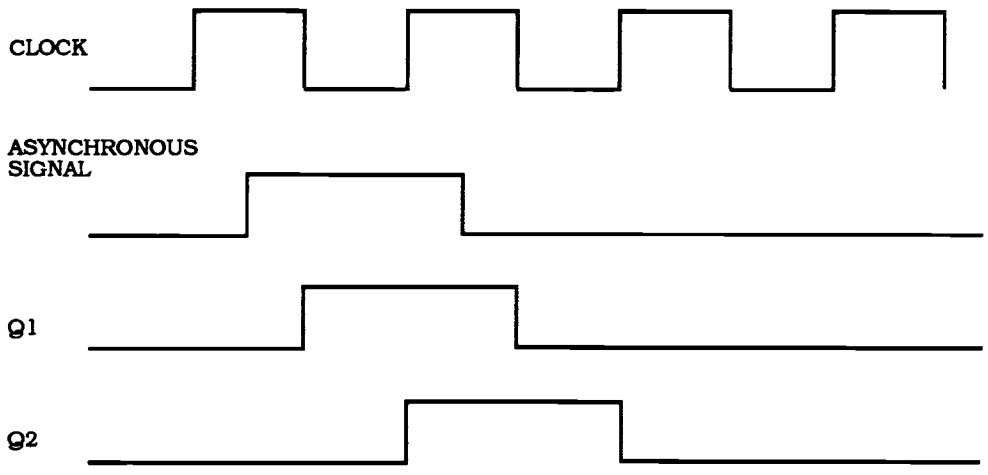


Figure 2.4 : Asynchronous Signal Satisfies Flipflop Timing Requirements

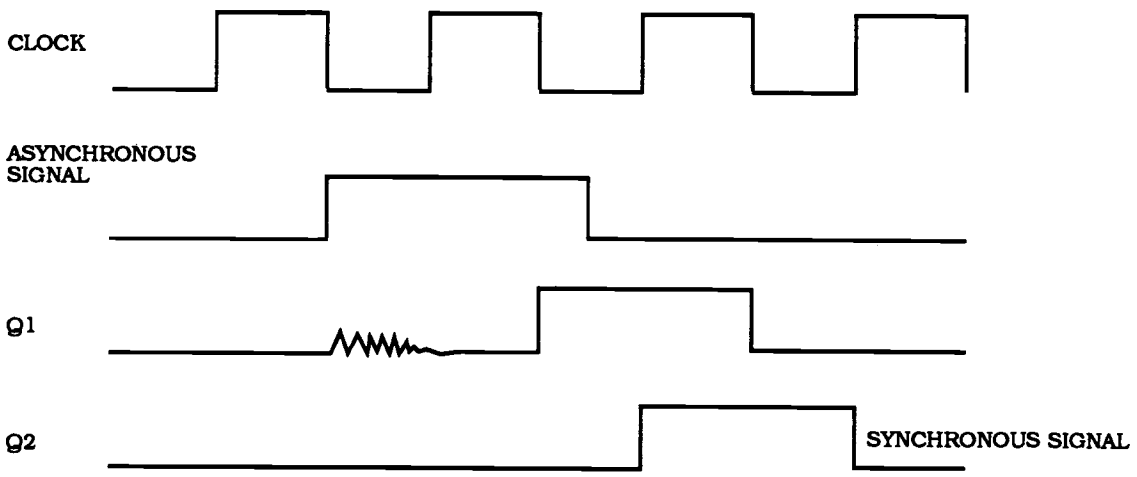


Figure 2.5 : Asynchronous Signal Arrives at the Clock Edge

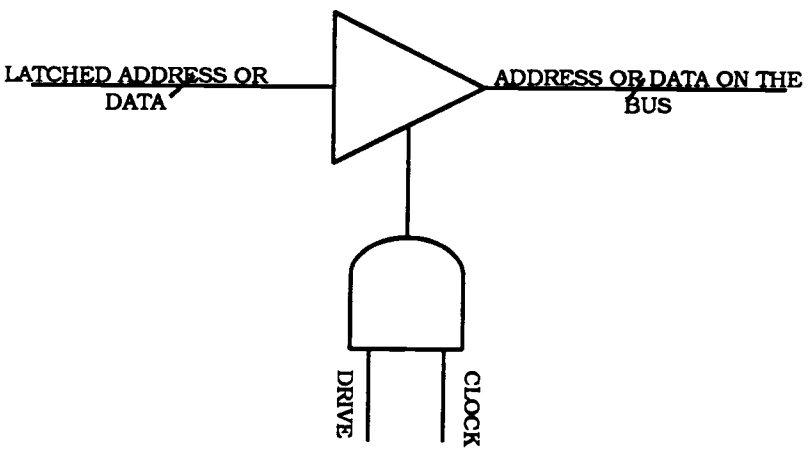


Figure 2.6 : Address/Data Driver

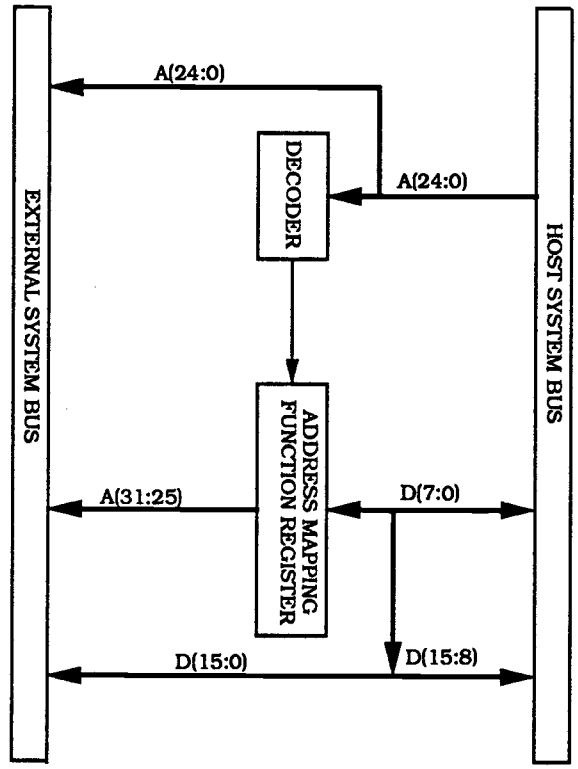


Figure 2.7 : Address Mapping

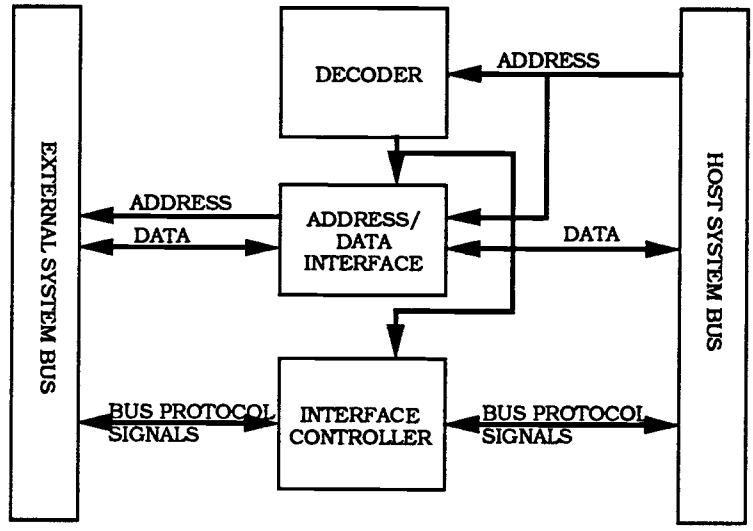


Figure 2.8 : Address Decoding

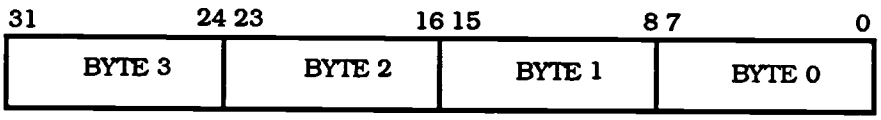


Figure 2.9 :Little Endian Representation

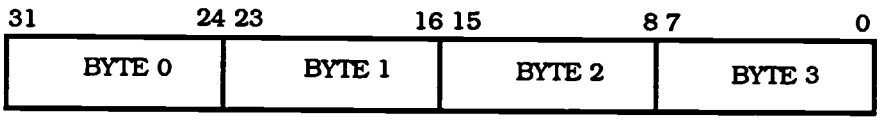


Figure 2.10 : Big Endian Representation

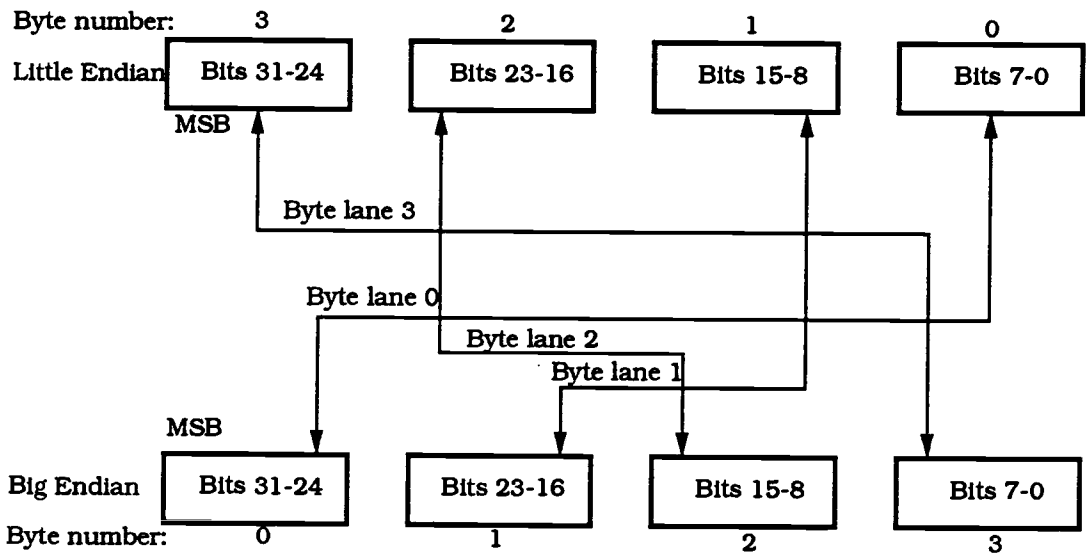


Figure 2.11 : Byte Swapping

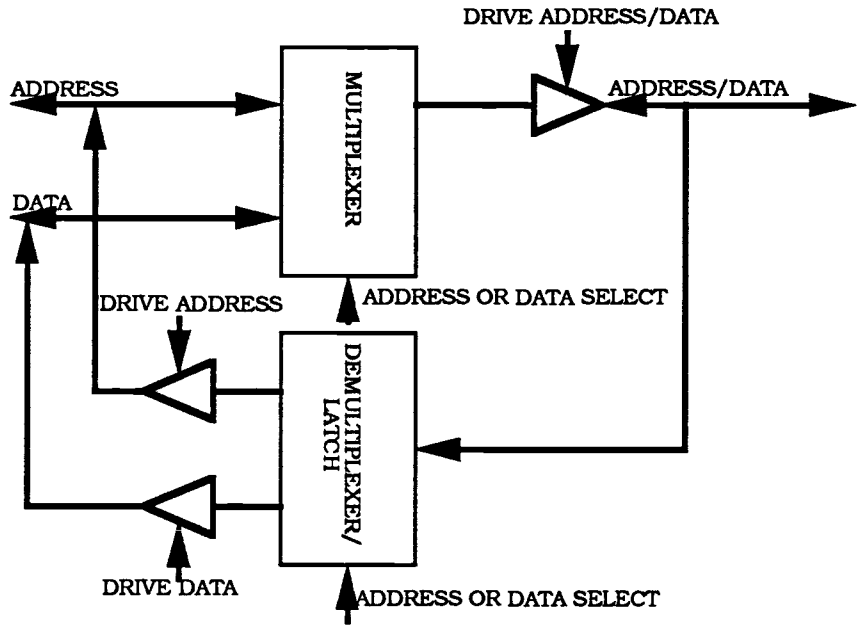


Figure 2.12 : Interface between a Multiplexed and a Non-Multiplexed Buses

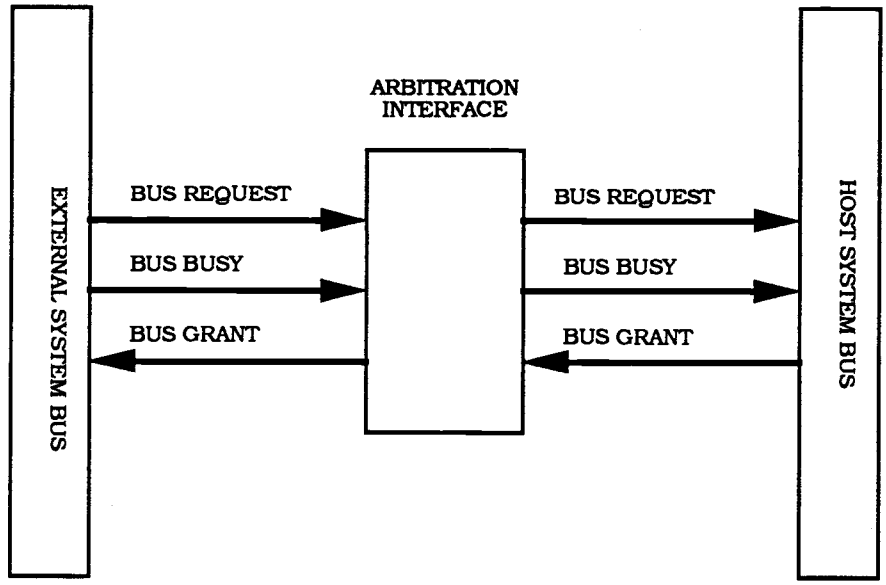


Figure 2.13 : Arbitration Interface between Two System Buses with Centralized Arbitration Schemes

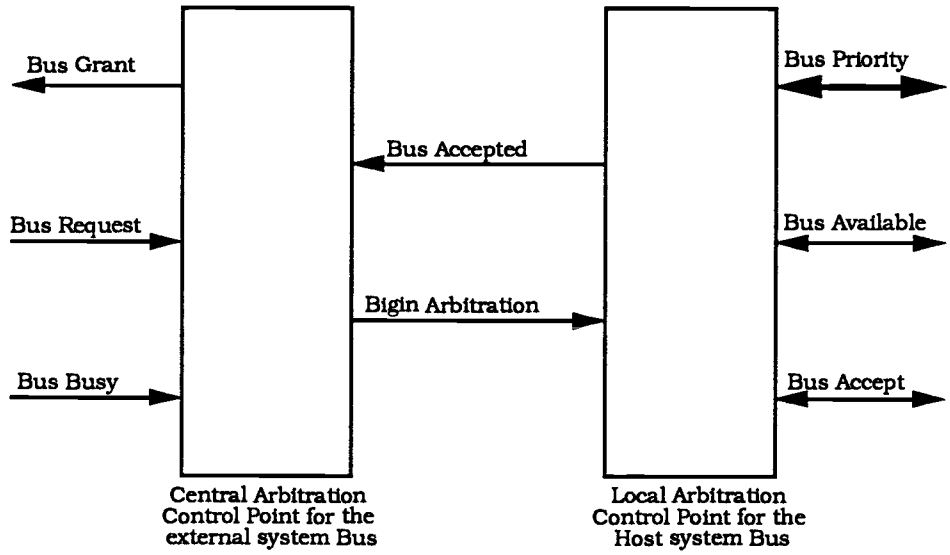


Figure 2.14 : Arbitration Interface between a Centralized Arbitration Scheme and a Decentralized Arbitration Scheme

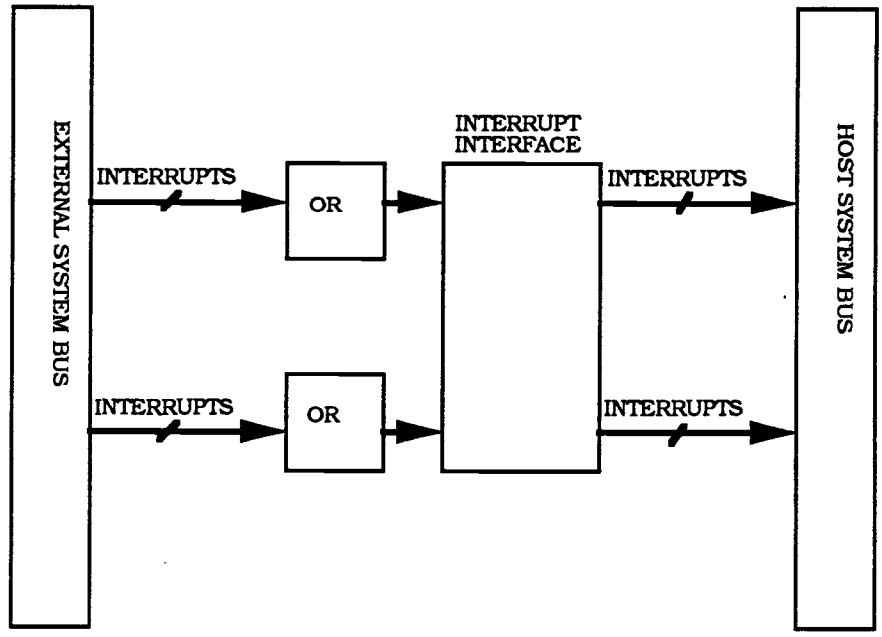


Figure 2.15 : Interrupt Interface

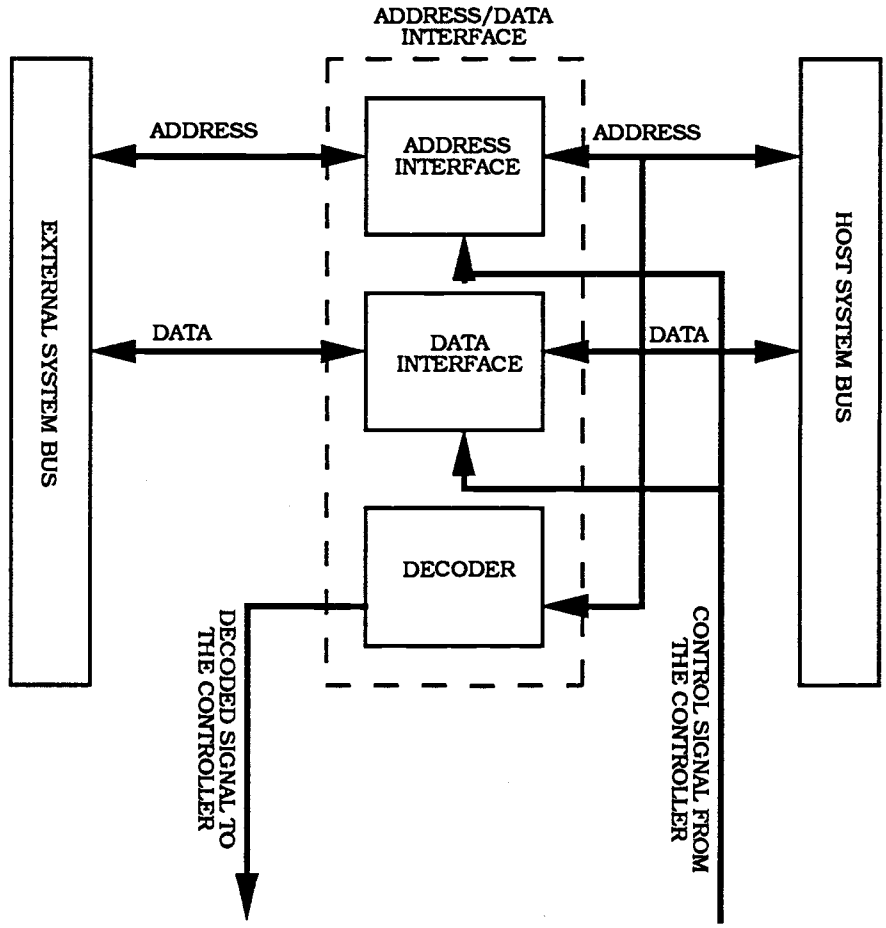


Figure 2.16 : General Block Diagram of a Address/Data Interface

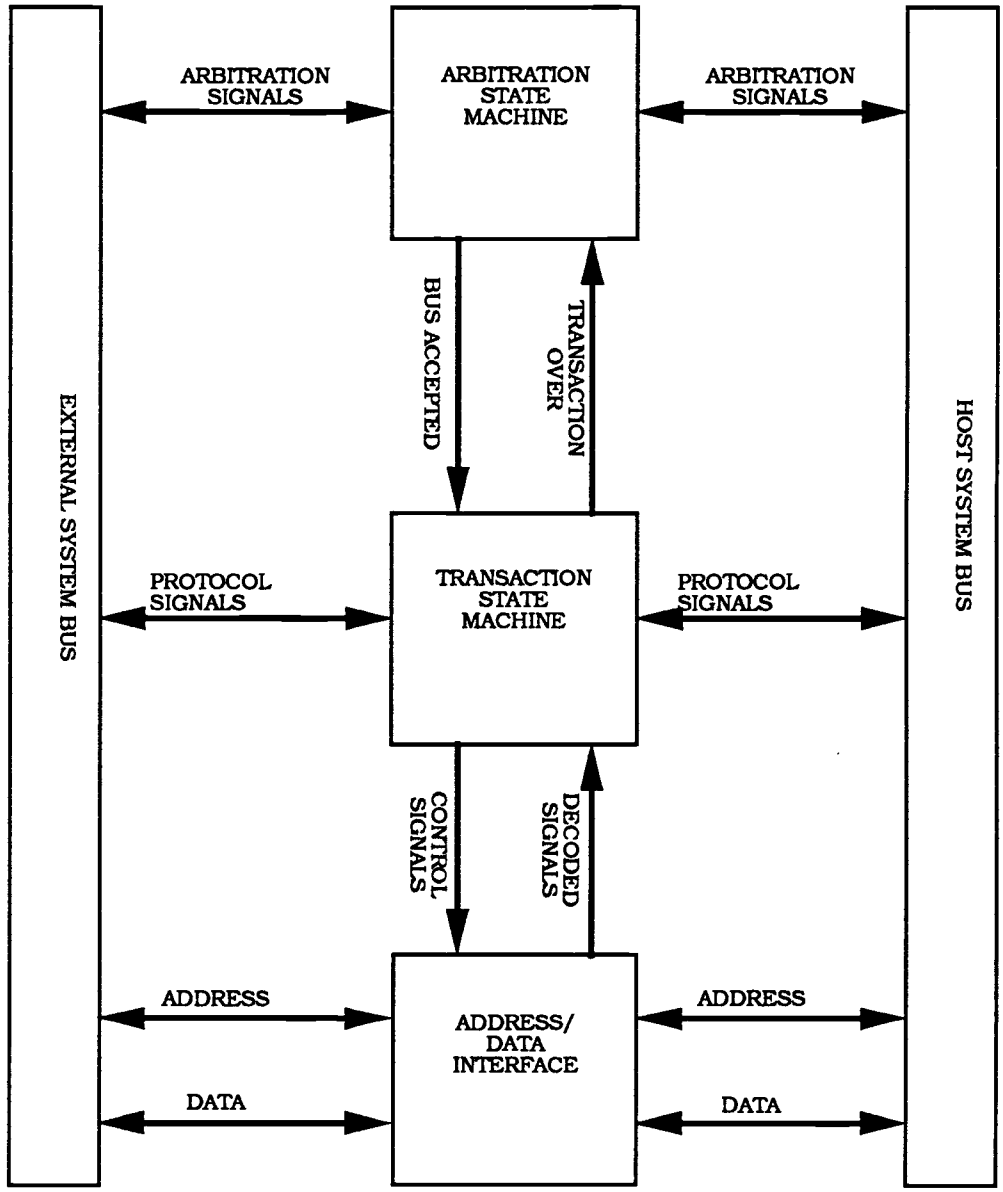


Figure 2.17 : General Block Diagram of a Interface between two System Buses

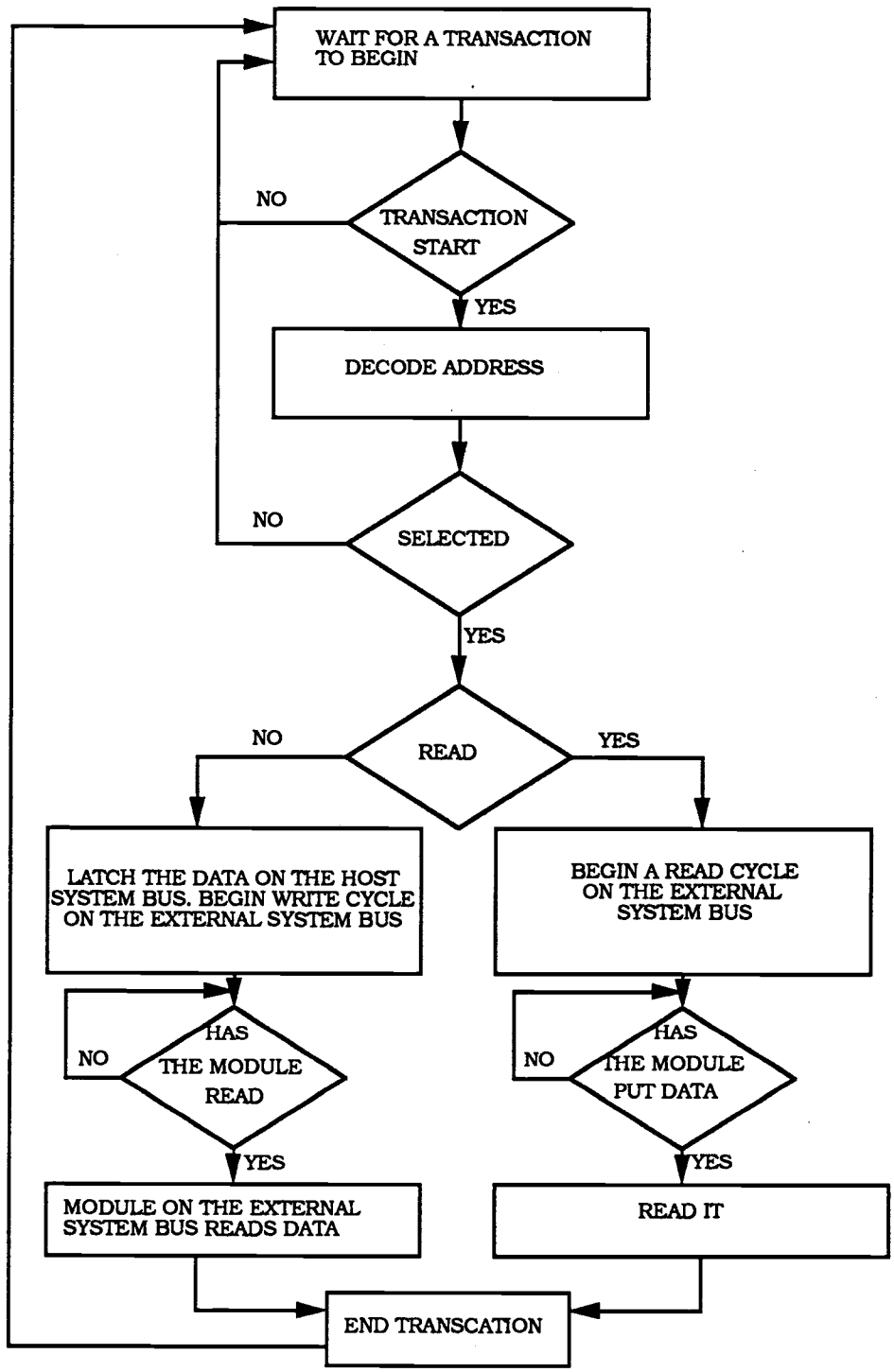


Figure 2.18 : Flow Chart of Interface Controller Operation

3. INTRODUCTION TO THE MICRO CHANNEL ARCHITECTURE

3.1 General Description

The Computers of today require a bus based architecture to increase processing speed, to use coprocessing or multiprocessing and to adapt to new peripherals. IBM, to fulfill these capabilities, developed the Micro Channel, 32-bit System bus to be used in Personnel System 2 higher end models. Micro Channel is used as both local CPU bus and System bus in PS/2. Material for this section is mostly derived from [12].

The Micro Channel is comprised of an Address bus, a Data bus, a Transfer Control bus, an Arbitration bus, and other support signals. The bus is asynchronous for Control and Data transfer between memory, I/O devices, and the main processor.

The salient features of the Micro Channel are as follows :

- An I/O Address width of 16-bits allows 8-, 16-, or 32- bit transfers within an address space of 64K. A memory address width of 24 and 32 bits allows 8-,16-, or 32-bit memory transfers with in 16 MBytes and 4 GBytes ranges respectively.
- Programmable Option Select feature that replaces hardware jumpers and switches for system configuration and permits installation of multiple identical feature cards. The definition file of a card programs the POS registers during the power-up sequence. Cards can be mapped anywhere in the memory. This makes it possible to instal multiple identical feature cards which can be mapped at different memory spaces.
- Level sensitive interrupts with interrupt sharing on all levels. This is to reduce the transient sensitivity of the interrupt controller.

- A serial DMA protocol that supports eight DMA channels for 8 or 16- bit DMA transfers.
- It has a central arbitration control point that allows upto15 devices to arbitrate for control of the Micro Channel. This also allows burst data transfers.

The Channel supports all signal, power, and ground signals to adapters. The PS/2 system board provides three types of channel connectors:

- 16-bit
- 16-bit with video extension
- 32-bit

The interface designed for this thesis deals with the 32-bit bus. The 32-bit channel is an extended 16-bit channel designed to accommodate 32-bit addressing, and 32-bit data transfers. It has 107 signal lines, 44 power and ground lines, a separate audio ground line, 21 reserved lines and four keyed positions in a dual 93-pin, 50-mil card edge connector.

The Micro Channel can behave as an extended local bus of Intel 80286, 80386 Microprocessors or as a System bus independent of the processor. All logic signals are TTL-compatible. A brief description of the relevant signal lines is given below.

3.2 Signal Description :

A0 - A31: Address bits 0 through 31 : These lines are used to address memory and I/O devices attached to the channel. These lines allow access to upto 4 GBytes of memory. Figure 3.1 shows the memory map of the Micro Channel. The lower 16 lines allow upto 64 KBytes of I/O address space. The valid addresses generated by the microprocessor are not latched and hence must be latched by the

slaves using either the trailing edge of -ADL (Address Decode Latch) signal or the leading edge of -CMD (Command) signal.

D0 - D31 : Data bits 0 through 31 : These lines provide data bus bits for the system microprocessor and slaves. It is made up of the high byte and low byte. During read cycles, data is valid on these lines after the leading edge but before the trailing edge of CMD, and must remain valid until after the trailing of CMD. During write cycle, data is valid throughout the period when CMD is active.

-ADL : -Address Decode Latch : This line, driven by the system micro-processor, is provided as a convenient mechanism for the slave to latch valid addresses and status bits. The addresses and the status bits can be latched with the trailing edge of -ADL or the leading edge of -CMD.

-CD DS 16 (n) : -Card Data Size 16 : This line is driven by 16-bit and 32-bit memory, I/O, or DMA slaves to provide a indication on the channel of a 16-bit or 32-bit data port at the location addressed.

-DS 16 RTN : -Data Size 16 Return : This output signal is a negative OR of the -CD DS 16 signals from each channel connector.

-SBHE : -System Byte High Enable : This line indicates and enables transfer of data on the high byte of the data bus (D8 - D15), and is used with A0 to distinguish between high byte (D8 - D15) and low byte (D0 - D7) transfers.

MADE 24 : Memory Address Enable 24 : This signal indicates when an extended address is used on the bus. If memory cycle is in progress, MADE- 24 inactive indicates an extended address space greater than 16M is being presented; MADE 24 active indicates an unextended address space less than or equal to 16M is being presented.

M/-IO : Memory/-Input Output : This signal distinguishes a memory cycle from an I/O cycle.

-S0, -S1 : -Status bits 0 and 1 : These lines indicate the start of a channel cycle and also define the type of channel cycle. When used with M/IO, memory read/write operations are distinguished from I/O read/write operations.

-CMD : -Command : This signal is used to define when data is valid on the data bus. The trailing edge of this signal indicates the end of the bus cycle.

-CD SFDBK (n) : -Card Selected Feedback : When a memory slave or an I/O slave is addressed by the system microprocessor, this signal is driven by the addressed slave as a positive acknowledgement of its presence at the specified address. This signal line is unique to each channel connector.

CD CHRDY (n) : Channel Ready : This normally active line is pulled inactive by a memory or I/O slave to extend the time to complete a channel operation. This line cannot be held inactive for more than 3.5μs. This line is unique to each channel connector.

ARB0 - ARB3 : Arbitration Bus Priority Levels : These lines make up the arbitration bus and are used by the participating entities to present their priority levels for arbitration. Up to 16 levels of priority are available.

ARB/-GNT : Arbitrate/-Grant : When high, this signal indicates that an arbitration cycle is in progress. When low, this signal is the acknowledgement from the central arbitration control point (CACP) to the winning arbiter that channel control has been granted.

-PREEMPT : -Preempt : This signal is used by bus entities to request the control of the channel, through arbitration. When this signal is activated an arbitration cycle is initiated.

-BURST : -Burst : This signal is driven by arbitrating participants to indicate to the CACP the extended use of the channel when transferring a block of data. This signal initiates a burst cycle.

-TC : -Terminal Count : This line provides a pulse on the channel during a read or a write command to indicate that the terminal count of the current DMA channel has been reached.

-IRQ 3 - 7, -IRQ 9 - 12, -IRQ 14 - 15 : -Interrupt Request : These are the level sensitive interrupt lines of the channel used by the slaves to request attention from the system microprocessor. These lines have priorities attached to them.

-CD SETUP (n) : -Card Setup : This signal is driven by system board logic to individually select channel connectors during system configuration and error recovery procedures. Each slot has an unique CDSETUP line. The system will generate a CDSETUP when an I/O access is made to addresses 0100h-0107h.

-CHCK : -Channel Check : This line is used to indicate a serious error on the system. It can be reset only by the error handler.

-CHRESET : Channel Reset : This signal is used by the system board logic to reset or initialize all adapters upon power on.

3.3 Basic Transfer Cycle :

This Section provides a general description of the basic transfer cycle of the Micro Channel.

The address bus, MADE 24, M/-IO, and -REFRESH become valid, beginning the cycle. The status signals and -ADL become active. In response to an unlatched address decode, MADE 24, and M/-IO, the adapter returns -CD SFDBK, -CD DS 16/32. The adapter also drives CHRDY inactive if the cycle is to be extended. Write data appears on the bus. -CMD becomes active, -ADL and status signals become inactive. The address lines become invalid in preparation for the next cycle. In response to an address change signals -CD SFDBK, -CD DS 16/32 are set inactive by the adapter. The system holds this state indefinitely until CD CHRDY is set active. The adapter places the read data on the bus. -CMD goes inactive, ending the cycle. Figure 3.2 shows the basic read/write cycle of the Micro Channel.

The Micro Channel allows multiple bus masters. The bus has signal lines that allow different devices to contend for the ownership of the bus. Devices wishing to drive the bus must submit to arbitration. Arbitration starts when any device activates the -PREEMPT line to request control the active master. The system board has a circuit, the Central Arbitration Control Point (CACP), which sets the ARB/-GNT line high to start a new arbitration cycle. Each device contending for the bus puts its 4-bit arbitration level on the ARB0 - ARB3 lines. Based on a certain priority, one device wins and drives the bus. The losing devices get of the bus and the CACP makes ARB/-GNT low. The devices can also request a burst cycle for huge data transfers of multiple words. The Micro Channel has a non-maskable interrupt (NMI), which has the highest priority and is used to handling error conditions on the bus.

Thus, the Micro Channel is an intelligent microcomputer bus that allows automatic adapter setup, multiple bus masters, different bus widths and a high data transfer rate.

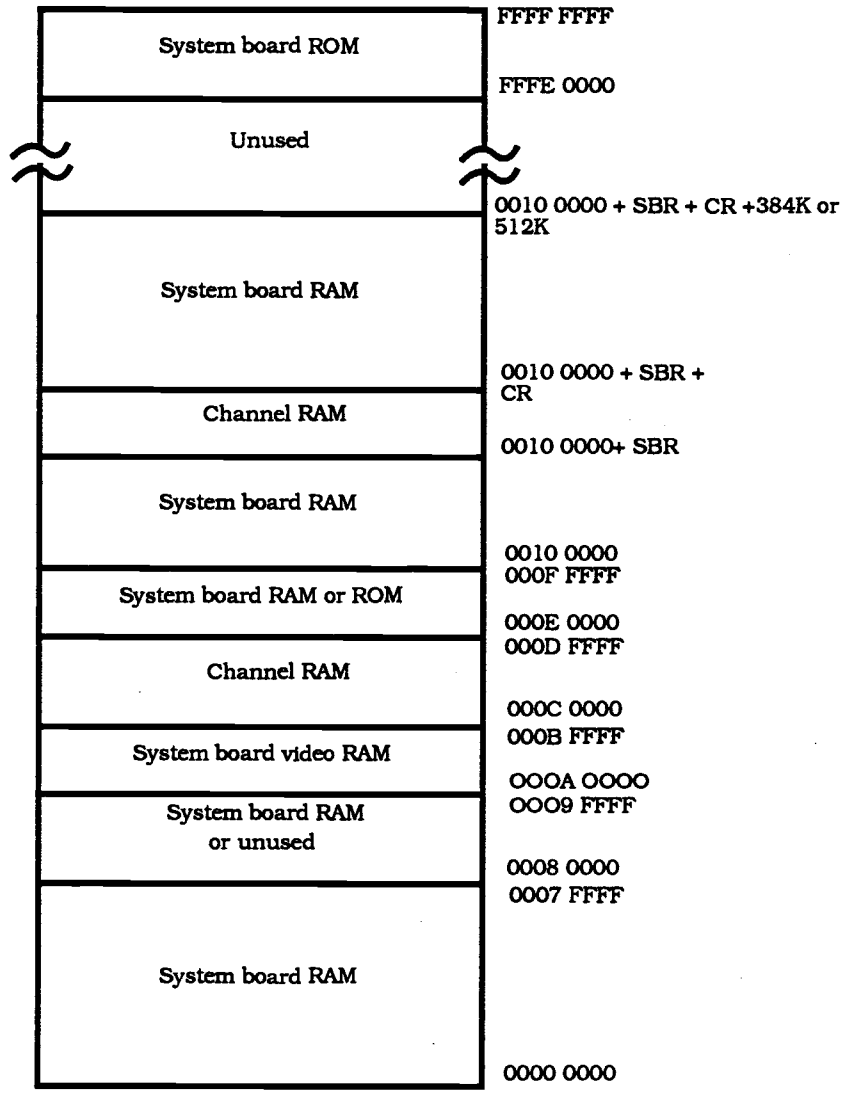


Figure 3.1 : Memory Map of the Micro Channel

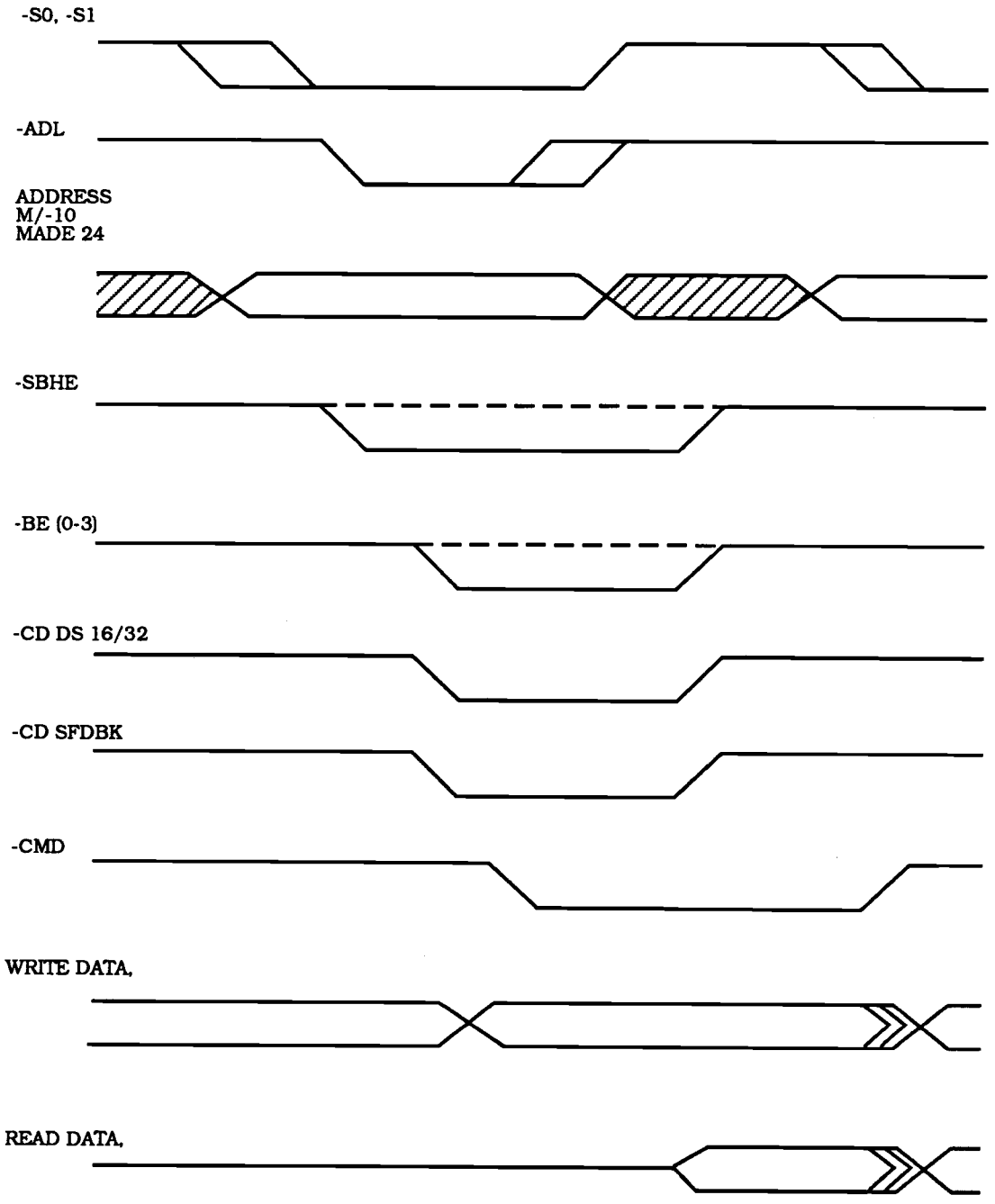


Figure 3.2 : Basic Read and Write Cycles of the Micro Channel

4. INTRODUCTION TO THE NUBUS ARCHITECTURE

4.1 General Description

The NuBus is a high performance (37.5 Megabytes per second) 32 bit Computer back plane bus which represents a distinctly minimalist approach. It is a system bus, independent of the Microprocessor Architecture. The NuBus architecture supports the multiprocessor system. The NuBus architecture was originally designed at MIT and developed further by Texas Instrument. The NuBus specification is accepted and published by IEEE as P1196 specifications in December 1986. Material for this section is mostly derived from [13].

The NuBus is synchronous (10MHz), multiplexed, multimaster bus which provides a strictly fair arbitration mechanism. NuBus signals can be grouped in to 4 classes based on the functions which they perform : utility signals, data transfer signals, arbitration signals and power.

The salient features of the NuBus architecture are :

- High addressing capability. With 32-bit addressing the memory map of the NuBus architectures extends to 4 GBytes.
- Optimized for 32 bit transfers, but supports 8-bit and 16-bit unjustified transfers.
- High data transfer speed with a clock of 10 MHz. Block transfers are available.
- System architecture independence.
- Simple protocol. I/O and interrupts are memory mapped and Reads and Writes are the only operations used.
- Small pin count. Address and data lines are multiplexed. Only 51 signals lines plus power and ground lines.

- Geographical addressing (ID lines) enables interface system to be free of DIP switches and jumpers.
- Distributed parallel arbitration ensures the fairness among the contending masters.

The NuBus has 51 signal lines, 45 power and ground lines in a single 96 pin DIN connector for all bus signals and power. All logic signals are TTL compatible. A brief description of the signal lines is given below.

4.2 Signal Descriptions :

4.2.1 Utility Signals :

Clock (CLK*), Reset (RESET*), power fail warning (PFW*), Non Master Request (NMRQ*), and the card slot identification (ID <3..0>*) are the utility signals provided in the NuBus protocol.

Clock (CLK*) : Synchronizes bus arbitration and data transfers. CLK* has a duty cycle of 25%. Usually bus signals are changed on the rising edge and sampled in the falling edge.

Card Slot Identification (ID <3...0>*) : These signals are binary encoded to specify the physical location of each card.

Non Master Request (NMR*) : An asynchronous signal asserted by slave boards to indicate a need for some service.

4.2.2 Bus Data Transaction Signals :

These signals are all three state lines, and include address/data, control and parity lines.

Address and Data (AD <31...0>*) : These lines are multiplexed to carry a 32 bit byte address at the beginning of each transaction and upto 32 bits of data later in the transaction. Address is valid when the start signal is active, hence should be latched with the falling edge of the clock* when the START* signal is active. Figure 4.1 shows the memory map of the NuBus. Data is valid when the slave card asserts ACK* signal. hence should be latched by the falling edge of the CLOCK* when a ACK* signal is active.

Transfer Mode (TM <1..0>*) : At the beginning of a transaction , these two lines indicate the type of the transaction being initiated. Later in the transaction, the responding module uses them success or failure. of the requested transaction. The transaction mode should be sampled with the falling edge of the clock when START* signal is active. The slave drives these signals during Acknowledgement cycle where ACK* signal is active. Hence the status code is latched with the falling edge of the clock where ACK* is active.

Start signal (START*) : This signal is asserted at the start of a transaction, and also initiates an arbitration contest. Additionally, when asserted in conjunction with the ACK* line, it denotes special non-transaction cycles called attention cycles.

Transfer Acknowledge (ACK*) : The usual use of this signal is to indicate the ending cycle of a transaction. It has a special use if asserted during the same cycle with START*.

4.2.3 Arbitration System Signals :

The signals in this group are all open - collector lines, and are used by the distributed arbitration logic to determine the next owner of the bus.

Bus Request (RQST*) : This line is asserted by modules to indicate their desire to own the bus. The fair arbitration scheme guaranties that all modules requesting the bus will obtain ownership within some determinable maximum time.

Arbitration signals (ARB<3..0>*) : These 4 lines are bussed and binary encoded in the same moment as the ID<3..0>* lines. During an arbitration contest, contending modules compare these lines with the binary value of their own ID <3..0>* lines, and drive ARB<3..0>* lines according to the rules of the distributed arbitration logic. The net effect of an arbitration contest is that, two cycles after starting a contest, the ARB<3..0>* lines carry the binary encoded number of the next bus owner.

4.2.4 Power lines :

Four voltages are defined for use by NuBus modules, +5V, +12V, -12V and -5.2Volts.

4.3 Single Data Cycle Transactions :

A single data cycle transaction is a read transaction or a write transaction in which only a single data value is transferred.

The master asserts START* and drives the AD*(31:0) lines with the desired address and drives the TM*(1:0) lines with the appropriate transfer mode to initiate the read or write transaction. The Master ensures that ACK* is unasserted. The adapter samples the AD*(31:0) and TM*(1:0) lines. The master stops driving the AD*(31:0), TM*(1:0), and ACK* lines in case of read cycle. It drives the data to be written onto the appropriate AD*(31:0) lines in case of write cycle. It also drives START* unasserted and waits for the acknowledge cycle. The addressed slave drives the requested data onto the AD*(31:0) lines in case of read cycle, samples the AD*(31:0)

lines to receive the data in case of write cycle. It also drives appropriate transaction code on the TM*(1:0) lines and asserts ACK*. The bus master and slave stop driving AD*(31:0), TM*(1:0), and ACK* lines, thus ending the cycle. Figures 4.2 and 4.3 show the read and write cycles of the NuBus.

The block transfer is a read or write transaction in which multiple data values are transferred. Figures 4.4 and 4.5 show the block transfer cycles of the NuBus.

Thus, the NuBus is a high performance, multiple master system bus that allows different bus widths and physical slot addressing with automatic configuration.

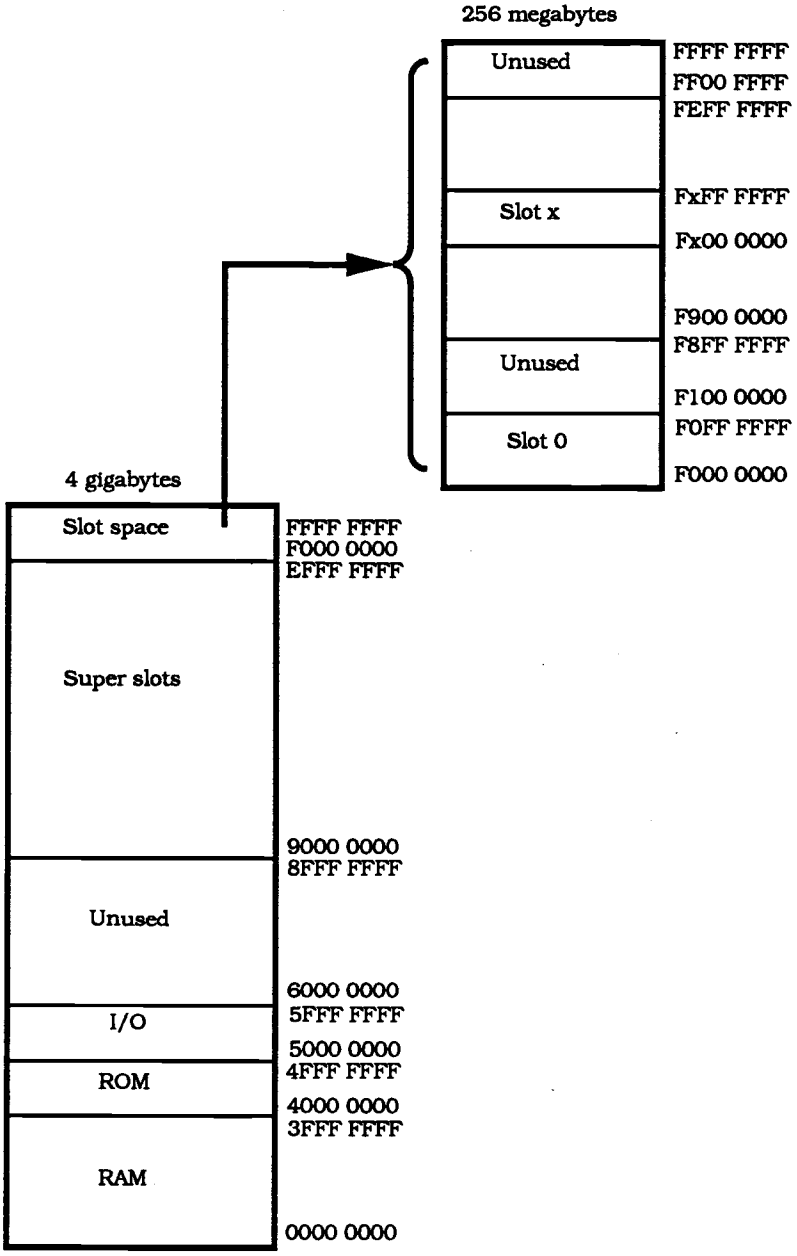


Figure 4.1 : Memory Map of the NuBus

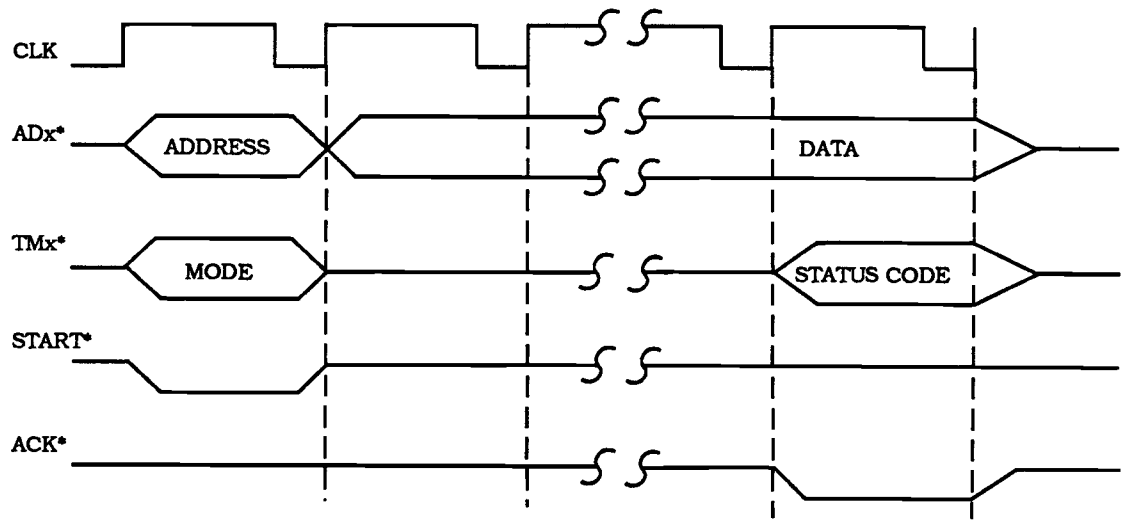


Figure 4.2 : Write Cycle of the NuBus

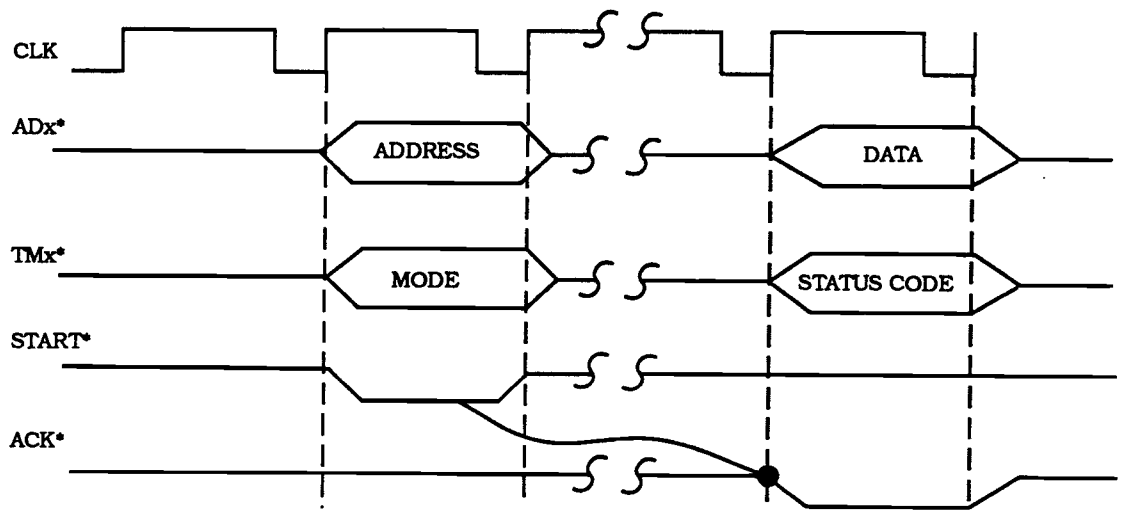


Figure 4.3 : Read Cycle of the NuBus

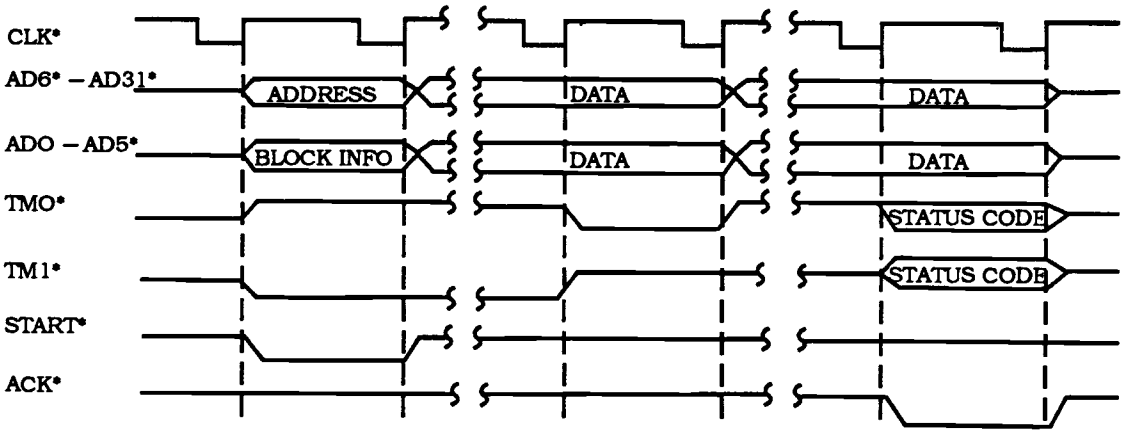


Figure 4.4 : Block Write Cycle of the NuBus

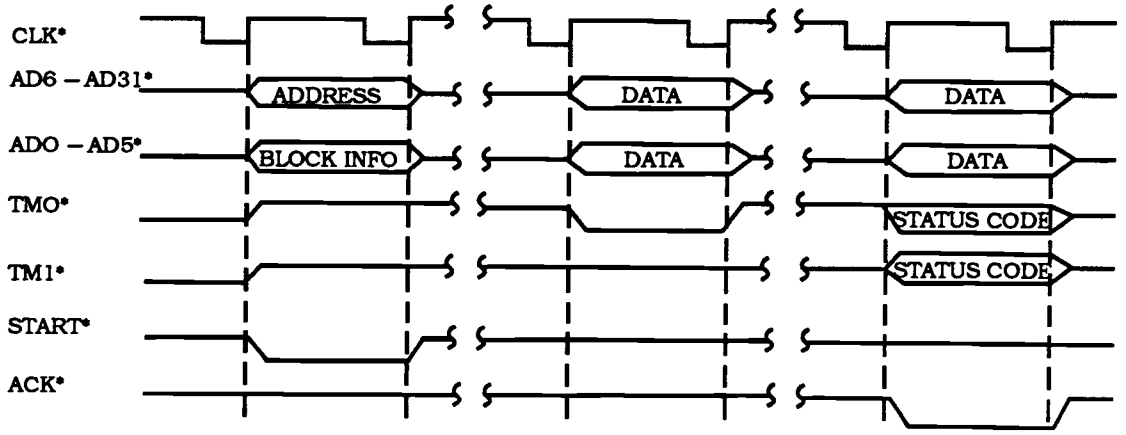


Figure 4.5 : Block Read Cycle of the NuBus

5. THE DESIGN OF MICRO CHANNEL INTERFACE FOR THE NU BUS ARCHITECTURE

5.1 Introduction

With the introduction of the IBM PC in 1981, IBM set standards in the area of personal computers. The original IBM PC has been steadily upgraded to form a whole family of upward compatible personal computers. These computers use expanded local buses like PC bus or AT bus for communication with the I/O devices. These buses are processor specific and are designed around the Intel 80x8x microprocessor architecture. They have problems accommodating large memory expansions. The PC bus is limited to one Megabyte of memory and the AT bus to 16 Megabytes.

Recent developments in the microprocessor industry have resulted in the evolution of the powerful 32 bit microprocessors. To take advantage of the power of these 32 bit microprocessors, to maximize hardware subsystem to subsystem transfers and to adapt to expanding processing rates, IBM developed a new I/O bus architecture termed the Micro Channel. It also developed a new family of personal computers, the Personal System 2 (PS/2) around the Micro Channel. IBM, with its open architecture policy, encourages other vendors to develop I/O subsystems around the Micro Channel. The functional characteristics, timing diagrams, electrical and mechanical specifications of the Micro Channel have been released in elaborate detail. A survey shows many I/O subsystems have already been devised for Micro Channel by IBM and other board manufacturers.

Meanwhile Apple Computer, INC. developed Macintosh II around IEEE 1196, NuBus - a simple 32 bit system bus. Since then, Macintosh II has captured a considerable share in the personal computer market. NuBus was developed by MIT and Texas

Instruments. This system bus is also designed to support the capabilities of present-day faster microprocessors, large memory and multiprocessing. A simple observation clearly shows the fact that Macintosh II is most commonly used for desktop publishing, rather than in control and engineering applications. The main reason for this is, not many I/O subsystems have been designed around NuBus.

5.2 The Differences between the Micro Channel and the NuBus :

There are several differences between the communication protocols of the Micro Channel and the NuBus. Micro Channel is both a local bus and a system bus. As a local bus, it optimizes the host CPU (Intel 80286 or 80386)-to-memory bandwidth, using special control signals. It is also a system bus, in that it is treated as a system resource. NuBus is a full system bus. It is independent of any processor architecture. NuBus provides mechanisms for true multiprocessing : Bus and Resource locks. Bus locking allows a processor to lock a bus for exclusive access. With resource locking, a shared resource, such as RAM on a card with its own local processor, is locked so that the local processor can't access it. Micro channel does not have direct provisions for bus or resource locking.

Micro Channel has an asynchronous communication protocol, where as the NuBus uses a synchronous protocol. The Micro Channel has a number of signals for coordinating asynchronous handshakes : The signals -ADL, -CMD, CD CHRDY provide the basic bus handshake edges. In contrast, the NuBus operations are relatively simple, requiring no special signals. NuBus transactions are referenced to a single, 10MHz clock signal. NuBus timing is more stringent than the Micro Channel's. *The signals and data should be sent and strobed* within 75 ns in the 100-ns clock cycle.

Address and data on the NuBus are multiplexed on 32 signal lines. The Micro Channel provides 64 individual lines for a non-

multiplexed address and data bus. The Micro Channel Architecture provides for separate memory and I/O address spaces. It supports 4 GBytes of memory space and 64 KBytes of I/O space. The NuBus provides for a single address space of 4 GBytes. Data can be transferred on the NuBus as 8-bit bytes, 16-bit halfwords, or 32-bit words. The Micro Channel can handle 8-bit, 16-bit, 24-bit, and 32-bit data. On the NuBus, cards are given 1/16th of the total 4 GBytes address space (256M bytes). NuBus allows up to 16 cards in a system and each card is given 1/16th of the total 256M card address space (16 MBytes per card). The card automatically assumes its proper address by sampling 4 signals on the NuBus connector which provide a card identification code.

The Micro Channel defines a number of discrete interrupts that can be shared among the boards. The NuBus, on the other hand, defines an interrupt per slot that is fed separately into the main logic for processing. Both buses use decentralized scheme for arbitration. The NuBus has fixed arbitration priority determined by the card's slot ID, with 0 being the lowest priority and Fh being the highest. For the Micro Channel, the arbitration level is stored on the card when it is configured into the system. The highest priority a card can have is level 0, and the lowest is Fh. The Micro Channel also has a Central Arbitration Control Point, which is some logic on the main system board that controls the start and winner of an arbitration contest. Both buses ensure fairness by preventing a higher-priority-level card from continuously withholding ownership of the bus from lower-priority-level entities. Card logic prevents the card just serviced from requesting bus ownership until all the pending requests are honored.

Both the Micro Channel and the NuBus define high-level mechanisms to integrate cards or devices into the bus system. This eliminates the need for jumpers or switches to set either a card's interrupt level or its address space. The Micro Channel's Programmable Option Select (POS) eliminates switches from the

system board and cards by replacing them with programmable registers. Automatic configuration routines store the POS data from adapter description files into a battery-powered CMOS memory for system configuration. The system loads the appropriate configuration data from CMOS memory into the card POS registers. This data sets cards arbitration level, the address range of the card's I/O ROM, and the I/O address range. Cards that fail to configure properly are disabled by the system. In case of NuBus, each card is required to have a special declaration ROM that holds the card specific configuration information. Information in the declaration ROM includes byte lanes (which bytes of the NuBus data path are used), a test pattern, a revision level, a ROM cyclic redundancy check for validating the contents of the declaration ROM, and a resource directory. The resource directory points to the device drivers required for the operation of the card.

5.3 Strategies Used in the design of Micro Channel Interface for the NuBus :

The objective of this design is to devise a Micro Channel interface to the NuBus, so that the I/O subsystems designed for the Micro channel can be used on the NuBus with little change in the hardware and software. A model of the design is shown in Figure 5.1 This design would make it possible for the NuBus systems to take advantage of strong hardware support which is available for the Micro Channel. This interface gives a Micro Channel slave interface to the NuBus.

As the two buses are dissimilar, completely transparent interface design is complex and expensive. Hence a nontransparent interface is designed.

As the NuBus is synchronous and the Micro Channel is asynchronous the NuBus clock is used for synchronization. All the

signals generated by the interface are synchronous with the NuBus clock. The Micro Channel input signals are sampled with respect to the NuBus clock using the master-slave flipflops as shown in Figure 5.2

The problem of address mapping is taken care by restricting the user to map the Micro Channel I/O cards in to the memory map of the NuBus card slot in which the card is to be mounted. The cards should decode all the 32 address lines. The device drivers of the cards must be modified accordingly. This eliminates the need for an address mapping mechanism for memory and I/O space. Figure 5.2 shows the address interface provided between these two buses.

The Micro Channel has non-multiplexed address/data buses and the NuBus has a multiplexed and inverted address/data bus. The address/data interface provides proper multiplexing function as shown in Figure 5.3. The NuBus address/data information is demultiplexed and inverted before putting on the Micro Channel address and data bus and viceversa.The control signals for this function are generated by the interface controller.

The data formats are similar on both the buses. Hence the data lines of the two buses can be connected directly with proper tri-stating.

A Micro Channel slave interface to the NuBus does not require the arbitration signals to be interfaced on both the buses. All the other signals on both the buses, which are necessary for slave read, write, and interrupt protocol are interfaced. The Micro Channel handshake signals and status signals are properly generated depending on the NuBus Start signal and the transfer mode signals. At the end of a transaction the NuBus Acknowledge and transfer mode signals are driven as specified in the NuBus specifications.

5.4 Goals for the Design :

A survey shows most of the cards designed for the Micro Channel are designed as the slaves on the bus, which use other masters to communicate on the bus. Hence this interface gives a Micro Channel slave interface to the NuBus. The arbitration signals need not be interfaced to design a slave interface.

As the two buses are dissimilar, completely transparent interface design is complex and expensive. Hence a nontransparent interface is designed.

The user must memory map the Micro Channel cards in the memory map of the Micro Channel such that the cards respond to the addresses which match with one of the available slot spaces of the NuBus. This approach is adapted to avoid complex hardware and software requirements of the address mapping.

The NuBus clock is used as the clock of the interface. This is a 100 ns clock. Hence the interface state machine goes through the bus conversion fairly slow. By using a faster clock for the interface the transfer rate between these two buses can be increased.

The NuBus does not have a programmable card configuration, instead it has configuration ROM. The ROM is provided in the interface and also the decoded control signals to address this ROM. This ROM should be programmed with proper firmware [17]. The POS registers of the Micro Channel should be implemented as ROM on the card, except the -CHCK, the most significant bit of the POS register 105. This bit is to be deasserted by the I/O device itself with the rising edge of the -CMD signal of the Micro Channel.

The Micro Channel does not support the NuBus block transfers. Special Hardware is designed in the interface to take care of this.

Thus a facility which Micro Channel does not support is taken advantage of. Some changes are required to be made in the Card driver software.

The Address/Data interface and the Interface controller are designed to implement this interface efficiently. Considering the speeds of the two buses, it is necessary to implement the interface as VLSI components. This reduces the real estate and power consumption of the interface. Discussion of the VLSI aspects of the design is not the objective of this thesis. The following chapters describe the design in detail.

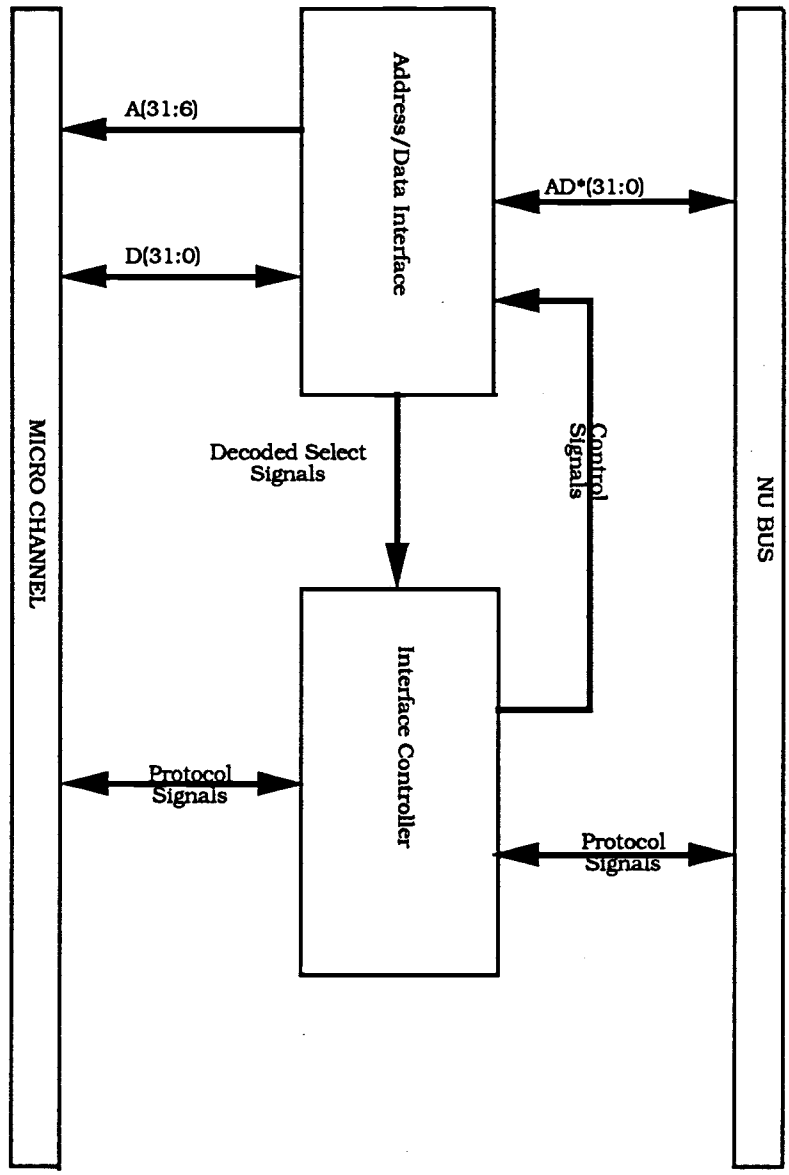


Figure 5.1 : Model of a Interface between the Micro Channel and the NuBus

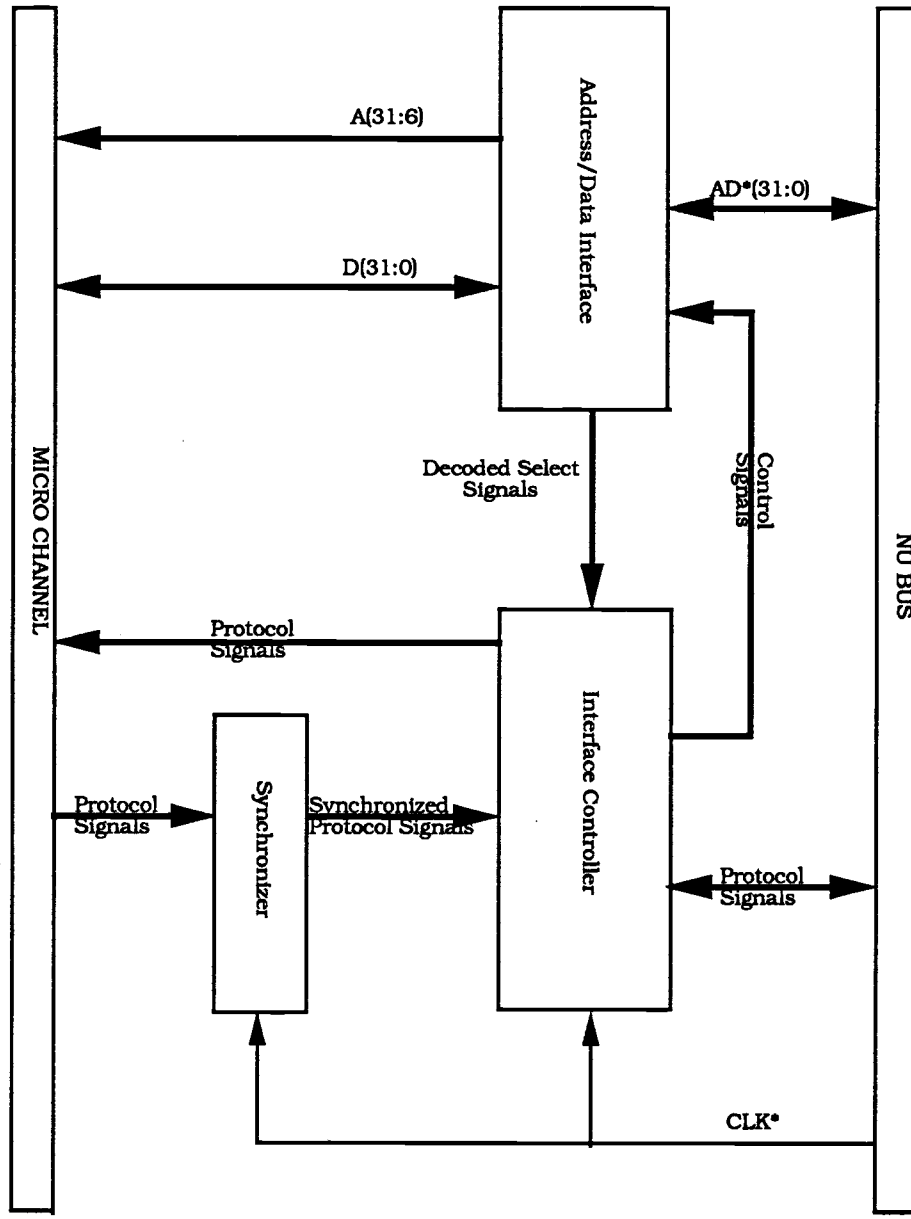


Figure 5.2 : Synchronization and Address Mapping for the Interface

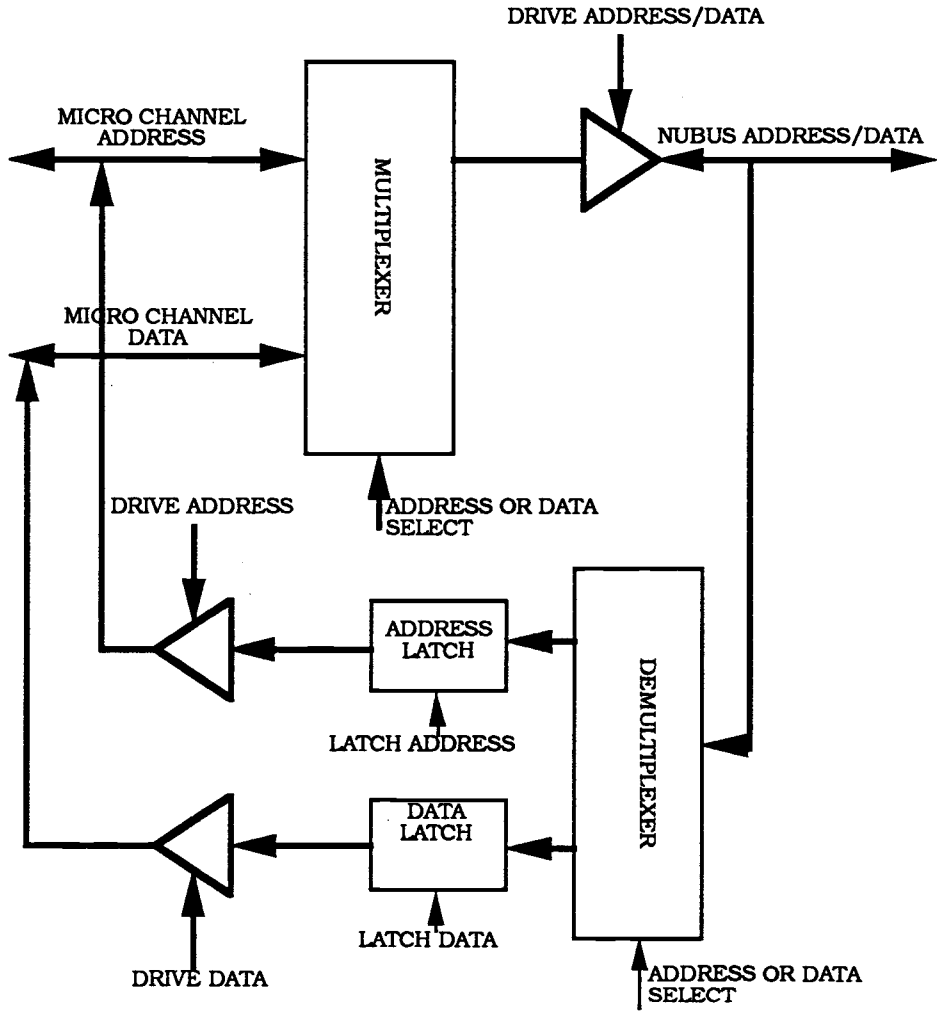


Figure 5.3 : Address/Data Interface for the Micro Channel and the NuBus

6. ADDRESS / DATA INTERFACE

6.1 General Description

The Address / Data Interface is designed to work with the Micro Channel to NuBus interface controller designed for this thesis. It consists of bus transceiver circuits, D-Type flipflop latches and control circuitry arranged for multiplexed transmission of address and data on NuBus. A comparator has been added to detect when a NuBus transfer cycle is requesting the local board. A decoder decodes the address for the configuration ROM of the NuBus. The block diagram of Address/Data interface is shown in Figure 6.1.

The -AEN, -DEN, and -ADEN inputs control the transceiver functions. These signals are generated by the interface controller. Three 32-bit i/o ports, A31-A0, D31-D0, and -AD31- -AD0 provide for address and data transfer. Address lines -A5- -A0 are given as output to feed as input to the Block Transfer logic of the Interface Controller.

When a NuBus performs write cycle on a local board, address information is saved on the rising edge of ACLK. During the last portion of the NuBus write cycle, data information is saved on the rising edge of the DCLK. The truth table for A(31:0) and D(31:0) is shown in Figure 6.2. The interface controller generates the ACLK and DCLK signals as required.

When a NuBus performs a read cycle, the address information is saved on the rising edge of the ACLK. During the last portion of the read cycle, data information is put on the NuBus by pulling -A/D high and -ADEN low. In fact -A/D is tied high to always enable data port D31-D0 to be selected as input to the -AD31- -AD0, as the slave local board does not drive the A31-A0 lines. The truth table for AD*(31:0) is shown in Figure 6.3.

The -IDEQ output is used to signal that the local board is being requested by the NuBus. This output is fed into Micro Channel to NuBus interface controller. -IDEQ goes active (low) when -AD31- -AD28 are low and -AD27- -AD24 match -ID3- -ID0. -IDEQ stays valid until the next ACLK occurs. The truth table for -IDEQ is shown in Figure 6.4. The -ROMCS is the decoded Configuration ROM select output.

6.2 Signal Descriptions :

The input and output signals to this Address/Data interface are shown in Figure 6.5.

A31-A0, Address bus : This 32-bit i/o port is connected to the Micro Channel address bus. When information is transferred between this Micro Channel port and the NuBus port (-AD31- -AD0), the data is inverted to conform to the Micro Channel and the NuBus specifications.

-A5- -A0, Latched Address Lines : These latched Address lines are provided to feed into the Interface Controller for Block transfer control of the NuBus.

ACLK, Address Clock : This input saves the address portion of NuBus read or write cycles. Address present at the -AD31- -AD0 inputs is clocked into the address register on the low to high transition of ACLK.

-A/D, -Address/Data Select : This input controls the address/data multiplexer. When -A/D is driven low, the Micro Channel port, A31-A0 is selected as input to the -AD31- -AD0 outputs. When -A/D is taken high, the Micro Channel data port A31-A0 is selected as input to the -AD31- -AD0 outputs.

-AD31- -AD0, Address/Data port : This 32-bit active low i/o port directly interfaces to the NuBus address/data lines. These lines are multiplexed to carry address information at the beginning of a NuBus cycle and data information later in the cycle.

-ADEN. Address/Data Output Enable : This active-low input enables the -AD31- -AD0 outputs. When -ADEN is taken high, the -AD31- -AD0 outputs are in the high-impedance state, allowing input from the NuBus.

-AEN, Address Enable : This active-low input enables the local address outputs, A31-A0, to place data onto the local board. When -AEN is taken high, the A31-A0 outputs are in the high-impedance state, allowing input from the local address bus.

D31-D0, Data Bus : This 32-bit i/o port is connected to the local board's data bus. When information is transferred between this port and the NuBus port (-AD31- -AD0), the data is inverted to conform to NuBus specifications.

DCLK, Data Clock : This input saves the data portion of NuBus write cycles. Data present at the -AD31- -AD0 inputs is clocked into the data register on the low to high transition of DCLK.

-DEN, Data Enable : This active-low input enables the local data port outputs, D31-D0, to place data onto the local board. When -DEN is taken high, the D31-D0 outputs are in the high-impedance state, allowing input from the local board.

-ID3- -ID0, Card Slot Identification : These four inputs accept binary-coded location information for each NuBus slot position on the backplane. These four lines are typically hard-wired logic levels unique to each NuBus slot connector. For convenient

implementation, the inputs have internal 10-kΩ pull-up resistors that ensure the logic high level when the inputs are left open circuited. The internal comparator uses these inputs to identify when the local hardware card is being accessed.

-IDEQ, Identification Equal : This active-low output is used to signal that the local board is being accessed by the NuBus. -IDEQ goes low whenever -AD31- -AD28 are low and -AD27- -AD24 match -ID3- -ID0. Since the internal comparator uses data from the address register, the address register must be clocked before the local board samples -IDEQ. -IDEQ is valid for the entire NuBus cycle after ACLK.

-ROMCS, ROM Select : This active-low output is the decoded chipselect for the configuration ROM which resides at FsFF FFFF to FsFF FE00 address. -ROMCS becomes active whenever the NuBus system intends to read the configuration ROM. It is valid for complete NuBus read cycle after ACLK.

When the NuBus master starts a transaction cycle the address/data interface decodes the address and asserts -IDEQ if it is addressed. If the address is for configuration ROM it asserts the -ROMCS. In response to these signals the Interface Controller asserts the ACLK signal which latches the address. In the next cycle if it is a write transaction DCLK is asserted to latch the write data. In the same cycle the address is driven on the address bus of the Micro Channel by driving the signal -AEN. The Micro Channel data bus is driven by the latched data if it is a write cycle by driving the signal -DEN. -AEN is driven for two clock cycles as required by the Micro Channel protocol. -DEN is driven till the end of the cycle. When the Micro Channel slave reads the data on the data bus -DEN signal is deasserted. In case of a read cycle the signal -ADEN is driven during the NuBus ACK cycle, when the Micro Channel slave puts the data on the Micro Channel data bus. -A/D is always driven high so that Micro Channel data bus is always connected to the AD*(31:0) lines of the

NuBus. The Interface Controller has to generate these control signals in proper sequence for the address/data interface to function properly.

The logic schematic of the Address/Data interface is shown in Figure 6.6. This Address/Data interface is a generic address/data interface which can be used for any address/data interface with little changes.

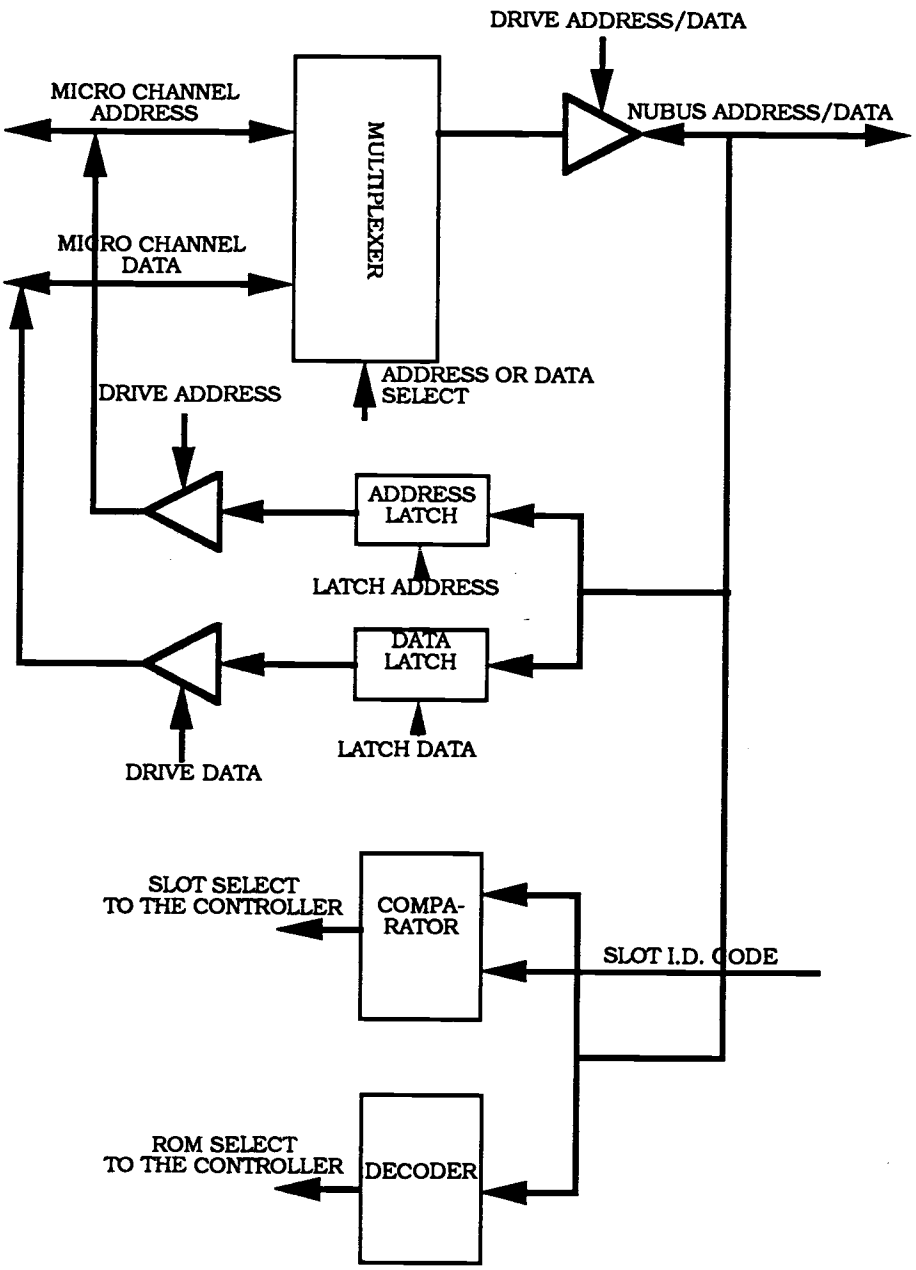


Figure 6.1 : Block Diagram of Address/Data Interface

A(31:0)	D(31:0)	-A/D	-ADEN	AD*(31:0)
H	X	L	L	L
L	X	L	L	H
X	H	H	L	L
X	L	H	L	H
X	X	X	H	Z

Figure 6.2 : Truth Table for AD*(31:0)

AD*(31:0)	ACLK, DCLK	-AEN, -DEN	A(31:0), D(31:0)
H	↑	L	L
L	↑	L	H
X	L	L	Qo
X	X	H	Z

Figure 6.3 : Truth Table for A(31:0) and D(31:0)

AD*(31:28)	AD*(27:24)	-IDEQ
Eo O	EQ ID*(3:0)	L
X	NE ID*(3:0)	H
NE O	X	H

Figure 6.4 : Truth Table for -IDEQ

Qo = NO CHANGE
z = HIGH IMPEDENCE

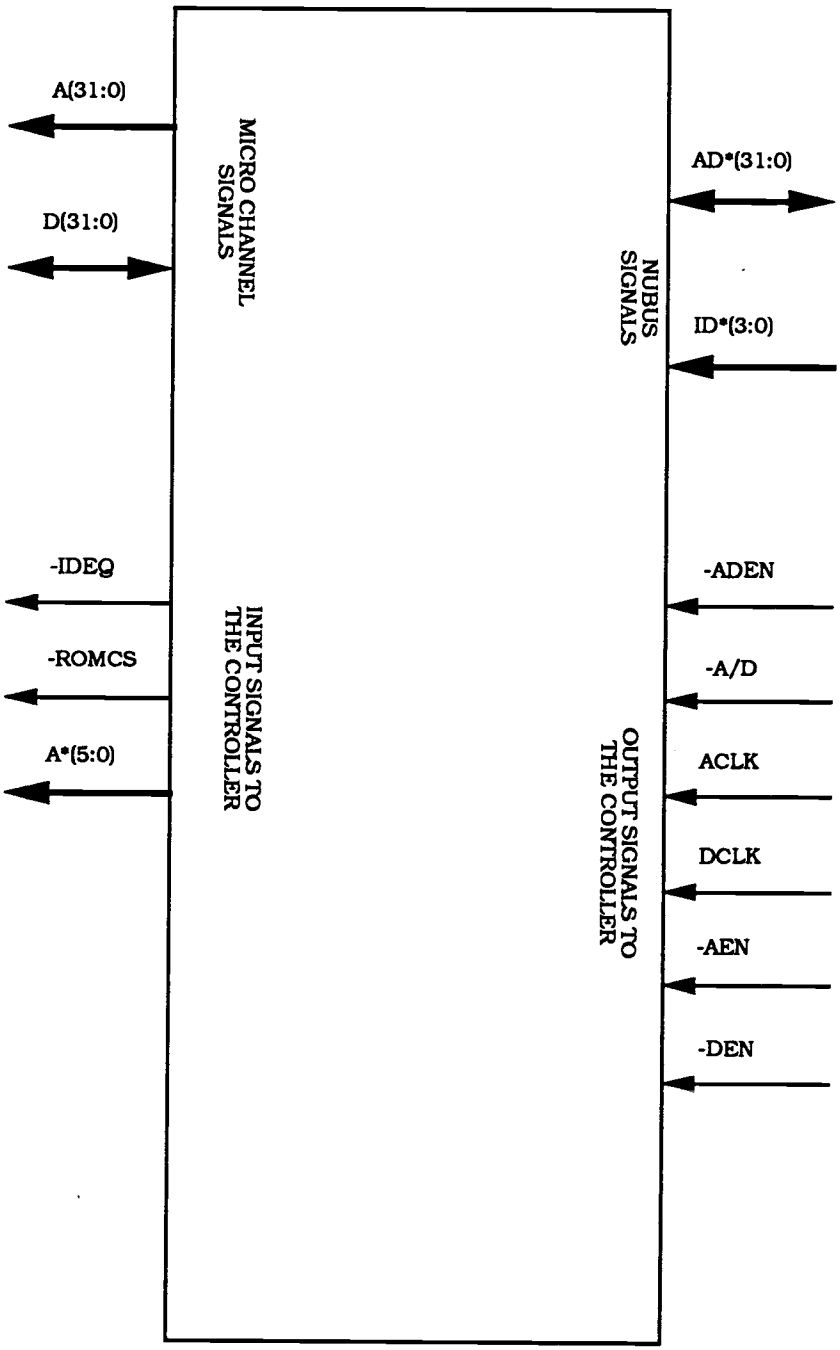


Figure 6.5 : Input/Output Signals of the Address/Data Interface

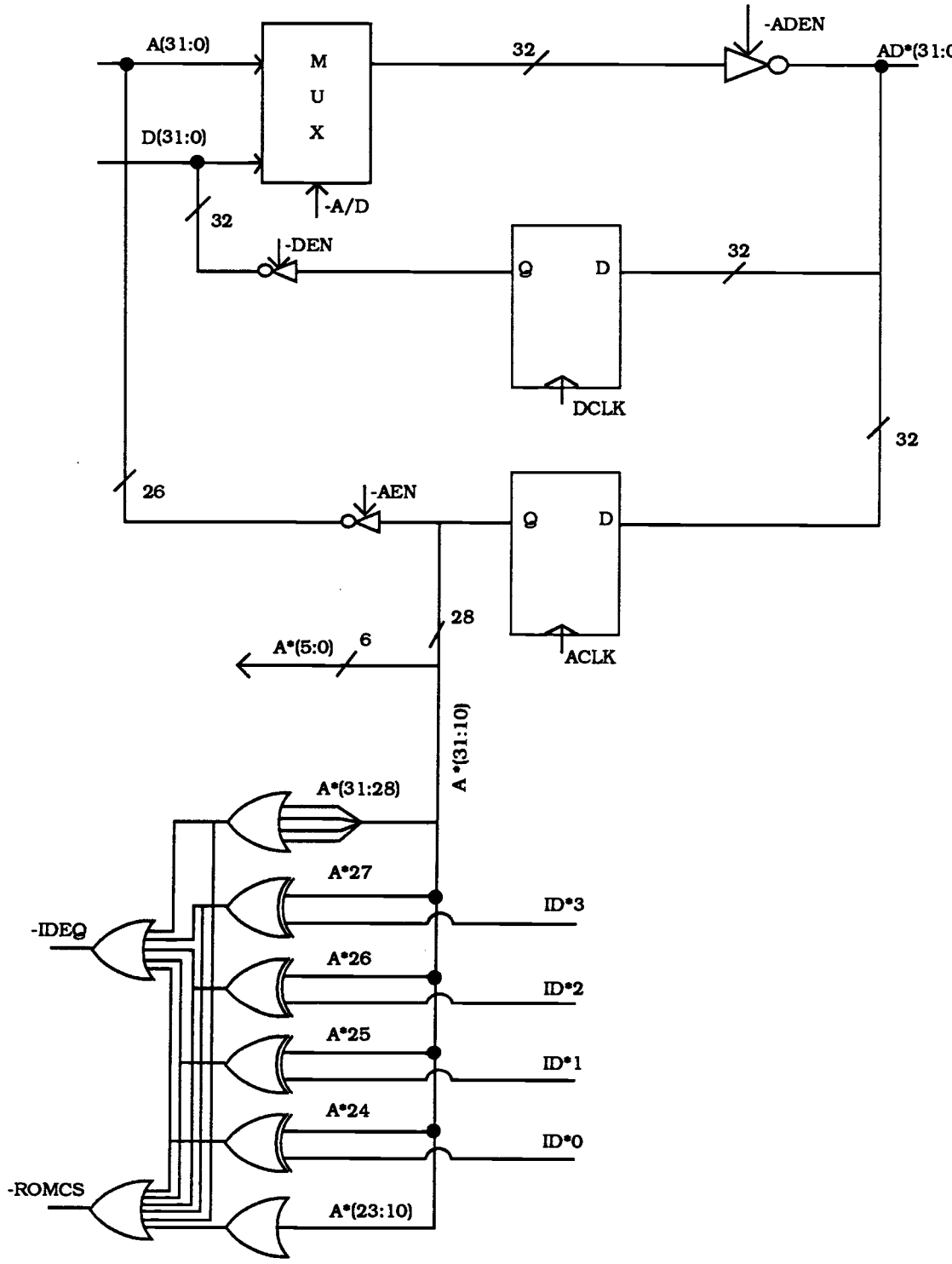


Figure 6.6 : Logic Diagram of the Address/Data Interface

7. MICRO CHANNEL TO NU BUS INTERFACE CONTROLLER

7.1 Introduction

This interface controller is designed to offer a slave level Micro Channel interface to the NuBus. The controller handles the NuBus signalling protocol in compliance with ANSI/IEEE std. 1196 - 1987 and the Micro Channel signalling in compliance with the Technical Reference Manual for Micro channel Architecture published by IBM. This controller does not support Micro Channel masters to be interfaced as masters on the NuBus. It supports NuBus block transfers.

Data and address multiplexing, demultiplexing and buffering is handled by the Address/Data Interface. The controller generates necessary control signals for the proper operation of Address/Data interface.

The Interface controller has a similar structure as explained in Section 5.3 of Chapter 5. It interfaces the control signals of the Micro Channel and the NuBus as defined in the protocols of these two buses. It interfaces the interrupts of the two buses. It generates the control signals required for the operation of the address/data interface. Thus the controller designed here has a similar structure as the general controller discussed earlier.

The controller recognizes the NuBus Start cycle and checks for ROM read, Block transfer, read or write transactions and drives the Micro Channel signals and the Address/Data control signals in correct sequence and with proper timing. At the end of the transaction it also drives the NuBus Transfer Mode signals and ACK* signal. A general flowchart of the controller sequential machine is shown in Figure 7.1. Figure 7.2 shows the detailed state diagram of the state machine designed for this Interface Controller. Figure 7.3

shows the state assignment table for the state machine. The state table of the controller is shown in Figure 7.4. The state machine which drives the different output signals of the Controller constitutes the main block of the controller. The Random logic block drives the Micro Channel outputs like -BE (3:0) and also provides some decoded inputs to the State machine. The ROM serves the purpose of the Configuration ROM of the NuBus. The Controller block diagram is shown in Figure 7.5.

The Micro Channel to NuBus Interface controller is comprised of three major signal groups. Micro Channel signals, NuBus signals, Address/ Data Interface - control signals. The Micro Channel signals directly interface with the card designed for Micro Channel. The NuBus signals provide interface with the NuBus. Address/Data interface control provides the buffering signals required to multiplex and demultiplex the NuBus address/data lines.

7.2 Signal Descriptions :

As previously explained, the input and output signals of the Micro Channel to NuBus Interface Controller can be functionally organized into three groups. The following paragraphs briefly describe the controller signals in each group. The input and output signals of the controller are shown in Figure 7.6.

7.2.1 Micro Channel signals :

-ADL : -Address Decode Latch : This is the Micro Channel signal which provides the convenient mechanism for the slave to latch valid address and status bits. This is driven active when the controller recognizes a NuBus START cycle and -IDEQ is active.

-SHBE : -System Byte High Enable : This signal indicates and enables transfer of data on the high byte of the data bus (D8 - D15), and is

used with A0 to distinguish between high byte (D8 - D15) and low byte (D0 - D7) transfers. This is decoded from the NuBus signals -TM0, -TM1, A0, A1.

MADE 24 : Memory Address Enable 24 : This line indicates when an extended address is used on the bus. MADE 24 inactive indicates an extended address space greater than 16MBytes is being presented. As in case of NuBus always a 32-bit address is presented, this signal is forced inactive.

M/-IO : Memory/-Input Output : This signal distinguishes a memory cycle from a I/O cycle. This signal is driven high during the read and write cycles as NuBus does not have a separate I/O address space.

-S0, -S1 : -status bits 0 and 1 : These signals indicate the start of a channel cycle and also define the type of the cycle. These signals are derived from the NuBus status signals -TM0 and -TM1.

-CMD : -Command : This signal is used to define when the data is valid on the data bus. -CMD trailing edge indicates the end of the bus cycle. It provides the slave the duration during which the data is valid on the bus. It is driven active when the slave drives its CD CHRDY signal high.

-CD SFDBK (n) : -Card Selected Feedback : This is the Micro Channel input to the controller which is driven by the addressed slave as a positive acknowledgement of its presence at the address specified.

CD CHRDY (n) : Channel Ready : This line, normally active (ready), is pulled inactive (not ready) by a slave to allow additional time to complete a channel operation. If this signal is held inactive for 3.5 microseconds a try later code is put on status lines of the NuBus in the ACK cycle.

-IRQ 3 - 7, -IRQ 9 - 12, and -IRQ 14 -15 : -Interrupt Request : These lines are used to signal the controller that an I/O slave requires attention. These signals are inverted and ORed to form the -NMRQ signal of the NuBus.

--CHCK : -Channel Check : This line is used to indicate a serious error which threatens continued operation of the system. It is driven by the slave card.

OSC : Oscillator : This line is high speed clock with a frequency of 14.31818 MHz ± 0.01%. The high level (more than 2.3 volts) pulse width and the low level pulse width (less than 0.8 volts) must not be less than 20 nanoseconds each. It is derived from a crystal oscillator.

CHRESET : Channel Reset : This signal is generated by the -RESET signal of the NuBus to reset or initialize the slave card.

-BE0 — -BE3 : -Byte Enable 0 through 3 : These signals indicate the data transfers of 8-, 16-, 32-bits. They also indicate the NuBus block transfer request. These are derived from the -TM0, -TM1, A0, A1 signals of the NuBus. Figures 7.7 and 7.8 show the truth table, equations and the logic diagram of these signals.

-REFRESH : -Refresh : This signal is always driven high by the controller.

X1 and X2 : Clock Inputs : These pins are the inputs for the Crystal Oscillator. A 14.31818 MHz crystal should be connected to these pins. The Micro Channel clock is derived from these pins. The clock generator diagram is shown in Figure 7.9

7.2.2 NuBus Signals :

-ACK : Transfer acknowledge : This bidirectional I/O pin signals the end of a transaction. It also signals attention cycles. Hence before a start cycle is started -ACK signal is sampled in the controller. Controller also drives this signal after driving the -CMD signal of the Micro Channel.

CLK* : Clock : The NuBus clock signal is tied directly to the controller. The data transactions are synchronized to this signal.

NMRQ* : NonMaster Request : This asynchronous output is asserted by the controller when one of the interrupt request lines of the Micro Channel is asserted.

RESET* : Reset :This asynchronous input monitors NuBus RESET* line. When active, it initializes the Controller and the Micro Channel card.

START* : Start : This input pin is asserted at the start of a NuBus transaction. Accordingly the controller begins the Micro Channel transaction.

TM0* and TM1* : Transfer Mode Signals : At the beginning of a transaction, these two lines indicate the type of transaction being initiated. Accordingly the controller initiates the read or write transaction or the block transfer transaction on the Micro Channel. At the end of the transaction the controller drives these lines to indicate success or failure of the requested transaction. Figure 7.10 shows the driving circuit for these signals.

7.2.3 Address/Data Interface- Control Signals :

-ADEN : Output Enable : This active low output enables data or address information to be placed onto the NuBus address/data bus. -ADEN is asserted on the driving edge of the NuBus Clock signal (CLK*) when the local board is the selected NuBus slave during an acknowledge cycle and the current cycle is a read.

-A/D : Address/Data Select : This output controls the multiplexing function of the address and data information onto the NuBus. When low, address information is indicated. When high, data information is indicated. In this design as the interface is a slave interface -A/D is always driven high to connect the D31-D0 data port to the NuBus AD*31-AD*0 port.

ACLK : Address Clock : This output loads NuBus address information onto the Address/Data interface. During both read and write start cycles, this output changes on the sample edge of the NuBus Clock signal (CLK*).

DCLK : Data Clock : This output loads NuBus data onto the address data interface. This output changes on the sample edge of the NuBus Clock (CLK*) when the local board is a selected NuBus slave and the current cycle is the write cycle.

-AEN : Address Enable : This active low output signal enables address information to be placed onto the Micro Channel address bus, when selected as a NuBus slave, -AEN goes low first cycle after the START cycle.

-DEN : Data Enable : This active low output enables data to be placed onto the Micro channel data bus, when selected as NuBus slave and the current cycle is the write cycle. This output remains active till the -CMD output of the Micro Channel is active.

-A(5:0) : Latched Address Lines : These are the inputs from the Address/Data interface and are fed into the Block Transfer logic of the controller.

The Interface Controller is designed to provide a Micro Channel slave interface to the NuBus. The controller supports the NuBus read and write transactions and the block transfers. It provides all the necessary Micro Channel signals for different transactions. The heart of the controller is a state machine. The state machine always waits in a state where it expects a Start cycle. When the -START signal is asserted, it checks for the -IDEQ signal from the Address/Data interface. If the local card is the selected slave it proceeds through its normal flow depending on the type of the transaction as shown in Figure 7.2. The state machine can be implemented using a standard PLA and the memory elements for the state variables. The logic equations which define each signal are listed in appendix A. Figures 7.11 and 7.12 show the timing diagram of the interface controller. It can be seen that proper signals are driven on both the buses, the Micro channel and the NuBus. The timing diagram also shows that the controller takes minimum of 5 NuBus cycles to complete a transaction between the two buses. Hence the fastest data transfer rate between these two buses that can be acheived with this interface is 2 MBytes per second. The controller supports the block transfers of the NuBus. The logic necessary for this purpose is shown in Figure 7.13

The design forces some changes in the Micro Channel card hardware. The POS registers on the cards should be implemented on a ROM. The configuration is fixed. The NuBus does not Configure the card. The card should be mapped on to the memory space of the card slot in which it resides. The card enable bit in the POS register 102 should always be active. The -CHCK bit of POS register 105 should indicate error condition when a error occurs in the card, and the card should reset that bit with rising edge of the signal -CMD

after setting the -CHCK bit. The NuBus Configuration ROM is provided on the interface. It should be properly programmed with the correct firmware [17].

7.3 Conclusion :

This thesis discusses the design strategies for the design of a interface between the two system buses. All the problems to be dealt while designing a interface between two system buses are identified for different combinations of the two buses and the solutions for these problems are also discussed. These strategies can be used as the guide lines for any interface design. With the design strategies and a good example the thesis offers a systematic approach for the interface design between the two system buses. This would make the interface design problems less time consuming and efficient. This is the most significant contribution this thesis makes to the world of bus interface design.

It was a challenge to match the signals on two dessimilar buses. The asynchronous property of the Micro Channel posed the problem of synchronization, which was overcome by using master-slave flipflops for synchronization with respect to the NuBus Clock signal. The problem of interfacing multiplexed address/data on the NuBus and the non-multiplexed address/data on the Micro Channel is solved as explained in Chapter 2. The goals set for the design in chapter 5 are met.

The design can be enhanced to be a Micro Channel Master interface to the NuBus, in which case the two systems on the Micro Channel and the NuBus can communicate and share resources. A faster transfer rate between these two buses can be achieved by using a faster clock for the interface and synchronizing signals of both the buses with respect to this clock signal.

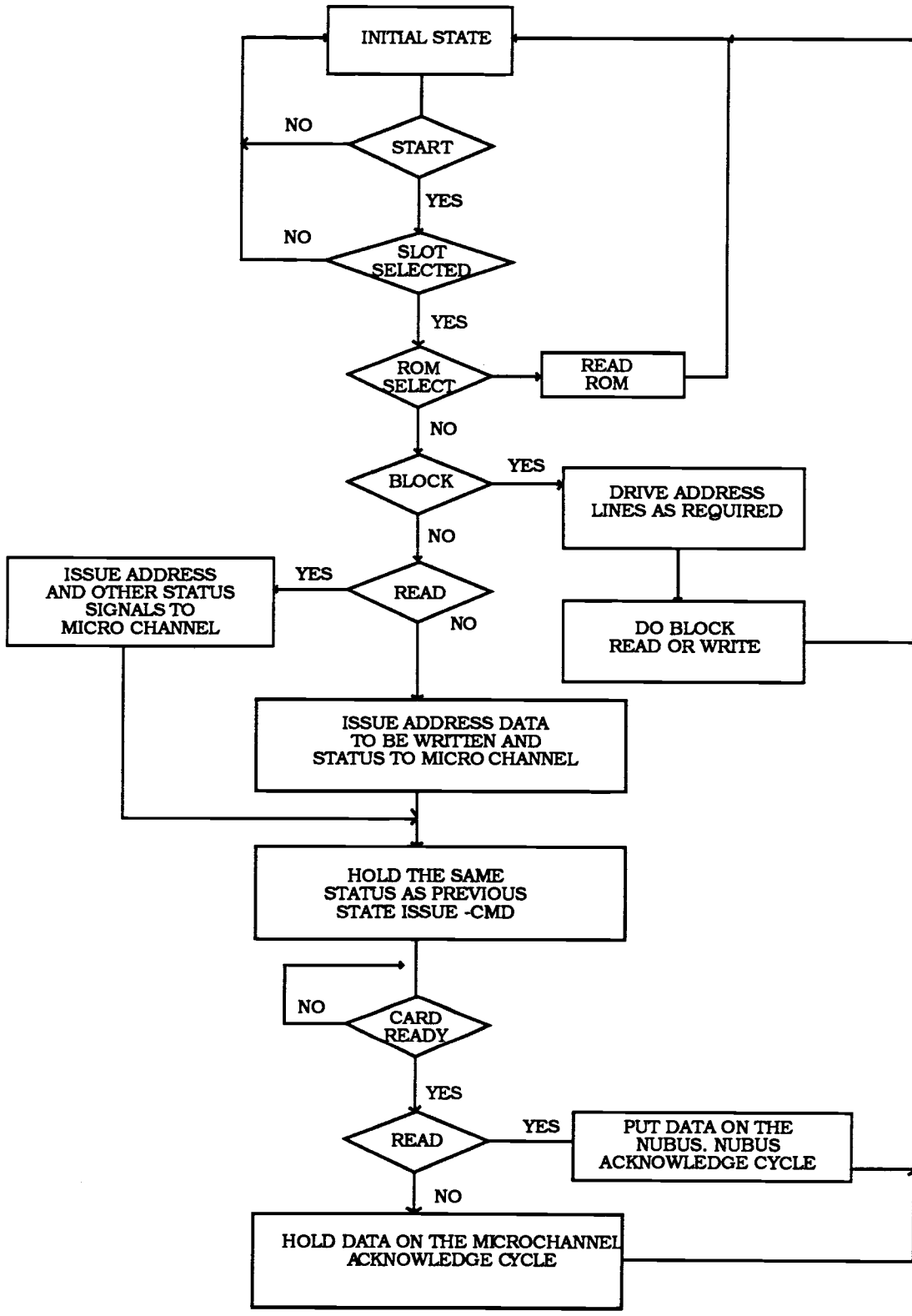


Figure 7.1 : General Flow Chart of Interface Controller

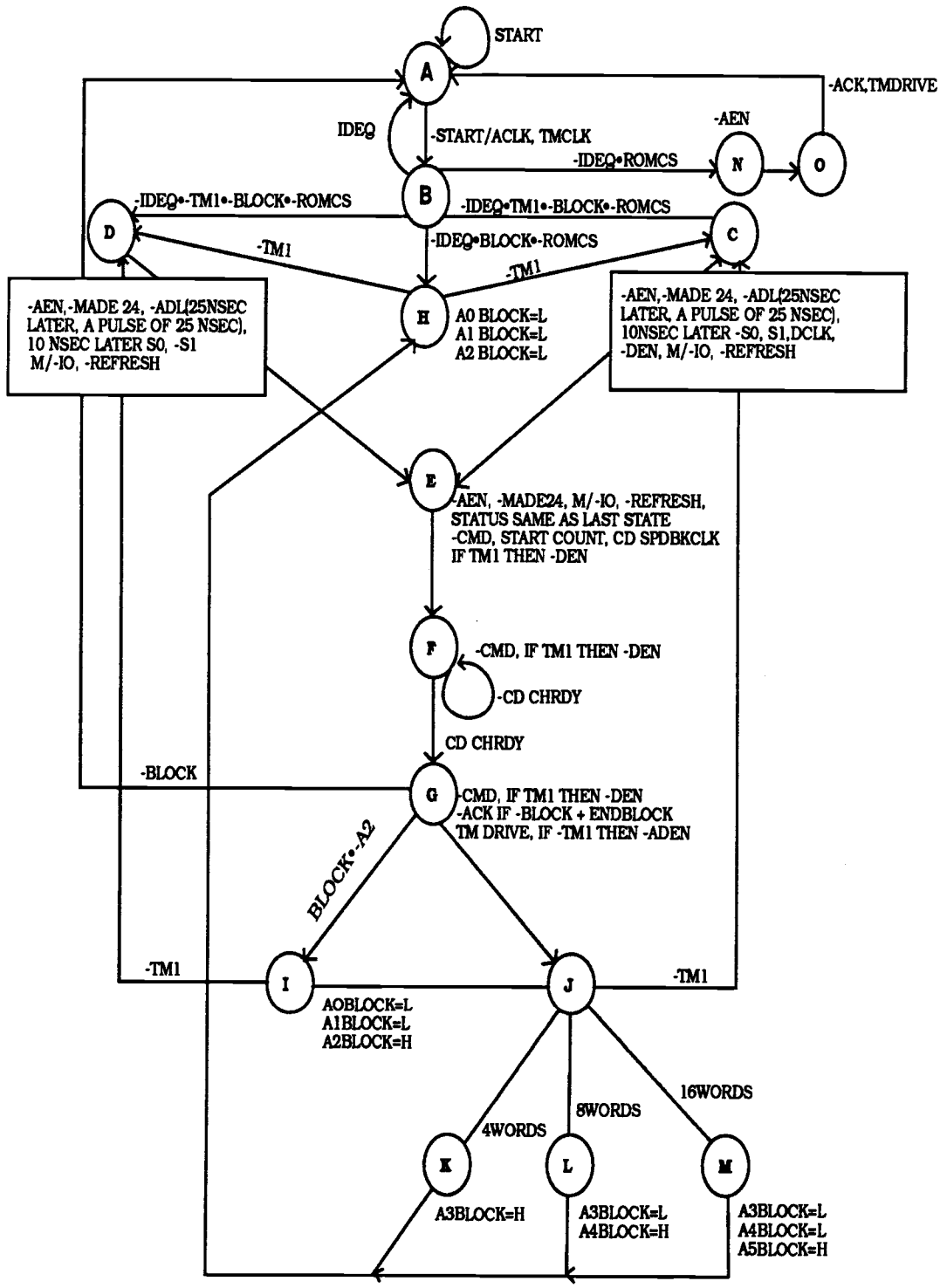
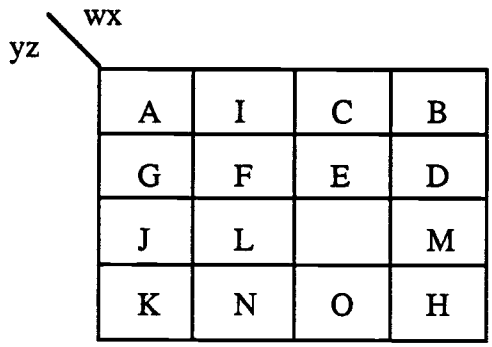


Figure 7.2 : State Diagram of the Interface Controller State Machine



State	State Assignment
A	0 0 0 0
B	1 0 0 0
C	1 1 0 0
D	1 0 0 1
E	1 1 0 1
F	0 1 0 1
G	0 0 0 1
H	1 0 1 0
I	0 1 0 0
J	0 0 1 1
K	0 0 1 0
L	0 1 1 1
M	1 0 1 1
N	0 1 1 0
O	1 1 1 0

Figure 7.3 : State Assignment

Present State	Inputs	Next State	Outputs
w x y z		Wn Xn yn Zn	
0 0 0 0	-START	0 0 0 0	DRIVE A5B
	START	1 0 0 0	ACLK,TMCLK, DRIVE A5B
1 0 0 0	-IDEQ	0 0 0 0	
	IDEQ.BLOCK.-ROMCS	1 0 1 0	
	IDEQ.-BLOCK.TM1. -ROMCS	1 1 0 0	
	IDEQ.-BLOCK.-TM1. -ROMCS	1 0 0 1	
	IDEQ.ROMCS	0 1 1 0	
1 1 0 0		1 1 0 1	-AEN,-MADE24,M/-IO,-REFR--SH,-ADL,-S0,S1,DCLK
1 0 0 1		1 1 0 1	-AEN,-MADE24,M/-IO, -REFRESH,-ADL,S0,-S1

Figure 7.4 : State Table of the Interface Controller State Machine

1 1 0 1	TM1	0 1 0 1	-AEN,-MADE24,M/-IO,-REFR- -SH,-S0,S1,-DEN,-CMD, START COUNT,CD SFDBKCL
	-TM1	0 1 0 1	-AEN,-MADE24,M/-IO,-REFR- -SH,S0,-S1,-DEN,-CMD, START COUNT,CD SFDBKCL
0 1 0 1	-CD CHRDY.TM1	0 1 0 1	-CMD,-DEN,START COUNT
	-CD CHRDY.-TM1	0 1 0 1	-CMD,START COUNT
	CD CHRDY.-BLOCK. -TM1	0 0 0 0	-CMD,-ACK,-ADEN,TMDRIVE
	CD CHRDY.-BLOCK. TM1	0 0 0 0	-CMD,-ACK,-DEN,TMDRIVE
	CD CHRDY.BLOCK. -TM1.-A2OUT	0 1 0 0	-CMD,-ADEN,TMDRIVE
	CD CHRDY.BLOCK. TM1.-A2OUT	0 1 0 0	-CMD,-DEN,TMDRIVE
	CD CHRDY.BLOCK. A2OUT.ENDBLOCK. -TM1	0 0 0 0	-CMD,-ACK,-ADEN,TMDRIVE

Figure 7.4 : Continued

	CD CHRDY.BLOCK. A2OUT.ENDBLOCK. TM1	0 0 0 0	-CMD,-ACK,-DEN,TMDRIVE
	CD CHRDY.BLOCK. A2OUT.-ENDBLOCK. -TM1	0 0 1 1	-CMD,-ADEN,TMDRIVE
	CD CHRDY.BLOCK. A2OUT.-ENDBLOCK. TM1	0 0 1 1	-CMD,-DEN,TMDRIVE
1 0 1 0	-TM1	1 0 0 1	AOBLOCK=L,AIBLOCK=L, A2BLOCK=L,DRIVEA2B
	TM1	1 1 0 0	AOBLOCK=L,AIBLOCK=L, A2BLOCK=L,DRIVEA2B
0 1 0 0	-TM1	1 0 0 1	AOBLOCK=L,AIBLOCK=L, A2BLOCK=H,DRIVEA2B
	TM1	1 1 0 0	AOBLOCK=L,AIBLOCK=L, A2BLOCK=H,DRIVEA2B
0 0 1 1	4WORDS.-8WORDS. -16WORDS	0 0 1 0	

Figure 7.4 : Continued

	-4WORDS.8WORDS. -16WORDS	0 1 1 1	
	-4WORDS.-8WORDS. -16WORDS	1 0 1 1	
0 0 1 0		1 0 1 0	A3BLOCK=H,DRIVEA3B
0 1 1 1		1 0 1 0	A3BLOCK=L,DRIVEA3B A4BLOCK=H,DRIVEA4B
1 0 1 1		1 0 1 0	A3BLOCK=L,DRIVEA3B A4BLOCK=L,DRIVEA4B A5BLOCK=5,DRIVEA5B
0 1 1 0		1 1 1 0	-AEN
1 1 1 0		0 0 0 0	-ACK,TMDRIVE

Figure 7.4 : Continued

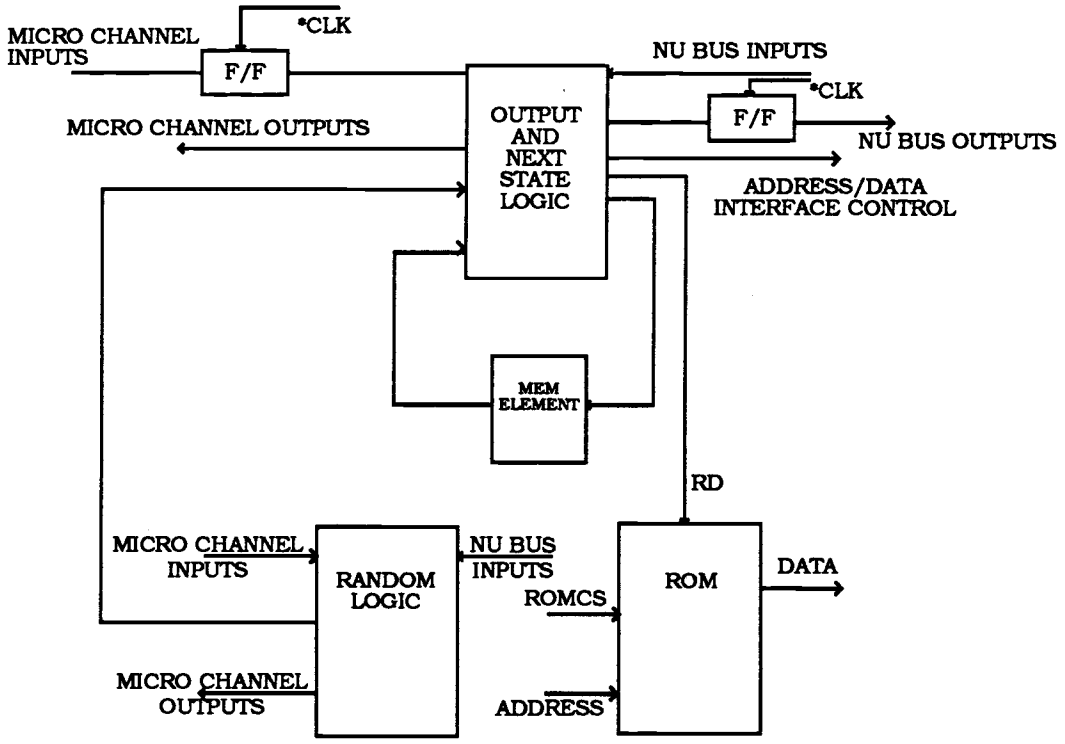


Figure 7.5 : Block Diagram of the Interface Controller

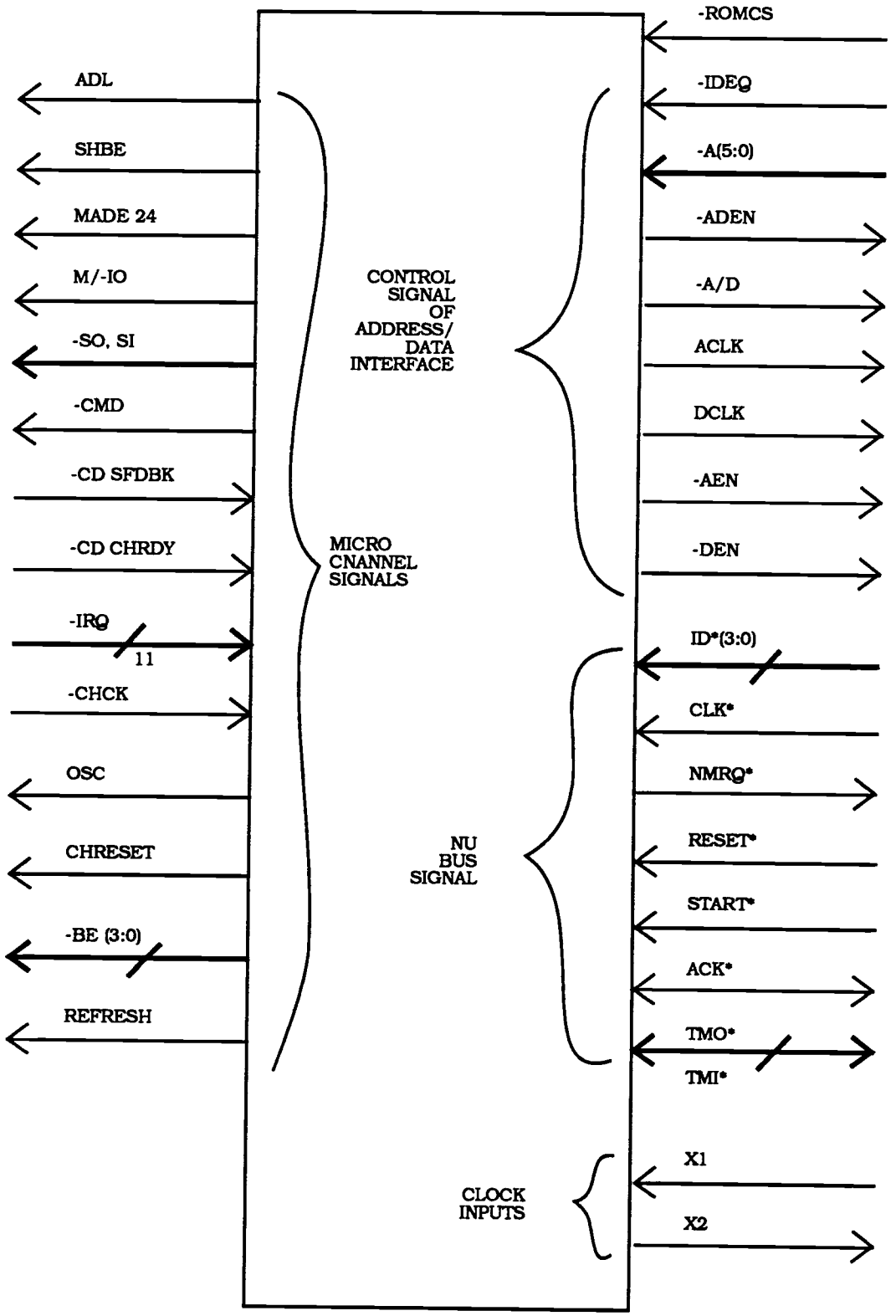


Figure 7.6 : Input/Output Signals of the Interface Controller

TM0*	A1	A0	-BE0	-BE1	-BE2	-BE3	Type of Cycle
L	L	L	L	H	H	H	Byte 0
L	L	H	H	L	H	H	Byte 1
L	H	L	H	H	L	H	Byrt 2
L	H	H	H	H	H	L	Byte 3
H	L	L	L	L	L	L	Full Word
H	L	H	L	L	H	H	1/2 Word 0
H	H	L	L	L	L	L	Block
H	H	H	H	H	L	L	1/2 Word 1

Note : TM1* = L Write Cycle.
TM1* = H Read cycle.

-BE0 = TM0L.A0 + TM0L.A1 + A1.A0
-BE1 = TM0L.-A0 + A1.A0
-BE2 = TM0L.-A1 + TM0L.A0 + -A1.A0
-BE3 = TM0L.A0 + -A1.A0

Note : TM0L is Latched TM0.

Figure 7.7 : Truth Table and Equations for Micro Channel signals -BE..

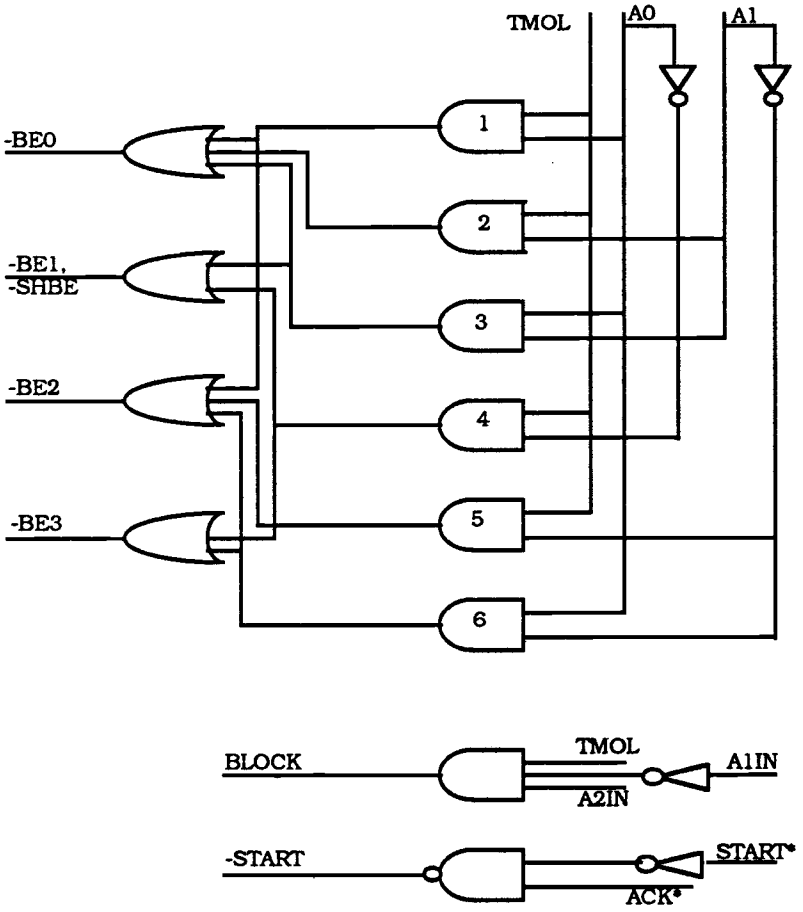


Figure 7.8 : Logic Diagrams

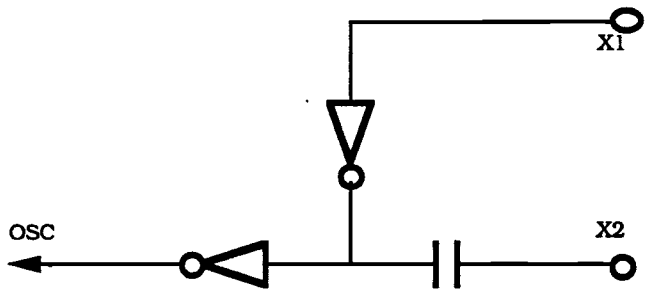


Figure 7.9 : Clock Generator Circuit

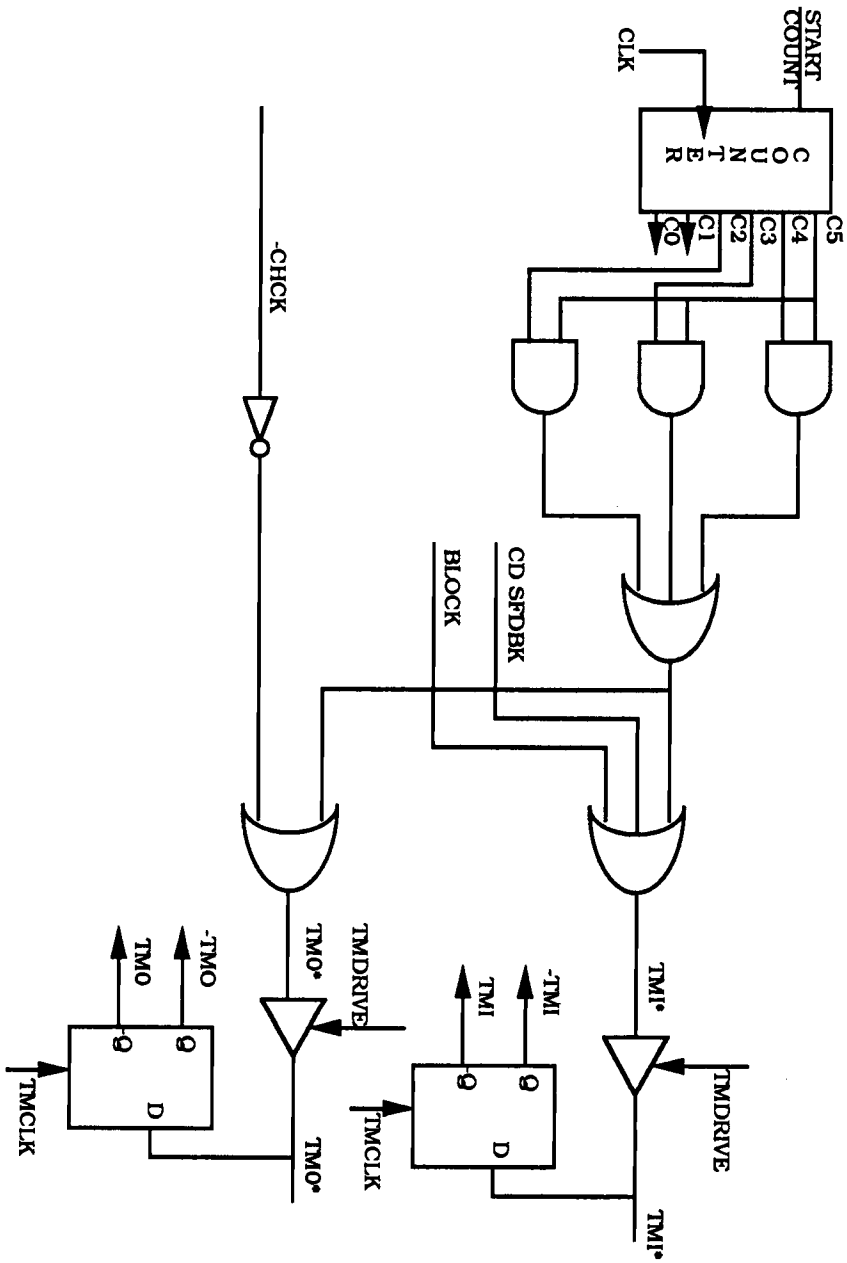


Figure 7.10 : Logic Diagram of Transfer Mode Signal Driving Circuit

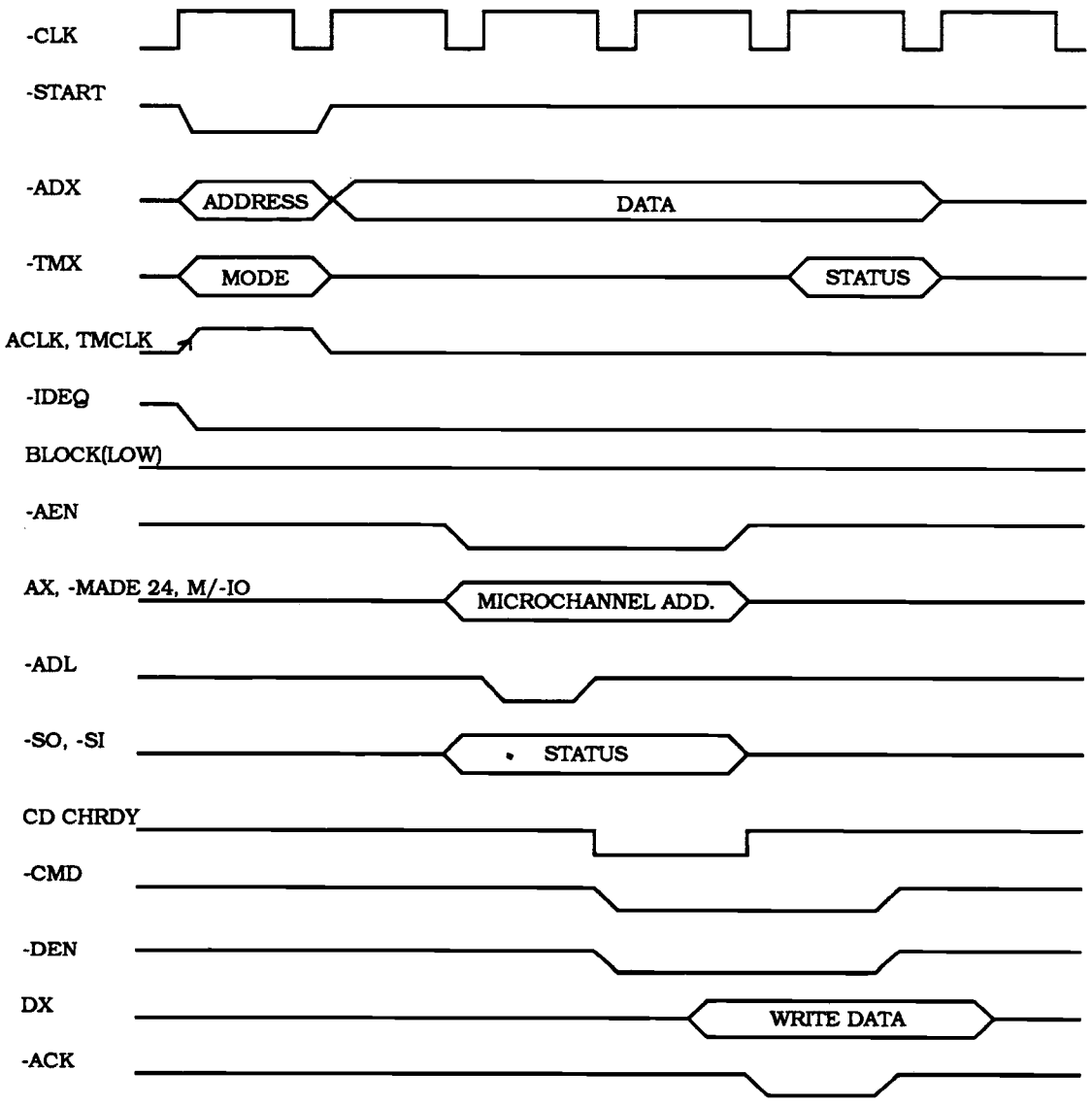


Figure 7.11 : Write Cycle of the Interface

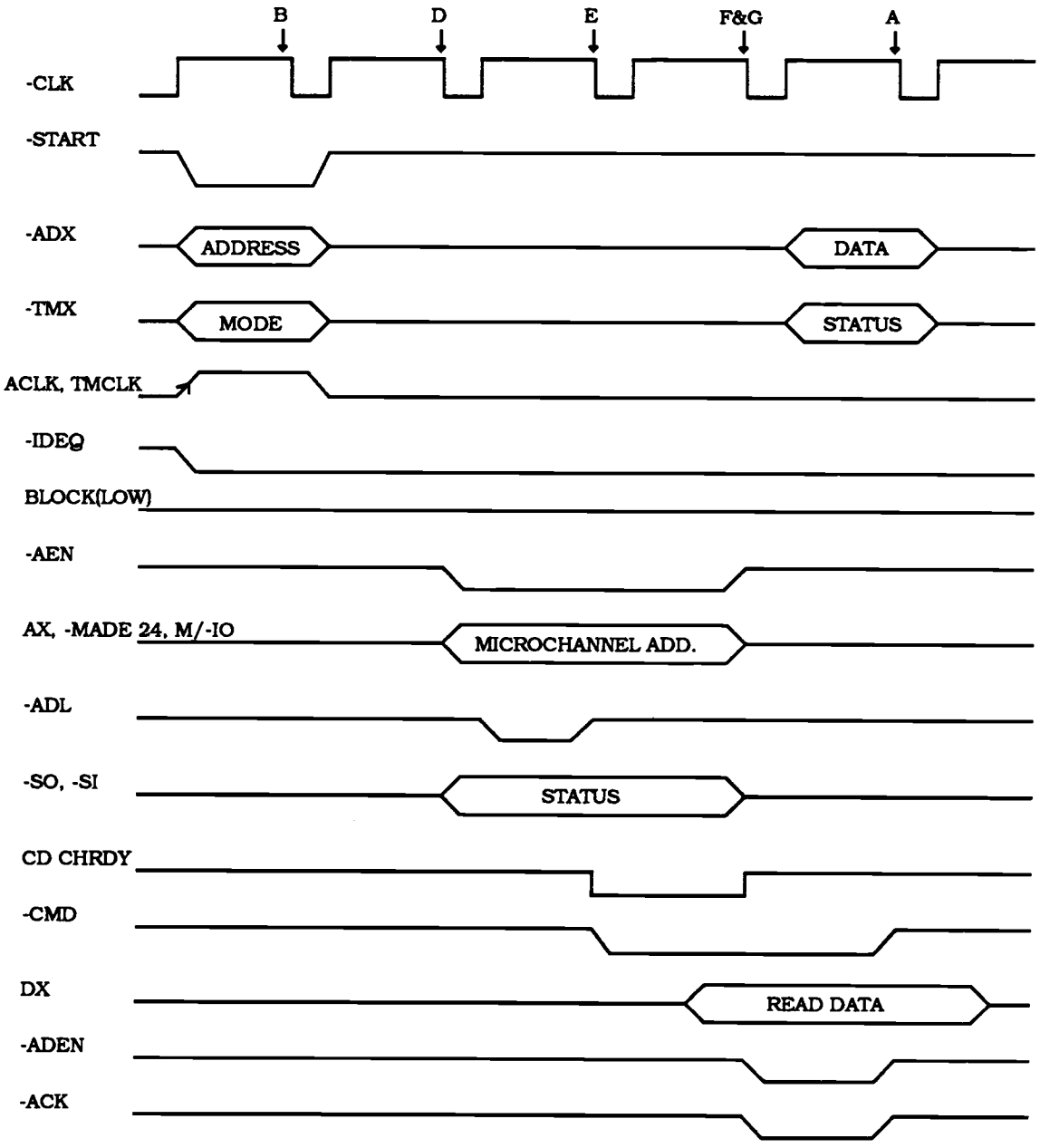


Figure 7.12 : Read Cycle of the Interface

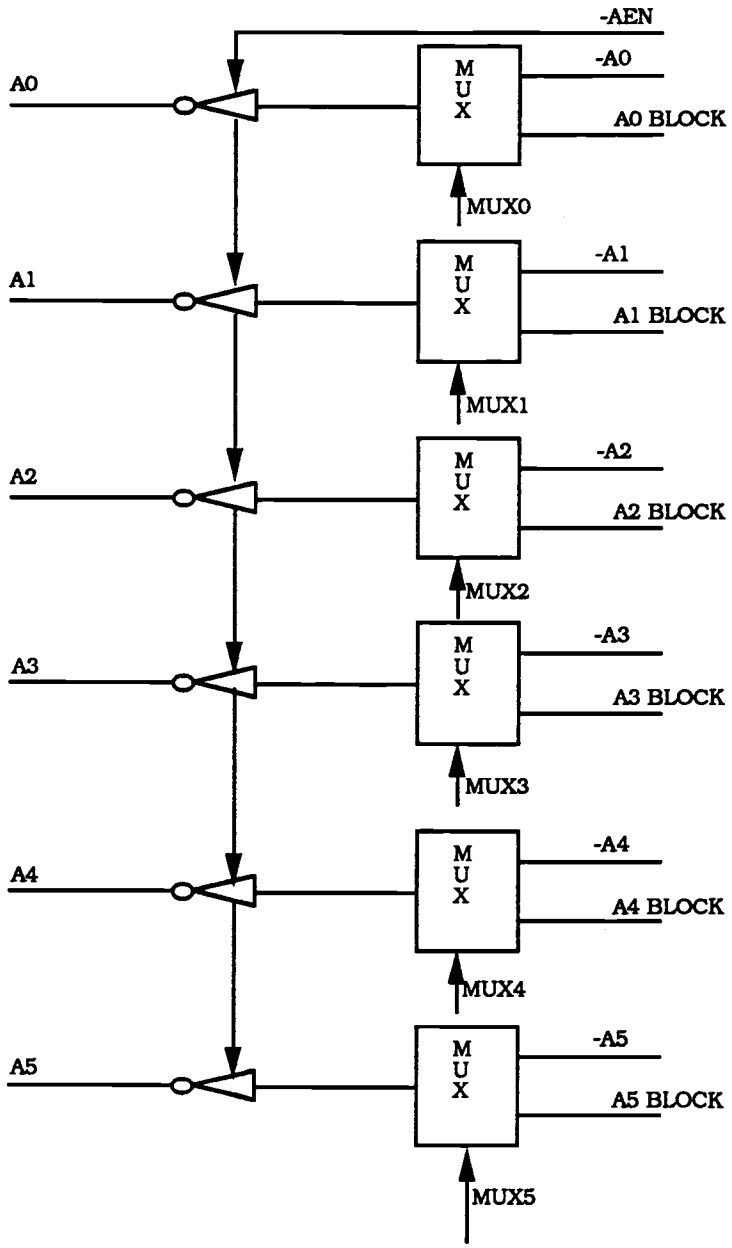


Figure 7.13 : Logic Diagram of Block Transfer Logic

BIBLIOGRAPHY

[1] Michael Slater, *Microprocessor Based Design : A comprehensive guide to effective hardware design*, Mayfield Pub. Co., Mountain View, Calif., 1987, Chapter 3.

[2] D. Del Corso, H. kirrman, and J. D. Nicoud, *Microcomputer Buses and Links*, Academic Pub. Co., London; Orlando, 1986.

[3] Arthur L. Dexter, *Microcomputer Bus Structures and Bus Interface Design*, M. Dekker, New York, 1986.

[4] Bill Robertson, "Interface Your PC to the Multibus", *EDN* January 8, 1987.

[5] International Business Machines, Personal System/2 Hardware Interface Technical Reference, IBM Corp., 1988.

[6] William C. (Bill) Nowlin, Jr., and Samuel K. (Sam) Ingram, "A Board Developer's Comparison of the PC-Style NuBus and IBM's Micro Channel Architecture", Texas Instruments Publication,1988.

[7] George White, "NuBus - A Simple, Cost-Effective Bus Interface", Texas Instruments Publication, 1988.

[8] Ciro Cornejo and Raymond Lee, "Comparing IBM's Micro Channel and Apple's NuBus", *BYTE*, Volume 12, Number12, 1987.

[9] Jon Shiell, "The 32-bit Micro Channel", *BYTE*, Volume12, Number 12, 1987.

[10] Martin Haeberli and Kirk Loevner, "The Macintosh II NuBus", *Mini Micro Systems*, November, 1987.

[11] Steve Ciarcia, "Under the Covers, The new IBM Micro Channel as seen from inside the PS/2 Model 50", *BYTE*, Volume 12, Number 8, 1987.

[12] International Business Machines, Micro Channel Architecture, IBM Corp., 1988.

[13] IEEE Standards Board, NuBus - A Simple 32-bit Backplane Bus, P1196 Specification, Draft 2.0, December 1986.

[14] Apple Computer, Inc., "Designing Cards and Drivers for Macintosh II and Macintosh SE", Inside Macintosh Library, March 1988.

[15] Texas Instruments, SN74ACT2440, NuBus Interface Controller specification, Texas Instruments Publication, 1988.

[16] Texas Instruments, SN74BCT2420, NuBus Address/Data Transceivers and registers specification, Texas Instruments Publication, 1988.

[17] Texas Instruments, Explorer, NuBus System Architecture General Description, December 1985.

[18] Johnson and Kassel, *The Multibus Design Guidebook*, McGraw-Hill, New York, 1984.

[19] Rolwin Lewis, "The Design of GPIB Interface for the Micro Channel", Iowa State university, May 1989

[20] Keith Dulac, "Design Considerations for a Three Port Bus Controller", *WESCON/87*, November 1987.

[21] Grahame Holmes, "Extending IBM Personal Computers to Interface With Real World Plant", *WESCON/84*, October 1984.

[22] Raymond. E. Floyed and Robert C. Stanley, "The IBM Personal Computers as Process controller", *WESCON/84*, October 1984.

[23] H. McG. Ross, "Communication of Peripheral Equipments to Computers, and an Outline of a Unified Method", IEE Conference Publication, Number 9, 1964.

[24] Thurber,K. J., Jenson,E. D., Jack,L. A., Kinney,L. L., Patton,P. C. and Anderson,L. C. (1972), "A systematic approach to the design of digital bussing structures". Proc. Fall Jr. Comput. Conf. 719 - 740.

[25] Roger R. Russ, "Getting the best of both the Buses", *Computer Design*, October 1983.

[26] Paul L. Borrill, "High-Speed 32-bit Buses for Forward-looking Computers" *IEEE Spectrum*, Volume 26, Number 7, 1989.

[27] IEEE, IEEE Standard Digital Interface for Programmable Instrumentation, IEEE Std. 488, 1978

APPENDIX

APPENDIX A

Design Equations

Equations for Next State Variables :

Zn = w.-x.-y.-z.IDEQ.-BLOCK.-TM1.-ROMCS + -y(w.z + x.w) + -TM1. - z(w.-x.y + -w.x.-y) + -w.-x.y.z.-4WORDS + -w.x.-y.z.BLOCK. A2out.-ENDBLOCK.CD CHRDY

Yn = w.-x.-y.-z.IDEQ(BLOCK + ROMCS) + -w.x.-y.z.BLOCK.A2out. -ENDBLOCK.CD CHRDY + -w.y + -x.y.z

Xn = w.-x.-y.-z.IDEQ(-BLOCK.TM1 + ROMCS) + w.x.-y + w.-y.z + -w.x. -y.z(BLOCK.-A2out + -CD CHRDY)+ -z.TM1(w.-x.y + -w.x) + -w. -x.-y.z.-4WORDS.8WORDS.-16WORDS + -w.x.y.-z

Wn = x.-y.-z + w.-x.z + -w.x.y + -x.y.-z + -w.-x.-y.-z.-START + w.-x.-y. -z.IDEQ.-ROMCS + -w.-x.-y.z. -4WORDS.-8WORDS.16WORDS

Equations For Control Signals of Address/Data Interface :

ACLK = -w.-x.-y.-z.-START.CLK

-AEN = -(w.x.-y + w.-y.z + -w.x.y.-z)

DCLK = w.x.-y.-z

-DEN = -(w.x.-y.-z + -w.x.-y.z).TM1

-ADEN = -(-w.x.-y.z.CD CHRDY.-TM1)

-A/D = HIGH

Note : HIGH = Logic 1 = Vcc.

Equations For NuBus Signals :

ACK* = -(-w.x.-y.z.CD CHRDY(-BLOCK + A2out.ENDBLOCK) + w.x.y.-z)

NMRQ*= -IRQ3.-IRQ4.-IRQ5.-IRQ6.-IRQ7.-IRQ9.-IRQ10.-IRQ11. -IRQ12. -IRQ14.-IRQ15

TM0* = C5.C4 + C5.C3 + C5.C2 + CHCK

TM1* = C5.C4 + C5.C3 + C5.C2 + CD SFDBK + BLOCK

TMCLK = -w.-x.-y.-z.-START.CLK

TMDRIVE = -w.x.-y.z.CD CHRDY + w.x.y.-z

Equations For Micro Channel Signals :

MADE 24 = -(w.x.-y + w.-y.z)

M/-IO = w.x.-y + w.-y.z

-REFRESH =-(w.x.-y + w.-y.z)

-SHBE = TMO.-A0 + A1.A0

-ADL = -(w.-y(x.-z + -x.z).-CLK)

-S0 = -(w.x.-y.-z + w.x.-y.TM1)

-S1 = -(w.-x.-y.z + w.-y.z.-TM1)

-CMD = -(x.-y.z)

CHRESET = -RESET*

-BE0 = TMO.A0 + TM0.A1 + A1.A0

-BE1 = TMO.-A0 + A1.A0

-BE2 = TMO.A0 + TM0.-A1 + -A1.A0

-BE3 = TMO.-A0 + -A1.A0

CD SFDBK CLK = w.x.-y.z

START COUNT = w.x.-y.z + -w.x.-y.z.-CD CHRDY

Equations For Block Support Logic Signals:

A0block = LOW

A1block = LOW

A2block = -w.x.-y.-z

A3block = -w.-x.y.-z

A4block = -w.x.y.z

A3block = w.-x.y.z

DRIVEA2B = -w.x.-y.-z + w.-x.y.-z

DRIVEA3B = -w.-x.y.-z + -w.x.y.z + w.-x.y.z

DRIVEA4B = -w.x.y.z + w.-x.y.z

DRIVEA2B = -w.-x.-y.-z + w.-x.y.z

MUX0 = BLOCK

MUX1 = BLOCK

MUX2 = BLOCK

MUX3 = BLOCK.-A3.A2

MUX3 = BLOCK.-A4.A3.A2

MUX3 = BLOCK.-A5.A4.A3.A2

4WORDS = A2block.BLOCK(A3 + A4 + A5)

8WORDS = A2block.A3block.BLOCK(A4 + A5)

4WORDS = A2block.A3block.A4block.BLOCK.A5