Design of an Interface for Multiple Microprocessors

Vairavan Palaniappan

A Thesis

in

The Department

of

Electrical Engineering

Presented in Partial Fulfilment of the Requirements
for the degree of Master of Engineering at
Concordia University
Montréal, Québec, Canada

May 1984

ABSTRACT

Design of an Interface for Multiple Microprocessors

Vairavan Palaniappan

For the last decade, one of the most active and exciting areas in computer architecture is the interconnection of multiple processors. Multiprocessors systems may range in organization from processors sharing a common memory to geographically isolated computer installation connected as a network. Present advances in semi-conductor technology used to implement the majority of microprocessors, limit the instruction execution rates such that many applications are compute bound rather than limited by other system bandwidths. One way to overcome the problem is to interconnect a number of processors together such that the overall performance of the system improves. In order for the processors of the system to communicate with each other, an interface circuit would be required.

This thesis is primarily concerned with the design and implementation of a simple and efficient multi-microprocessor interface. It is shown that the interface increases the general system performance by speeding up the data transfer between the processors. A simulation model of the interface is presented and the validity of the model is discussed based on the simulation results using GPSS V (General Purpose Simulation System V). The interface is built using TTL, MOS technology SSI and MSI integrated circuits and tested with two Intel 8085-A microcomputer (SDK-85). Certain applications of the interface in the areas of digital signal processing and distributed processing are discussed. Finally some suggestions for further investigation are presented which will enhance the performance of the interface.

ACKNOWLEDGEMENTS

I would like to express my sincere thanks to my thesis supervisor, Dr. M. Omair Ahmad for his invaluable advice and guidance throughout the course of this research work and for his numerous suggestions for improvements during the preparation of the manuscript. The comments of the committee members are gratefully acknowledged.

I wish to thank my brother-in-law, Mr. K.R. Subramanian for his encouragement and interest in my studies.

I extend my sincere thanks to Mr. Pierre Chevrier, Technical Officer, Department of Electrical Engineering for his cooperation during the hardware implementation of the project, and to Mrs. Monica Etwaroo for her excellent typing of the thesis.

I would also like to thank my fellow graduate students for their moral support during my studies.

I gratefully acknowledge the financial assistance from Natural Sciences and Engineering Research Council of Canada.

DEDICATED

TO

MY FATHER MR. VEST. VAIRAVAN

AND

MY MOTHER MRS. VAIRAVAN UNNAMALAI

WITH LOVE

TABLE OF CONTENTS

PAGE

LIST OF FIGURES

PAGE

LIST OF TABLES

CHAPTER I

INTRODUCTION

1.1 MULTIPROCESSOR SYSTEMS

Many problems such as on-board processing on satellites, intersatellite linking, low-frequency sonar signal analysis, digital signal processing for speech and image analysis etc. [1,2] require computational speed faster than what is attainable by using a uniprocessor system. The speed of a uniprocessor system is limited by physical laws. In recent years, advances in intergrated circuit technology have made multiprocessor systems to experience dramatic growth and increasingly widespread applicability. This has increased the processing speed of systems by several orders of magnitude.

Multiprocessor architectures are grossly characterized by two attributes: first, they include multiple, autonomous processors, and second, all processors share most, and often all, of primary memory [3]. Fig. 1.1 shows the basic structure of a multiprocessor system. It consists of P processors, M memory modules and an interconnection network. The system may perform functions in either a load sharing or a resource sharing mode or possibly in a mode which is a combination of the two [4]. The interconnection network requires some mechanism for inter processor exchange of informations. One of the most challenging problems in designing and implementing the hardware of a multiprocessor system, is the switching structure [5]. A multiprocessor interface, allows

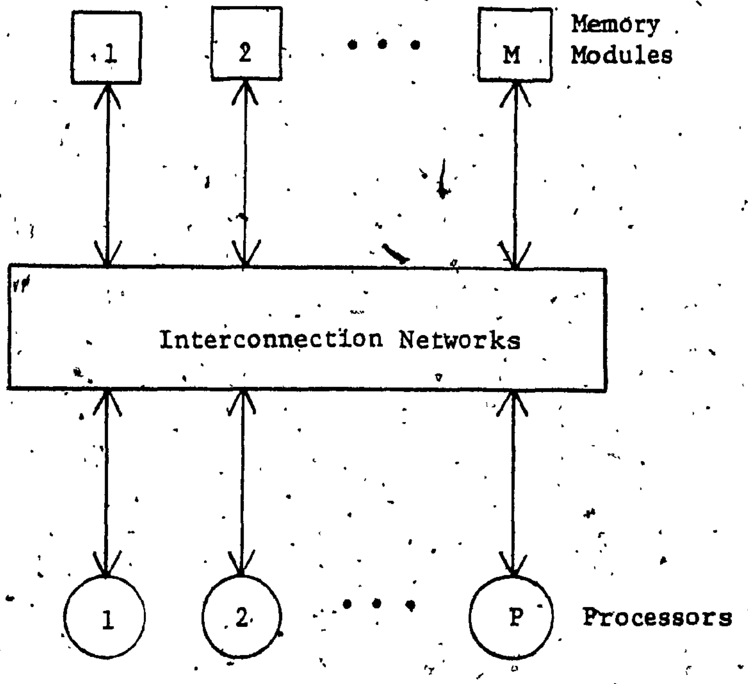


Figure 1.1 Multiprocessor System

processors of varied capabilities to communicate with each other. The degree of complexity of such an interface depends on the type of interprocessor communication desired and on the desired interconnection topology [6].

1.2 LOGICAL STRUCTURE OF A MULTIPROCESSOR SYSTEM

Logical structure refers to the way the control responsibility is distributed among the system processing elements. From the logical structure point of view, there are two types of organization - vertical and horizontal. In a vertical structure, elements are hierarchically structured. There is a global processor with multiple local processors, implying a master-slave relation. Logically all the elements are not equal. I/O processing is done by local processors, thus achieving high throughput. In a horizontal structure, all the elements are logically equal implying master-master relation [6]. In general, horizontal structure is more flexible and requires more sophisticated coordination than vertical structure and is capable of dynamic load sharing.

1.3 PHYSICAL STRUCTURE OF A MULTIPROCESSOR SYSTEM

Physical structure of a multiprocessor system refers to the method of information exchange between processors and is a function of interprocessor communication arrangements and interconnection topology.

1.3.1 Interconnection Topology

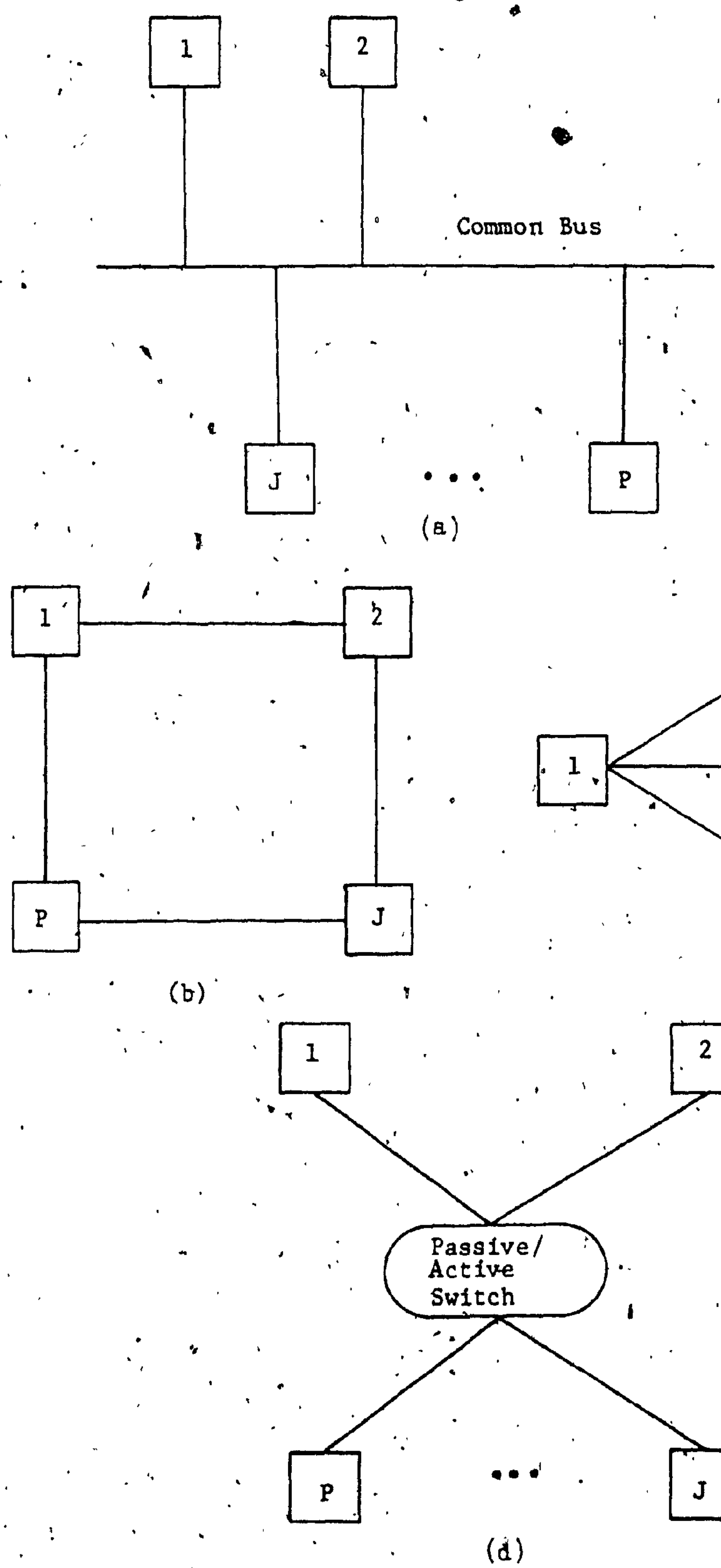
Physically there are many ways of interconnecting P processing elements in a system. Reliability and expandability are the two most important factors, establishing the interconnection scheme. The four most basic interconnection schemes are bus, star, ring, and fully connected structures as shown in Fig. 1.2. Other topologies are combination of these schemes [6,7]. Each of these interconnection schemes has its merits depending on the application.

1.3.2 Task Allocation

Overall system speed can be improved based on task allocation and partitioning in a computer network. For example, a multiprocessor system can be implemented in such a way as to support a pipeline or parallel partitioning as illustrated in Fig. 1.3 [1]. In pipeline partitioning, system functions (tasks) are divided among several processors such that each processor performs a specific task, and pass on the result to another processor, whereas, a parallel partitioning allows each processor to perform its task independently in parallel with others.

1.3.3 Interprocessor Communication Arrangements

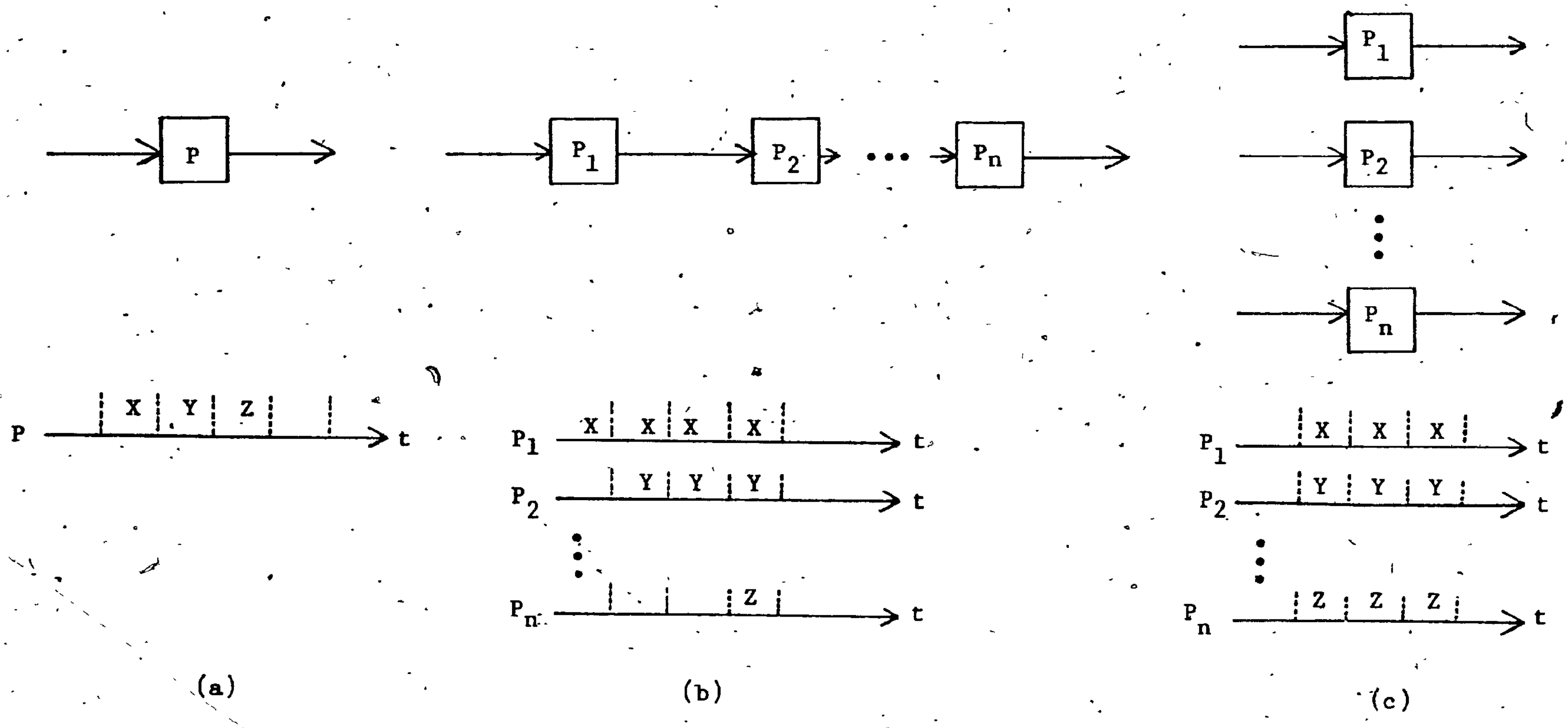
Speed, responsiveness, and throughput capacity of a multiprocessor system can be improved by proper balance between processing and interprocessor communications. Data transfer between the processors can be carried out either via a common memory structure referred to as



1...P - Processors

(a) Common Bus (b) Ring
(c) Fully Connected (d) Star

Figure 1.2, Interconnection Topologies



P-Processor X,Y,Z - Tasks

Figure 1.3 (a) Single Processor (b) Pipeline Partitioning (c) Parallel Partitioning.

centralized structure or a bus structure known as distributed structure [4].

Shared Memory

In this class of systems, the principal means of interaction between processors is provided by a common memory. Mutiprocessor systems which employ this interconnection are called as tightly coupled structure. A schematic of a tightly coupled structure is shown in Fig. 1.4 [4,6].

All the processors in the tightly coupled system can get access to all the memories and I/Os, and other system resources that are shared among processors. The major limitation of this system is the possibility of common memory access conflicts. In contrast to tightly coupled systems, loosely coupled system as shown in Fig. 1.4 do not share memory. At the hardware level there has to be an explicit communication interface between the processors. Combinations of these two extreme structures form a structure called moderately coupled structure.

The three important processor-memory physical interconnection schemes are

i) Time shared/common bus shared memory,

ii) Cross bar switch shared memory, and

iii) Multibus/multiport shared memory

Each of these types of the shared memory organizations has certain attributes that affect its suitability for a

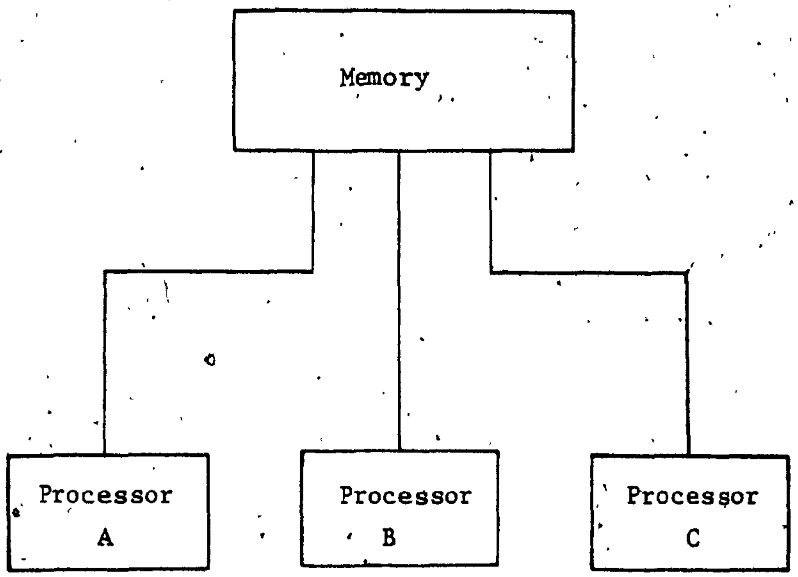


Figure 1.4 (a) Tightly coupled system

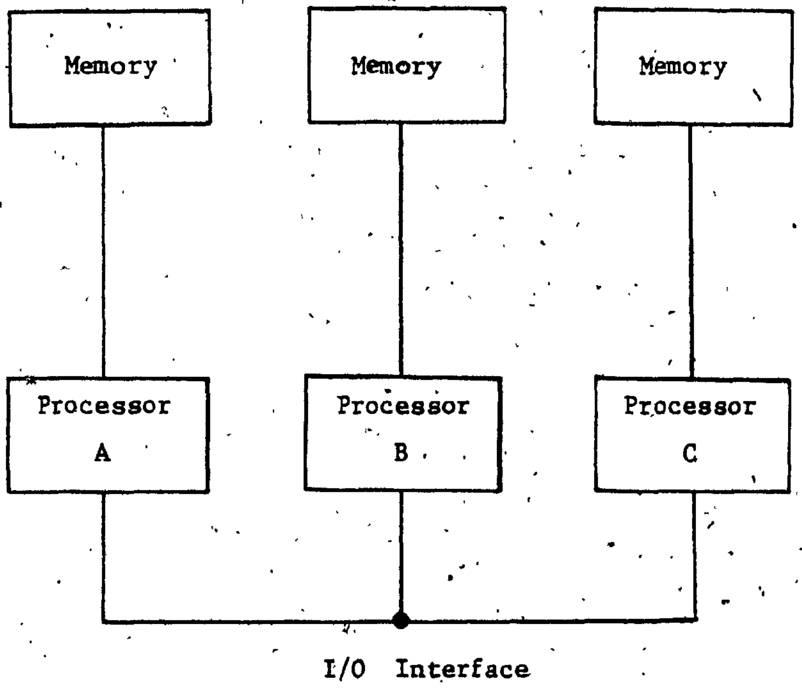


Figure 1.4 (b) Loosely coupled system

particular application. These attributes are related to cost, reliability, speed, throughput capacity, modularity, logical complexity, and physical dispersibility, etc.

(i) Time Shared/Common Bus, Shared Memory

The simplest interconnection system for a multiprocessor is a common communication path connecting all the functional units as shown in Fig. 1.5. Common bus consists of a memory arbitration logic, some shared memory, shared I/O, and a bus switch interface. The obvious drawback of this organization is low bus data transfer rate due to both bus contention and memory contention [8-10]. For this reason private memory and private I/O are highly desirable [11].

Dedicated bus structure (one per processor) as described in Fig. 1.6 is a solution to the above problem [10]. Multibus interconnection which connects all processors and all memory modules to all buses as shown in Fig. 1.7, is another solution. This interconnection network provides throughput which is intermediate between the single bus and the cross bar, with a correspondong intermediate cost [12]. Multiple bus network is less expensive than the cross bar. To reduce the cost further the network with partial buses has been described in [12] which is fault tolerant because it can operate in a degraded mode after the failure of a subset of the buses. The standard multibus connection scheme shown in Fig. 1.7 is redundant and expensive for a

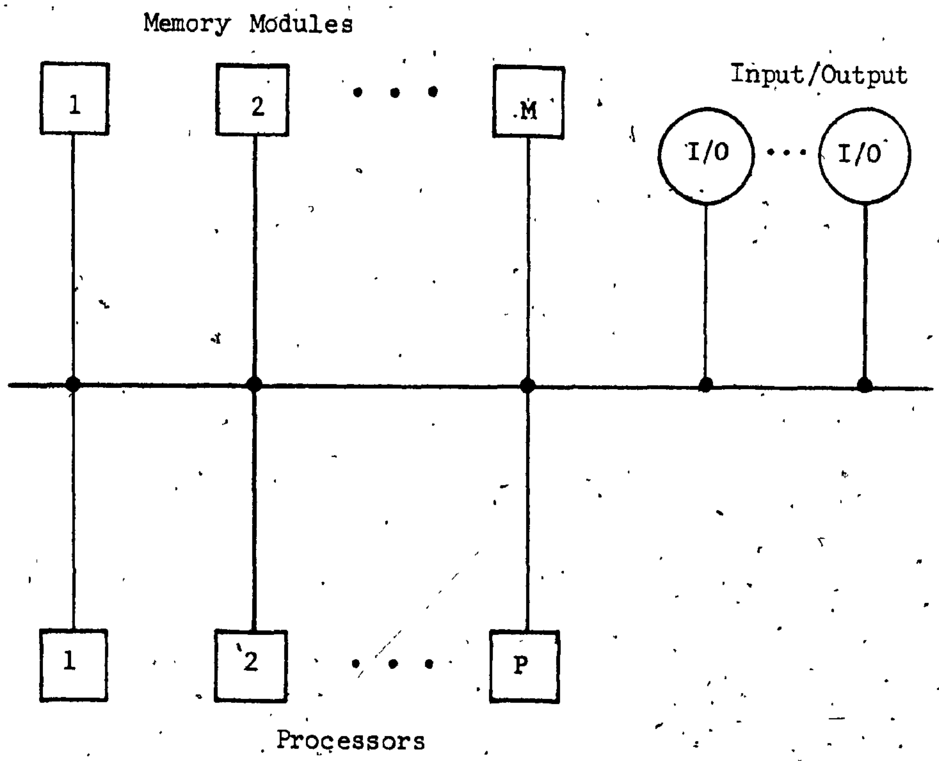


Figure 1.5 Single Bus Interconnection

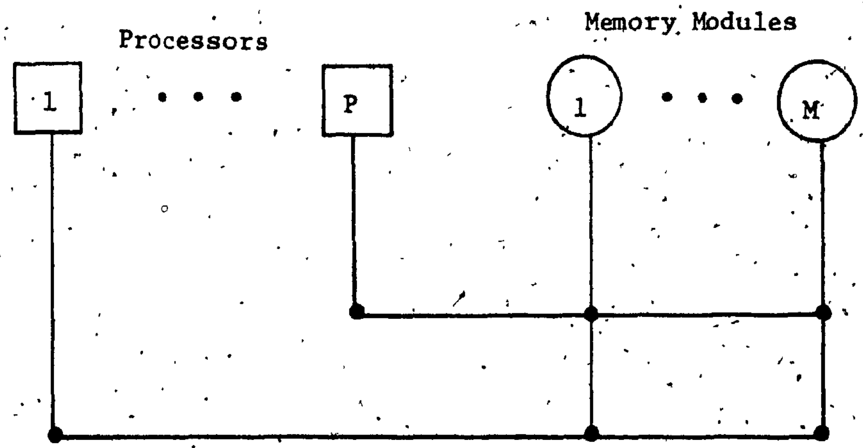


Figure 1.6 Dedicated Bus Interconnection

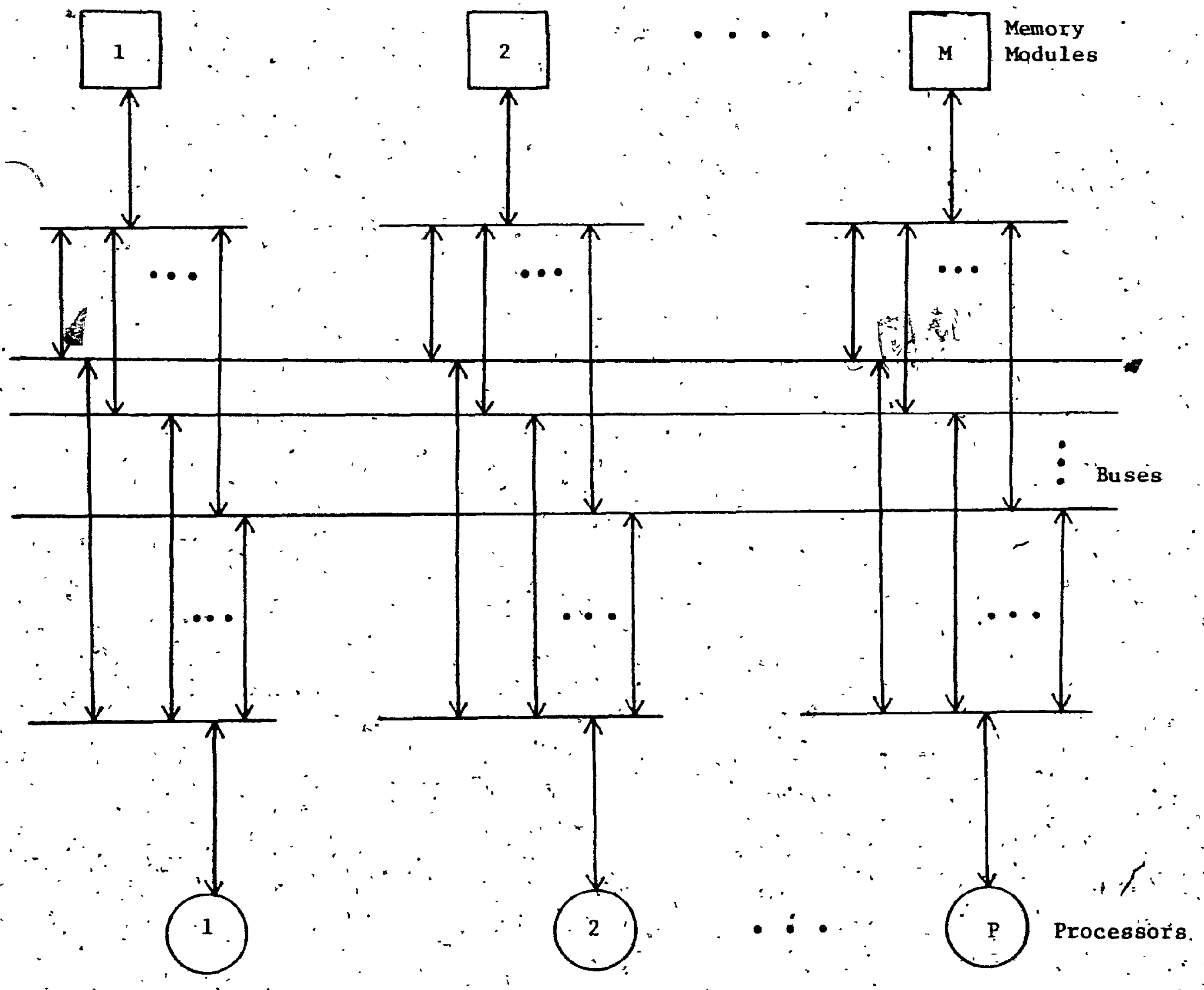


Figure 1.7 Complete Multiple Bus Interconnection

for relatively large number of buses [13]. Rhombic, balanced, cyclie, staircase, and trapezoidal networks discussed in [13] are the reduced connection schemes that produce the same throughput as the standard interconnection. The schemes are optimal with respect to number of connections, easy to arbitrate, reliable when a bus fails and expandable. The reduction is specially significant when the number of buses is relatively large.

(ii) Crossbar Switch Shared Memory

This is relatively complex and most expensive scheme which supports simultaneous transfers between all the processors and memory units. Bus interface logic functions are done by matrix switch which is complex, costly to control, and physically large. Fig. 1.8 is a crossbar-based multiprocessor system. If there are P processors and M memory modules, the crossbar requires PxM switches. The crossbar provides the largest potential bandwidth because there are no conflicts in the network. But it is less fault tolerant than the multiple bus structure, since a failure in one of the buses disconnects completely one memory module [12]. A way to reduce the wiring between processors and memory module is by using serial transmission of partial words at a frequency higher than that of processors, but this technique increases the cost of the system [9].

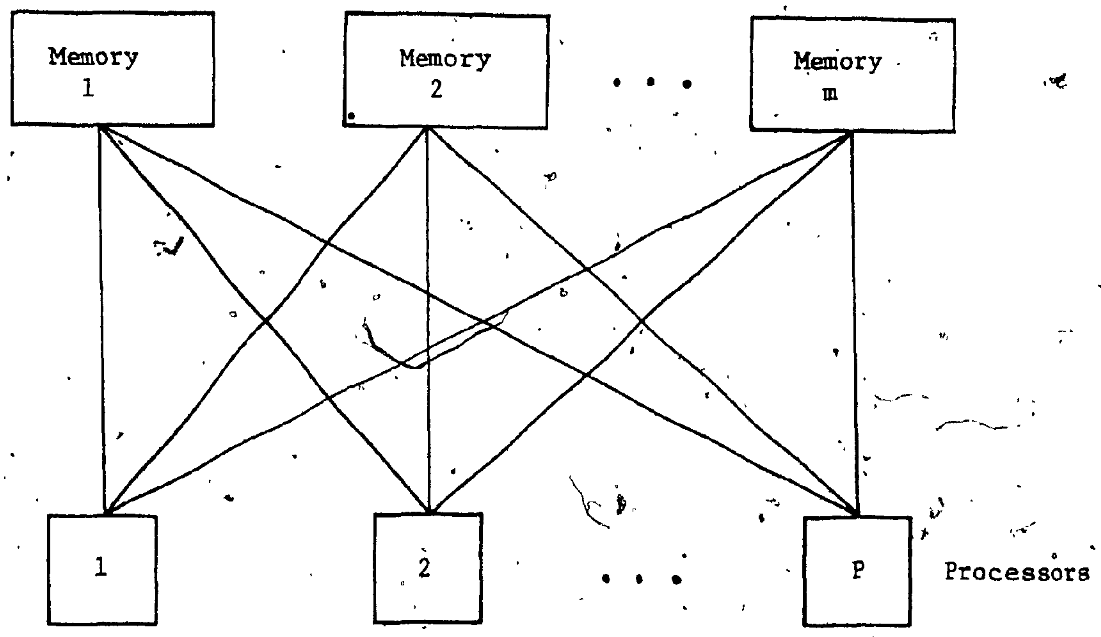


Figure 1.8 Multiported Memory System

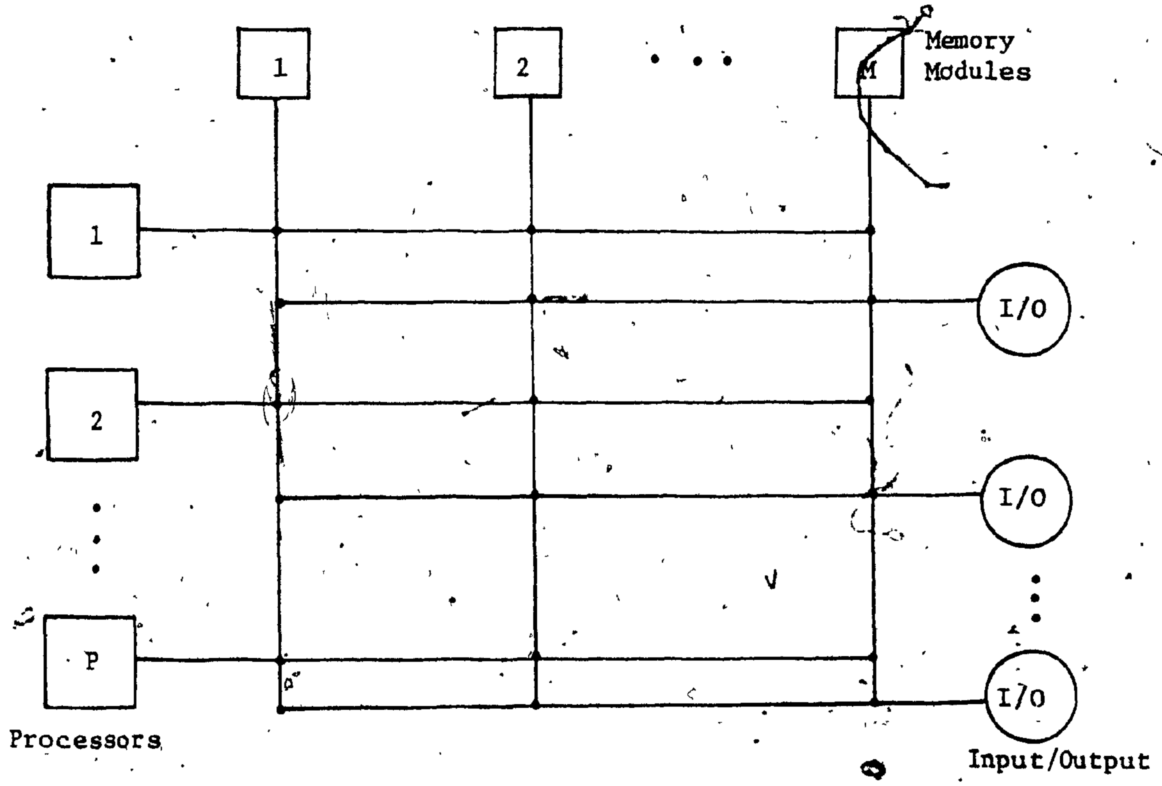


Figure 1.9 Crossbar Switch-Based Multiprocessor System

(iii) Multibus/Multiport Shared Memory

A multiport memory system, as depicted in Fig. 1.9, employs multiple dedicated buses connecting processors and memories, and accessing memory conflicts are resolved through hardwired fixed priorities. A large number of cables and connectors are required. Control and switching circuitry required by a memory unit increases the cost of the system. But very high total transfer rates may be achieved [14,15]. There generally is no speed differential between multiport and matrix arrangements [9]. But the difference is the wiring complexity. The DPS-1 (Distributed processing system) multiprocessor described in [10], uses a combination of time shared/common bus and a dual port memory.

Two important processor-memory logical interconnection schemes are virtual shared memory and mailbox shared memory as described below.

(i) Virtual Shared Memory

Virtual shared memory systems allows the use of the shared memory into virtual memory environment. It requires address translator hardware and various segmentation techniques to provide addressing capability for a larger memory space beyond the size of any processor's real memory. All the processors in the system are equal and can access any memory. This scheme is ideal for low speed applications. One of the disadvantages of this system is fixed memory management policy. CM* is a more sophisticated shared memory with virtual memory environment which can accommodate unlimited number of processors [16]. This is an example of hierarchial shared path networks [17].

(ii) Mailbox Shared Memory

This logical interconnection scheme uses shared memory in a mailbox fashion. In fact, insofar as multi-microcomputer systems are concerned, the primary use of a common memory would be to act as a message center,, where each processor can leave messages for other processors and pick up messages intended for it. Such an organization is illustrated in Fig. 1.10. If there are P processing systems, P mailboxes, each containing P-1 compartments,are required. Any processor can scan its mailbox to establish if there are any messages for it. This scheme is more advantegous than vertical memory environment since there is no extra overhead in address translation associated with each memory reference.

1.4 TWO EXISTING INTERFACE SCHEMES

The basic element of communication between processors is a message to be transferred between processors. . No distinction is made between different types of messages such as request for service, data blocks, etc [10]. For multiprocessor system to operate and function properly, it is necessary that each processor in the system be able to communicate with

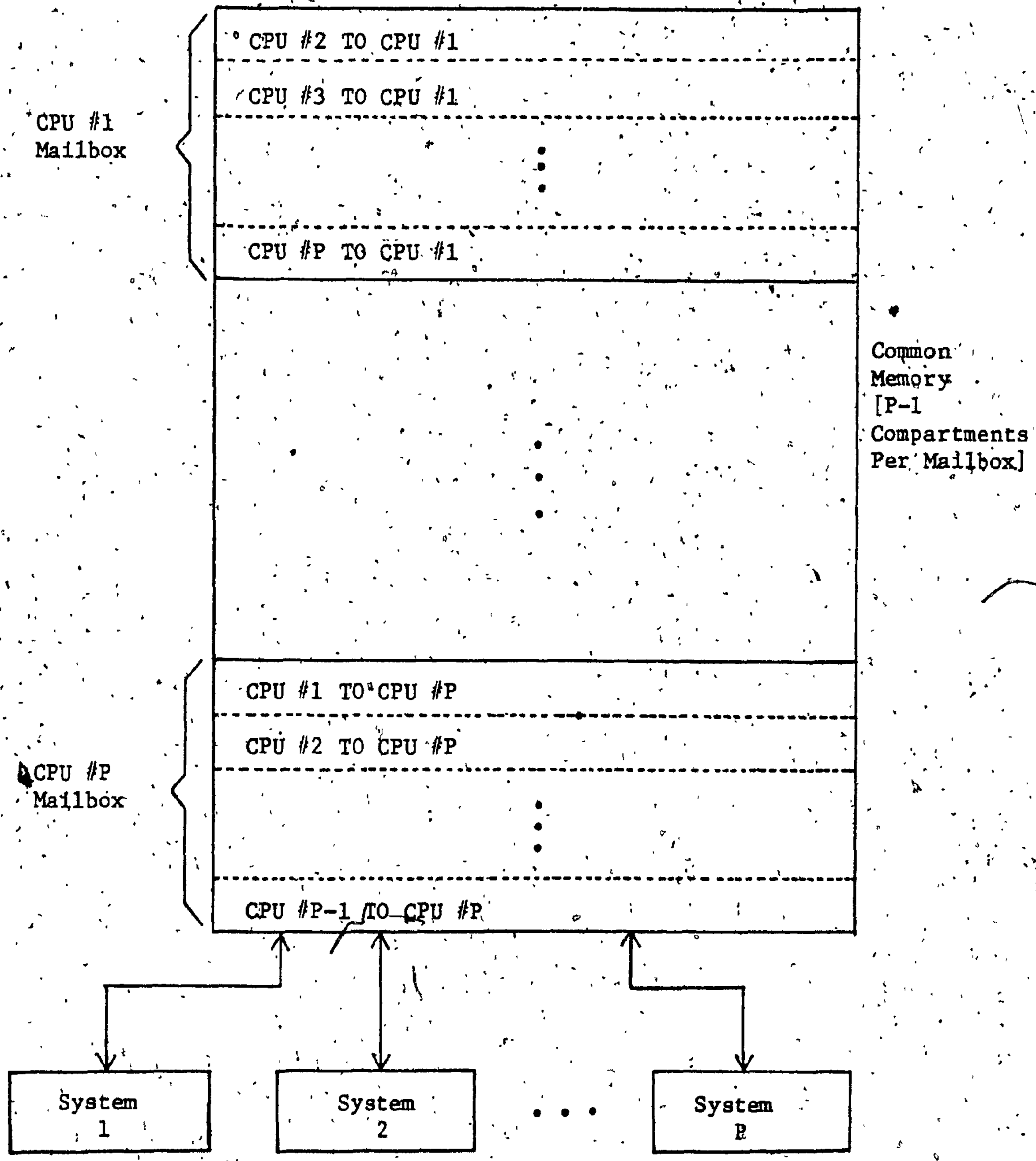


Figure 1.10 Mailbox Shared Memory Organization

others. Thus an interface known as multiprocessor interface, is necessary to provide efficient communication between the processors of varied capabilities.

The IEEE 488 standard defines an Instrumentation bus [18]. There are potential problems that could be encountered by attempting to use this bus for high speed interprocessor message communication. Each processor has one interface network which connects the processor to the bus and has the capability to assume one or more of three modes of operations— listener, talker or controller. A controller is an interface of the processor that controls the bus and has the ability to put any processor interface in either a listener or a talker mode [19].

In [20] another multiprocessor interface circuit has been described. This interface scheme accomodates only two processors. Each processor in the system uses an MC 68230 Parallel Interface/Timer (PI/T). The direction of data transfer between the processors is established by a TTL controller. A processor will initate and maintain data transfers in one direction, unless a request for data transmission from the higher priority processor is received by the controller. If such a request is received, the transmissions with lower priority processor will be suspended until the transmission with higher priority processor is completed.

In the above examples of the interface schemes each processor has it's own interface circuit. Consequently, wiring complexity and cost of implementation increases as the number of processors in the system increases. In addition, speed of data transfer is reduced. For example, when processor i wants to communicate with any of the processors in the system, at the time when processors j and k are communicating, the processor i keeps on testing the status of the situation till current communication between processors j and k is completed. During this period processors i's request is not recognised by the controller. If the controller did have the facility to recognize processor i's request, it would immediately establish the communication path between processor i and it's needed processor, once the current communication between processors j and k is over. Similarly, if a processor wants to send data to another processor, it must wait till the current communication is completed. This wastes the processing time of the processor in question.

1.5 PROPOSED MULTIPROCESSOR INTERFACE

Multiprocessing experiments have shown that because of contention on shared resources, speed increases equal to the increase in processors are achieved only over a limited range, thereafter performance will actually decrease if additional processors are added to the system. Hence a primary concern in a multiprocessor system is how to use large number of processors effectively for a given application. An outstanding generic research question is how to determine the amount of performance speedup (or parallelism) possible in an application and how to achieve that speed effectively.

The purpose of this investigation is to design, build, and test a general purpose multiprocessor interface. The interface scheme presented in this thesis is simple, flexible, and it can accomodate a large number of processors of varied capabilities. The interface basically functioning on first-come first-served basis, can at any instant receive the data from one processor and send the data to another processor simultaneously or receive the request for data from one processor and send the data to another processor simultaneously.

1.6 THESIS ORGANIZATION

Chapter II describes the architecture of the proposed interface in a detailed manner. A simulation model of the interface is presented in order to predict the behaviour of the interface, this model is then simulated using GPSS (General Purpose Simulation System) language.

In Chapter III design and implementation of the three modules of the interface are discussed.

In Chapter IV some applications of the interface in cascade, parallel, and distributed multiprocessing environments are discussed. In addition, some extensions to the research work in this thesis and suggestions for further study are also discussed.

Finally Chapter V summarizes the investigation carried out in this thesis.

CHAPTER II

INTERFACE ARCHITECTURE AND ITS SIMULATION USING GPSS

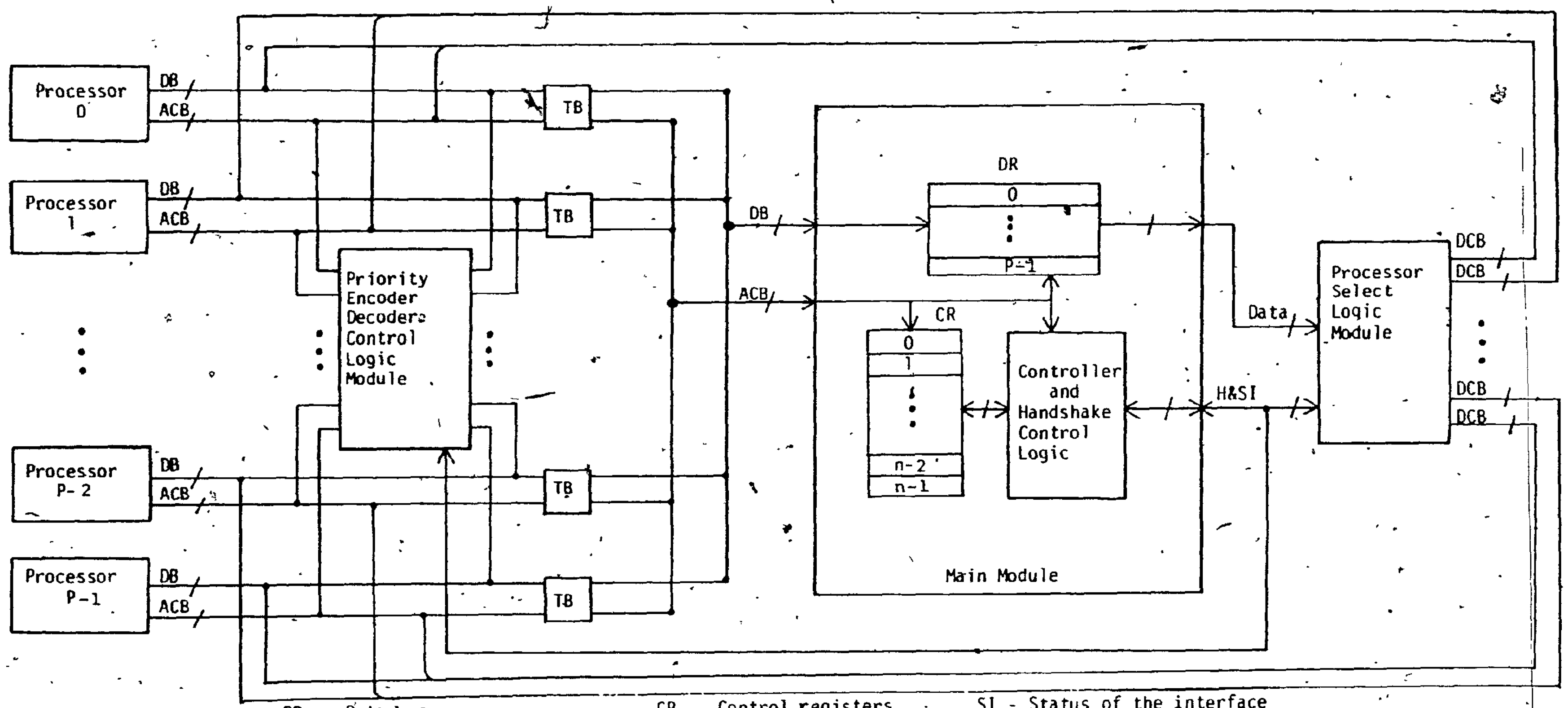
2.1 INTRODUCTION

Architecture of a simple, general purpose multiprocessor interface system, consisting of three logic modules, is described in this Chapter. As it would be useful to predict the behaviour of the proposed system, before it is built, the interface is modelled and simulated by using GPSS language. Discussion on the results obtained by computer simulations are also presented.

2.2 THE ARCHITECTURE OF THE MULTIPROCESSOR INTERFACE

The primary goal of the interface is to interact with two different processors simultaneously by accepting the data from one processor and sending the data to another processor or accepting request for data from a processor. The accepted requests will be serviced one by one on first-come first-serviced basis. The block diagram of the multiprocessor interface is shown in Fig. 2.1.

The interface can be divided into three logic modules. Each module has functions as described below. Design and implementation of each module will be discussed in Chapter III.


Processor 0
Processor 1
Processor P-2
Processor P-1
DB
ACB
TB
Priority Encoder Decoders Control Logic Module
DR
0
P-1
CR
0
1
n-2
n-1
Controller and Handshake Control Logic
Main Module
Data
H&SI
Processor Select Logic Module
DCB

DB - Data bus
ACB - Address and control bus
TB - Tri-state buffers

CR - Control registers
DR - Data registers
DCB - Data control bus

SI - Status of the interface
H - Handshake lines

Figure 2.1 Multiprocessor Interface

(i) The Main Module

Main module plays a major role in transferring and receiving data from processors. It contains P data registers each of which is dedicated to a processor in order to store data to be transferred to other processors on request. There are 2K-bit ($2^K \geq P$) n control registers to store a request for data from a processor on first-come first-served basis. Controller and handshake control logic are to control the transfer of data between processors and interface and coordinate various functions of the interface.

(ii) The Priority Encoder Decoder Control Logic Module (PEDC)

The PEDC logic module containing encoders and decoders provides information on the status of the interface to an accessing processor. It also assigns priority to each processor in the system, when more than one processor is trying to access the interface at a time.

(iii) The Processor Select Logic Module (PS)

The processor select logic module is controlled by the main module. The PS module routes the data and a handshake signal to the right processor and accepts a handshake signal from the processor for main module.

Each processor's address, data, and control bus is connected to the PEDC logic module and the main module. There are two kinds of operation in which a processor would interact with the interface.

(a) A processor has data to be stored in it's data register in the interface for use by other processors.

(b) A processor makes a request for data from another processor.

In the former case, the processor first tests the availability of the address bus to the interface (main module). If the bus is free, the processor transfers the data to its data register in the main module. A successful data transfer is then acknowledged to the processor by a handshake signal (data accepted) from the controller and handshake control logic. The sequence of events which take place for the data transfer is shown in Fig. 2.2. In the latter case, if processor i wants the data from processor j, processor i first tests the status of the interface (SI) by checking the availability of a control register and the availability of the address bus to the main module. If both conditions are met, then the request is recorded by storing the addresses of processors i and j in a control register. Once the stored request is ready for execution by the controller and handshake control logic, higher- and lower-order K bits of the control registers are decoded, respectively, as source and destination of the data. Actual data transfer to the processor takes place on first-come first-served basis. The main module sends a handshake signal (data ready) and data from processor j's data register to processor i through the PS logic module. Upon the receipt of this

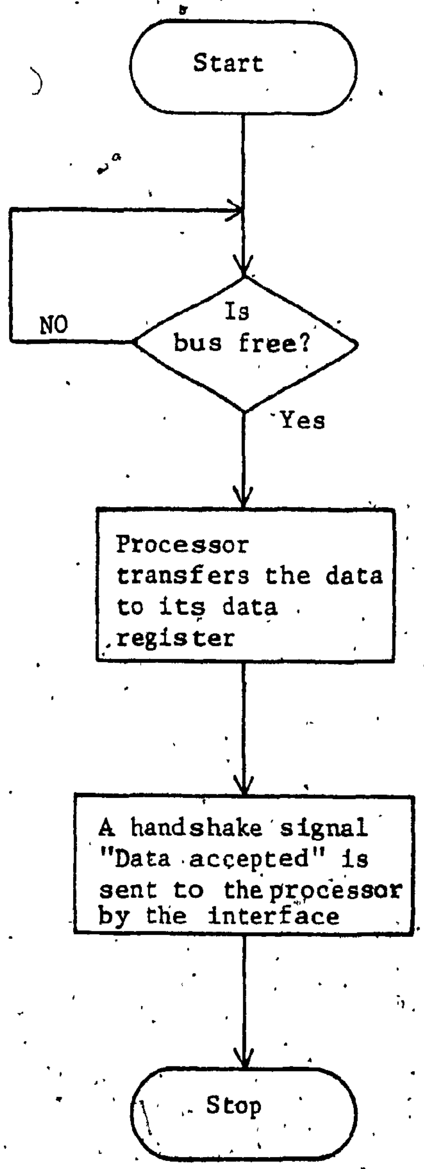

Start
Is bus free?
NO
Yes
Processor transfers the data to its data register
A handshake signal "Data accepted" is sent to the processor by the interface
Stop

Figure 2.2 Sequence of Events for Data Transfer to its Data Register by a Processor.

handshake signal, processor i latches the data and sends back another handshake signal (data received) to the main module through the PS logic module. The controller and handshake control logic are then ready to take another processor's request for execution. Fig. 2.3 delineates the sequence of events which take place for the data transfer from the interface.

The three types of conflicts that may arise during the interaction between the interface and processors are as follows:

(i) Two or more processors may initate the sequence of operations to store the request for data in a control register simultaneously.

(ii) Two or more processors may initate the sequency of operations to store the data to its data register simultaneously.

(iii) Different processors may simultaneously initate the sequence of operations to store the data to its data register and to store the request for data to a control register.

In the first two cases, the priority is pre-assigned to the processors by the PEDC logic module. While in the latter case, the priority is given by the PEDC logic module to the processor which has the highest priority among the set of processors trying to store data to it's data register. Fig. 2.4 depicts which processor has the eligibility to interact with the interface when such conflicts arise.

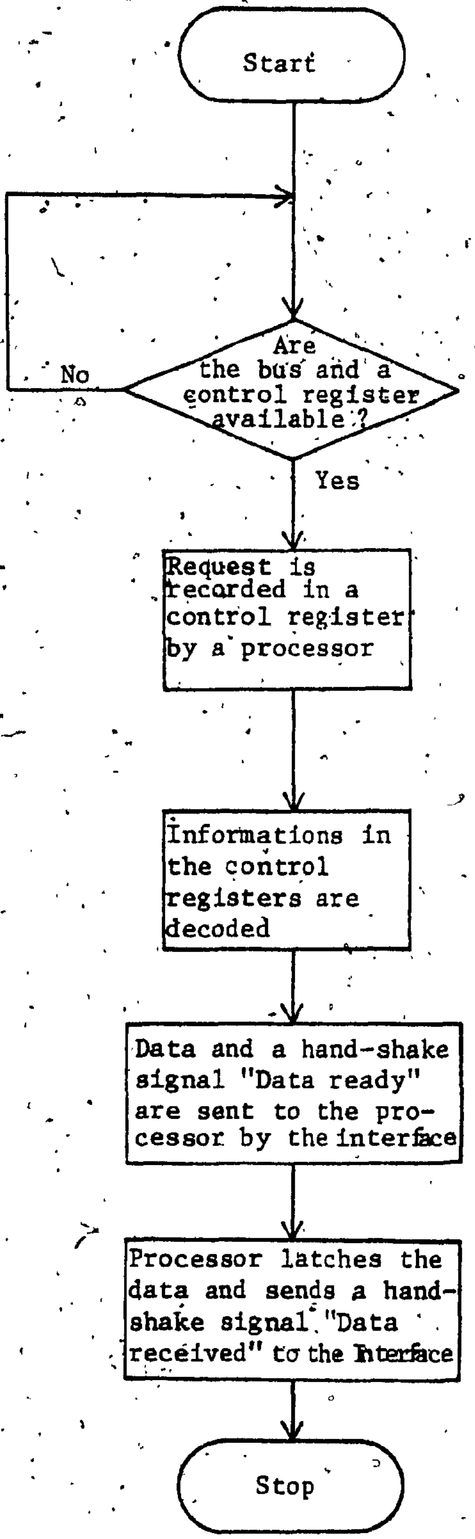

Start
Are the bus and a control register available ?
No
Yes
Request is recorded in a control register by a processor
Informations in the control registers are decoded
Data and a hand-shake signal "Data ready" are sent to the processor by the interface
Processor latches the data and sends a hand-shake signal "Data received" to the Interface
Stop

Figure 2.3 Sequence of Events for Data Transfer to a Processor by the Interface.

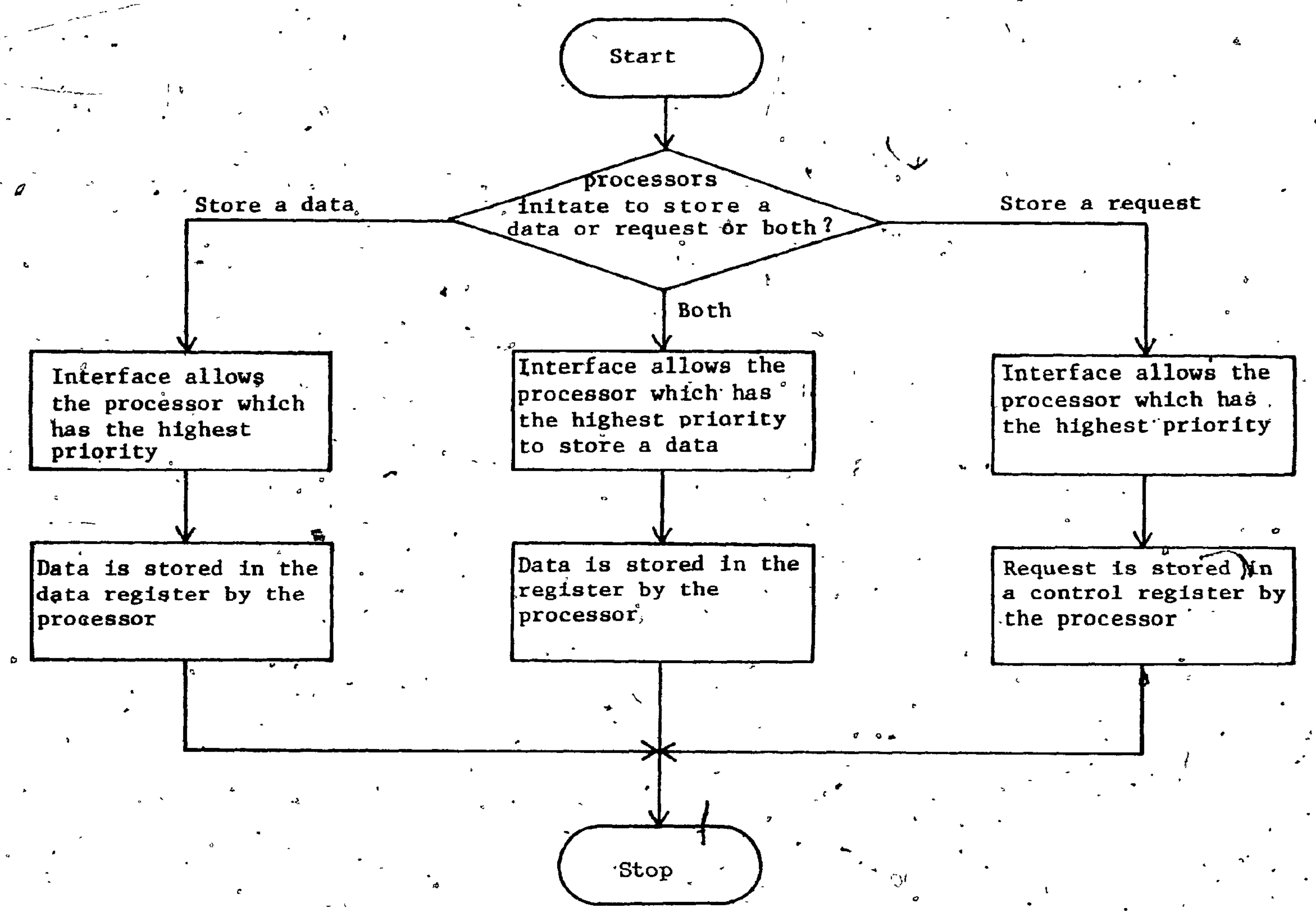

Start
processors initate to store a data or request or both?
Store a data
Both
Store a request
Interface allows the processor which has the highest priority
Interface allows the processor which has the highest priority to store a data
Interface allows the processor which has the highest priority
Data is stored in the data register by the processor
Data is stored in the register by the processor
Request is stored in a control register by the processor
Stop

Figure 2,4 Flow Chart for Determining which Processor is to Interact with the Interface.

2.3 SIMULATION MODEL

In most analytical models used to evaluate the performance of interleaved memories in multiprocessor systems, memory request arrivals are assumed to be independent of each other and rejected requests are discarded. But in pratice the above assumption cannot be justified. However, analytical models for practical cases without this assumption are very difficult to obtian [21]. Thus the performance evaluation in this thesis will be carried out through simulation. In order to proceed with computer simulation of the interface, a suitable model of the architecture would be required. Our objective in simulation are twofold.

1) To study the effect of storing of processors' requests in the control register.
2) To find optimum number of control registers in the interface.

A simulation model of the interface described in Section 2.2 is shown in Fig. 2.5, where circles represent read requests and triangles represent write requests from different processors. As discussed earlier write requests have higher priority than read requests. Squares represent server. There are three different servers. Circle or triangle inside a square signifies the request currently in service. The bus server, depending on the type of request, sends the request to the appropriate server. The dotted line indicates the rejected request path.

There are two kinds of requests issued by a processor —

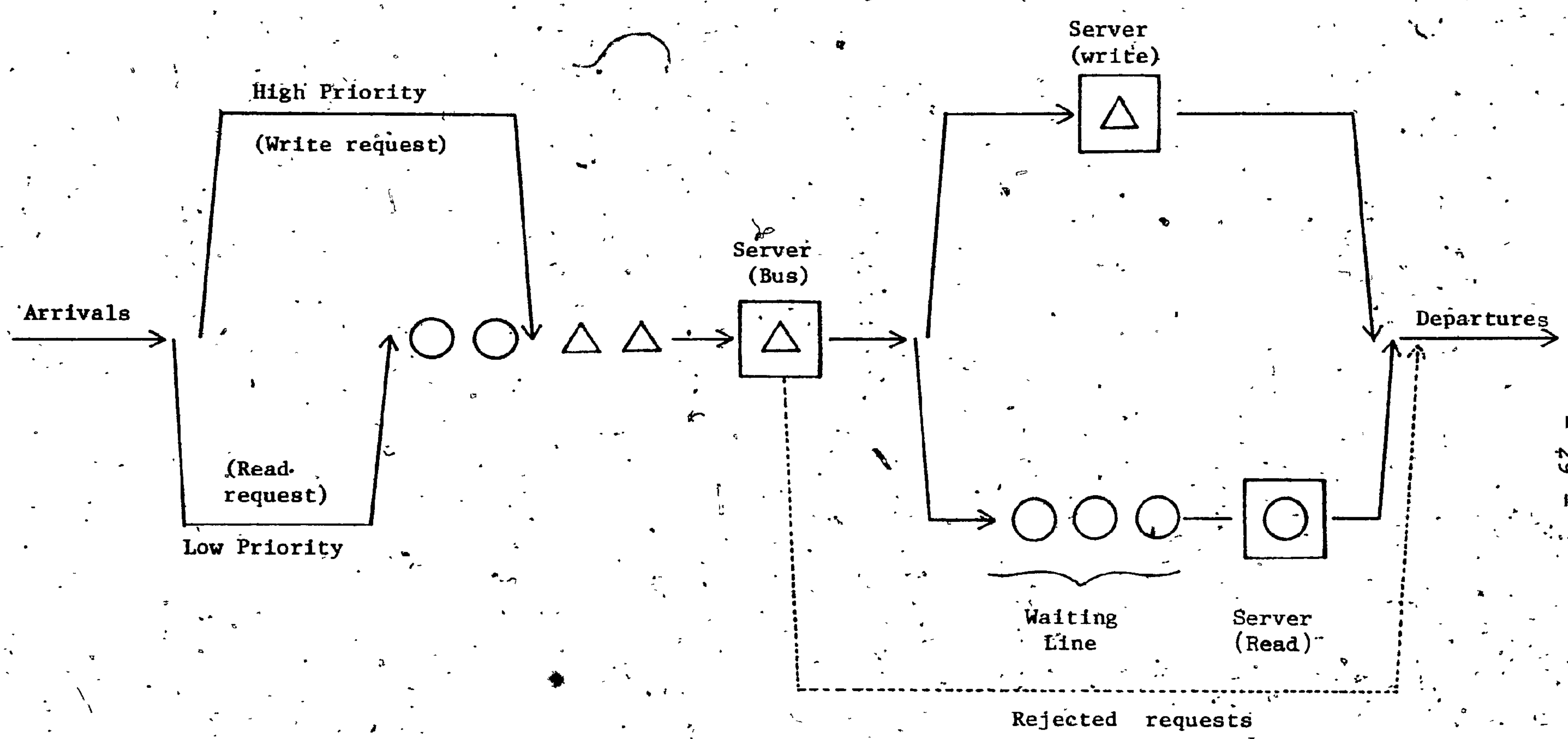


Figure 2.5 Multiple Server Queuing System with Priority Distinctions.

read and write. We will assume in the simulation model that the ratio of read request to write request is 7:3. The interface accepts one request at a time. When there are multiple requests from processors, one of the requests is accepted by the interface and others are rejected. Further we will assume that in the simulation model the rejected requests are discarded so that independence and randomness assumption can be justified. In practice, however, processors resubmit the request to the interface. Exactly one request is issued at a time by a processor and the processor whose request is being serviced must not issue a new request. In the case of simultaneous read and write requests from processors to the interface, the interface accepts the write request. Since we are interested mainly in the overall performance of the interface and not in the relative performance of one processor over another, the service discipline becomes irrelevent. However, for practical situations a simple priority scheme is devised in the design of the interface for service discipline. This scheme assigns priorities to different processors such that one processor takes priority over another.

An accepted write request takes one write memory cycle or write I/O cycle time unit to complete it's service in the interface. Subsequently, the accepted read requests are stored in control registers in the order of their arrival and will have to wait for service for a duration of time called waiting time. This waiting time depends on the total number of control registers in the interface. We will also assume the total number if control registers is four and each processor takes its own time units to test the condition to interact with interface as explained in the previous section.

2.4 SIMULATION USING GPSS

Once a simulation model has been developed, next we have to consider the feasibility of programming the simulation model and to establish a data structure that forms the system image. Therefore a general purpose simulation language with powerful statements and facilities would be desirable. General Purpose Simulation System (GPSS) is one such higher level language suitable for the simulation of the model that has been developed in Section 2.3. As a simulation programming language, GPSS V contains special features for reproducing the dynamic behaviour of systems which operate in time and in which changes of state occur at desirable points in time. It offers programming convenience, and at the same time serves as a vehicle for concept artrilation [22]. Now we will describe how the simulation described in the previous section can be expressed as a GPSS program.

The model to be simulated in GPSS is represented as a block diagram, shown in Fig. 2.6. Each processor's request

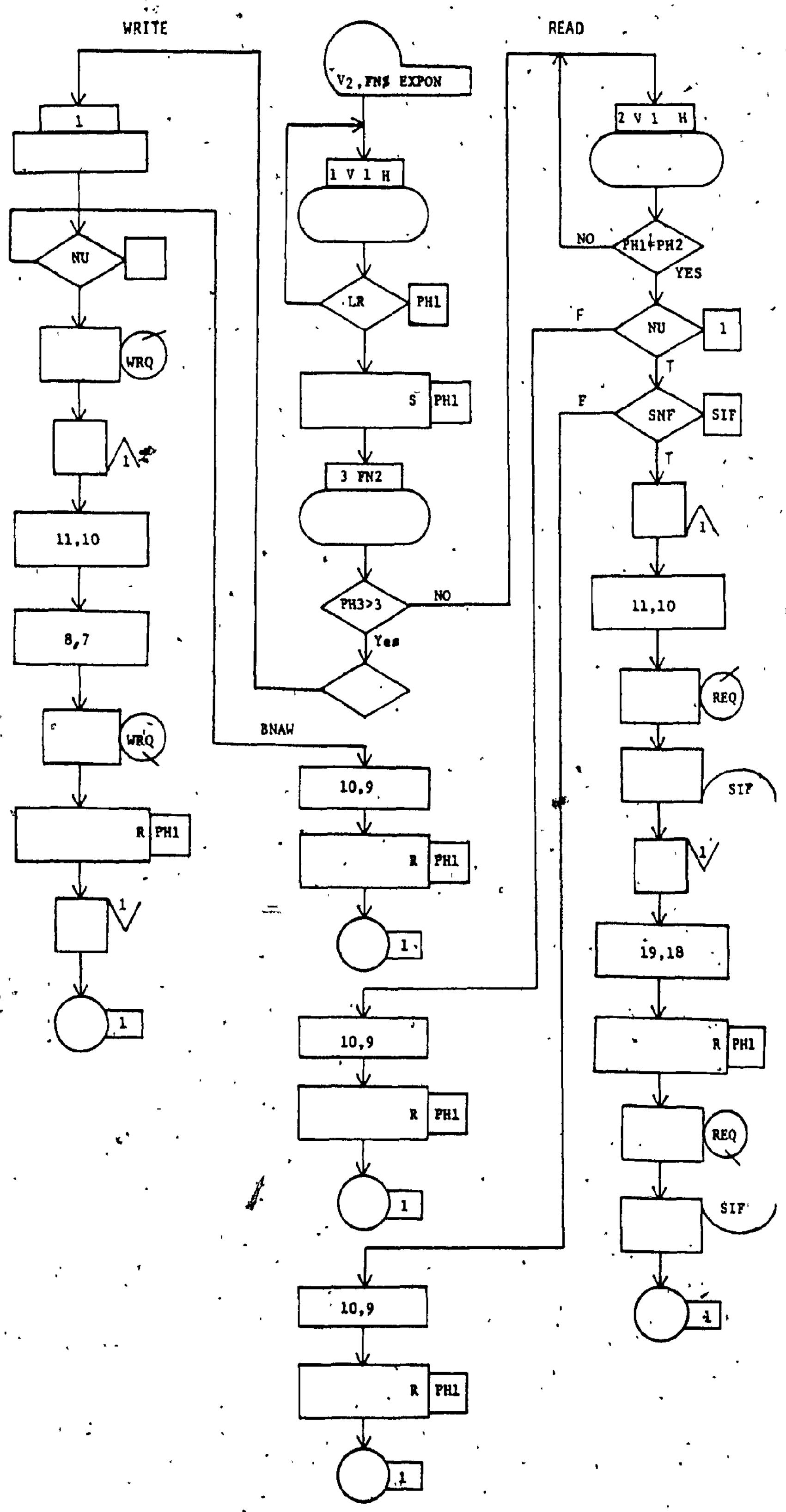


Figure 2.6 GPSS Block Diagram

is represented by one transaction. The unit of time chosen is one micro second. The mean inter arrival time varies with respect to total number of processors linked with the interface. Each new request can come from any of the processors which are not busy with interface, with equal probability and that its destination is equally likely to be any processor other than itself.

A GENERATE block is used to create a series of transactions by using the function EXPON and V_2 (VARIABLE 2). These functions are used to control the generation of transactions. Next the transaction is sent to ASSIGN block to select and record a processor. GATE block determines whether the selected processor has already sent a request to the interface or not. If the processor has not sent a request already, the transaction moves to the next block otherwise the transaction is sent back to ASSIGN block to select another processor. The logic block acts as a switch and sets the status of selected processor in ON mode (busy). Next it is decided whether the request is read or write. Source of the ASSIGN block, FN2, picks a number from function 2 which is defined as a discrete function. It sets the ratio of Read to Write request as 7:3. TEST and TRANSFER blocks are used to send the transaction to Read and Write group respectively, after establishing the request as being Read or Write. Once a transaction arrives at ASSIGN block of the Read group, it picks a destination processor. TEST block prevents the transaction from picking the same processor as the one the transaction already picked. The transaction is then sent to two GATE blocks where the availability of the bus and control register in the interface are checked. If they are not available the flow of transaction is branched to BNAR or SINA where it resets the status of the processor to OFF mode (not busy). Finally, the TERMINATE block removes the transaction from the simulation. The key variables used in the simulation program are listed in Table 2.1. Once the bus and a control register are available, the transaction is eligible to move to SEIZE block and make the bus busy. The transaction holds the bus with the help of ADVANCE block till it tests the status of the interface and places its read request in a control register. It has been assumed that the slowest processor in the multiprocessor system is Intel 8085A functioning with 0.5 MHz clock frequency and the fastest processor is MC 68000 with 10 MHz clock frequency. Execution times of Read, Write, Test and other instructions vary according to the speed of a processor. The RELEASE block makes the bus free. ENTER block keeps track of control register with the help of STORAGE statement. The second ADVANCE block makes the transaction to wait according to their order of arrival and total number of control registers in the interface. QUEUE and DEPART are used to gather statistics of the transactions and maximum entries in the QUEUE during simulation. After successful completion of data transfer to the processor, the transaction resets the status

TABLE 2.1 Variables Used in GPSS V Program for the Simulation of the Interface

VARIABLES	DESCRIPTION
OPRO	Origin processor.
DPRO	Destination processor.
BNAR	Bus not available to read the data.
SINA	Status of the interface (control register) is not available to read the data.
BNAW	Bus not available to write the data.
SIF	Status of the interface (control register).
REQ	Read queue which is used in read group to collect the statistics.
WRQ	Write queue, which is used in write group to collect the statistics.

of the processor in OFF mode and gets terminated from the simulation. The main difference between Read and Write group is that the Write group has a priority block. Once this transaction arrives to this block, it has got higher priority than the read request transaction. When there is a tie between read request transaction and write request transaction to enter into SEIZE block, the SEIZE block allows the write request transaction.

The GPSS V program listings and outputs are found in Appendices A and B.

2.5 PERFORMANCE RESULTS

Simulation program has been run for different sets of processors. A computer output is included in Appendices A and B. We observe that, for six processors, maximum queue length for read request is 2. Thereafter, even through the total number of control registers were set to four during the simulation run, the maximum queue length was three. It is because the bus gets saturated before the control registers are exhausted. Hence the optimum number of control registers in the interface is a function of total number of processors, the frequency of interprocessor data transfers required, and speed of the processors in the system. Fig. 2.7 and Fig. 2.8 are respectively, graphs of percent of total requests rejected versus number of processors and percent of read requests rejected versus number of processors. It is evident from the graphs that the total number of

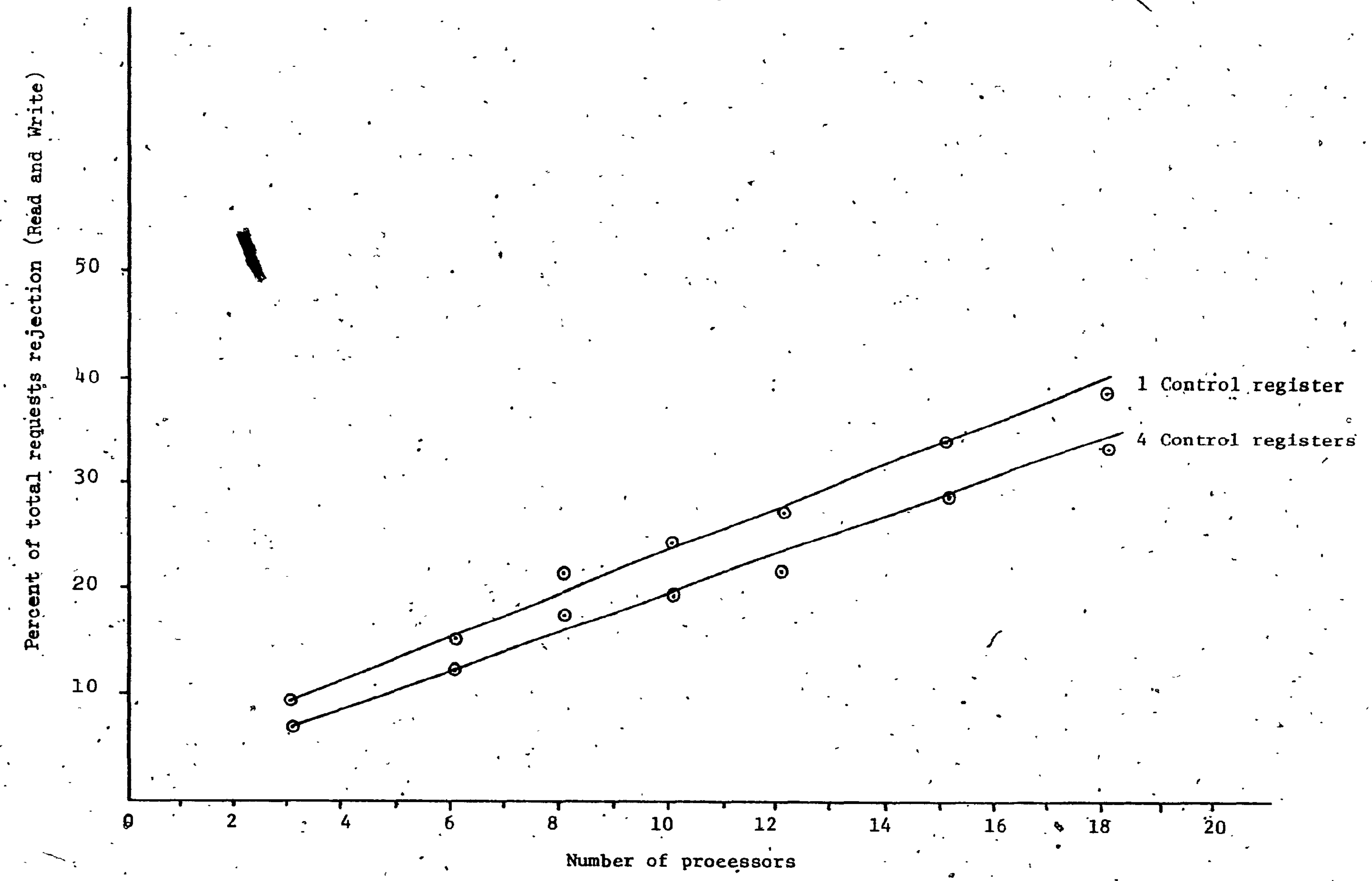


Figure 2.7 Percent of Request Rejection Versus Number of Processors

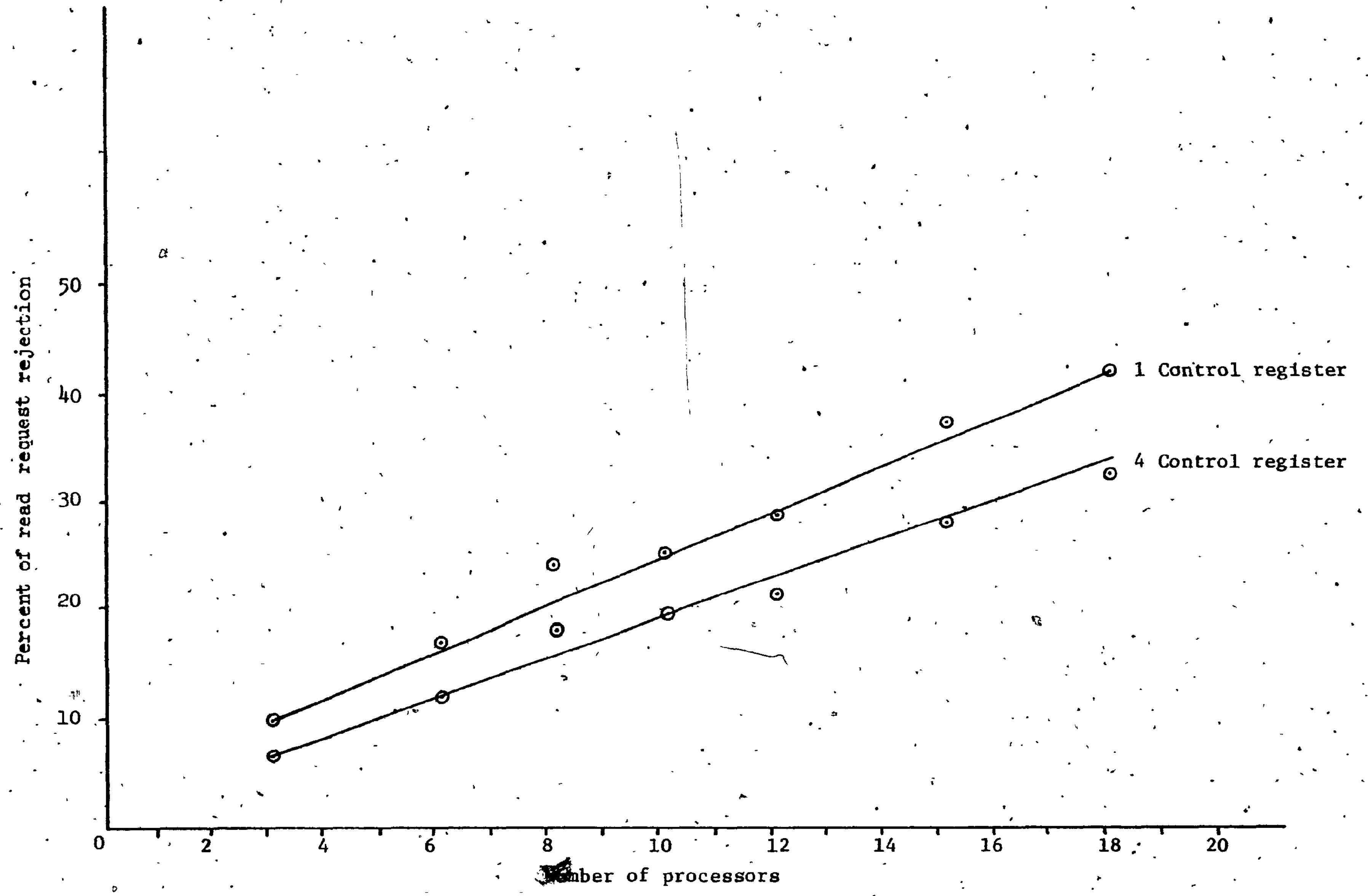


Figure 2.8 Percent of Read Request Rejection Versus Number of Processors

rejection of requests is increased as the number of processors increases. It is also clear that the rejection rate is more when the interface has one control register than when it has four control registers. The processors of the rejected request cannot make a retrial in less than an instruction cycle length. But the processors of the stored request does not have to make a retrial as it gets the data very quickly. Hence an increased number of control registers can be more effectively used to improve the total performance of a multiprocessor system. Fig. 2.9 is the graph of percent of rejected write request versus number of processors. From Fig. 2.9, it is seen that the use of an optimum number of control registers in the interface reduces the rejection of write request. It is because the stored requests don't have a chance to compete with write requests to access the interface. It is clear that the interface can handle up to seven processors effectively in this case. In the main module, if the total number of control registers exceeds the optimum number of control registers, then these extra registers and a few of the logic devices in the controller and handshake control logic will remain idle. To avoid this situation it is desirable to design the main module with optimum number of control registers. However, it is observed from the simulation results that, for a fixed number of control registers, the rejection rate increases linearly as the number of processors increases up to around 18. There after the rejection rate is expected to be increased linearly with a different slope.

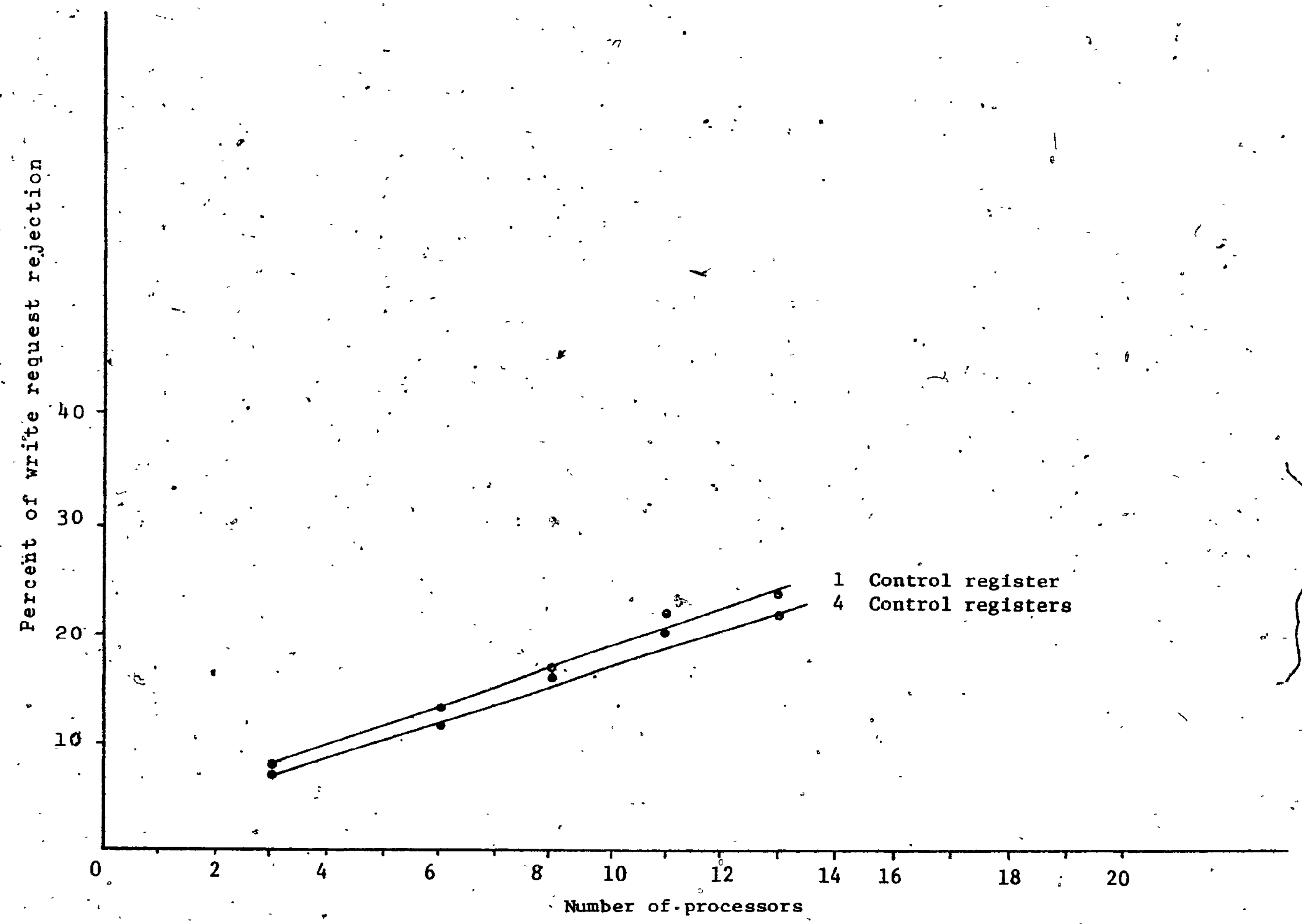


Figure 2.9 Percent of Write Request Rejection Versus Number of Processors

CHAPTER III

DESIGN AND IMPLEMENTATION

3.1 INTRODUCTION

Design specification of the interface would involve the analysis of the requirements and description of the modules in detail. In Chapter II the interface was divided into three logic modules. This chapter deals with the design and implementation of each of the modules.

3.2 CONTROL SIGNALS

The physical structure previously discussed provides the basis for communication between processors. In addition, a logical structure must be developed to allow meaningful communication. This structure consists of various signals. These signals allow two communicating entities to cooperate. Before one can discuss the design philosophy of each logic module, it is necessary to get familiar with different control signals generated by the interface and processors while interacting with each other. Interface issues the handshake signals H_1 (data accepted) and H_2 (request received), respectively, after accepting data from a processor and after storing the request for data by a processor. Interface sends a handshake signal H_3 (data ready) to the processor in order to indicate to the processor to latch the data on the data bus. A successful data transfer is acknowledged to the interface by the handshake signal H_4 (data received) from the processor. Description of various control signals are summarized in Table 3.1.

3.3 THE PEDC LOGIC MODULE

The main functions of the PEDC logic module are,

(i) to resolve the tie between the processors when they all simultaneously try to interact with the interface.

(ii) to indicate the processors about the current status of the control registers in the interface and the availability of the bus to the interface.

A state diagram describing the requirements of the PEDC logic module is shown in Fig. 3.1. It consists of a set of states with a designated initial state and a set of transitions among the states. Control passes from one state to another when external events stimulate the module. Associated with each of these transitions is an action which the module must perform. Description of the state diagram is given in Appendix C.

Fig. 3.2 delineates the block diagram of the PEDC logic module. It consists of three pairs of priority encoder-decoder devices. We will call one pair of priority encoder-decoder device as Read PED and another pair as

TABLE 3.1 Signals Generated During the Interaction Between a Processor and the Interface.

	Signals	Description
H_1	Data accepted	Interface issues this handshake signal after accepting data from a processor.
H_2	Request received	After storing the request for data by a processor, interface issues this handshake signal.
H_3	Data ready	Interface indicates the processor by using this handshake signal to latch the data on the data bus.
H_4	Data received	A successful data transfer is acknowledged to the interface by this handshake signal from the processor.
SI	Status of the interface	This signal signifies the availability of a control register.

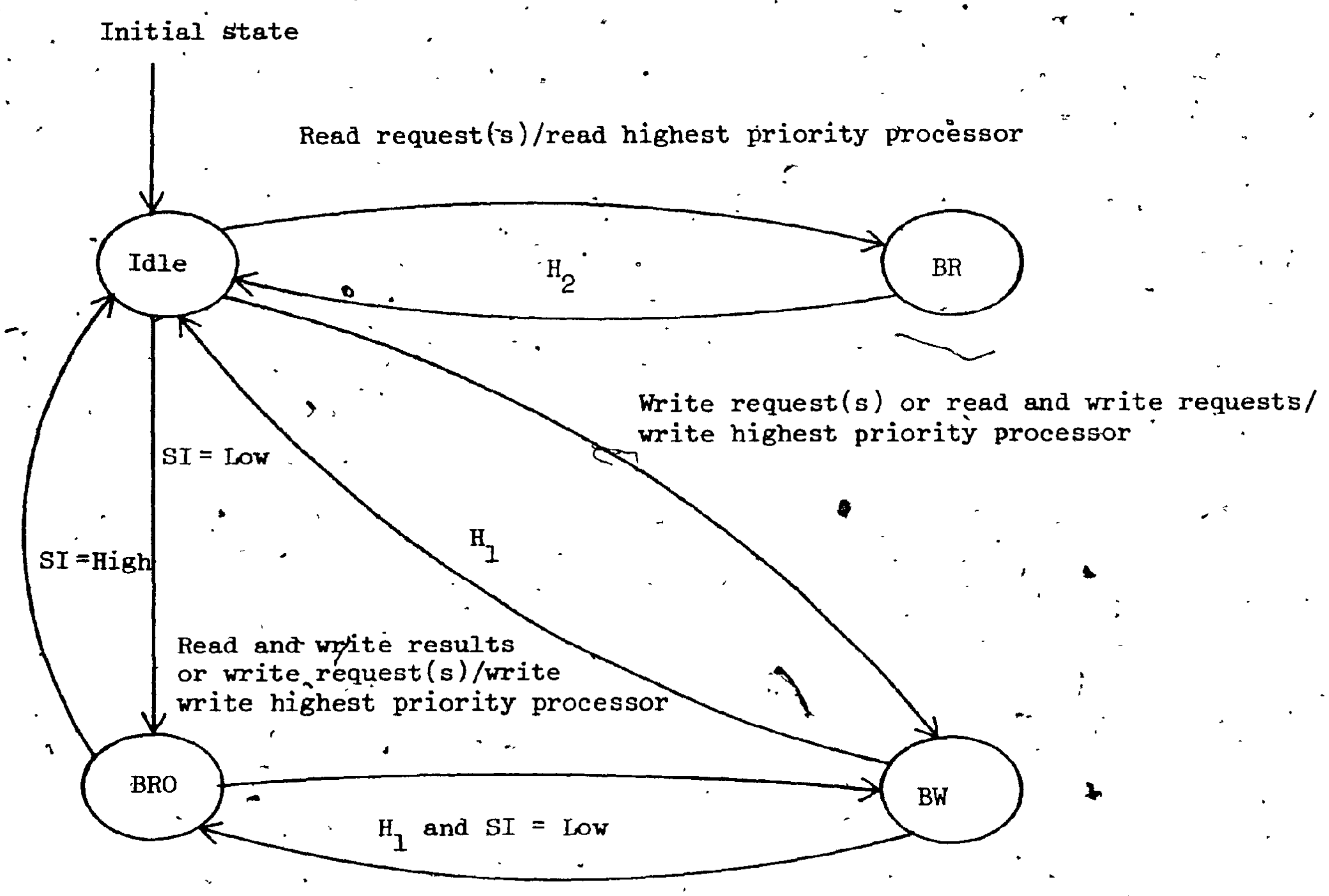
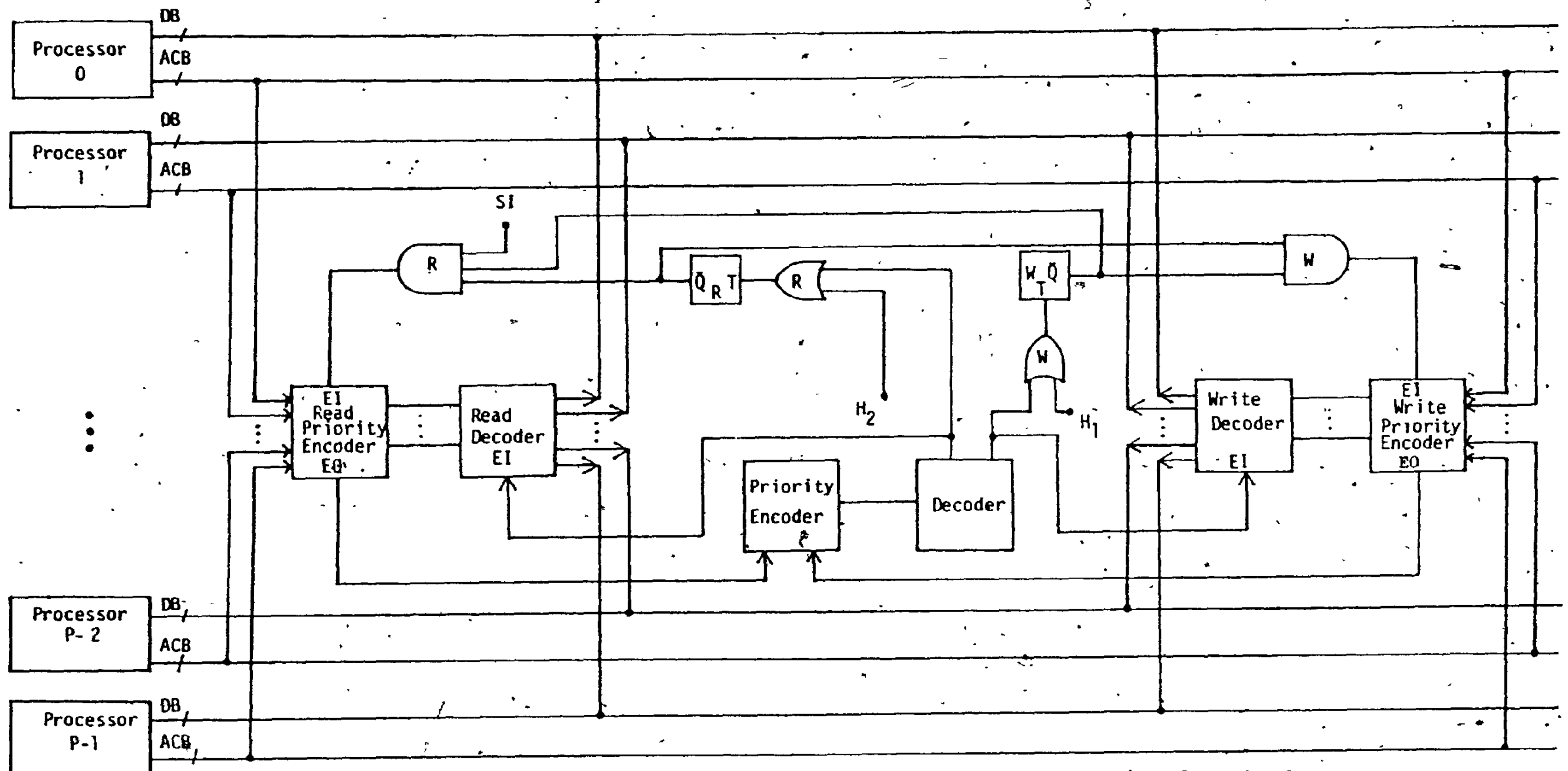

Initial state
Read request(s)/read highest priority processor
Idle
H2
BR
Write request(s) or read and write requests/
write highest priority processor
SI = Low
H1
SI =High
Read and write results
or write request(s)/write
write highest priority processor
BRO
BW
H1 and SI = Low

Figure 3.1 PEDC Logic Module State Diagram



DB - Data bus
ACB - Address and control bus
T - T-flip-Flop

EI - Enable input
EO - Enable output

SI - Status of the interface signal
H_1, H_2 - Handshake signals

Figure 3.2 Priority Encoder Decoder Control Logic Module

Write PED, dedicated for read request and write request, respectively. The third pair, which is accessible to both read and write requests is called Common PED. Read PED and Write PED, each has a set of logic devices consisting of an OR gate and an AND gate and a T-flip-flop.

Priority encoder assigns a priority among the processors. Enable input (EI) terminal of each priority encoder of this logic module is initially set to be active (high). Logic level of the EI terminal of read priority encoder depends on the availability of a control register which is signified by SI line from the main module, and also on the availability of the bus to the interface which is reflected by state of T-flip-flops with the help of Common PED. On the other hand, for write priority encoder, logic level of EI terminal depends only on the availability of bus. When the EI terminal of the decoders is not active (low), all the output terminals will be disabled. When none of the decoders input lines is active, the enable output (EO) will be disabled, even though the EI terminal is active.

When there are simultaneous read request from the processors, the output logic signal of the Read PED will correspond to active input with the highest priority. This signal indicates the processor to proceed further. Since this signal is available only for about 10n sec, it is necessary to latch this signal until the processor in question reads it. It can be done by using D-latches or monostable multivibrator. The ~~R~~T-flip-flop gets affected through the Common PED and the EI terminal of both encoders will be disabled. Once the request is stored in a control register, the main module sends a handshake signal H_2 (request received) and turns the affected T-flip-flop to its original state. The above is also true in the case of simultaneous write requests from the processors except that instead of R-T-flip-flop W-T-flip-flop gets affected. The main module sends back a handshake signal H_1 (data accepted) after accepting the data from the processor and turns the affected T-flip-flop to its original state. When there are simultaneous read requests and write requests from the processors, the Common PED and the Write PED allows the write request processor which has the highest priority at that time to proceed further.

3.4 THE MAIN MODULE

The primary functions of the main module are

(i) to generate signals as described in Table 3.1 to control and coordinate the various functions of the interface, and

(ii) to accept the data from one processor and to send the data to another processor simultaneously or to accept the request for data from one processor and to send the data to another processor simultaneously through the processor select logic module.

The main module consists of data registers, control registers, controller, and handshake control logic. First we will describe how a data is to be stored in a data register and then how a request is to be stored in a control register for execution. A state diagram describing the requirements of the main module is shown in Fig. 3.3. Description of the state diagram is given in Appendix C.

3.4.1 Write Request

There are P data register pairs for P processors, one dedicated to each processor. Registers in each pair are called primary data register (PDR) and secondary data register (SDR) as shown in Fig. 3.4. After checking the availability of the address bus to the interface, a processor through tri-state buffers sends the address of its data register and data to the interface. This process can take place even during the time the interface is transfering a data to a processor. Fig. 3.4 depicts the various logic devices involved during the write operation. The decoder which is enabled with the help of W signal of the processor, decodes the address and activates the strobe terminal of the processor's PDR. The decoder does not require all the address lines of the processor. A word is stored in the PDR as soon as its strobe terminal is active. At the same time its EO terminal becomes active which helps to acknowledge the successful receipt of the word to the processor through an OR gate. The stored data in the PDR is always available to its SDR. Generated output signal of the OR

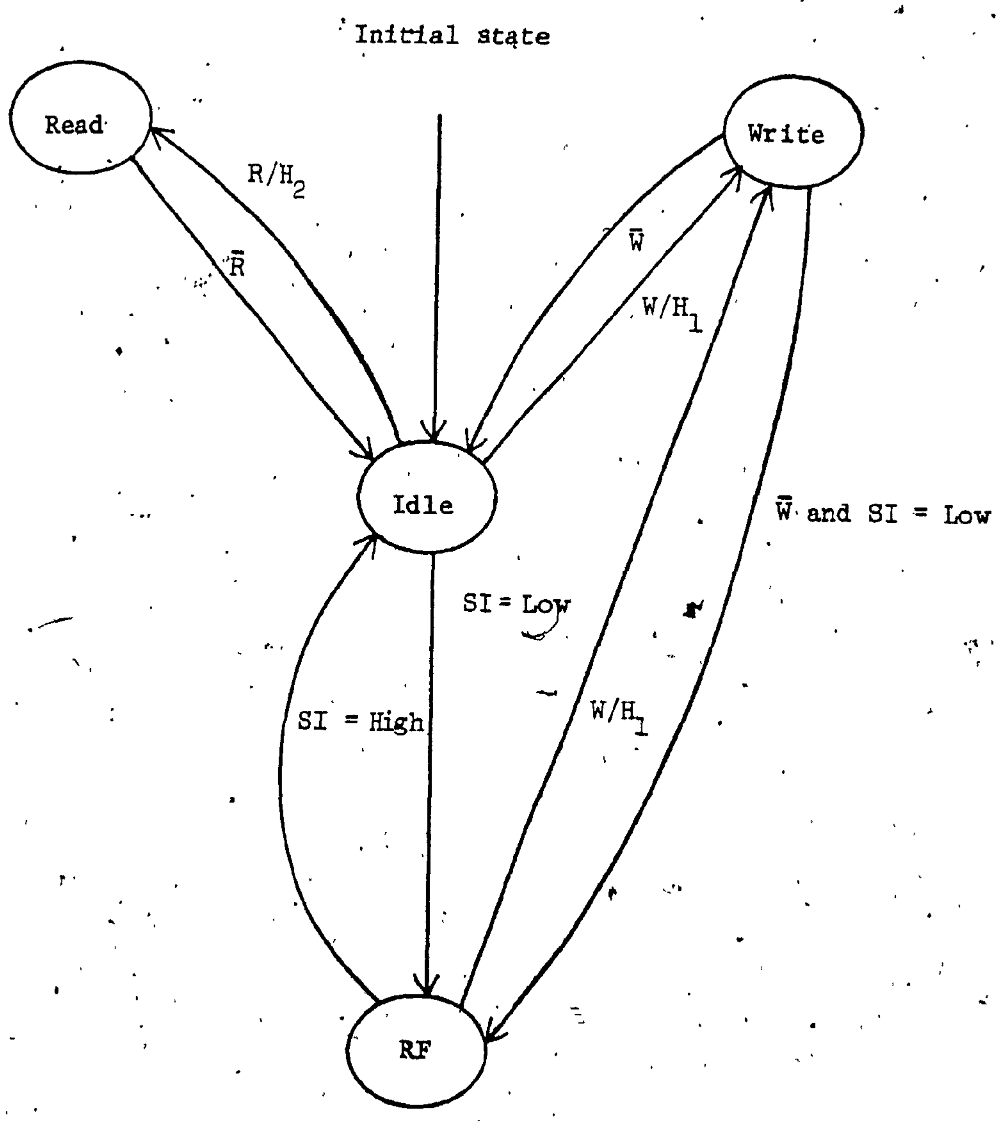
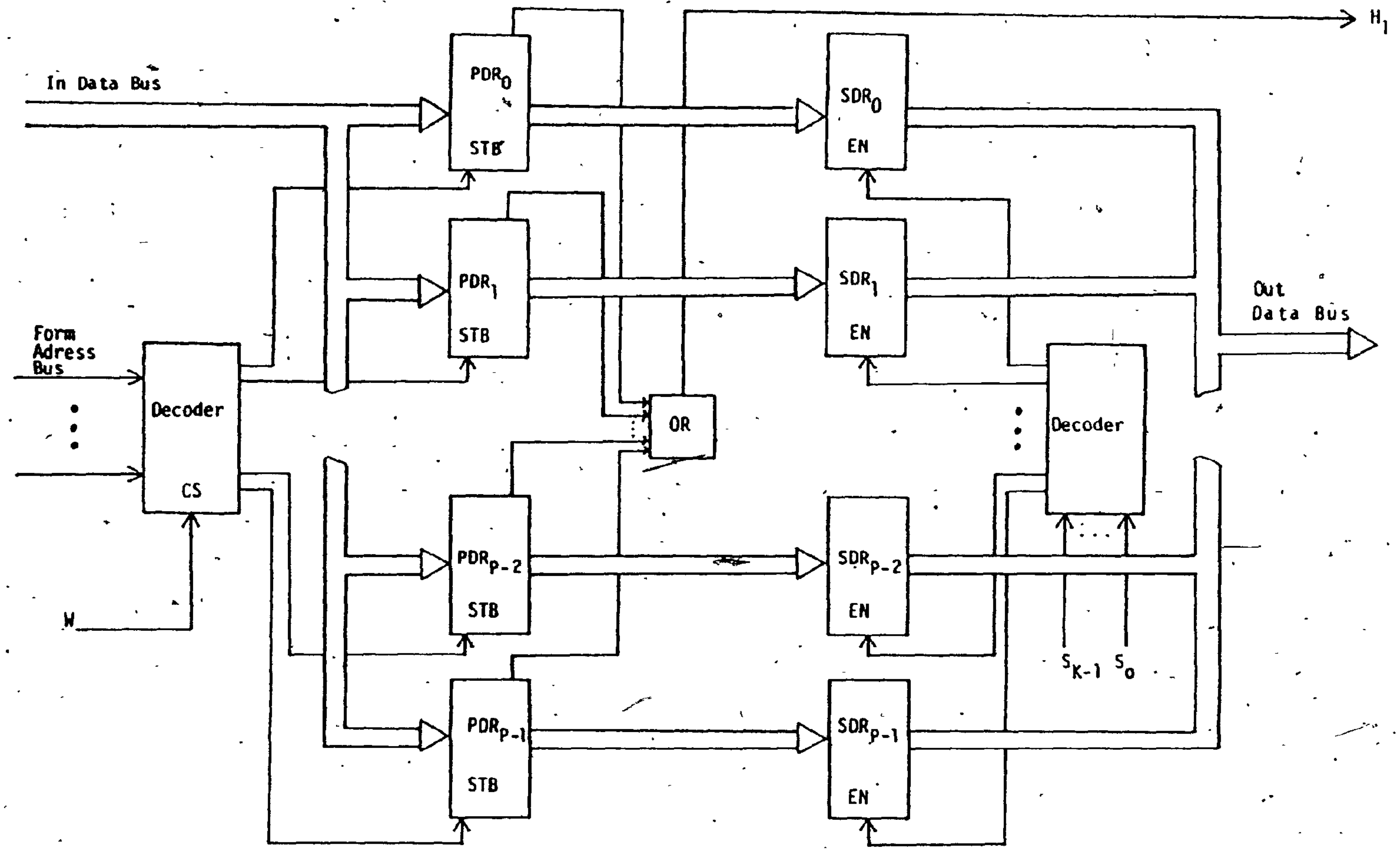


Figure 3.3 Main Module State Diagram



W - Write signal
CS - Chip select

PDR - Primary data register
STB - Strobe

SDR - Secondary data register
EN - Enable

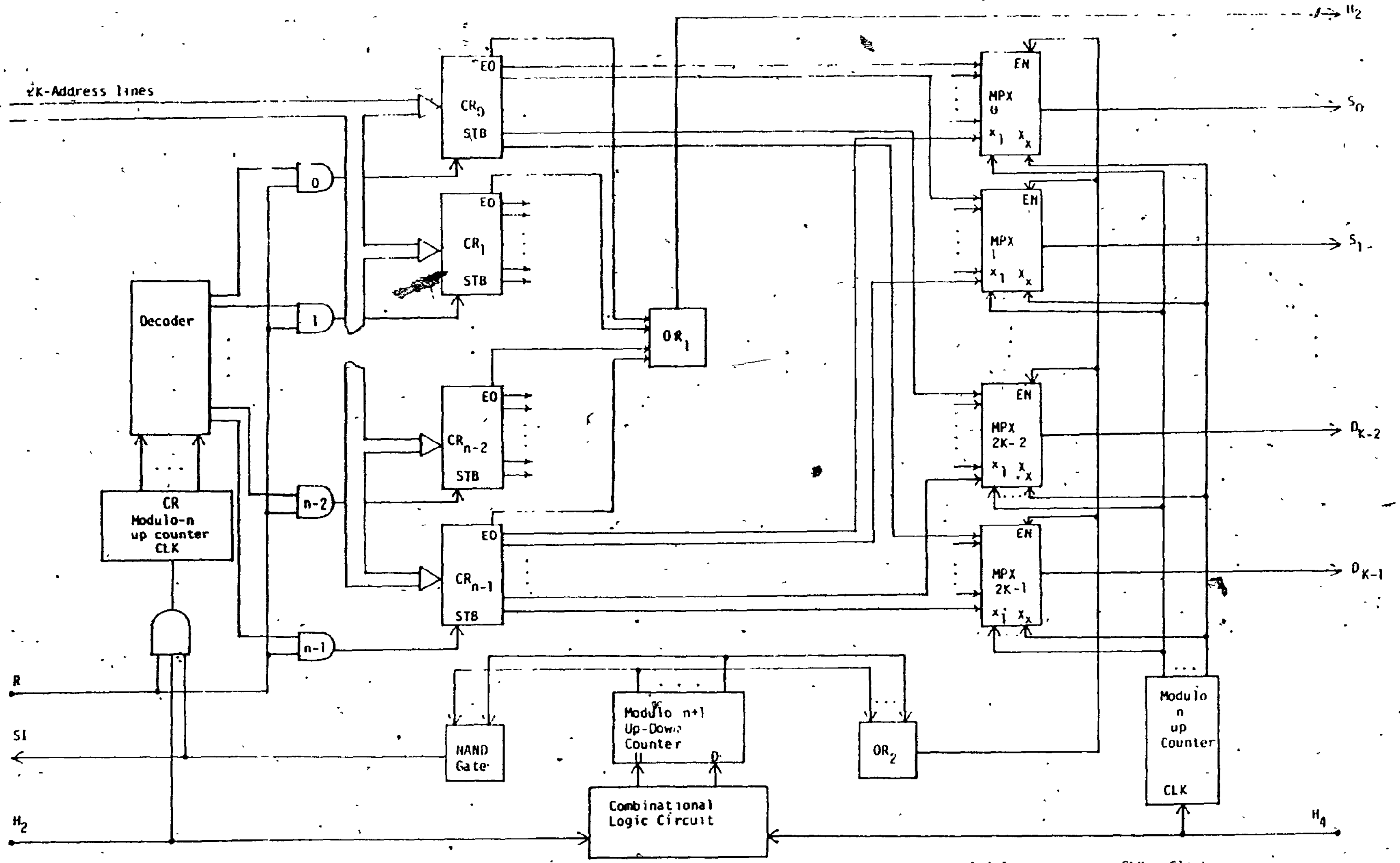
OR - OR gate
$S_{K-1} \cdots S_o$ Source bits
H_1 Handshake signal

Figure 3.4 Various Logic Devices Involved During Write and Execution of a Request

gate becomes handshake signal H_1 (data accepted). This handshake signal is available only for a duration of the order of 10 nsec, thus minimizing bus contention problem by removing the processors data bus, address bus, and control bus signals with the help of tri-state buffers and making the interface bus free for use by other processors.

3.4.2 Read Request

There are n control registers in the main module to store a request for data from a processor. Requests are stored in the control registers on a first-come first-served basis even during the transfer of a data to a processor by the interface. As discussed in Chapter II, optimum number of control registers depends upon the multiprocessing environment. Each control register contains 2K bits where $2^K \geq P$ (P = total number of processors). These bits are used to identify all the processors in the multiprocessor system. We will call the high-order K bits and the low-order K bits, respectively, as source bits and destination bits of a control register. Functions of these bits will be discussed in the next section. Fig. 3.5 delineates the various logic devices involved during a read request by a processor and during the excution of the request. The processor, after checking the availability of a control register and the bus to the interface, sends a request for data to the interface. This request is nothing but an address generated by the processor. Only 2K of the generated address line signals -



R - Read
SI - Status of the interface
H_2,H_4 - Handshake signals

CR - Control register
EO - Enable output
STB - Strobe

MPX - Multiplexer
S - Source bit
D - Destination bit

CLK - Clock
OR_1,OR_2 - OR Gates
U - UP clock, D-Down clock

Figure 3.5 Various Logic Devices Involved During Storing a Request and Execution of a Request.

need to be stored in the next available control register. Since there are n control registers, a modulo n up counter is used. We will call it as CR modulo n up counter, since it keeps track of control registers according to first-come first-serve basis through a decoder. One of the input of the AND gate of a control register is always made active by the decoder. Initially all the counters are set to be zero. For example as soon as the 2K address lines and R line are made active by a processor through the tri-state buffers, the top AND gate activates the strobe terminal CR_o. The request then is stored in the CR_o. The CR_o acknowledges the successful receipt of the request to the PEDC logic module and to other counters through an OR gate OR_1 with the help of its EO terminal as shown in Fig. 3.5. The output signal of this OR_1 gate is used as handshake signal H_2 (request received). This handshake signal helps to increase the CR modulo n up counter, and modulo n+1 up and down counter through the combinational logic circuit by one. At the same time the processor's bus signals are isolated from the interface with the help of tri-state buffers. Fig. 3.6 describes timing relations between various signals.

3.4.3 Execution of a Request

The controller coordinates and controls the various functions of the interface. It mainly consists of modulo n up counter, a modulo n+1 up and down counter, decoders and multiplexers as shown in Figs. 3.4 and 3.5. The state

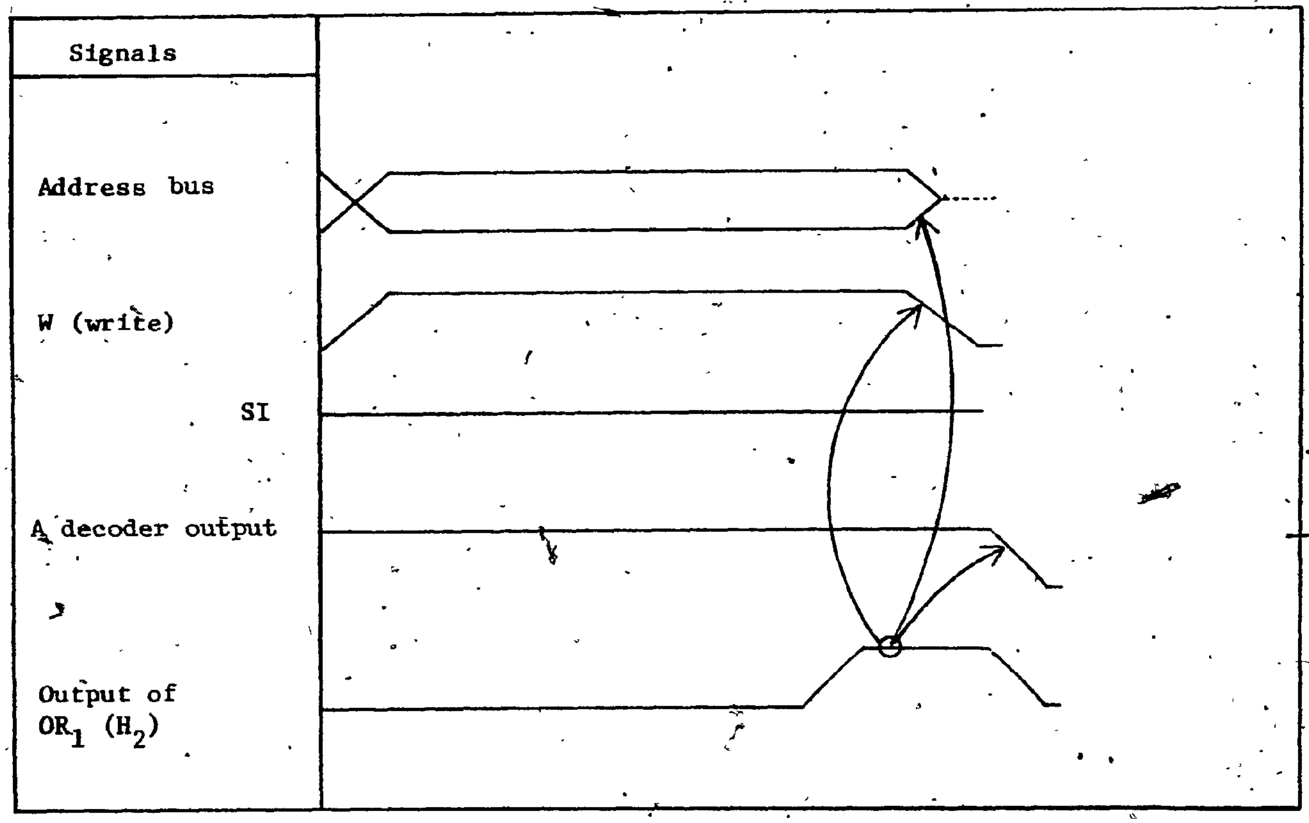


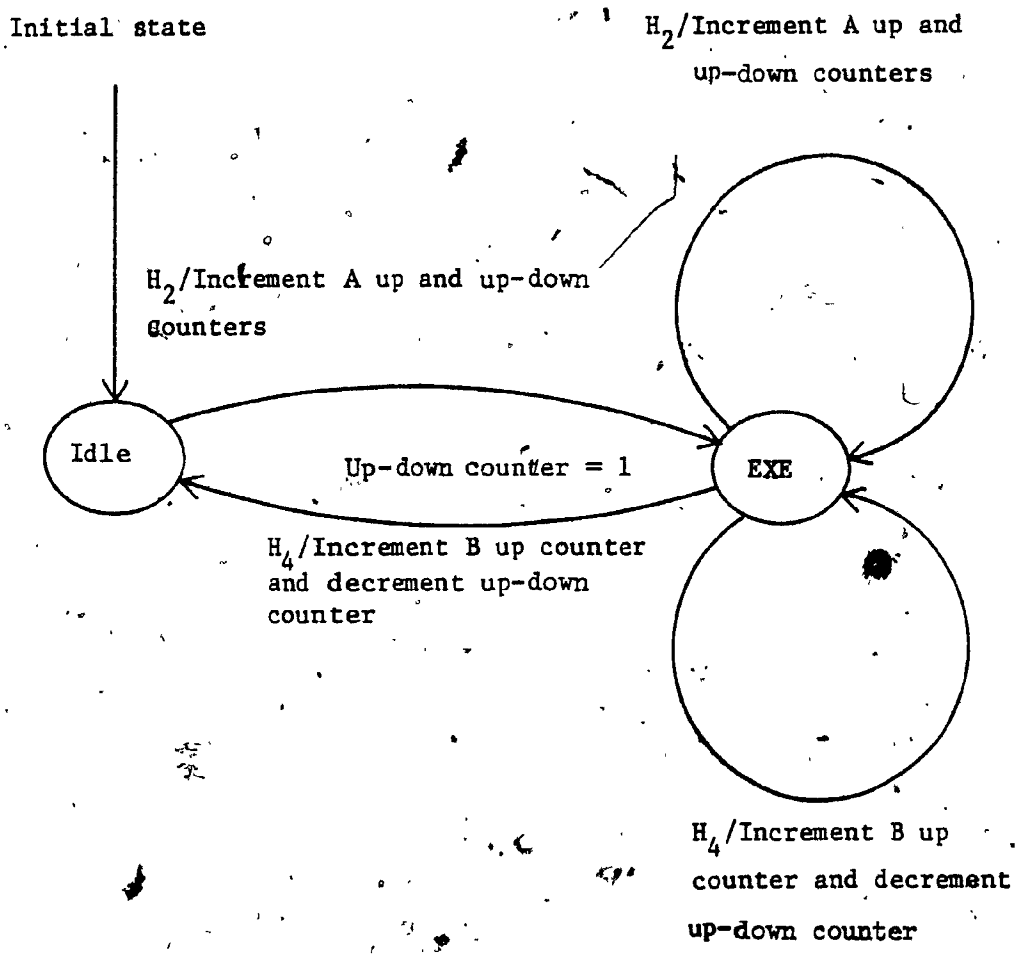
Figure 3.6 Timing Diagram

diagram of the controller is shown in Fig. 3.7. Description of the state diagram is given in Appendix C.

Each control register has 2K bits. So there are 2K multiplexers with n inputs and x select inputs, where $2^x \geq n$. A modulo n up counter is used to keep track of execution of requests one by one. Modulo n+1 up and down counter acts as a supervisor and gives informations about the current status of the module to various logic devices. When this counter's output is in BCD 0 , this means there is no more request waiting for service and all the multiplexers are disabled through the OR_2 logic gate. When all the CR are full, the count becomes n+1 which forces the SI line to be in low logic level through the NAND logic gate. The modulo n up counter and modulo n+1 up and down counter select a CR and output the contents of the CR through all the multiplexers. The source bits help to find wanted processor's data and destination bits are used to find the destination processor of the data. Source bits are decoded by a decoder as shown in Fig. 3.4 and the corresponding SDR is selected. The SDR latches the input signals and outputs it on the data bus.

3.5 THE PROCESSORS SELECT LOGIC MODULE

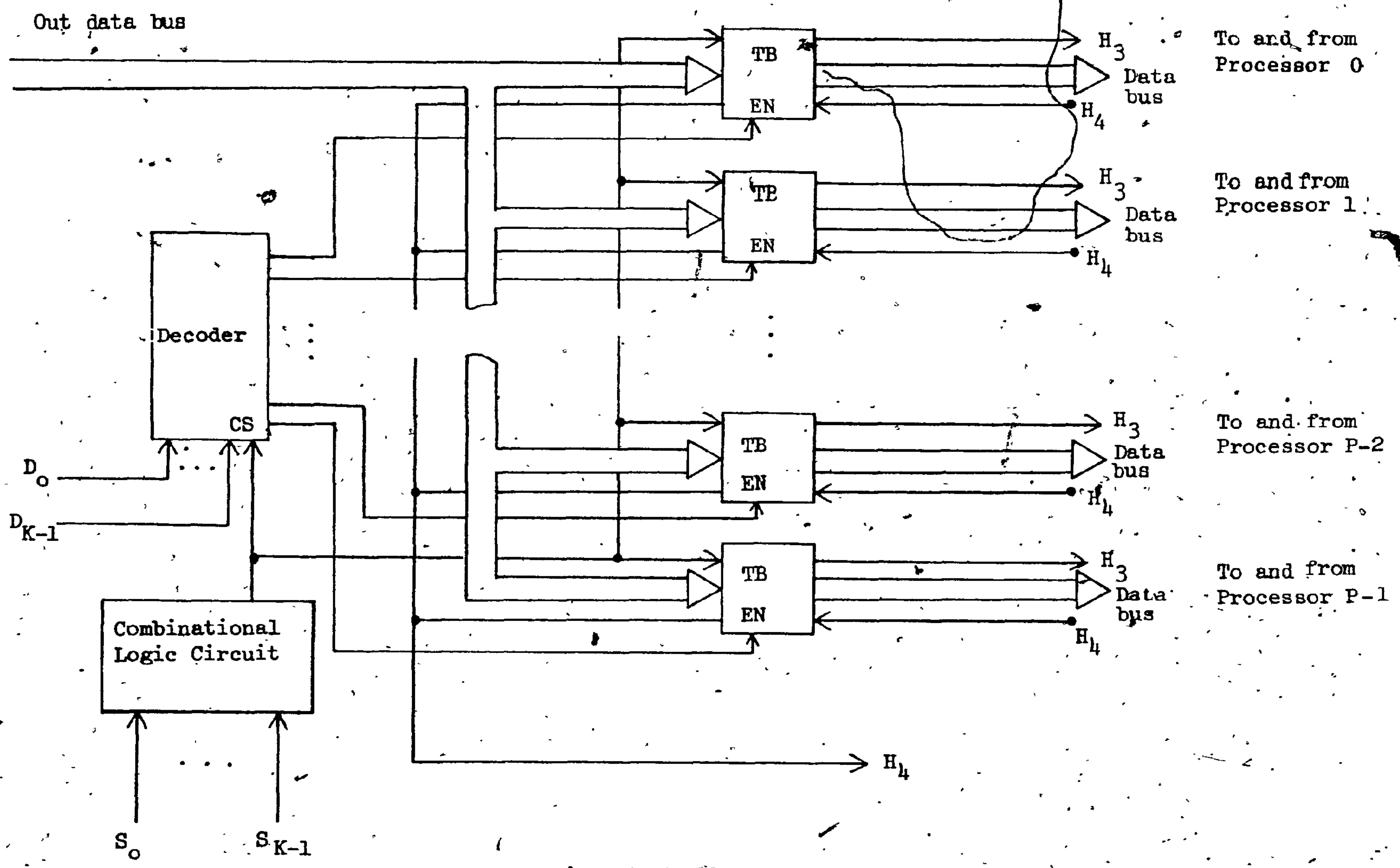
Processors select logic module mainly consists of a decoder and P sets of tri-state buffers or drivers with three states, each set dedicated to one processor as shown in Fig. 3.8. In Fig. 3.5, even though all the multiplexers are not enabled, their outputs are either in logic 1 or logic 0 state. Since a processor doesn't need the data from its


Initial state
H_2/Increment A up and up-down counters
H_2/Increment A up and up-down counters
Idle
Up-down counter = 1
EXE
H_4/Increment B up counter and decrement up-down counter
H_4/Increment B up counter and decrement up-down counter

A - CR modulo-n up counter

B - Modulo-n up counter

Figure 3.7 State Diagram of the Controller



S - Source bit
D - Destination bit
CS - Chip select

TB - Tri-state buffers
EN - Enable
H_3,H_4 - Handshake signals

Figure 3.8 Processor Select Logic Module

own data register, a combinational logic circuit is required to select the decoder only when the outputs of the multiplexers are combinations of low and high logic levels. The output signal of the combinational logic circuits is also used to generate the handshake signal H_3 (data ready). With the help of the destination bits, a set of tri-state buffers is selected through the decoder and the data and handshake signal H_3 are sent to the processor simultaneously. Upon the receipt of this handshake signal by the processor, processor latches the data and acknowledges the successful receipt to the interface by the handshake signal H_4 (data received). This handshake signal increments the modulo n up counter and decrements modulo n+1 up and down counter by one. In Fig. 3.5, the modulo n+1 up and down counter can simultaneously receive H_2 and H_4 handshake signals. In such a case, the combinational circuit allows none of the above signals to the counter. Truth table and the circuit of the combinational logic are shown in Fig. 3.9. When modulo n up counter is updated the interface starts to execute the next request which is stored in the next CR.

3.6 IMPLEMENTATION OF THE INTERFACE

The recent advances in semi-conductor technology have resulted in cheaper, faster, and easier-to-use digital components for a wide range of applications. After analyzing various trade offs for the construction of the interface, decision was made to implement each module by using TTL and MOS family ICs.

Truth Table

Inputs		Outputs	
H_2	H_4	U	D
0	0	0	0
0	1	0	1
1	0	1	0
1	1	0	0

$U = H_2 \bar{H}_4$

$D = \bar{H}_2 H_4$

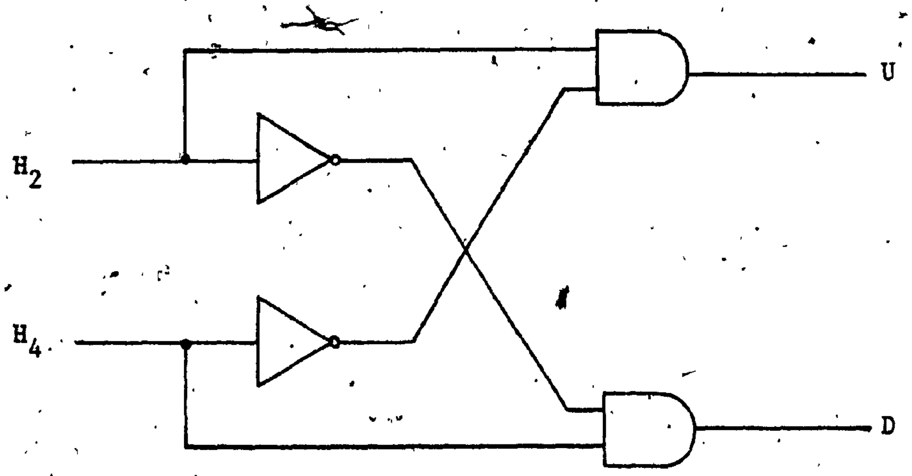
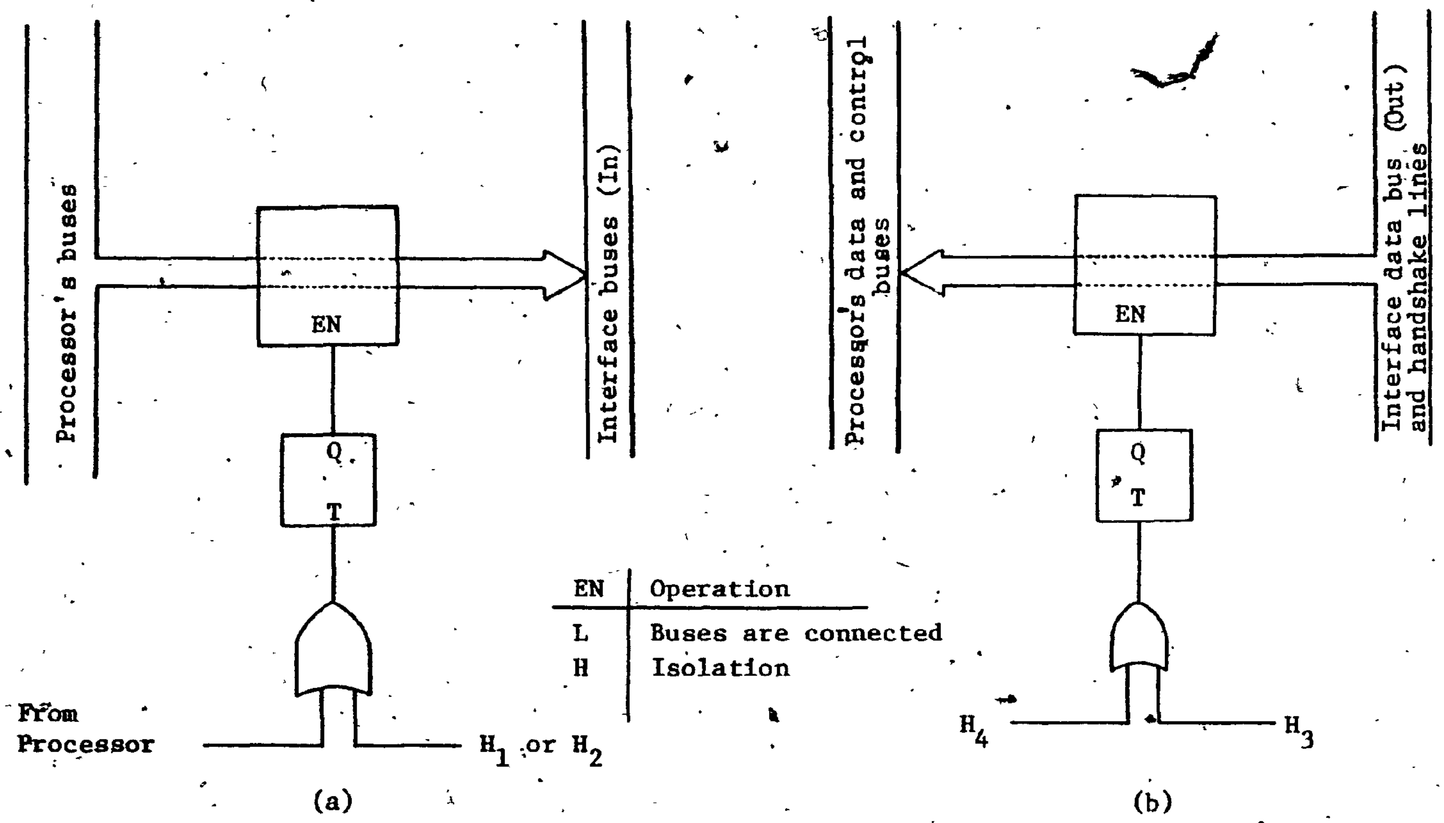


Figure 3.9 Combinational Logic Circuit



EN	Operation
L	Buses are connected
H	Isolation

H_1, H_2, H_3, H_4 - Handshake signals

T - T-flip-flop

L - Low EN-Enable

H - High

Figure 3.10 (a) Tri-State Buffers or Bus Receivers with Three States
(b) Tri-State Buffers or Bus Drives with Three States

TTL MSI priority encoders and decoders are available with short propagation delay times (in the range of 8 to 20 ns) which perform the same functions as encoders and decoders in the PEDC logic module described in the previous section. Cascading facility has also been provided to allow expansion without the need for external circuitry.

Each processor's address, data, and control buses are connected to the interface dynamically only when a processor requests for data or write a data through tri-state buffers or receivers with tri-state outputs. This device, in general, allows data transmission from processor to interface as shown in Fig. 3.10(a), or from interface to processor through tri state buffers or drivers with three states, as shown in Fig. 3.10(b). Enable input can be used to disable the device so that buses are effectively isolated with the help of T-flip-flop. In Fig. 3.10(a) processor enables the receivers or tri-state buffers while it sends the data or requests for data to the interface. The interface disables it after accepting the data or request. But in Fig. 3.10(b), interface enables the drivers or tri state buffers while it sends the data to the processor and then processor disables it after latching the data. Programmable counters, where the outputs may be preset to either level, are also available. These counters are easily cascadable for expansion without requiring external circuitry. MOS technology and TTL compatable parallel-in, parallel-out data registers which do the same functions as data register and control register as described in the previous section, are available

Figs. 3.11 and 3.12 show the photographs of the built interface connected with 2 Intel 8085-A microcomputer (SDK-85)

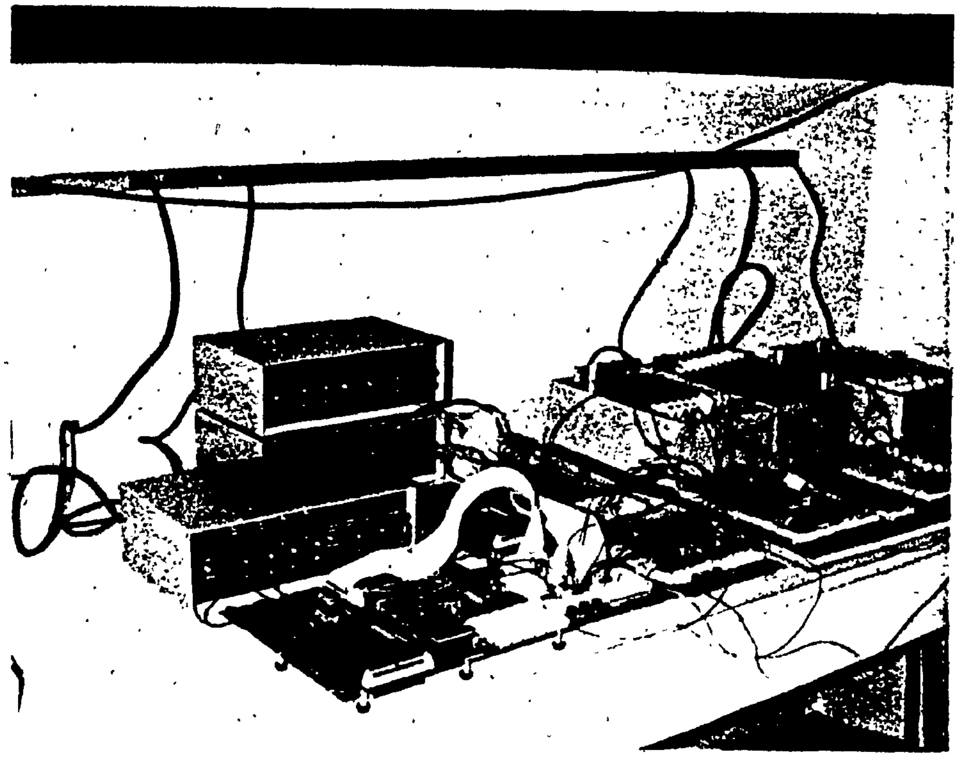


Figure 3.11 Front View of the Interface with Two Intel 8085 (SDK-85)

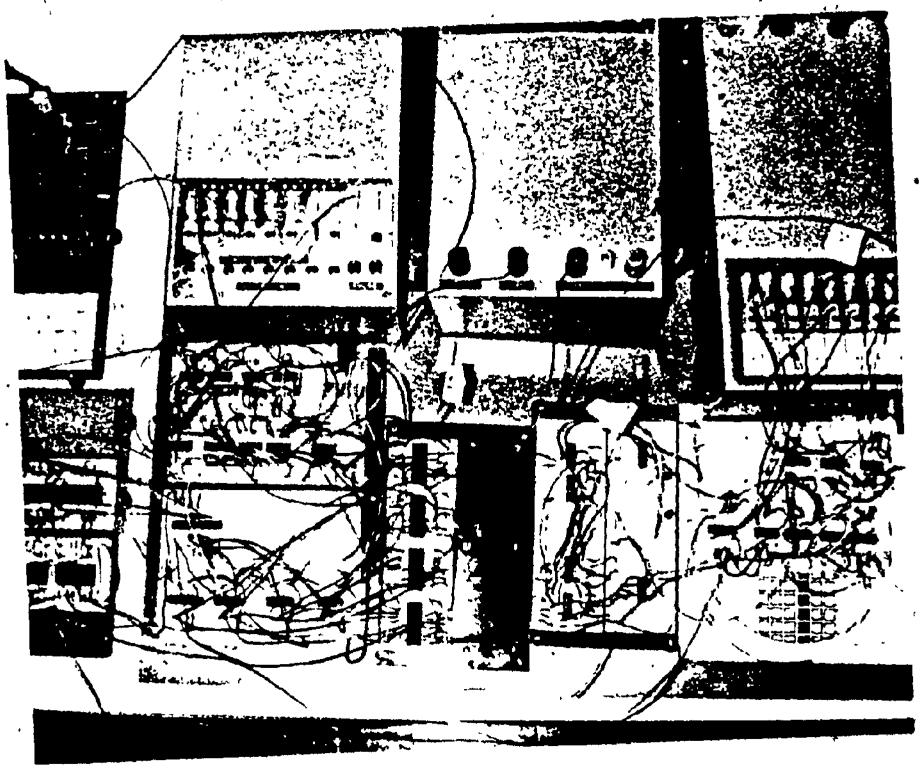


Figure 3.12 Top View of the Interface

CHAPTER IV

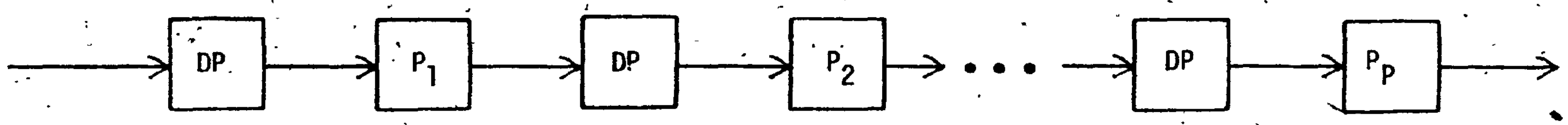
APPLICATIONS AND EXTENSIONS

4.1 INTRODUCTION

Multiprocessing offers attractive computational gains along the dimensions of reliability, flexibility, cost/performance etc. This chapter discusses the application of the interface in the area of digital signal processing and distributed processing. Possible extensions of the research work undertaken in this thesis are also explored.

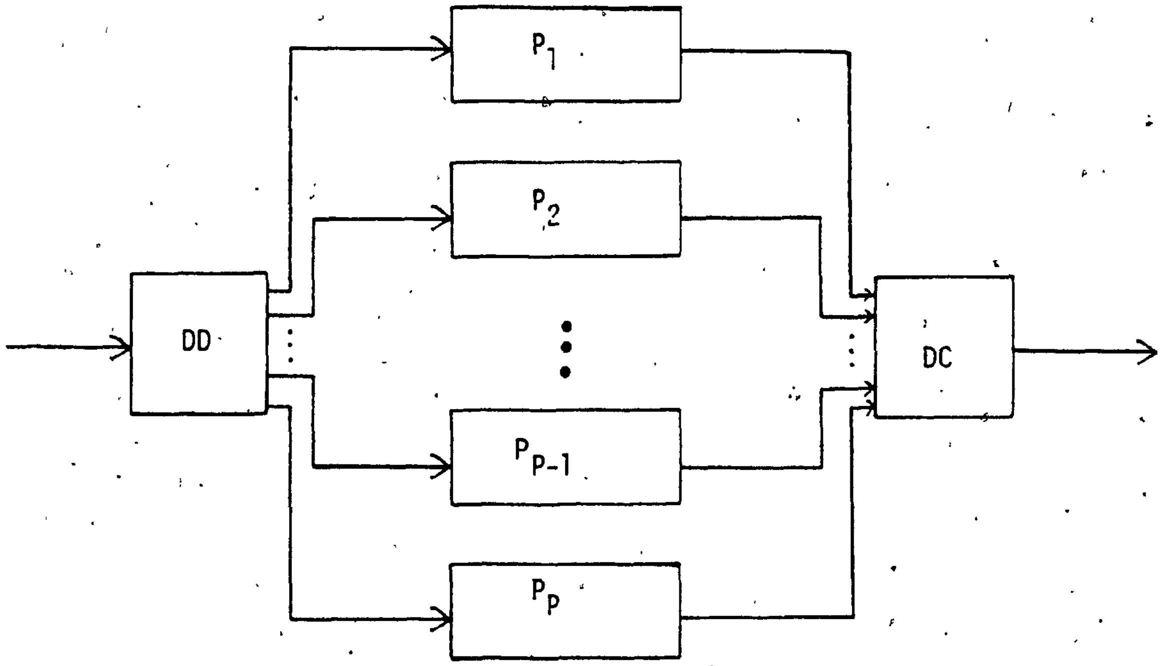
4.2 APPLICATION OF THE INTERFACE IN DIGITAL SIGNAL PROCESSING

Capabilities of multiprocessing systems are useful in signal processing system design where improved performance is mandatory. In [23] it has been shown that the cascade and parallel cluster structures could be used to implement the DFT, and FIR and IIR filters. Different types of algorithms for computing DFT and the design of different types of filters will not be discussed here as it is beyond the scope of this thesis. Figs. 4.1 and 4.2 show the cascade processor cluster and parallel processor cluster, respectively, for implementing FFT. For an N-point FFT, each processor in the cascade cluster structure performs a given piece of the FFT in some fixed sequence and outputs an N-point array. It is the responsibility of the data passer (DP) to collect the data from



DP - Data passer P - Processor

Figure 4.1 Cascade Processor Cluster



DD - Data distributor DC - Data collector
P - Processor

Figure 4.2 Parallel Processor Cluster

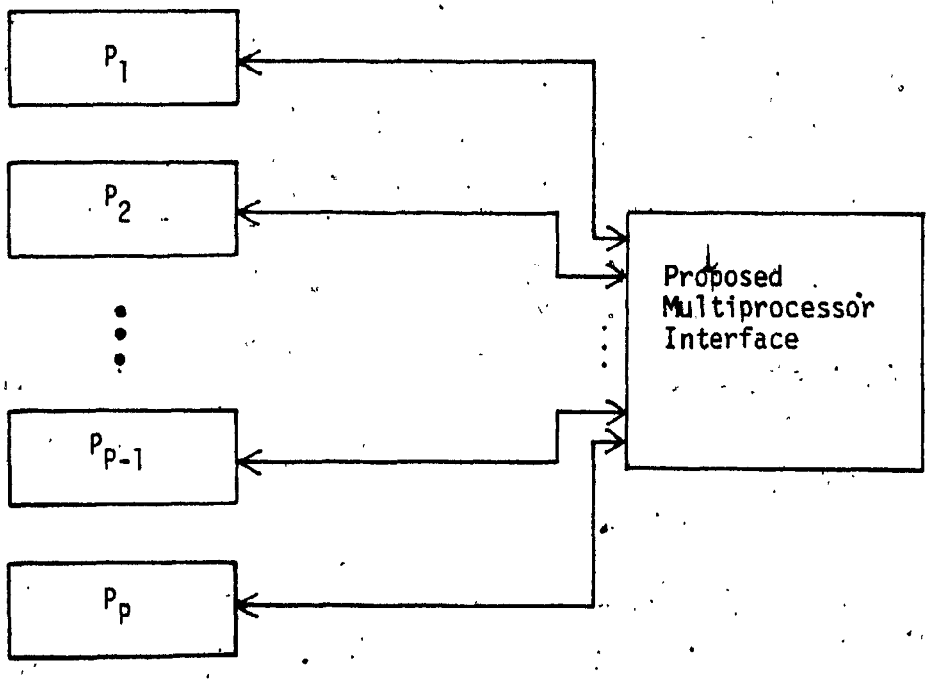


Figure 4.3 Implementation of FFT

the processor preceding it and to pass the data to the processor succeeding it in cascade. As noted in [23], since each processor in the cascade structure must have a DP, the cost of providing even simple external DPs may be high. Also, the data corresponding to the transform of each stage must be input and output to the respective processors, no matter how small is the processing load of that processor. Each stage in the structure requires 5% of total time to input and output the data. In the parallel processor cluster structure, data distributor (DD) transfers the data to a processor in some fixed sequence. After completion of the transform, the output data is collected by the data collector (DC) and reformated as necessary. DD and DC have complicated job, since they must perform reordering of data to dispatch input points to the processors.

Fig. 4.3 shows a processor cluster structure used to implement FFT. Without altering this structure it is possible to make the structure to act as a cascade processor cluster structure or parallel processor cluster structure. The communication between the processors is established through the interface. When the structure has to behave as a cascade structure, processor P_i will send the data to its data register in the interface for its successor P_{i+1}. This successor processor can make a request for this data immediately after the interface accepts the data from the predecessor processor. Thus the interface increases the speed of the data transfer. In

Fig. 4.1 a successor can make the request for data only after its predecessor completes its write cycle. When the structure behaves like the one shown in Fig. 4.2, each processor sends the request for data one after another in some fixed sequence and sends the data to its data register. As discussed above these data are available for next processing even before the processors write cycle is completed.

4.3 APPLICATION OF THE INTERFACE IN DISTRIBUTED PROCESSING

Another application of the interface is in distributed computing system as shown in Fig. 4.4. Distributed computing can be considered as the physical separation of intelligence of the system. Any processor in the system can communicate with another processor on first-come first-served basis as described earlier. In case processor P_i wants to know whether a new data has arrived to the interface from processor P_j or not, a semaphore technique can be used as illustrated in Fig. 4.5. When the semaphore is at logic 0, P_j sends the data for P_i to the interface and sets the semaphore to logic 1. P_i now can take the data and clear the semaphore to logic 0.

In [24] a distributed processing system (DPS) which is geographically separated, functionally organized, task oriented, and connected on an efficient high speed serial data bus, as shown in Fig. 4.6 has been described. The purpose of this DPS is to satisfy military requirements.

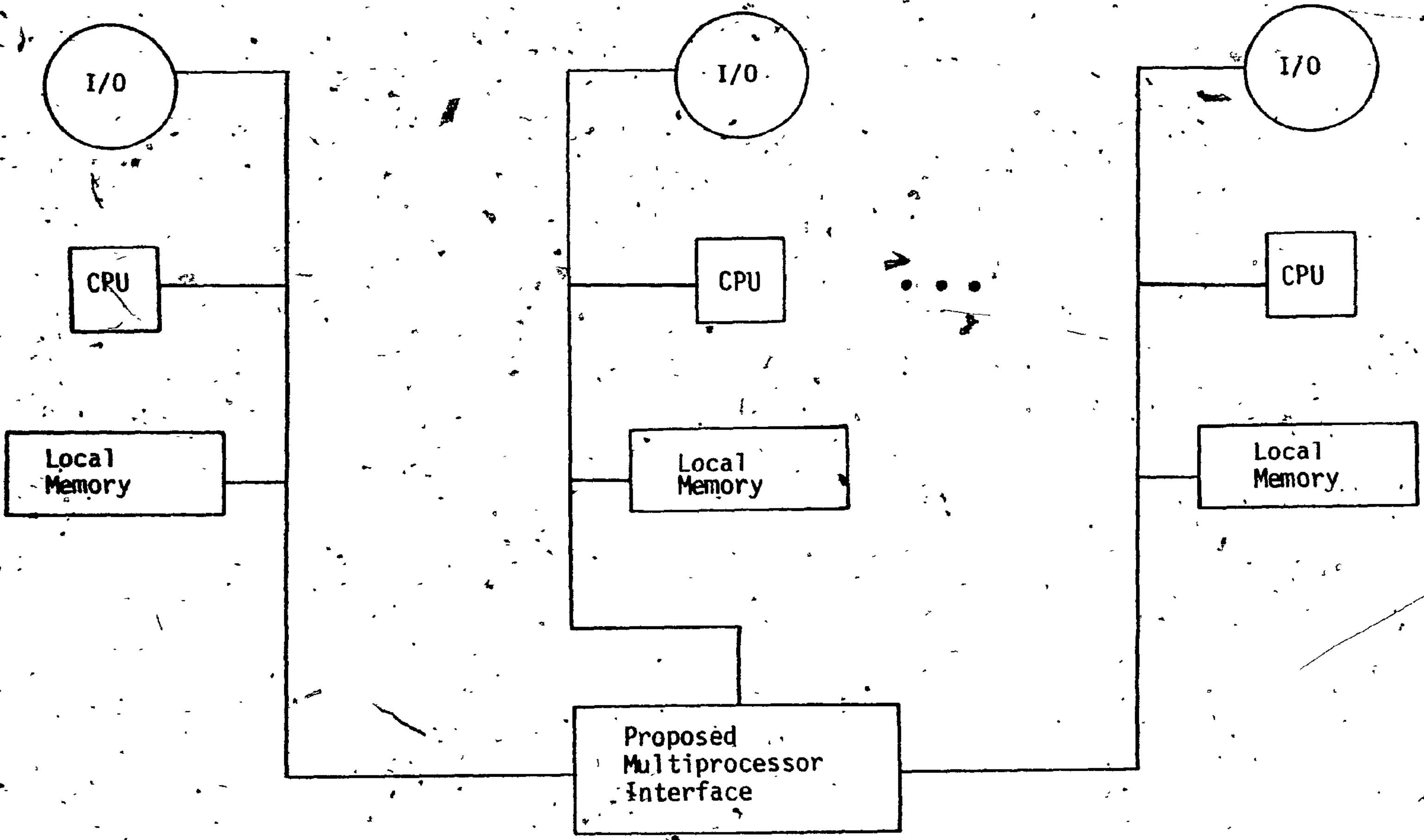
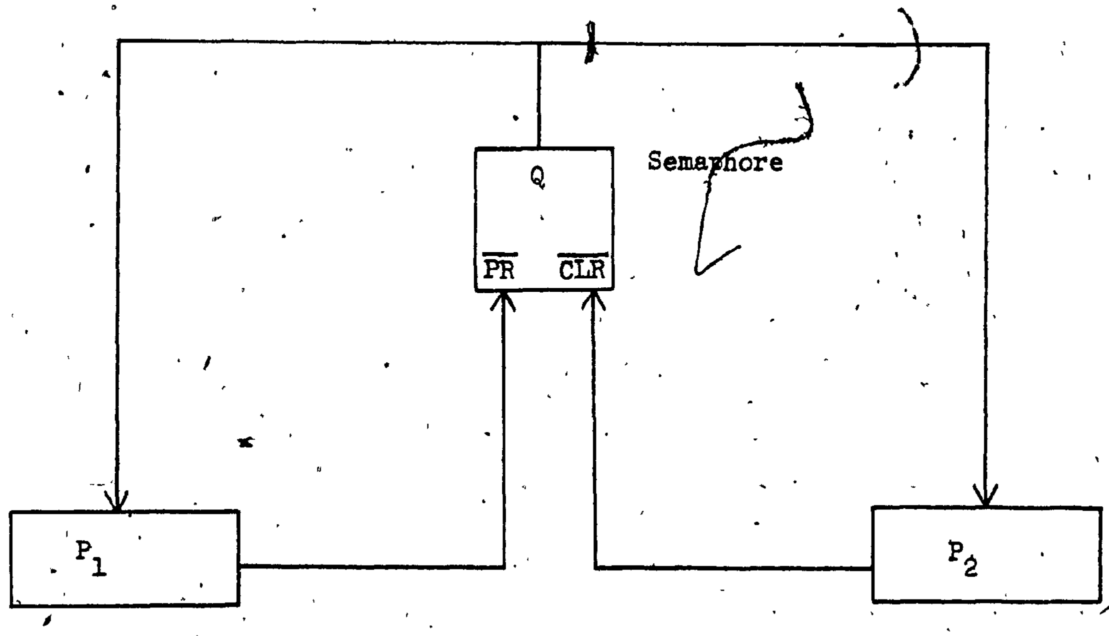


Figure 4.4 Distributed System



P	- Processor	Q	- Output
$\overline{PR}$	- Preset	$\overline{CLR}$	- Clear

Figure 4.5 Conditional I/O Technique with Semaphore.

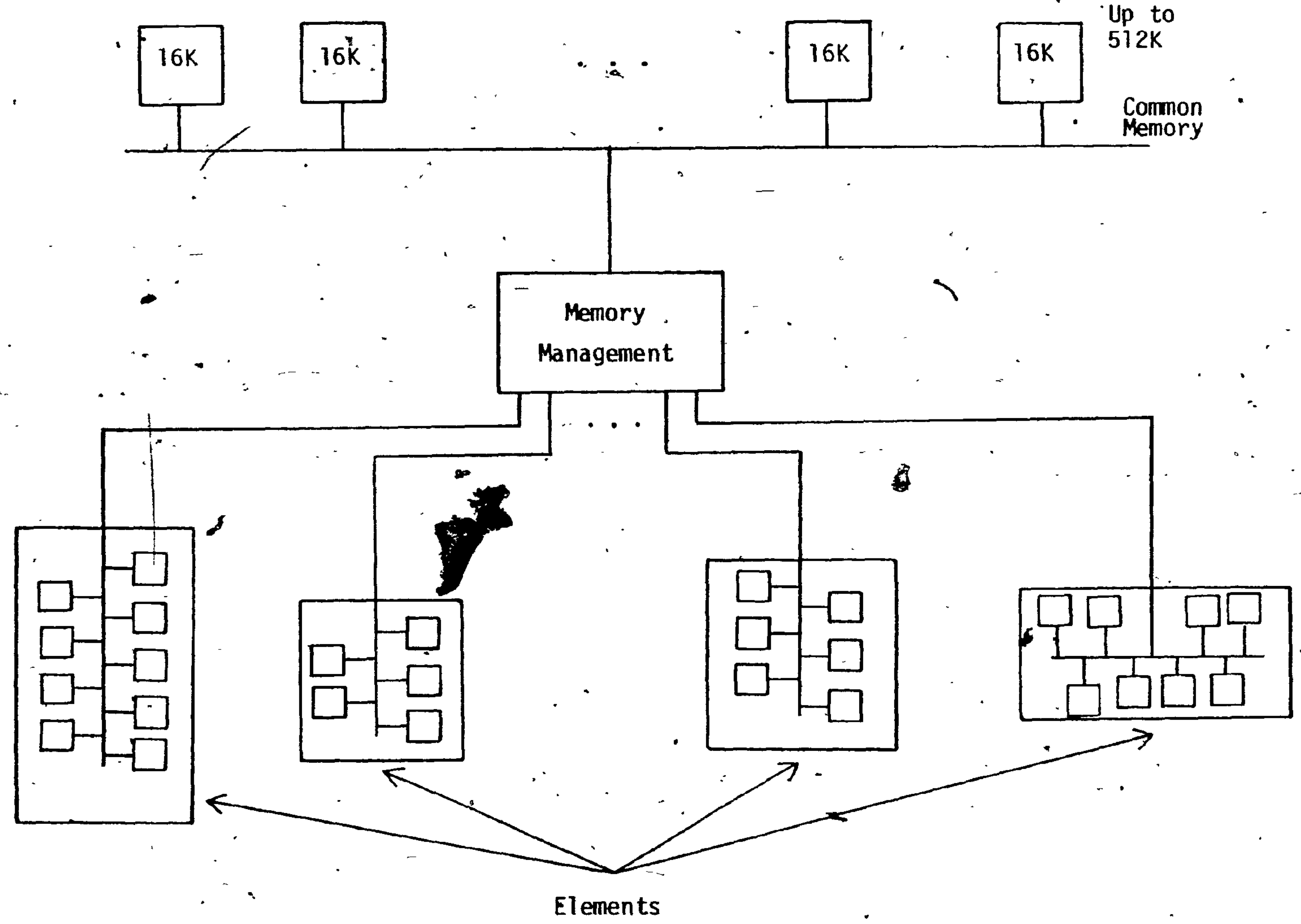
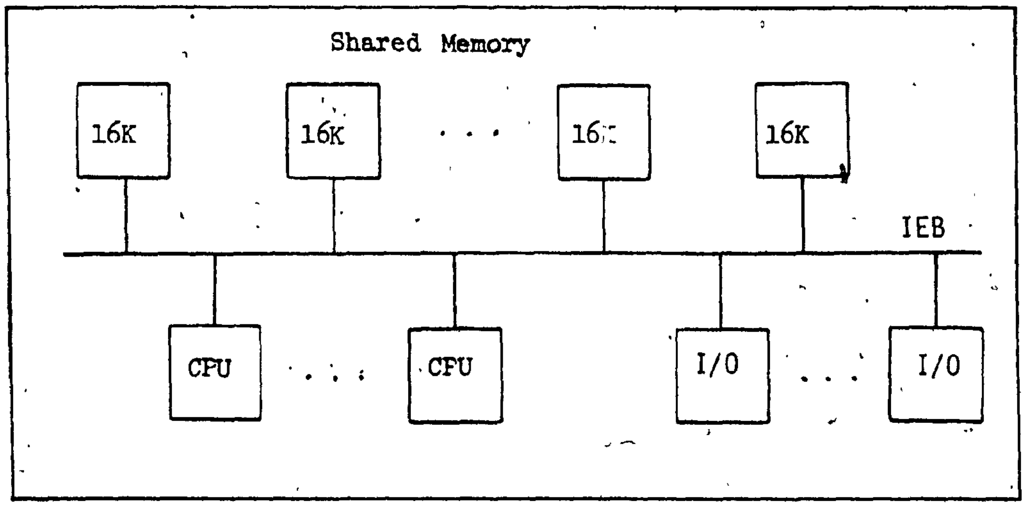


Figure 4.6 DPS Cluster Multiple Elements

Processing is distributed among several processing elements. These elements may be found in different parts of the system such as transmission, reception, and display subsystems, each with its own processing capabilities as shown in Fig. 4.7. . All CPUs, memories and I/O interface cards are connected to interface with intra-element bus (IEB) as shown in Fig. 4.7. The memory may be accessible by all the CPUs or may be dedicated to support a single CPU. The configuration is totally up to the designer. If the memory is accessible by all the CPUs, the transfer of data between the processing elements takes place through the memory. If the memory is to be a private memory, the processing element requires interface to communicate with other elements. Fig. 4.8 shows that the proposed interface makes the communications between the processing element easier whether the memory module is a dedicated one or not. The arrangement also reduces the IEB contention problem. Since the system is mainly used in processing of radar signals, the speed of data transfer between the processors is of primary importance. The interface helps to speed up the data transfer.

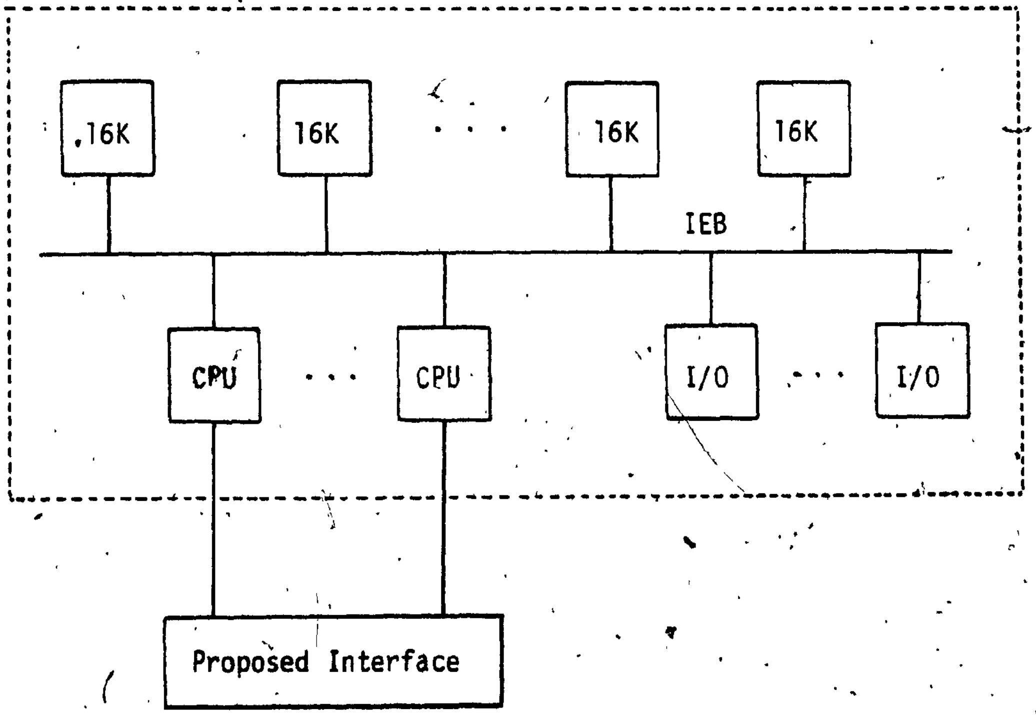
4.4 SUGGESTIONS FOR EXTENSIONS

In this section some possible modifications and extensions of the investigation carried out in this thesis, which may result in a more efficient performance of the interface and hence an improved multiprocessing, will be discussed.



IEB - Intra-element bus

Figure 4. 7 DPS Single Element



IEB - Intra-element bus

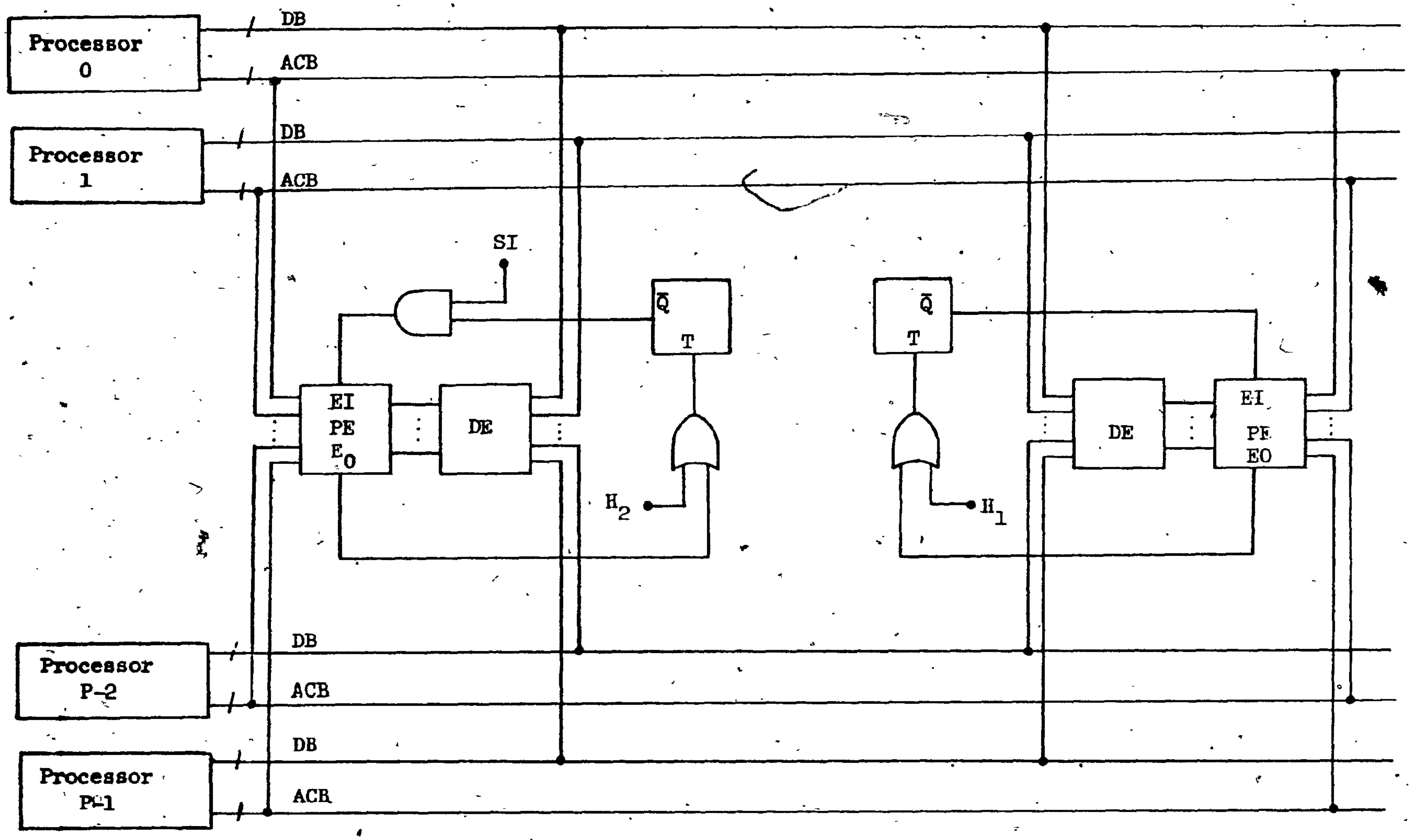
Figure 4.8 DPS Single Element with Proposed Interface.

4.4.1 Dedicated Bus

Dedicated bus can be effectively used to minimize the single bus contention problem. However, it was seen in the previous Chapter that there is only one address bus to the interface. It was also found through the computer simulation results that the rejection of read requests is also due to write requests and vice-versa, and the bus gets saturated before the control registers do. If we introduce another address bus to the interface, so that one is dedicated to read requests and the other to write requests, read rejection and write rejection rate will be reduced considerably. Thus a larger number of control registers can be used in the main module. This way the interface will be able to communicate with three processors at any one time. It can take the data from one processor, take the request for data from another processor, and send the data to some other processor simultaneously. If processor P_i wants the data from processor P_j, processor P_j sends the data to the interface and processor P_i can send the request for this data to the interface simultaneously. Thus the speed of data transfer can be increased further. Introduction of an additional address bus in the existing interface structure requires very little changes. In PEDC logic module now, the logic level at EI terminal of the write priority encoder becomes a function of availability of write address bus to the interface and logic level at EI terminal of read priority encoder becomes a function of availability of a control register and the read address bus to the interface. Fig. 4.9 illustrates the modified PEDC logic module. Both address buses will be connected to the main module separately.

4.4.2 Block of Data Transfer

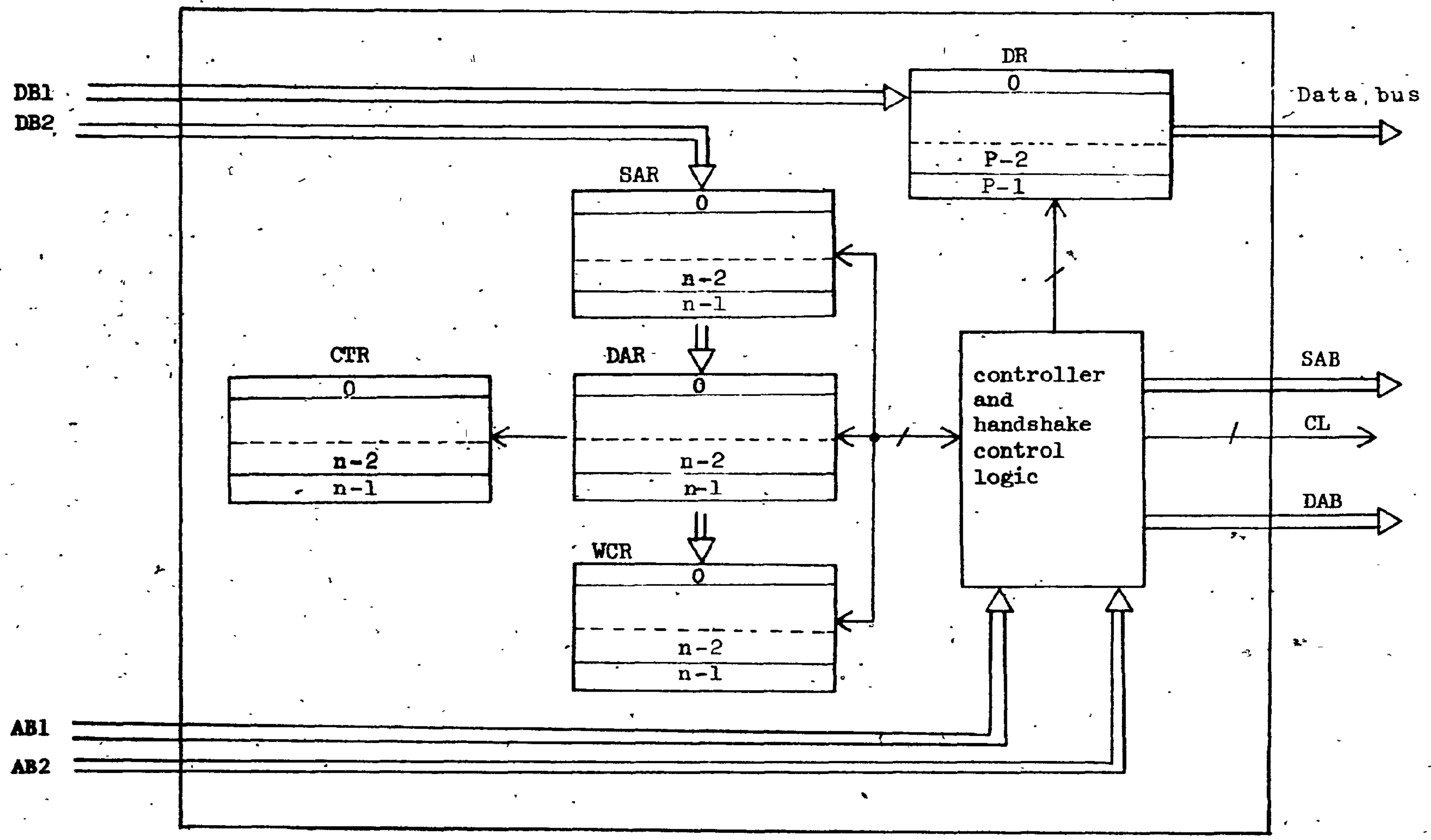
In order to reduce the common resource contention in a multiprocessor system, technique of block of data transfer from one memory module to another memory module can be used. To transfer a block of data by the interface, the main module with dedicated buses as discussed in Section 4.4.1 has to be modified. It is possible to achieve this goal by introducing clusters of registers along with the existing cluster of control registers. These clusters of registers can be used to store the starting address of the source memory module from where the data has to be taken, starting address of the destination memory module where the data has to be stored, word count to identify the total number of data to be transferred. Cluster of control registers, as before, will be used to identify the address of source and destination processors. A dedicated data bus will be required which will help to store the above information for block data transfer in the clusters of registers sequentially. Fig. 4.10 shows the block diagram of the modified main module. Request for data or request for block of data transfer will be stored on first-come first-served basis. Two outgoing address buses from the interface will be needed, so that



DB-Data bus
ACB-Address and control bus
EI-Enable input
EO-Enable output

PE-Priority encoder
DE-Decoder
H_1,H_2 -Handshake signals
SI-Status of the interface

Figure 4.9 Modified PEDC Logic Module



DB1	Data bus 1	CTR	Control registers	DR	Data register
DB2	Data bus 2	SAR	Source address register	SAB	Source address bus
AB1	Address bus 1	DAR	Destination address register	DAB	Destination address bus
AB2	Address bus 2	WCR	Word count register	CL	Control lines

Figure 4.10 Modified Main Module

each address bus can be connected to source memory module's address bus and destination memory module's address bus respectively. When the request for block data transfer is ready for execution, the interface will send a signal called Bus Request to the intended processors and in turn processors will send back signal Bus Grant to the interface. The interface will then put the destination starting address on the address bus and source starting address on the other bus and connect the two corresponding memory module data buses. The processor select logic module also will need some altrations. Once the data transfer is completed, the interface will give Bus Grant acknowledge signal to both the processors and start executing the next request. This way the interface will be able to take the data from one processor, take the request for data or block of data transfer from another processor and send the data to some other processor or transfer a block of data between the memory modules simultaneously.

CHAPTER V

CONCLUSIONS

A general purpose multimicroprocessor interface which speeds up data transfer between processors has been designed, built, and tested. The design method adopted in this investigation has been simple and adaptable to an evnironment where a large number of processors are to be used. Due to the cost consideration, however the interface has been built and tested using only two Intel 8085-A (SDK 85) microcomputers. The computer simulation studies have demonstrated that for an efficient use of the interface an optimum number of control registers to store the request for data by processors is required in the main module.

The interface designed has several features. The system where the interface is to be used does not require an overall synchronization, that is, each processor in the system can operate asynchronously with different clock rates. Without altering the physical structure of the multiprocessor system, the system can easily be made to act as a cascade or parallel structure. In a particular application, if the data exchange between the processors takes place in some fixed sequence and there is no simultaneous requests from the processors, the processors do not have to test the conditions for the availability of the interface. Two suggestions have been made to speed up data transfer between the processors even further. They are through the use of the dedicated buses for read and write requests and blocks of data transfer between the processors. Dedicated bus technique is easy to implement. An address bus to the interface consists of a few address lines which are connected to a processor's address lines dynamically only at the time of interaction between the processor and the interface. Block of data transfer involves data transfers between the memory modules which can be implemented by cycle stealing technique.

REFERENCES

1. J.P. Barthmaier, "Multiprocessing System Mixes 8-and 16 - Bit Microcomputers," Computer Design, PP. 137 - 144, Feb. 1980.

2. K.N. Karna and E.W. Dusio, "Communication Satellite Software," Computer, PP. 15 - 16, April 1983.

3. A.K. Jones and P. Schwarz, "Experience Using Multi-processor Systems - A Status Report," Computing Surveys, Vol. 12, No. 2, PP. 121 - 165, June 1980.

4. Cay Weitzman, Distributed Micro/Minicomputer Systems, Englewood Cliffs, N.J., Prentice Hall,1980

5. S.H. Fuller, J.K. Ousterhout, L. Raskin, P.I. Rubinfeld, P.J. Sindhu, and R.J. Swan, "Multi-Microprocessors: An Overview and Working Example," PROC. IEEE, Vol. 66, No. 2, PP. 216-228, Feb. 1978.

6. E.T. Fathi and M. Krieger, "Multiple Microprocessors Systems: What, Why and When," Computer, PP. 23-31, March 1983.

7. P.M. Russo, "Interprocessor Communication for Multi-Micro Computer Systems," Computer, PP. 67-76, April 1977.

8. B.A. Bowen and R.J.A. Buhr, The logical Design of Multiprocessor Systems, Englewood Cliffs, N.J. Prentice Hall, 1980

9. R.L. Davis, S. Zucker, and C.M. Campbell, "A Building Block Approach to Multiprocessing," Spring Joint Computer Conference, pp. 658-703, 1972.

10. K.A. Elmquist, "Architectural and Design Perspectives in a Modular Multimicroprocessor, The DPS-1," National Computer Conference, PP. 587-593, 1979.

11. Anderson G.A. and E.D. Jenson, "Computer Interconnection Structures: Taxonomy, characteristics, and Examples," Computing Surveys, Vol. 7, No.4, PP. 197-213, Dec. 1975.

12. T. Lang, M. Valero and I. Alagre, " Bandwidth of Cross Bar and Multiple Bus Conections for Multiprocessors," IEEE Trans.Comput., Vol. C-31 No-12, PP. 1227 - 1234, Dec.1982.

13. T. Lang, M. Valero, and M.A. Fiol, "Reduction of Connections for Multibus Organization," IEEE Trans. comput., Vol. C-32, No. 8, PP. 707-714, Aug.1983

14. Baer J.L., " A Survey of some Theoritical Aspects of Multiprocessing," Computing Surveys, Vol. 5, No.1. PP. 31-80, March 1980.

15. Enslow P.H., "Multiprocessor Organization-A Survey," Computing Surveys, Vol.9, No.1. PP. 103-129, March 1977.

16. R.J. Swan, S.H. Fuller, D.P. Siewiorek, "CM*-A Modular Multimicroprocessor," National Computer Conference, PP. 637 - 644, 1977.

17. H. Siegel, R.J. McMiller, and P.T. Mueller, "A Survey Interconnection methods for Reconfigurable Parallel Processing Systems," Proc. NCC-AFIPS, V01. 48, PP. 529-542, June 1979.

18. D.C. Loughny, "IEEE Standard 488 and Microprocessor Synergism," Proc. IEEE, Vol. 66, No.2, PP. 162-172, Feb. 1978.

19. R. Gilbert, "The General Purpose Interface Bus," IEEE Micro, PP. 41-51, Feb. 1982.

20. K. Eckert, "A Multiprocessor Interface," IEEE Micro, PP. 67-70, Feb. 1982.

21. F.A. Briggs, "Effects of Buffered Memory Requests in Multiprocessor Systems," Simuletter, Vol.11, No 1., PP. 73-81, Fall 1979.

22. Thomas J. Schriber, Simulation Using GPSS, New-York, John Willey & Sons, 1974.

23. F. Mintzer, " Parallel and Cascade Microprocessor Implementation for Digital Signal Processing," IEEE Trans. Acoustics, Speech, and Signal Processing, Vol. 29, No. 5, PP. 1018-1027, Oct. 1981.

24. R. Mauriello et al., " A Distributed Processing System," Proc. Trends and Application: 1979, PP. 1-10.

APPENDIX A

PROGRAM LISTINGS FOR 4 CONTROL REGISTERS

```
BLOCK                                                                        STATEMENT
NUMBER  *LOC    OPERATION  A,B,C,D,E,F,G,H,I          COMMENTS               NUMBER
        *                                                                         1
        *                                                                         2
                SIMULATE                                                          3
        *                                                                         4
         EXPCN FUNCTION     RN1,C24              EXPONENTIAL DIST                 5
        0.0,0.0/0.1,0.104/0.2,0.222/0.3,0.355/0.4,0.509/0.5,0.69                  6
        0.6,0.915/0.7,1.2/0.75,1.38/0.8,1.6/0.84,1.83/0.88,2.12                   7
        0.9,2.3/0.92,2.52/0.94,2.81/0.95,2.99/0.96,3.2/0.97,3.5                   8
        0.98,3.9/0.99,4.6/0.995,5.3/0.998,6.2/0.999,7/0.9997,8                    9
        *                                                                        10
         2      VARIABLE    500/XH1                                              11
        *                                                                        12
  1             GENERATE    V2,FN$EXPCN          CREATE REQUESTS                 13
  2      OPRO   ASSIGN      1,V1,PH              PICK A ORIGIN PROCESSOR         14
  3             GATE LR     PH1,OPRO             TEST FOR OPRO BUSY              15
  4             LOGICS      PH1                  MAKE OPRO BUSY                  16
  5             ASSIGN      3,FN2,PH             GENERATE READ/WRITE             17
  6             TEST G      PH3,3,READ           FIND READ/WRITE                 18
  7             TRANSFER    ,WRITE                                               19
        *                                                                        20
        *                                                                        21
  8      BNAR   ADVANCE     10,9                 TIME FOR TEST INST              22
  9             LOGICR      PH1                  MAKE OPRO FREE                  23
 10             TERMINATE   1                    BUS NOT AVAILABLE TO READ       24
        *                                                                        25
        *                                                                        26
 11      SINA   ADVANCE     10,9                 TIME FOR TEST INST              27
 12             LOGICR      PH1                  MAKE OPRO FREE                  28
 13             TERMINATE   1                    CR IS NOT AVAILABLE             29
        *                                                                        30
        *                                                                        31
 14      READ   ASSIGN      2,V1,PH              PICK A DEST PROCESSOR           32
 15             TEST NE     PH1,PH2,READ         RETRY IF OPRO=DPRO              33
 16             GATE NU     1,BNAR               CHECK FOR BUS                   34
 17             GATE SNF    SIF,SINA             CONDITION SI,FULL/NOT           35
 18             SEIZE       1                    MAKE BUS BUSY                   36
 19             ADVANCE     11,10                TIME FOR TEST INST              37
 20             QUEUE       REQ                                                  38
 21             ENTER       SIF                  GET THE STATUS                  39
 22             RELEASE     1                    BUS HANGS UP                    40
 23             ADVANCE     15,14                DATA TRANSFER                   41
 24             LOGICR      PH1                  MAKE OPRO FREE                  42
 25             DEPART      REQ                                                  43
 26             LEAVE       SIF                                                  44
 27             TERMINATE   1                    OPRO GOT THE DATA               45
        *                                                                        46
        *                                                                        47
 28      BNAW   ADVANCE     10,9                 TIME FOR TEST INST              48
 29             LOGICR      PH1                  MAKE OPRO FREE                  49
 30             TERMINATE   1                    BUS NOT AVAILABLE TO WRITE      50
        *                                                                        51
        *                                                                        52
 31      WRITE  PRIORITY    1                    GET PRIORITY                    53
 32             GATE NU     1,BNAW               CHECK FOR BUS                   54
 33             QUEUE       WRQ                                                  55
```

```
34          SEIZE       1                 MAKE EUS BUSY               56
35          ADVANCE     11,10             TIME FOR TEST INST          57
36          ACVANCE     8,7               DATA-TRANSFER               58
37          DEPART      WRQ                                           59
38          LOGICR      PHI               MAKE CFRO FREE              60
39          RELEASE     1                 BUS HANGS JP                61
40          TERMINATE   1                 CFFC STORE THE CATA         62
        *                                                             63
        *                                                             64
        SIF     STORAGE     4                 NO OF REGISTERS             65
        *                                                             66
                INITIAL     XH1,8             NO OF PROCESSORS            67
        *                                                             68
        1       VARIABLE    XH1*RN1/1000+1    PICK A PROCESSOR            69
        *                                                             70
        *                                                             71
        2       FUNCTION    RN2,D5                                    72
       0.2,1/0.4,2/0.7,3/0.9,4/1,5                                    73
        *                                                             74
                START       1000                                      75
                END                                                   76
```

```
RELATIVE CLOCK          60299  ABSOLUTE CLOCK          60299
BLOCK COUNTS
BLOCK CURRENT      TOTAL   BLOCK CURRENT     TOTAL   BLOCK CURRENT     TOTAL   BLOCK CURRENT     TOTAL
    1       0       1000      11       0         0      21       0       565      31       0       304
    2       0       1044      12       0         0      22       0       566      32       0       304
    3       0       1044      13       0         0      23       0       566      33       0       256
    4       0       1000      14       0       601      24       0       565      34       0       256
    5       0       1000      15       0       801      25       0       566      35       0       256
    6       0       1000      16       0       656      26       0       566      36       0       256
    7       0        304      17       0       566      27       0       566      37       0       256
    8       0        130      18       0       566      28       0        48      38       0       256
    9       0        130      19       0       566      29       0        48      39       0       256
   10       0        130      20       0       566      30       0        48      40       0       256
```

```
                              ****************************************
                              *                                      *
                              *              FACILITIES              *
                              *                                      *
                              ****************************************

FACILITY   NUMBER    AVERAGE    -AVERAGE  UTILIZATION  DURING-    CURRENT    PERCENT         TRANSACTION NUMBER
           ENTRIES   TIME/TRAN   TOTAL    AVAIL.    UNAVAIL.      STATUS     AVAILABILITY    SEIZING  PREEMPTING
                                 TIME     TIME      TIME
    1        822      13.290     .181                                        100.0


                              ****************************************
                              *                                      *
                              *               STORAGES               *
                              *                                      *
                              ****************************************

STORAGE   CAPACITY   AVERAGE    ENTRIES   AVERAGE    -AVERAGE  UTILIZATION  DURING-    CURRENT   PERCENT        CURRENT    MAXIMUM
                     CONTENTS             TIME/UNIT   TOTAL    AVAIL.    UNAVAIL.      STATUS    AVAILABILITY   CONTENTS   CONTENTS
                                                      TIME     TIME      TIME
  SIF         4       .141       566      14.993      .035                                       100.0                        3


                              ****************************************
                              *                                      *
                              *                QUEUES                *
                              *                                      *
                              ****************************************

QUEUE    MAXIMUM   AVERAGE    TOTAL     ZERO      PERCENT   AVERAGE     $AVERAGE    TABLE    CURRENT
         CONTENTS  CONTENTS   ENTRIES   ENTRIES   ZEROS     TIME/TRANS  TIME/TRANS  NUMBER   CONTENTS
 REQ         3       .140       566                  .0      14.992      14.992
 ARQ         1       .077       256                  .0      18.355      18.355
$AVERAGE TIME/TRANS = AVERAGE TIME/TRANS EXCLUDING ZERO ENTRIES
```

APPENDIX B

PROGRAM LISTINGS FOR 1 CONTROL REGISTER

```
BLOCK                                                                      STATEMENT
NUMBER  *LOC   OPERATION  A,B,C,D,E,F,G,H,I          COMMENTS                NUMBER
        *                                                                      1
        *                                                                      2
               SIMULATE                                                        3
        *                                                                      4
        EXPCN FUNCTION    RN1,C24            EXPONENTIAL DIST                  5
        0.0,0.0/0.1,0.104/0.2,0.222/0.3,0.355/0.4,0.509/0.5,0.69               6
        0.6,0.915/0.7,1.2/0.75,1.38/0.8,1.6/0.84,1.83/0.88,2.12                7
        0.9,2.3/0.92,2.52/0.94,2.81/0.95,2.99/0.96,3.2/0.97,3.5                8
        0.98,3.9/0.99,4.6/0.995,5.3/0.998,6.2/0.999,7/0.9997,3                 9
        *                                                                     10
        2      VARIABLE   500/XH1                                             11
        *                                                                     12
  1            GENERATE   V2,FN$EXPCN        CREATE REQUESTS                  13
  2     OPRO   ASSIGN     1,V1,PH            PICK A ORIGIN PROCESSOR          14
  3            GATE LR    PH1,OPRO           TEST FOR OPRO BUSY               15
  4            LOGICS     PH1                MAKE OPRO BUSY                   16
  5            ASSIGN     3,FN2,PH           GENERATE READ/WRITE              17
  6            TEST G     PH3,3,READ         FIND READ/WRITE                  18
  7            TRANSFER   ,WRITE                                              19
        *                                                                     20
        *                                                                     21
  8     BNAR   ADVANCE    10,9               TIME FOR TEST INST               22
  9            LOGICR     PH1                MAKE OPRO FREE                   23
 10            TERMINATE  1                  BUS NOT AVAILABLE TO READ        24
        *                                                                     25
        *                                                                     26
 11     SINA   ADVANCE    10,9               TIME FOR TEST INST               27
 12            LOGICR     PH1                MAKE OPRO FREE                   28
 13            TERMINATE  1                  CR IS NOT AVAILABLE              29
        *                                                                     30
        *                                                                     31
 14     READ   ASSIGN     2,V1,PH            PICK A DEST PROCESSOR            32
 15            TEST NE    PH1,PH2,READ       RETRY IF OPRO=DPRO               33
 15            GATE NU    1,BNAR             CHECK FOR BUS                    34
 17            GATE SNF   SIF,SINA           CONDITION SI,FULL/NOT            35
 18            SEIZE      1                  MAKE BUS BUSY                    36
 19            ADVANCE    11,10              TIME FOR TEST INST               37
 20            QUEUE      REQ                                                 38
 21            ENTER      SIF                GET THE STATUS                   39
 22            RELEASE    1                  BUS HANGS UP                     40
 23            ADVANCE    7,6                DATA TRANSFER                    41
 24            LOGICR     PH1                MAKE OPRO FREE                   42
 25            DEPART     REQ                                                 43
 26            LEAVE      SIF                                                 44
 27            TERMINATE  1                  OPRO GOT THE DATA                45
        *                                                                     46
        *                                                                     47
 28     BNAW   ADVANCE    10,9               TIME FOR TEST INST               48
 29            LOGICR     PHI                MAKE OPRO FREE                   49
 30            TERMINATE  1                  BUS NOT AVAILABLE TO WRITE       50
        *                                                                     51
        *                                                                     52
 31     WRITE  PRIORITY   1                  GET PRIORITY                     53
 32            GATE NU    1,BNAW             CHECK FOR BUS                    54
 33            QUEUE      WRQ                                                 55
```

```
34         SEIZE      1                  MAKE EUS BJSY
35         ADVANCE    11,10              TIME FOR TEST INST
36         ADVANCE    8,7                DATA TRANSFER
37         DEPART     WRO
38         LOGICR     PH1                MAKE OPRO FREE
39         RELEASE    1                  BUS HANGS JP
40         TERMINATE  1                  OPRO STORE THE DATA
      *
      *
       SIF STORAGE    1                  NO OF REGISTERS
      *
           INITIAL    XH1,8              NO CF PROCESSORS
      *
       1   VARIABLE   XH1*RN1/1000+1     PICK A PROCESSOR
      *
      *
       2   FUNCTION   RN2,D5
      0.2,1/0.4,2/0.7,3/0.9,4/1,5
      *
           START      1000
           END
```

```
RELATIVE CLOCK         58778  ABSOLUTE CLOCK        58778
BLOCK COUNTS
BLOCK CURRENT   TOTAL   BLOCK CURRENT   TOTAL   BLOCK CURRENT   TOTAL   BLOCK CURRENT   TOTAL
    1     0     1000      11      0       43      21      0      527      31      0      304
    2     0     1030      12      0       43      22      0      527      32      0      304
    3     0     1030      13      0       43      23      0      527      33      0      256
    4     0     1000      14      0      795      24      0      527      34      0      256
    5     0     1000      15      0      795      25      0      527      35      0      256
    6     0     1000      16      0      696      26      0      527      36      0      256
    7     0      304      17      0      570      27      0      527      37      0      255
    8     0      126      18      0      527      28      0       48      38      0      256
    9     0      126      19      0      527      29      0       43      39      0      256
   10     0      126      20      0      527      30      0       48      40      0      256
```

```
                         ****************************************
                         *                                      *
                         *             FACILITIES               *
                         *                                      *
                         ****************************************

FACILITY   NUMBER    AVERAGE    -AVERAGE  UTILIZATION  DURING-   CURRENT    PERCENT      TRANSACTION NUMBER
           ENTRIES   TIME/TRAN   TOTAL     AVAIL.    UNAVAIL.    STATUS   AVAILABILITY   SEIZING  PREEMPTING
                                 TIME      TIME      TIME.
    1        783      13.704     .182                                        100.0


                         ****************************************
                         *                                      *
                         *              STORAGES                *
                         *                                      *
                         ****************************************

STORAGE   CAPACITY   AVERAGE   ENTRIES   AVERAGE   -AVERAGE  UTILIZATION  DURING-   CURRENT     PERCENT     CURRENT    MAXIMUM
                     CONTENTS            TIME/UNIT   TOTAL     AVAIL.   UNAVAIL.   STATUS   AVAILABILITY  CONTENTS   CONTENTS
                                                     TIME      TIME      TIME
  SIF         1        .063      527      7.025      .062                                   100.0                      1


                         ****************************************
                         *                                      *
                         *               QUEUES                 *
                         *                                      *
                         ****************************************

QUEUE    MAXIMUM    AVERAGE    TOTAL    ZERO     PERCENT     AVERAGE     $AVERAGE     TABLE    CURRENT
         CONTENTS   CONTENTS   ENTRIES  ENTRIES  ZEROS      TIME/TRANS   TIME/TRANS   NUMBER   CONTENTS
  REQ       1         .062      527                .0          7.024        7.024
  WRQ       1         .082      256                .0         19.003       19.003
$AVERAGE TIME/TRANS = AVERAGE TIME/TRANS EXCLUDING ZERO ENTRIES
```

APPENDIX C

DESCRIPTION OF STATE DIAGRAMS

DESCRIPTION OF THE STATE DIAGRAM OF THE PEDC LOGIC MODULE

(1) From State Idle

(a) when the module receives read request(s), it allows the highest priority read request processors to access the interface and goes to the state BR (Module is busy because of read request).

(b) when the module receives write request(s) or read and write requests, it allows the highest priority write request processor to access the interface and goes to the state BW (Module is busy because of write request).

(c) when SI line becomes low, the module goes to the state BRO (Module's read part only busy).

(2) From State BR

(a) when the module receives handshake signal H_2, it goes to the state Idle.

(3) From State BW

(a) when the module receives handshake signal H_1, it goes to the state Idle.

(b) when the module receives handshake signal H_1 and at the same time the logic level of SI line is low, it goes to the state BRO.

(4) From State BRO

(a) when SI line becomes high, the module goes to the state Idle.

(b) when the module receives read and write requests or write request(s), it allows the highest priority write request processor to access the interface and goes to the state BW.

DESCRIPTION OF THE STATE DIAGRAM OF THE MAIN MODULE

(1) From State Idle

(a) when the module receivers R(Read) signal from a processor, it goes to the state Read and accepts a request and issues a handshake signal H_2.

(b) when the module receives W(Write) signal from a processor, it goes to Write state and accepts a data and issues a handshake signal H_1.

(c) when SI line becomes low, the module goes to RF state (all the control registers are full).

(2) From State Read

(a) when R signal becomes active low, the module goes to the state Idle.

(3) From State Write

(a) when W becomes active low, the module goes to the state Idle.

(b) when W becomes active low and at the same time the logic level SI line is low, the module goes to the state RF.

(4) From State RF

(a) when SI line becomes high, it goes to the state Idle.

(b) When W becomes active it goes to the state-write and issues a handshake signal H_1.

DESCRIPTION OF THE STATE DIAGRAM OF THE CONTROLLER

(1) From State Idle.

(a) when the controller receives handshake signal H_2, CR modulo n up counter and n+1 up and down counter gets incrementated and goes to the state Execution (where the request are received one by one).

(2) From State EXE

(a) when the modulo n+1 up and down counter are equal to one and the controller receives the handshake signal H_4, it goes to the state Idle and the modulo n+1 up and down counter gets decremented.

(b) when the controller receives H_4 signal, the up counter gets incremented and the modulo n+1 up and down counter gets decremented and stays in the same state.

(c) when the controller receives H_2 signal, the CR modulo n up counter and the modulo n+1 up and down counter gets incremented and stays in the same state.