AN INTERFACE DESIGN BETWEEN THE PDP11/45

AND THE MICROPROCESSOR LABORATORY

APPROVED BY SUPERVISORY COMMITTEE:

Yu-cheng Liu
CHAIRMAN

Glenn A. Gibson

John W. Stamey

Michael E. Keeler
DEAN OF GRADUATE SCHOOL

AN INTERFACE DESIGN BETWEEN THE PDP11/45 AND THE MICROPROCESSOR LABORATORY

by

HANY HUSSEIN AMMAR

THESIS

Presented to the Faculty of the Graduate School of

The University of Texas at El Paso

in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE

THE UNIVERSITY OF TEXAS AT EL PASO

August, 1980

ACKNOWLEDGEMENTS

I would like to express my sincere appreciation to my graduate advisor Dr. Yu-Cheng Liu, for guiding me in this work and for making my graduate study at UTEP a truly meaningful experience. To professors Glenn A. Gibson and J. Starner, I wish to express my appreciation for their review and valuable suggestions. And thanks to all professors and fellow students in the Electrical Engineering Department.

TABLE OF CONTENTS

LIST OF FIGURES

Page

CHAPTER I

INTRODUCTION

This work addresses the problem of designing an interface between two digital computer systems so that they can communicate and cooperate with each other to accomplish a certain task. The word "interface" in the data communication field refers to interactions between two independent data processing systems.

The interface between any two digital systems could be accomplished hardwarewise, softwarewise, or a combination of both. This work will concentrate on the software approach.

The computer laboratory in the Electrical Engineering Department at UTEP consists of a PDP11/45 minicomputer operated under a time-sharing operating system, dual disk drive, two DEC tape drives, and other major peripherals including a card reader, a line printer, several CRT terminals and two DEC writers. The department also has a microprocessor laboratory that contains an Intel Microcomputer Development System (MDS) designed around the 8080 microprocessor, and twelve MCS-86 system design stations (SDK86) based on the 8086 microprocessor.

The Intel MDS system provides the necessary facilities to develop software for both the 8080 and the 8086. Although the system includes a monitor, text editor, assemblers, and other utility routines, it is a single user oriented system. The idea behind this work was to interface the MDS sys-

tem to the multi-user PDP11/45 computer system so that the users can simultaneously develop their 8080 or 8086 based assembler programs with the PDP11/45 and assemble them through the existing 8080 or 8086 cross-assemblers. The interface allows the user to transfer these programs to the microprocessor laboratory for execution. This will substantially reduce the time that each user has to wait if the program development is done with the MDS system. In addition, this allows the MDS system to share resources, such as mass storage devices, line printer, card reader, etc., with the PDP11/45.

This document is organized into five chapters as outlined below. Chapter two describes generally the software interface between two digital computers, the levels of the interface, the protocols that govern the communication between the two systems, and how a physical connection could be established between them.

Chapter three describes briefly the features of the PDP11/45 minicomputer system, and the Intel MDS system. Also a detailed description of how to establish a physical connection between the two systems, and the interface routine will be given.

Chapter four describes the MSD-86 system design station (SDK86), and how to connect it to the PDP11/45. The software which allows the user to down load 8086 absolute module from the PDP11/45 will also be presented.

In Appendix A several examples that illustrate the usage of the interface routines will be given. In Appendix B

the source code of these routines is listed.

CHAPTER II

THE INTERFACE BETWEEN TWO DIGITAL COMPUTER SYSTEMS

2.1 Introduction

The idea of interfacing two or more computer systems emerged from the growing technique of distributed processing. Distributed processing means that a single logical set of processing functions is implemented across two or more processors, so that each performs some part of the total processing required. Distributed processing is often accompanied by the formation of a distributed data base. A distributed data base is a collection of data elements stored at multiple systems and are interrelated. It allows a process (program execution) at one system to have access to data stored at another system. Distributed systems take many forms covering a diverse range of system architecture. The very term distributed processing may invoke radically different images of technology and problem solutions depending upon the user.

The common thread linking the different types of distributed systems is the use of two or more cooperating computing elements to perform jobs that were heretofore performed on large computers or not performed at all. Distributed systems have evolved as one of the existing alternatives to centralized facilities because in many circumstances centralized

facilities have not provided optimum or effective solutions. For example, a large computer might be too costly for applications which do not need extensive computing capability. The complex hardware and operating system software required for a large facility may fail more often than is acceptable for some time-critical jobs. In addition, a local system may not have the resources required for a specific job.

A network of cooperating processors can under the proper circumstances alleviate the above problem. In the past, a major setback to the distributed approach has been the cost. The technology of computing has been such that a collection of small computers equivalent in capability to a large computer cost more than the large computer. However, the situation has been changed in the past few years. Low cost computing power in the form of minicomputers and microcomputers presents an attractive alternative to large computers for many applications.

With cost no longer a barrier the inherent advantages of distributed processing can be applied to a wide array of computing problems. As mentioned in reference (1), there are two major advantages of distributed processing:

1- Resources sharing - a collection of interacting systems allow users of one system to take advantage of resources that exist at another system. These resources could consist of programs, data bases and computing power.

2- Reliability - redundancy can be achieved in a relatively inexpensive manner in a distributed system. The entire system does not have to be replicated as in the case with a

single computer. Only an incremented number of processors must be added to achieve the required degree of availability. Simpler and hence, more reliable software structure may be achieved in a collection of small computers.

Back to our main concern which is the design of an interface between two digital computer systems, the interface can be divided into three parts or three levels that are listed below.

1- Level one, deals with how to establish a physical connection between the systems.

2- Level two, deals with the protocols governing the communication between the two systems (type of data, message framing and error detection).

3- Level three,deals with the interface software package and its interaction with the system user.

The following three sections will discuss these three levels of interface. Sections two, three, and four will discuss levels three, two, and one respectively.

2.2 Major Types of Interface

The interface software package that resides at each system depends greatly on the type of interface and the task the interface between the two systems should accomplish. There are two major types of interface that will be discussed below.

Master-slave type interface

In this type of interface one system acts as a master and the other acts as a slave. The master system will prepare and send proper commands to the operating system of the slave computer. The slave will execute these commands and send back its response and status to the master system.

As mentioned above, the interface software package depends on the task that should be accomplished. For example, one of the simplest tasks is to exchange files between the two systems. A file may be a source-level program or a job (an executable program and its accompanying data) which is transmitted from one system to another for load leveling purposes. In this type of interface as shown in figure 2.1, process P in the master system will interact with process Q in the slave system to transfer a file from the master to the slave. Also processes R and S will interact to accomplish the file transfer in the other direction (i.e. from the slave to the master).

If a user of the master system wishes to transfer a file to the slave system, he will send command to the master operating system to start execution of process P. Process P will send a proper command to the operating system of the slave to start execution of process Q. Then the two processes will interact together to accomplish the file transfer. Similarly if the user wants to fetch a file from the slave system, he will use process R which will interact with process S over

the slave system. These processes will be explained in detail in Chapter III when the interface between the Minicomputer PDP11/45 and the Intel Microcomputer Development System (MDS) is discussed.

The master-slave type interface is used in the vertically (hierarchically) distributed systems. In such a system, the interconnected devices from a hierarchy, sharing tasks in a structured way with each component being controlled by the higher level member(s) of the hierarchy.

As an example of the hierarchically distributed system, figure 2.2 shows a manufacturing control system. In this system the large scale information processor is not directly in real time process control. It maintains the master database, which is used for overall scheduling and control of the manufacturing process.

At the next level of the hierarchy, there are several satellite processors implemented using minicomputers. Each satellite handles a part of the manufacturing process, and maintains its local database, which is a subset of the control database and contains only the data applicable to the local task.

The next lower level of the hierarchy consists of terminal concentrators. These minicomputers link the satellites to the lowest level devices concentrating data from many devices onto a few communication links and/or direct cables to each satellite processor.

At the lowest level, microcomputers monitor and

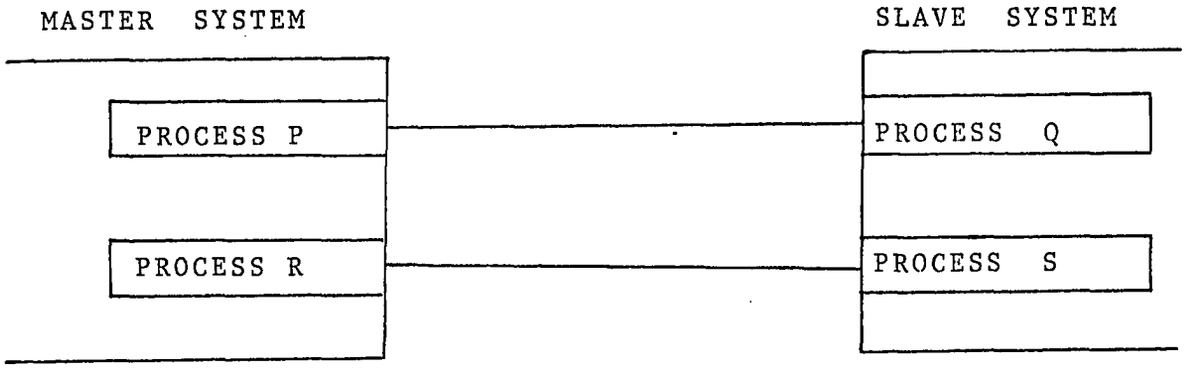

Figure 2.1 Master-Slave Type Interface

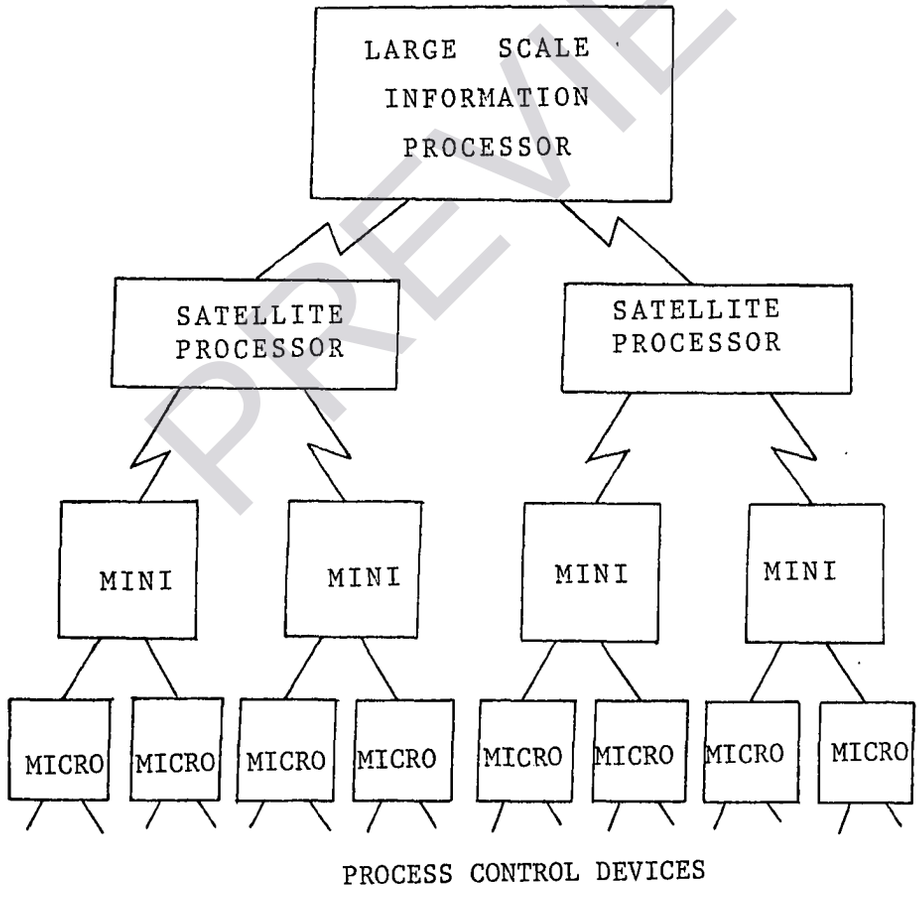

Figure 2.2 Hierarchically Distributed System
(Adapted from reference (1))

control the factory equipment. Process status information is sent up to the satellite processors while control commands are received from the satellite to guide the manufacturing process.

Horizontal Interface

In this type of interface the two systems cooperate at an equal level so that both systems could send commands to each other. For the task mentioned before where the two systems communicate by exchanging files, the users of any of the two systems should be able to transfer to or fetch a file from the other system. Figure 2.3 shows the processes needed to accomplish this task, they perform the same jobs described before in figure 2.1.

This type of interface is used in the horizontally distributed systems. The horizontal distribution of processing functions involves the interconnection of two or more components which are logically equal. The components may be physically unlike and of different capacity and power. The important aspect is their logical relationship within the distributed system. Figure 2.4 shows an example of horizontally distributed system in which three processors cooperate to achieve dynamic load leveling, in other words spreading the processing/database access load among them. The three processors have the same capabilities so that jobs/transactions can be moved freely among them. It might be desirable to execute a specific job in a

particular computer because it requires access to a database associated with that computer. On other occasions it might be desirable to move jobs, possibly accompanied by their data, from an overloaded processor to one which has available capacity. The choices for load leveling can be summarized as follows.

1- Move the process to data

2- Move data to the process

In a batch-oriented environment the tendency is to transmit a job or a subdivision of a job to the processor where the database is located. Because the batch job often does extensive processing against the database and database size is typically large, it is more economical to transmit the job and its input/output data than to transmit the database.

As shown in figure 2.4, if a job is entered at city A but requires access to database at city B, then the job will normally be transmitted to processor B, and executed there. The output will be returned to city A if necessary.

For inquiry/response applications, usually only a small part of the database is accessed. It is therefore feasible to transmit the data to the process and only the updates if any will be returned to the computer where the data are stored. In this case the process as specified by its process-state and programs may be considerably larger than the data involved.

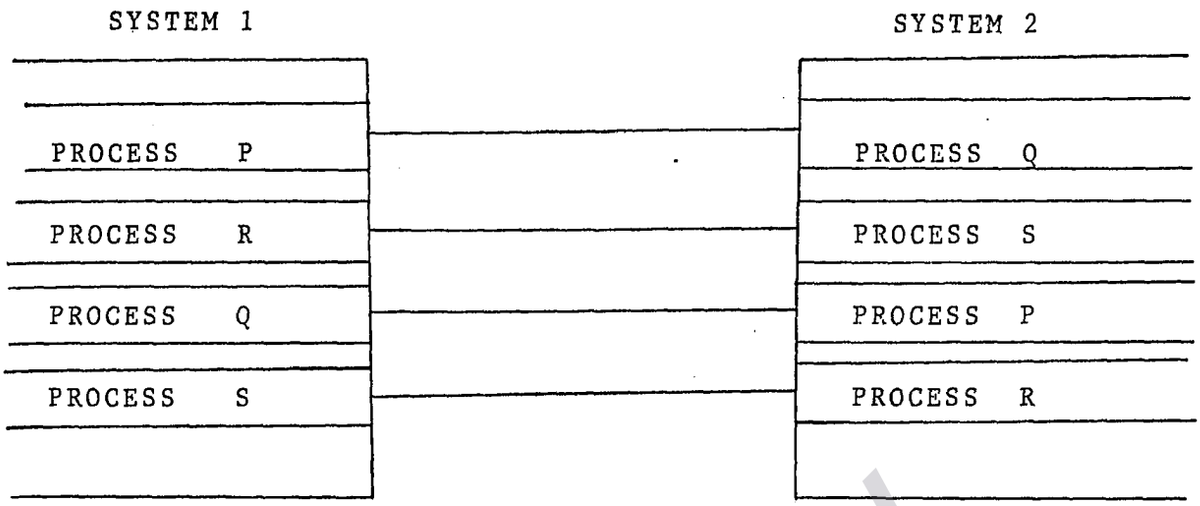

Figure 2.3 Horizontal Interface

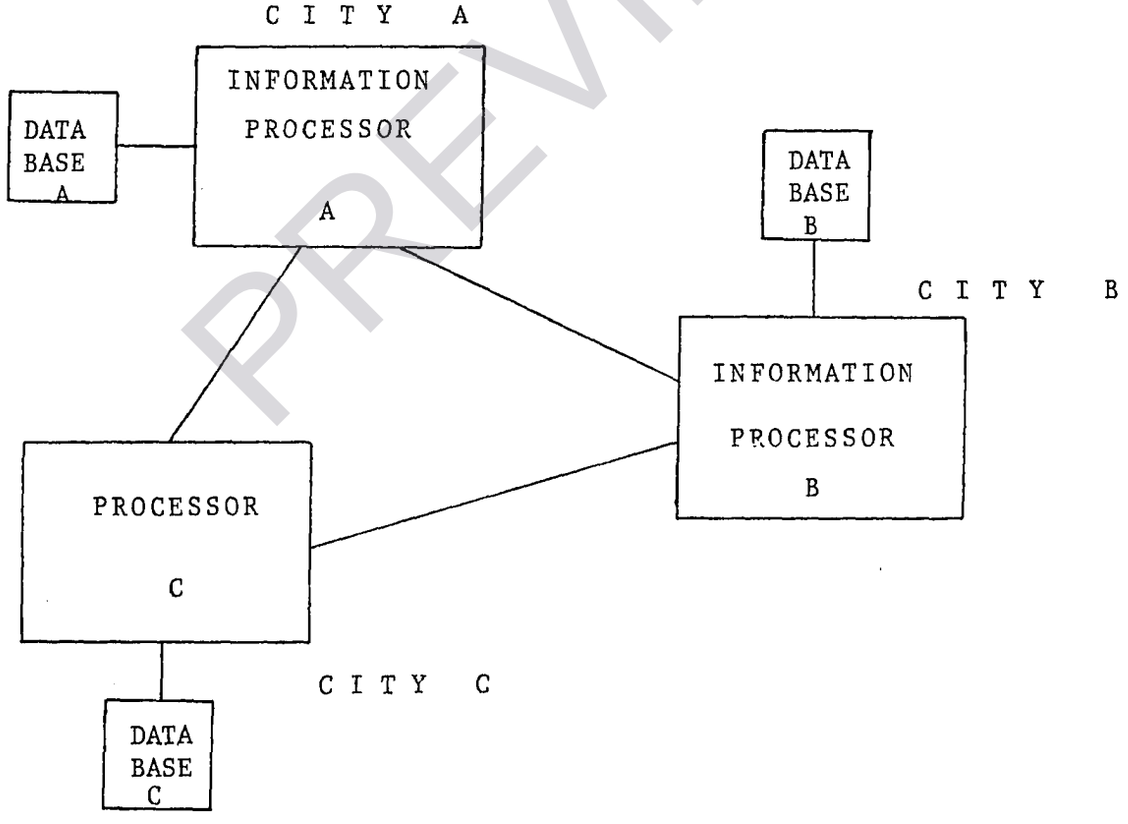

Figure 2.4 Interconnected Computer Service Centers (Adapted from reference (1))

2.3 Communication Protocols

A protocol is basically a set of rules that govern the communication between two systems or two processes. These rules are designed to solve operating problems in the following areas.

1- Framing - The determination of which groups of characters constitute a message.

2- Error control - The detection of errors by parity checks or by checksums, the acceptance of correct messages and the request for retransmission of faulty messages.

3- Transparency - The transmission of information (binary data) that contains bit patterns which resemble the control characters used to implement functions 1 and 2 above, without receiving signal identifying those bit patterns as control characters.

Protocols may be divided into three categories according to the message framing technique used. These are character-oriented, byte-count-oriented, and bit-oriented protocols. In this work only the first two categories will be discussed briefly because they have been used in the interface examples in Chapters III and IV. For further details refer to reference (3).

Character-Oriented Protocol

This protocol uses special characters to indicate the

beginning and the end of a message or a block of text, the acceptance of correct messages, and the request for retransmission of faulty messages.

Message framing is accomplished by means of two characters STX and ETB, whereSTX means start of text and ETB means end of a block of characters. ETB requires a response from the receiving process indicating its status. This response is accomplished by means of either character ACK, which indicates that the previous block was accepted without error, or NACK, which indicates that the previous block was received in error and the receiving process is waiting for retransmission of this block. The end of transmission is accomplished by an EOT character, which signals the receiver to terminate its receive sequence. Also the start of transmission is initiated by an ENQ character, by which the transmitter tells the receiver "I have some data to send to you." This requires the receiver to respond with an ACK character.

These characters (STX, ETB, ACK, NACK, EOT, and ENQ) are special control characters, and the bit combinations to form them may be found in the character set for ASCII, EBCDIC and the six bit transcode.

In case of transmitting transparent data (uncoded or binary data), the data may contain bit patterns which resemble the control characters. To avoid false control characters the transmitting process attaches an additional special character called DLE (Data Link Escape) to each real control character to

inform the receiver that it is going to receive transparent data and also to distinguish between real and false control characters. To distinguish between real and false DLE characters the transmitter also attaches a DLE character to each data pattern that resemble DLE. The receiver will discard one DLE and treat the second as data whenever it receives two consecutive DLE characters.

Byte-Count-Oriented Protocol

In this type of protocol the message has a header, which consists of a count that indicates how many bytes follow in the data portion of the message and other information, for example the type of the message. The data portion which comes next is the specified length and is followed by the block check characters (e.g. checksum character). Figure 2.5 shows a typical message format.

START OF TEXT	BYTE COUNT	INFORMATION ABOUT THE MESSAGE	DATA	CHECK CHARACTERS

Figure 2.5 A Typical Message Format
(Adapted from reference (3))

The message acceptance and retransmission procedures are accomplished by means of the special characters ACK and