HOST/CC AND CC/CC ASYNCHRONOUS CONTROL LINE DRIVER IN THE MIMICS NETWORK

by

ERWIN LYNN BEHME

B.S., Kansas State University, 1975

A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1977

Approved by:

Virg Wallentine
Major Professor

TABLE OF CONTENTS

THIS BOOK CONTAINS NUMEROUS PAGES WITH DIAGRAMS THAT ARE CROOKED COMPARED TO THE REST OF THE INFORMATION ON THE PAGE.

THIS IS AS RECEIVED FROM CUSTOMER.

LIST OF FIGURES

ACKNOWLEDGEMENTS

I would like to thank Dr. Virgil Wallentine, my major professor, for suggesting this project and for his help and guidance. I would like to thank Drs. William Hankley and Myron Calhoun for their suggestions concerning the paper. Also, I would especially like to thank Dave Neal, Jim Ratliff, Gary Anderson, and Earl Harris for all the help they have given me. Finally, I would like to thank the Computer Science Department and the Computing Center for the use of their equipment in running the programs and printing the report.

CHAPTER 1

Introduction

The MIni - MIcro Computer System (MIMICS) is being developed at Kansas State University as a network of mini and micro-computers for a distributed data base system. This network is designed to use mini and micro-computers in order to provide the maximum amount of computing power at a minimum cost. MIMICS functions can be at geographically dispersed locations or in clustered activities and only the speed at which these functions are accomplished is affected. The computers in the network are not limited to one manufacturer or type, therefore the links between them must be universal. One type of link used between the computers is an asynchronous line. This paper contains a description of the design and implementation of an Asynchronous Control Line Driver (ACLDR) in the MIMICS network for these lines. The driver handles the functions necessary for sending and receiving of control information between computers within a cluster of the network.

1.1 Structure of the Paper

The remainder of this chapter contains an overview of the MIMICS network architecture based on the description in reference WHA76. It also describes the function of an Asynchronous Control Line Driver (ACLDR) in this network. Chapter 2 presents the reasons for using asynchronous lines in MIMICS. In it are described the asynchronous lines in

general, and then specifically, Interdata's Programmable Asynchronous Single Line Adapter which is used to link two Interdata machines together for this report. Chapter 3 gives the description of the Asynchronous Control Line Driver as implemented using Interdata's Common Assembler Language. Chapter 4 presents the Concurrent PASCAL version of this driver which runs under a KERNEL on the Interdata 16 bit machines. It also describes a special entry point in that Concurrent PASCAL KERNEL which the driver uses. Chapter 5 gives a comparison of the two versions of the driver plus a summary of the work completed and some extensions to conclude the report.

1.2 MIMICS Network Architecture

This section presents an overview of the MIMICS network architecture. It is not intended to give a detailed description of the entire network, but just the necessary elements to give the reader an understanding of how the Asynchronous Control Line Driver functions in MIMICS. This overview is based on reference WHA76, which gives a much more complete look at the network.

The general configuration of the MIMICS network is shown in Figure 1.1. It consists of nodes, which are known as "clusters", connected by low speed synchronous communication paths. It is not necessary that the clusters be geographically remote from each other but they probably will be. Each cluster contains one or more central processing units (CPU's), all located "close" to each other

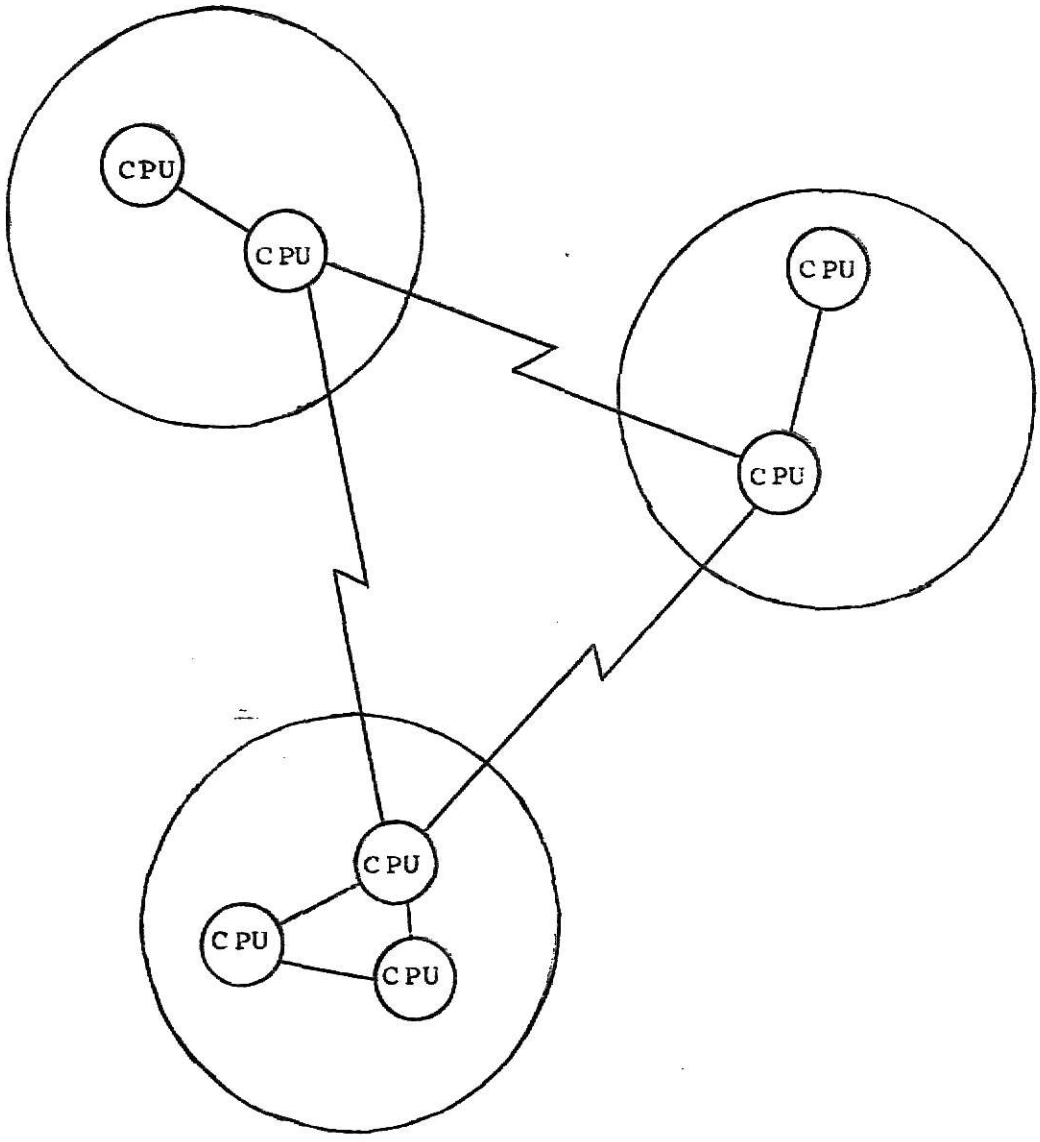

Figure 1.1

General Configuration of MIMICS

(on the order of tens to hundreds of feet). Localizing of these CPU's in a cluster provides for interconnection with high speed synchronous data paths (perhaps 5-10 Megabytes per second transfer rate).

This is the view of the network that the average user might get. The actual makeup of MIMICS is much more complicated. Figure 1.2 shows a possible machine configuration in a cluster. It contains three host CPU's and two Communication Controller (CC) CPU's. The double lines represent high speed data paths and the single lines represent low speed control lines. The circles on the single lines are the hardware interfaces which operate the lines. Most of the burden of network communication is given to the CC's to lessen the load on the host CPU's. Each host is connected to a CC via a KSUBUS. This bus provides high speed data tranfers from the memory of one CPU to that of another on the same KSUBUS under control of local Data Movers (DM). It also allows this fast transfer of data from one KSUBUS to another KSUBUS in the same cluster through remote Data Movers. The job of a CC is to communicate with the hosts on its own KSUBUS and with the other CC's in the cluster to set up the Data Movers to accomplish this movement of data. One CC in each cluster is designated as the enter/exit CC and its function is to handle communication from its cluster to remote clusters across synchronous lines. Asynchronous lines carry control information between the different CPU's within the cluster.

When a CPU requests some data through its operating

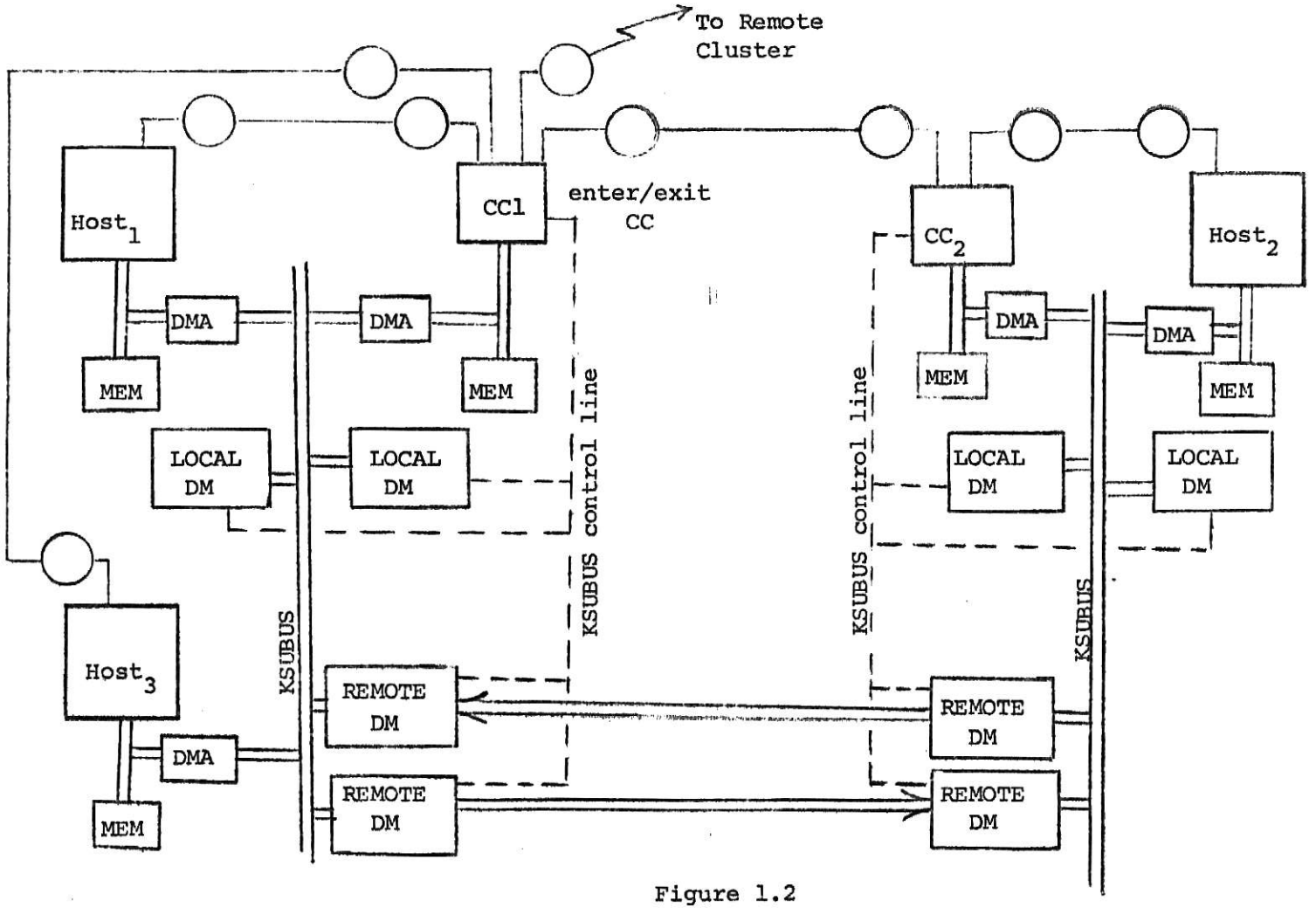

Figure 1.2

Possible Machine Configuration in a Cluster

system, the control computer associated with that host is contacted via the asynchronous line link. The request is then passed to the network message system which takes the necessary steps to find the requested data. There are many distributions of data and many ways of supplying that data to the requesting tasks.

1 - If the requested data exists in the host's memory or the memory of another host on the same KSUBUS, then the CC sets up a high speed copy of the data into the memory area of the requesting user task.

2 - If the data resides in the CC's memory then a similar high speed copy is executed.

3 - If the data is in the memory of a host in the cluster but not on the same KSUBUS, then the CC's associated with the hosts arrange for the transfer of the data using the Remote Data Movers that connect the two KSUBUS's.

4 - If the requested data exists in a remote cluster, it is transferred first to the enter/exit CC of the cluster where the data resides, then across a synchronous line to the enter/exit CC of the cluster of the requesting host, and finally to that host across the KSUBUS.

Of course, these are only a few of the possibilities, but they should give the reader a feel for the types of transfers that may be needed. These examples are also illustrated in Figure 1.3, which was reprinted, with permission, from reference WHA76.

There are two types of computer-to-computer connections using asynchronous lines in the MIMICS network. The first

type is the link between a host computer and its communication controller (CC). In Figure 1.3, the data requests are sent across these asynchronous control lines. The second type, which also uses asynchronous lines, is the connection between communication controllers in a cluster of the network. This type is shown in Figure 1.3b.

The requests for data that are sent across the control lines are set up in the Asynchronous Communication Processes. These processes then call a driver for the asynchronous lines to transfer the requests. The design and implementation of this Asynchronous Control Line Driver is presented in this paper.

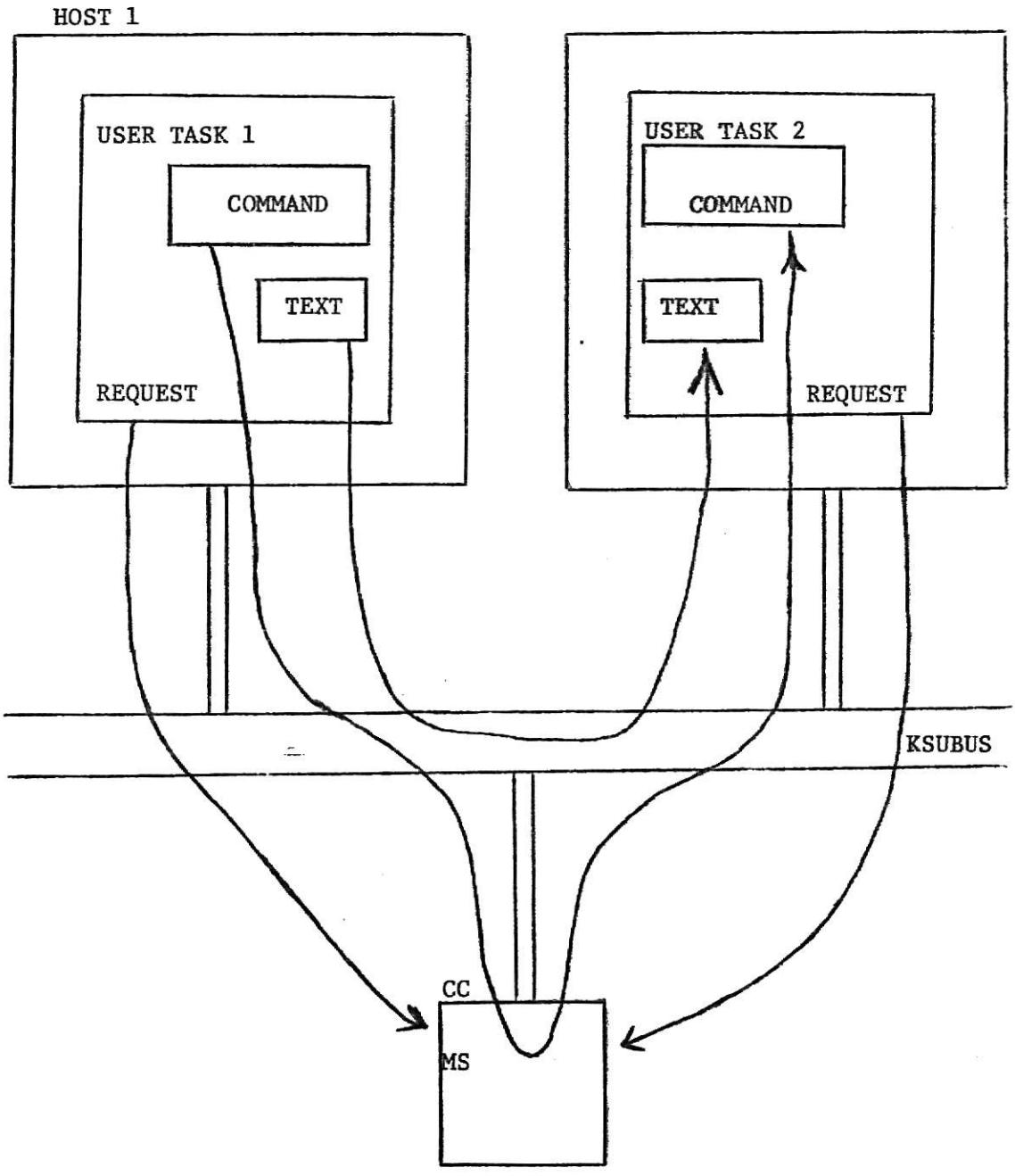

Figure 1.3a

Message Data Flow in MIMICS:
User Tasks with a Common CC,
(Either same host or two hosts on same KSUBUS)

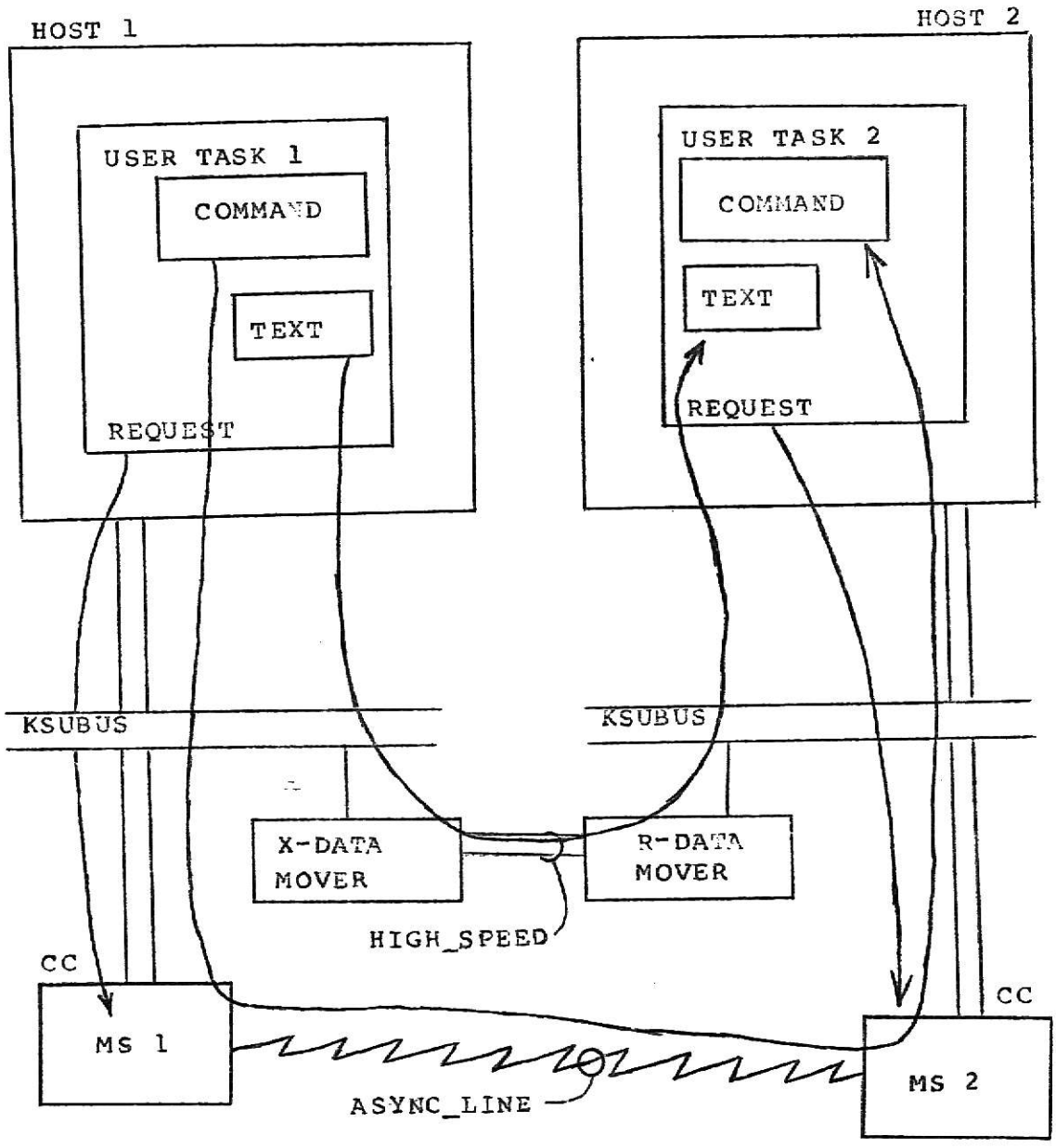

Figure 1.3b

Message Data Flow in MIMICS:
User Tasks in the same Cluster,
but not the same CC.

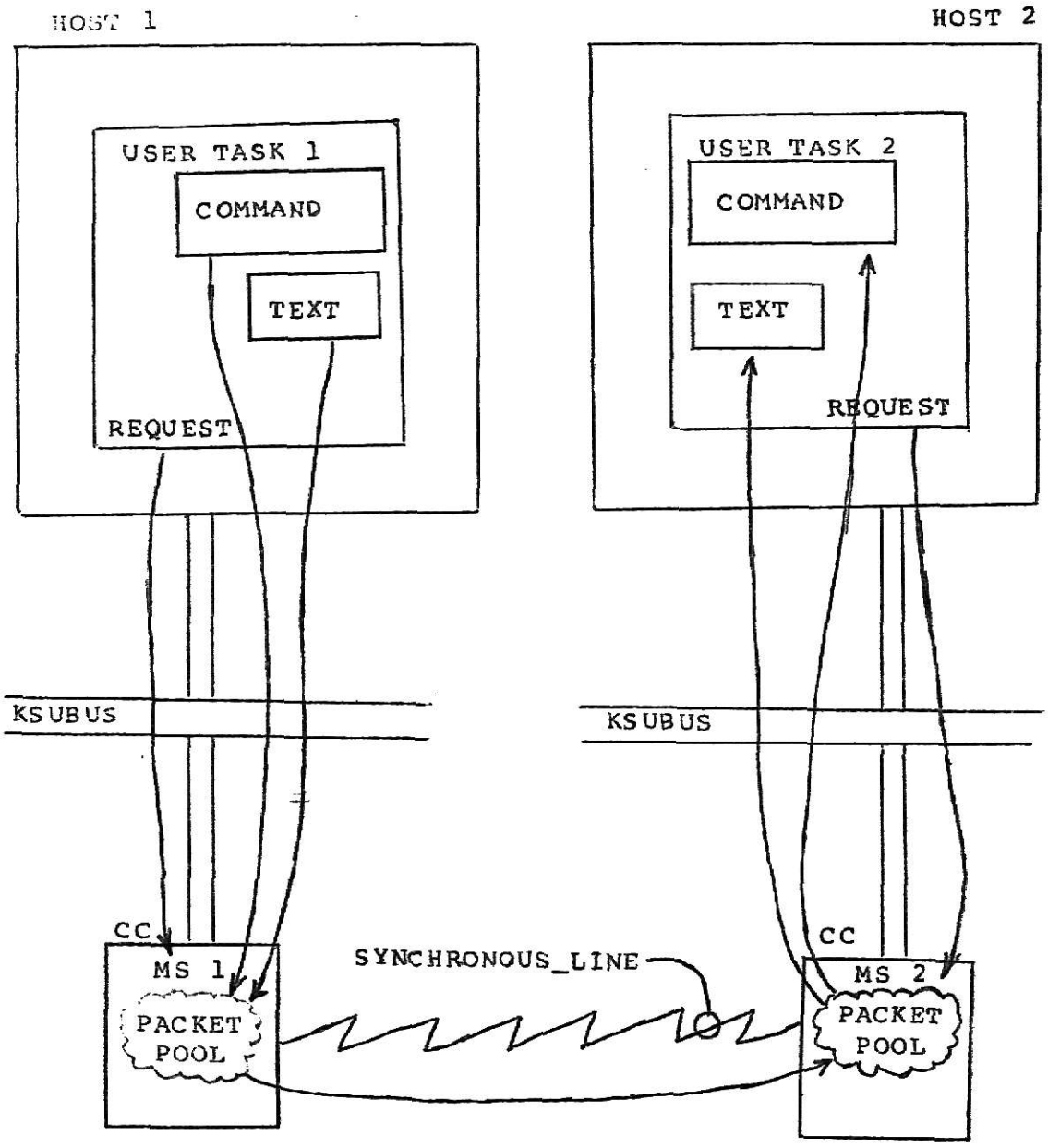

Figure 1.3c

Message Flow in MIMICS:
User Tasks in Different Clusters

CHAPTER 2

Asynchronous Lines

One major problem in designing a network is how to connect the computers together without having to build special interfaces between the machines. One method chosen to transmit control messages for the MIMICS network is to handle communication across asynchronous lines. This chapter contains the basis for this decision, the characteristics of asynchronous lines, and how they are used on the Interdata 16-bit machines.

2.1 Motivation for use of Asynchronous Lines

In an effort to minimize the amount of money and time spent to develop the MIMICS network, asynchronous lines were chosen to carry control messages between computers. This choice was made because "off-the-shelf hardware" for these lines can be used, they are basically universal, and they are relatively straight-forward to operate.

Most manufacturers of computers have designed and built asynchronous line interface hardware that is compatible with their machines, thus eliminating the need for the network developers to design, build, and debug them. This greatly reduces the time and money spent for linking the computers together.

RS-232C is the Electronic Industries Association interface standard for Asynchronous Lines. This is very important because a network can be made up of many different

kinds of computers and if the connections between them are similar, then the interface drivers which control these connections can be similar also. Because of the RS-232C standard, greater portability of the control line driver between machines may be obtained.

All asynchronous interfaces which are RS-232C compatible have common output signals of Data Terminal Ready, Request to Send, and one output data line, plus common input signals of Carrier, Data Set Ready, Clear to Send, Ring and one input data line. These wires are connected to different data sets according to the needs of that data set. The use of these signals for this report will be discussed in the following sections.

Since the interface hardware handles all functions of the actual transmitting and receiving of data across the wire, the device driver is only concerned with transfering data to and from the interface and handling interrupts from that interface. The handling of interrupts differs depending on the machine used, but the different hardware interfaces usually interrupt in like manner, i.e. an interrupt is generated when a character enters the interface and one is generated when a character leaves the interface.

2.2 Characteristics of Asynchronous Lines

Asynchronous lines are characterized by data transmission with an unknown length of time between characters. The data flow across the line is independent of the operation of the central processor and the timing needed

for the hardware interface is part of each character being transmitted. This timing is in the form of a start bit at the beginning of each character and one or two stop bits at the end of the character. These bits are attached to and deleted from the character by the hardware.

In Figure 2.1 we show how the character "S" is transmitted on an asynchronous line. The start bit informs the hardware interface of an incoming character. The 7-bit character is next. The 8th bit of this character is used for parity, followed by either one or two stop bits. In some interfaces, such as Interdata's PASLA [INT01], the value of this bit is determined and checked by the hardware for possible errors in transmission, but in others it must be software controlled. Certain interfaces, like the PASLA, also allow the character length to be as little as 5 bits long. The length of the character is either hardwired on the interface board or under program control.

Since asynchronous lines are predominantly used with modems or terminals, special signals known as Ring, Data Terminal Ready(DTR), Data Set Ready(DSR), Carrier(CARR), Request to Send(RQ2S), and Clear to Send(CL2S) are used along with Transmit Data(TDATA) and Receive Data(RDATA). With a hook up between a terminal and a computer using modems and a telephone line similar to Figure 2.2, the connection and transfer of data from computer to the terminal would be obtained as described in INT05:

1. The operator dials the computer using the terminal's telephone.

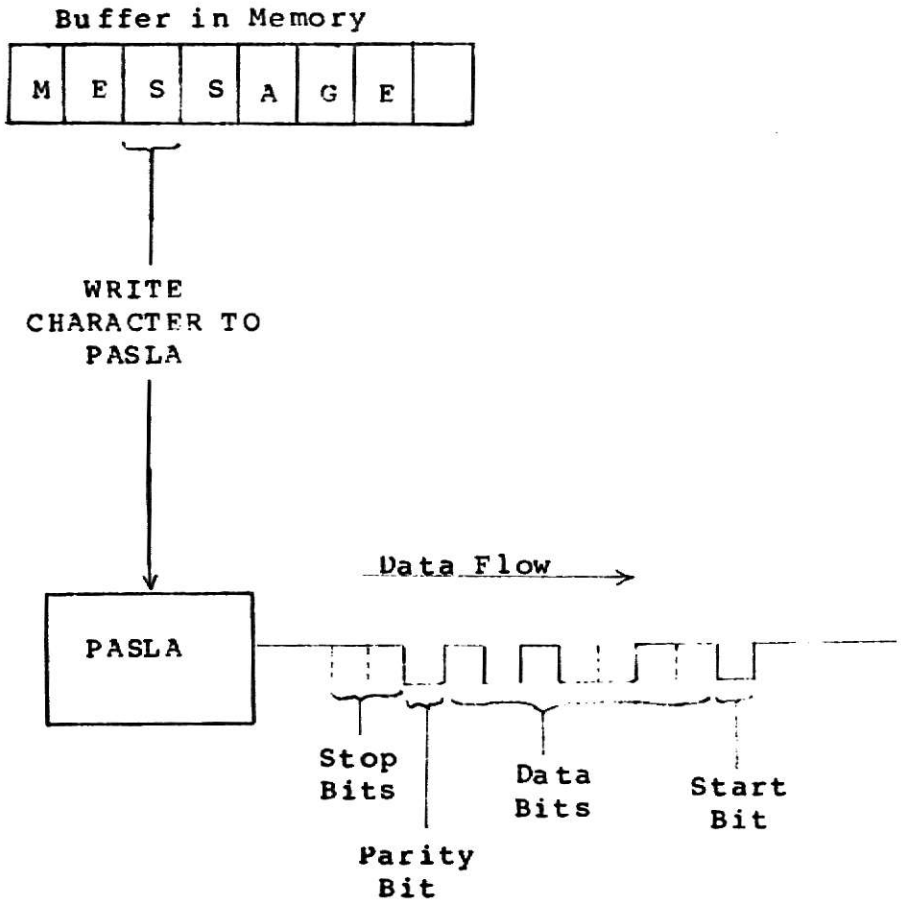

Figure 2.1

Transmission of the Character "S"
on an Asynchronous Line

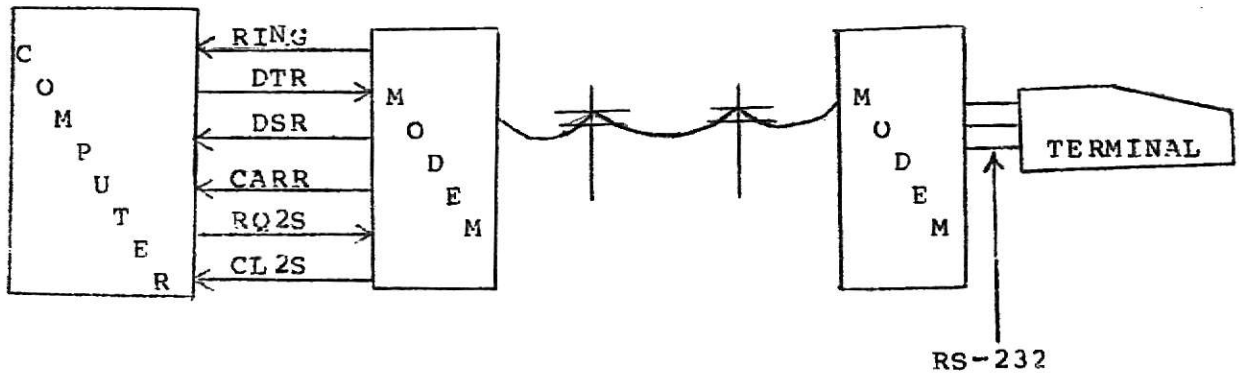

Figure 2.2

Terminal/Computer Connection Using Modems

2. A ringing signal goes through the answering modem to the computer.

3. The computer returns a DATA TERMINAL READY (DTR) signal to the answering modem.

4. The answering modem sends a tone signal to the originating modem. The operator hears the tone and presses the data button of the originating modem.

5. The originating modem sends a DATA SET READY (DSR) signal to the terminal.

6. The answering modem sends a DATA SET READY (DSR) signal to the computer.

7. The modems are now in the data mode.

8. The computer can raise REQUEST TO SEND (RQ2S) which informs the answering modem it wants to transmit data.

9. The answering modem then responds with CLEAR TO SEND (CL2S) and begins transmitting a carrier signal.

10. The originating modem detects CARRIER ON (CO) and informs the terminal that the computer wishes to transmit.

11. On detection of CL2S, the computer can start sending data to the terminal.

12. The terminal receives the data as transmitted.

13. When the computer has finished sending all data, it drops REQUEST TO SEND (RQ2S).

14. The answering modem then stops sending carrier.

Steps 8 through 14 can be repeated (possibly with roles reversed) until either end terminates transmission.

For this report, the number of wires running between machines was kept to a minimum, therefore, several of these signals were not needed. The only ones used were Data Terminal Ready (DTR), Carrier(CARR), Transmit Data (TDATA),

and Receive Data (RDATA). In Figure 2.3 we show that DTR was used to activate the CARR signal of the other machine, and as with normal connections, TDATA was tied to RDATA of each machine. A common ground wire was also run between the machines.

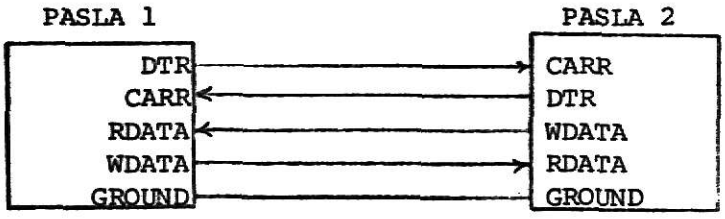

Figure 2.3

Connection of two PASLAs for MIMICS

2.3 Interdata PASLA

For implementation on the Interdata machines the asynchronous lines are controlled by Interdata's Programmable Asynchronous Single Line Adapter (PASLA) [INT01]. The PASLA is a hardware device that is connected to the Multiplexor Bus and provides an interface between the Interdata computer and a number of data sets such as a CRT or a Modem. Depending on the type of data set, the PASLA can be wired for Half-duplex or Full-duplex operation. It can also be wired for two different baud rates, ranging from 47 Baud up to 16K Baud (the value of K is 1024). The choice of which rate to use plus several other options is determined by the programmmer communicating with the PASLA.

He accomplishes this by using the machine language instructions Sense Status, Output Command, Write Data, and Read Data [INT02].

The Sense Status (SS or SSR) instruction is used to interrogate the status of the line and to test whether a character transfer was complete and correct. When an SS instruction is used, the device to which it is directed returns an 8-bit status byte. The format of the Sense Status instruction is:

```
SSR  R1,A2(X2)  Sense Status
SS   R1,R2      Sense Status Register
```

R1 contains an 8-bit device address and the returned status is placed in the second operand. The format of the status byte of the PASLA is shown in Figure 2.4 and explained below.

bits	0	1	2	3	4	5	6	7
	OV	PF or CL2S	FR ERR	RCR	BSY	EX	CARR OFF	RING

Figure 2.4

STATUS BYTE

OV The Overflow bit is 1 when a character that was received by the PASLA was not read before another one was received. The last character to be received is the one that exists in the PASLA.

PF The Parity Flag bit is 1 in read mode if received parity does not agree with the programmmed parity. It is 1 in transmit mode if the Clear to Send(CL2S) signal is not being sent from the data set.

FR ERR The Framing Error bit is 1 if an incoming character has no stop bit.

RCR The Reverse Channel Receive bit is an option with some half-duplex data sets and is used to indicate the state (whether transmitting or receiving) of the data set.

BSY The busy bit is 1 whenever a character is being transmitted or received and indicates that the processor cannot transfer data to or from the PASLA without mutilating the character being transmitted or received.

EX The Examine bit is disabled on the transmit side of the PASLA in Full-duplex mode and is set to one on the receive side if any one of OV ,PF,

or FR ERR bits are set.

CARR_OFF — The Carrier Off bit is one if the Carrier Signal is no longer being received.

RING — The Ring bit indicates that a ring signal is coming from the data set.

In order to set up the PASLA, the programmer needs some way of communicating with it and this is done with the Output Command instruction. The format of the Output Command instruction is:

```
OC   R1,A2(X2)  Output Command
OCR  R1,R2      Output Command Register
```

R1 contains an 8-bit device address and the second operand contains the command byte that is sent to the device. This command byte is of two forms which are shown in Figure 2.5 and explained below:

bits	0	1	2	3	4	5	6	7
COMMAND BYTE 1	DIS	EN	DTR	ECHO PLEX	RCT or DTB	TRANS LB	WRT or RD	1

bits	0	1	2	3	4	5	6	7
COMMAND BYTE 2	X	CLK	BIT SEL		STOP BIT	PARITY		0

Figure 2.5

COMMAND BYTES

Bit 7 tells the PASLA which command is being sent to it and Bit 0 of COMMAND 2 is unused.

PASLA COMMAND 1:

DIS/EN — These two bits are separate for Transmit and Receive. They control the interrupts of the PASLA in the following manner:

0	0	No Change
0	1	Enable
1	0	Disable(interrupt queued)
1	1	Complement(Change State)

DTR — When the Data Terminal Ready bit is 1 then the DTR signal going out of the PASLA is made active

ECHO-PLEX — When this bit is 1, any incoming character is transmitted back out.

RCT/DTB — Reverse Channnel Transmit and Data Terminal Busy are options of certain data sets and are unused in this implementation.

TRANS LB — When on, the PASLA transmits a continuous space. It is unused used in this implementation.

WRT/RD — Inorder to get an interrupt when a character has been transmitted this bit must be active otherwise the hardware holds the busy signal high until a character is received.

PASLA COMMAND 2:

CLK — The PASLA board can be wired for two different baud rates and if this bit is 0 then the lower of the two one is chosen, otherwise the higher one is used.

BIT SEL — These two bits select how many data bits there are per character, not including parity.

Bit 2	3	Number of data bits
0	0	5
0	1	6
1	0	7
1	1	8

STOP BIT — When this bit is 0 then only one stop bit is transmitted, otherwise two stop bits are sent.

PARITY — Parity is set according to the following table:

Bit 5	6	Parity
1	0	ODD
1	1	EVEN
0	x	NONE

Since the only control signal being sent out of the PASLA is Data Terminal Ready and the only one received is the Carrier signal, bits 3 through 6 of COMMAND 1 are not needed and are set to zero. After COMMAND 2 is issued to the PASLA to set the baud rate, the number of data bits, the number of stop bits, and the type of parity to match the two machines which are connected, COMMAND 1 is issued to turn DTR on and set the interrupt conditions for the PASLA. COMMAND 1 must be issued once for the receive side and once for the transmit side if both are to be used. If a character is to be sent to the PASLA then a Write Data instruction is issued. The format of the Write Data instruction is:

```
WDR  R1,A2(X2)  Write Data
WD   R1,R2      Write Data Register
```

where R1 contains an 8-bit device address and the second operand contains the 8-bit byte to be transferred to the PASLA. When the byte is transmitted out of the PASLA, an interrupt is generated by the device if enabled. If interrupts are not desired then the WD instruction can be issued and status of the PASLA can be sensed until the busy flag goes to zero before sending another character. This is known as a "busy wait loop". The disadvantage of using "busy wait loop" is that it ties up the processor until the character has been transmitted. A "busy wait loop" can also be used to wait on a character to be received by the PASLA

or an interrupt can indicate to the processor that a character is available. When a character is in the PASLA, the program can get that character by issuing a Read Data instruction. The format of the Read Data instruction is:

```
RD   R1,A2(X2)  Read Data
RDR  R1,R2      Read Data Register
```

where R1 contains an 8-bit device address and the character is put into the second operand.

2.4 Example of a CRT Driver Using a PASLA

To give the reader a better feel for the operation of a PASLA, an example is given next. It shows the types of output commands that are issued to control the PASLA for a CRT in full duplex mode.

This PASLA is wired for full-duplex mode and a baud rate to match the speed of the CRT. For this example, the format of the characters transferred to and from the CRT will have one start bit, seven bits of data, odd parity, and two stop bits. Because of this configuration, the Output Command issued to set up the PASLA would use a COMMAND 2 byte of the value, binary '01101100'. This value can also be represented as hexadecimal '6C' (X'6C'). The next two Output Commands would be issued to set up the receive and the transmit sides of the PASLA to enable interrupts and activate Data Terminal Ready. The COMMAND 2 byte can be sent to either side but the COMMAND 1 byte has to be sent to both, if both are to be used. The receive side command byte would be a X'61' and the transmit side would be a X'63'.