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

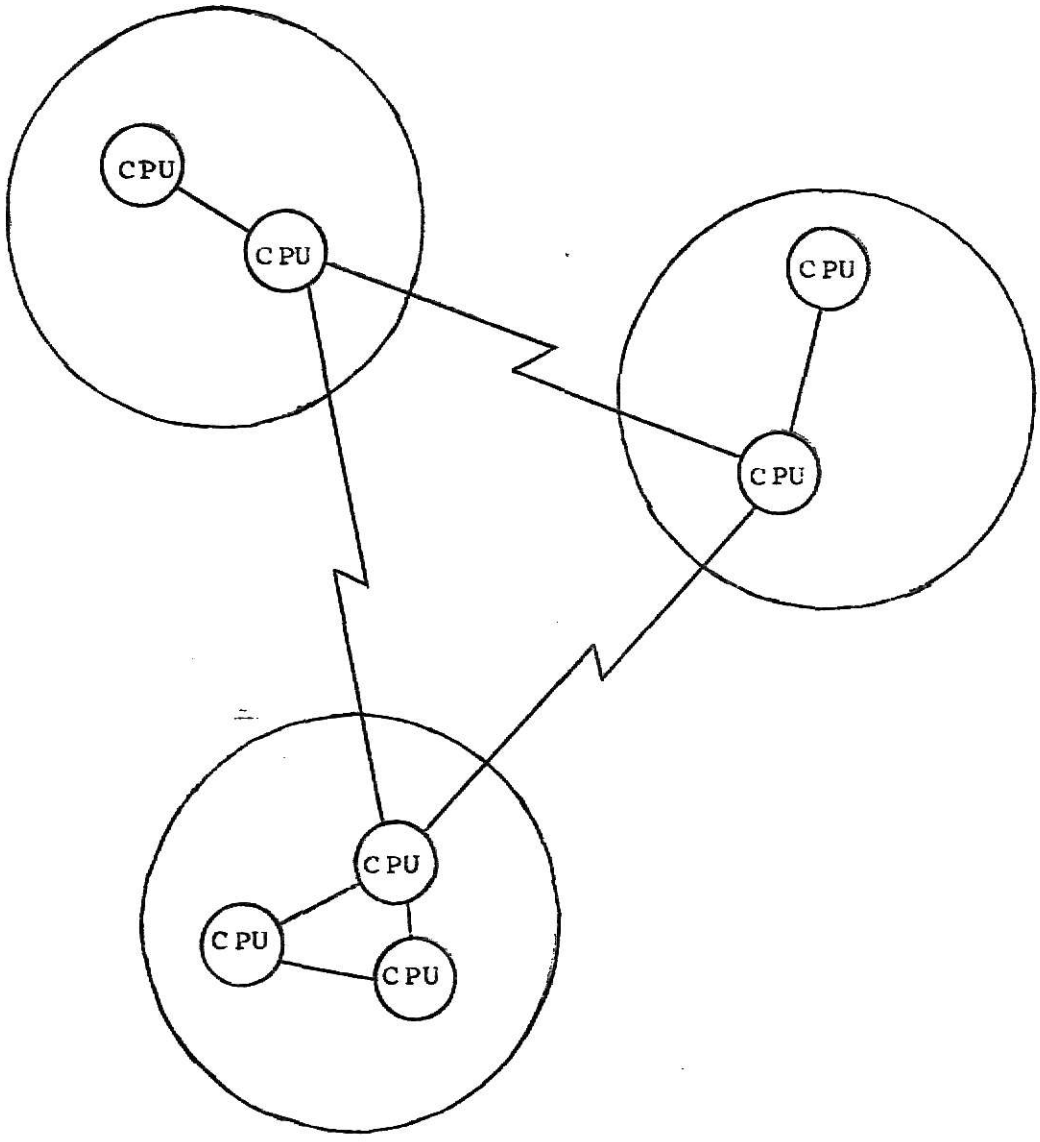

Figure 1.1

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

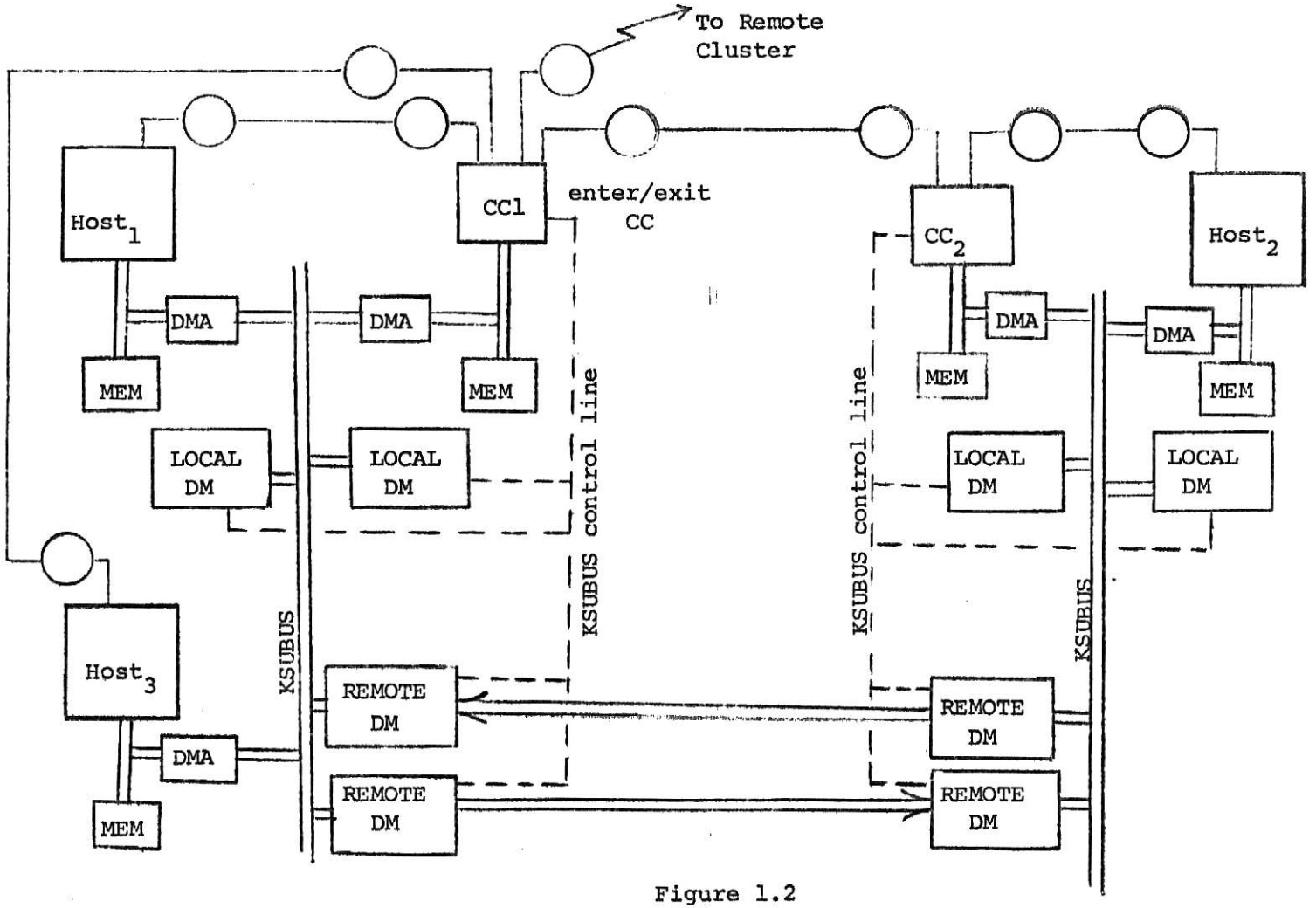

Figure 1.2

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.

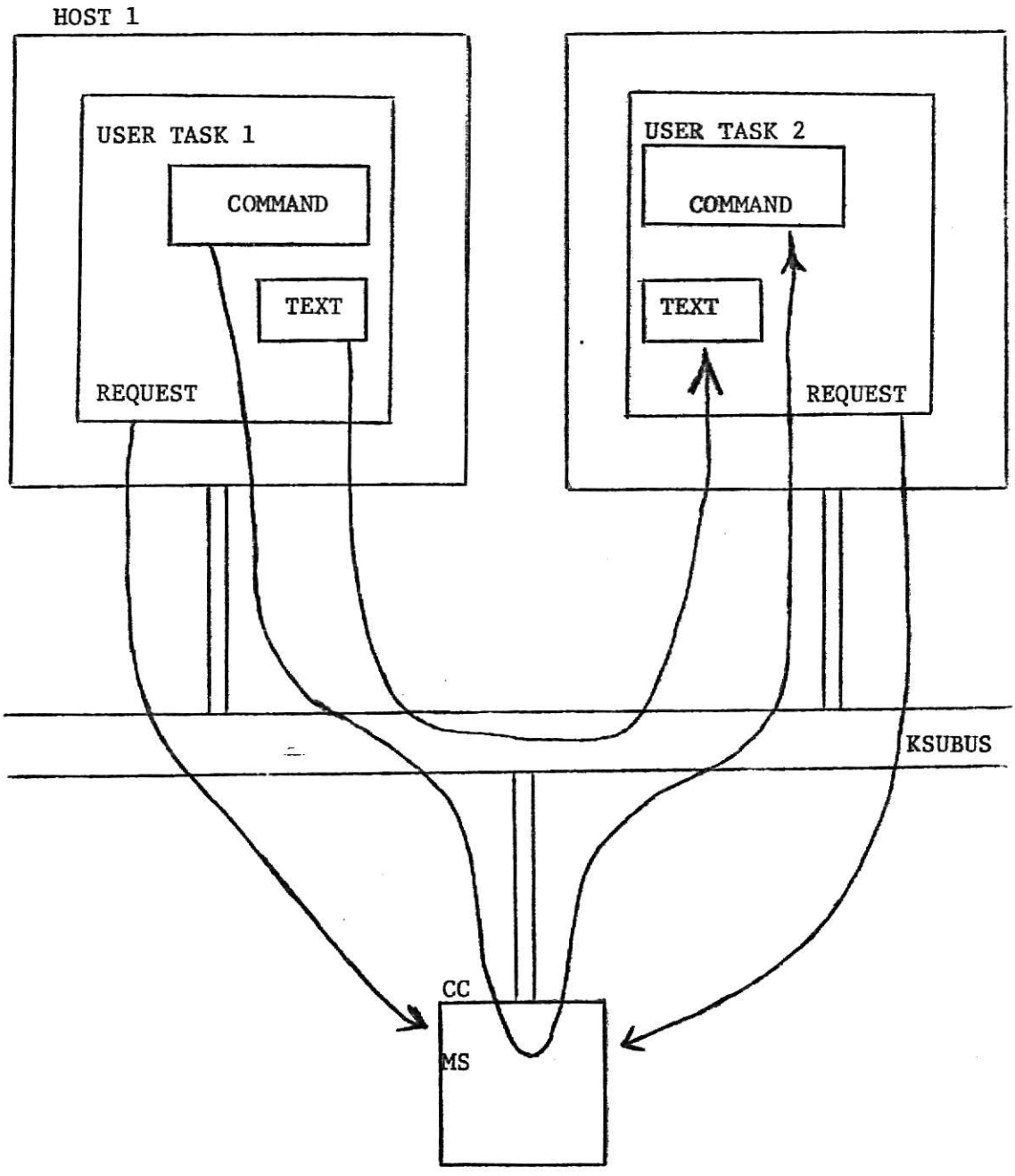

Figure 1.3a

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

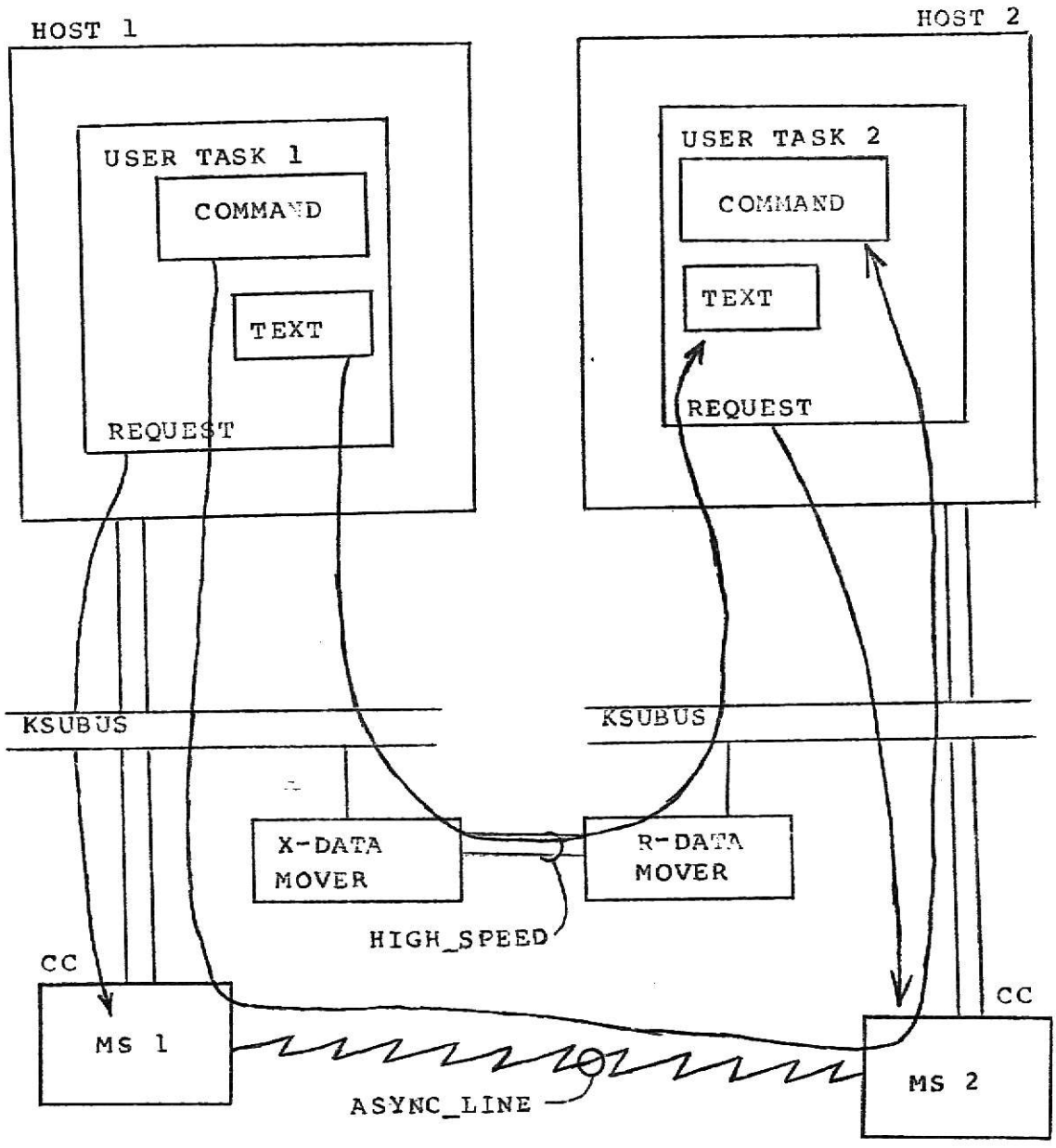

Figure 1.3b

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

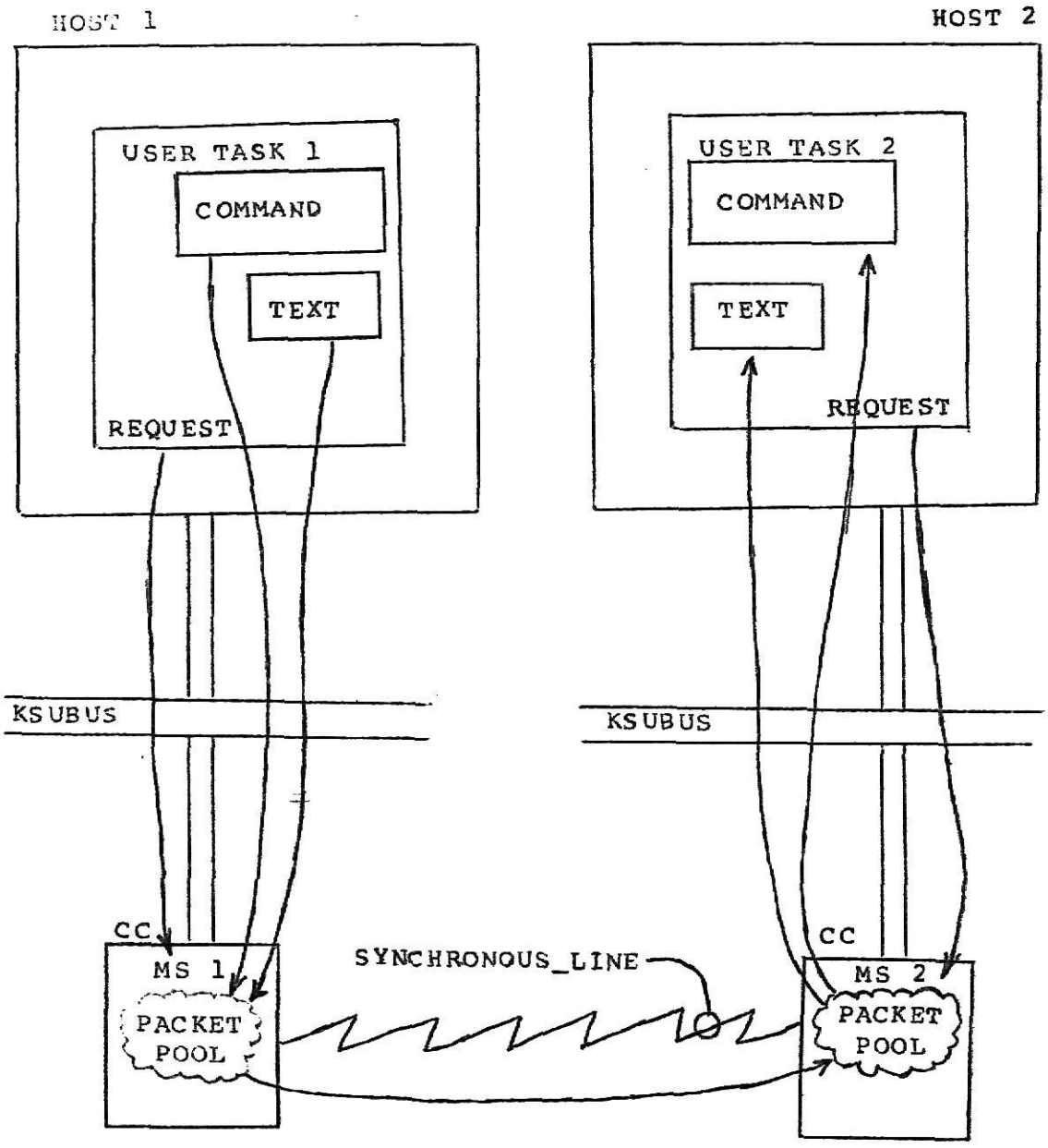

Figure 1.3c

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.

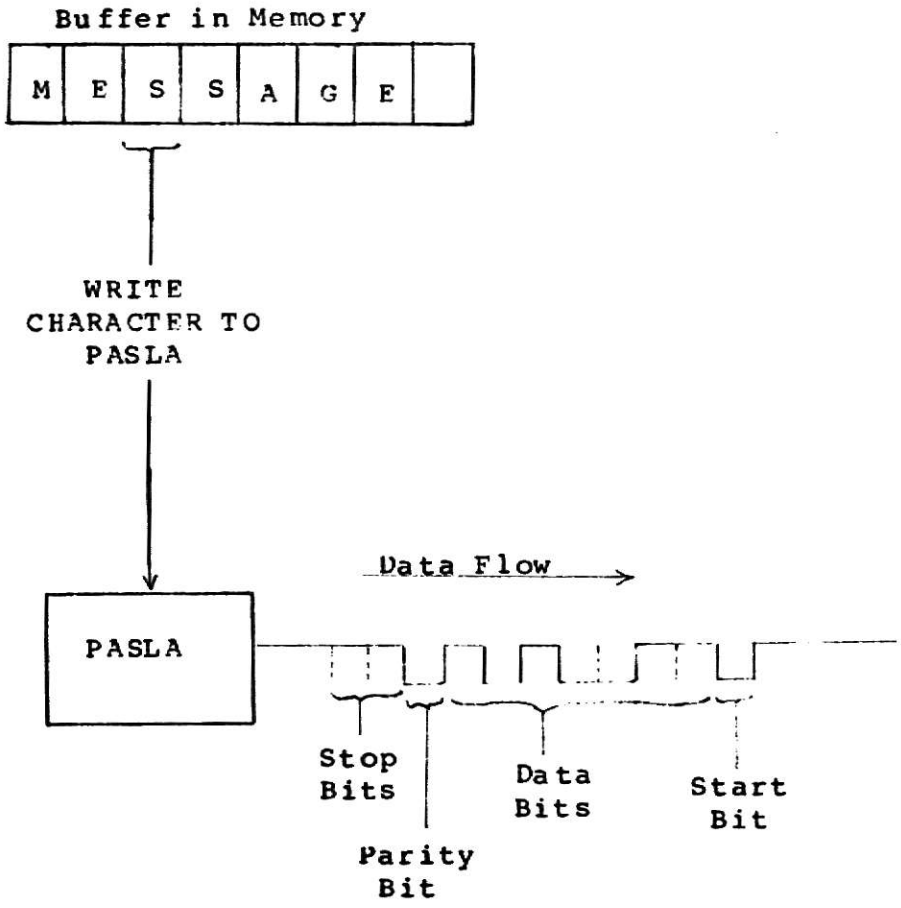

Figure 2.1

Transmission of the Character "S"
on an Asynchronous Line

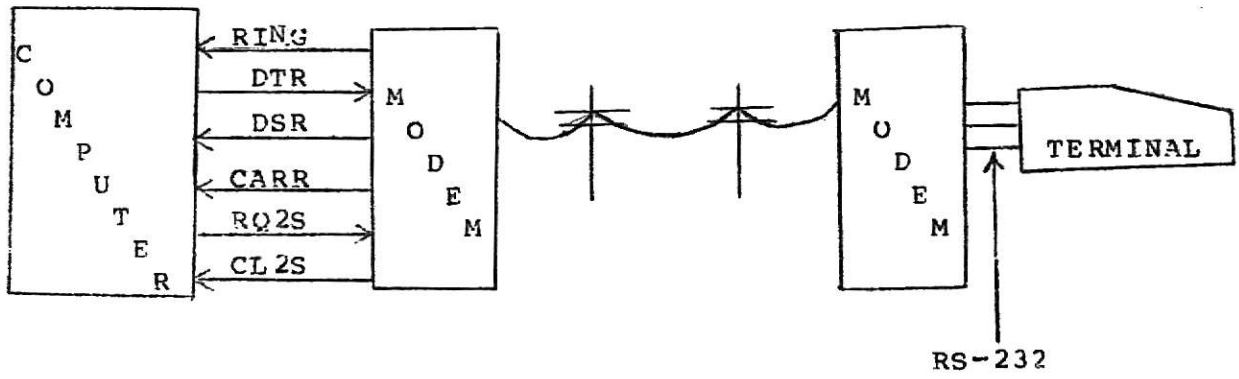

Figure 2.2

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.

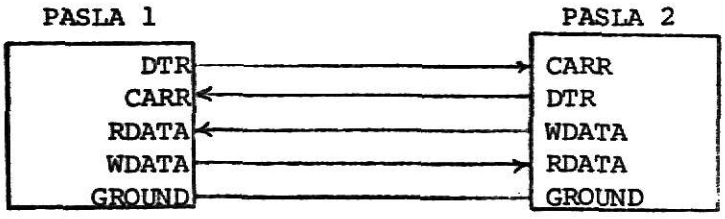

Figure 2.3

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'.

After the PASLA has been initialized and a character is received from the CRT, an interrupt is generated if they are enabled. When the processor recognizes the interrupt, it sends control to a routine which usually reads the character out of the PASLA. When a character is to be transmitted, it is written out to the PASLA. The PASLA then sends it out on the asynchronous line. When the character leaves the PASLA, an interrupt is generated, if enabled, to inform a routine that another character can be sent.

The following program is a simple example which reads in ten characters from a CRT and then echoes all ten back out to the CRT, followed by a carriage return and a line feed. The main routine sets up the PASLA, waits for the message to be read in, starts the transmission of the message by an Output Command to enable interrupts, and then halts. The RCV routine is entered each time there is an interrupt on the receive side of the PASLA. It reads the character and puts it into a buffer. The XMT routine is entered whenever an interrupt occurs on the transmit side of the PASLA. The first interrupt is caused by Data Terminal Ready becoming active from the Output Command. Subsequent interrupts are caused by a character leaving the PASLA, indicating another one can be sent. When the buffer is empty, the Write Data instruction is skipped, terminating interrupts on the transmit side. The code for this example follows:

PROG= *NONE* ASSEMBLED BY CAL/16 00-00

```
                                    2          TARGT  16
                    0001            3  RDEV    EQU    1
                    0002            4  XDEV    EQU    2
                    0003            5  FLAG    EQU    3
                    0004            6  BPTR    EQU    4
                    0005            7  CHAR    EQU    5
0000R  0766                         8          XHR    6,6
0002R  9576                         9          EPSR   7,6          DISABLE PROCESSOR INTERRUPTS
0004R  4810 0058R                  10          LH     RDEV,RADDR   GET RECEIVE DEVICE ADDRESS
0008R  4820 005AR                  11          LH     XDEV,XADDR   GET TRANSMIT DEVICE ADDRESS
000CR  DE10 005CR                  12          OC     RDEV,SETUP   SET UP PASLA
0010R  DE10 005DR                  13          OC     RDEV,ENRCV   ENABLE RECEIVE INTERRUPTS
0014R  9B15                        14          RDR    RDEV,CHAR    DUMMY READ TO INSURE BUSY IS SET
0016R  9567                        15          EPSR   6,7          ENABLE PROCESSOR INTERRUPTS
0018R  C840 005FR                  16          LHI    BPTR,BUFFER  BUFFER POINTER
001CR  C540 0069R                  17  LOOP    CLHI   BPTR,BUFFND1 WAIT FOR MESSAGE
0020R  4230 001CR                  18          BNE    LOOP
0024R  C840 005FR                  19          LHI    BPTR,BUFFER
0028R  DE20 005ER                  20          OC     XDEV,ENXMT   ENABLE TRANSMISSION AND START
002CR  4300 002CR                  21  STOP    B      STOP
                                   22  *
                                   23  ***** RECEIVE INTERRUPT SERVICE ROUTINE
                                   24  *
0030R  0000                        25  RCV     DC     0,0,X'3000'  OLD PSW AND NEW STATUS
0032R  0000
0034R  3000
0036R  DB14 0000                   26          RD     RDEV,0(BPTR) STORE CHARACTER IN BUFFER
003AR  2641                        27          AIS    BPTR,1       INCREMENT BUFFER POINTER
003CR  C200 0030R                  28          LPSW   RCV          RETURN
                                   29  *
                                   30  ***** TRANSMIT INTERRUPT SERVICE ROUTINE
                                   31  *
0040R  0000                        32  XMT     DC     0,0,X'3000'  OLD PSW AND NEW STATUS
0042R  0000
0044R  3000
0046R  C540 006BR                  33          CLHI   BPTR,BUFFND2 END OF MESSAGE ?
004AR  4330 0054R                  34          BE     XDONE        BRANCH IF YES
004ER  DA24 0000                   35          WD     XDEV,0(BPTR) SEND OUT CHARACTER
0052R  2641                        36          AIS    BPTR,1       INCREMENT BUFFER POINTER
0054R  C200 0040R                  37  XDONE   LPSW   XMT          RETURN
                                   38  *
                                   39  *
0058R  0016                        40  RADDR   DC     X'16'        RECEIVE DEVICE ADDRESS
005AR  0017                        41  XADDR   DC     X'17'        TRANSMIT DEVICE ADDRESS
005CR  6C                          42  SETUP   DB     X'6C'        PASLA SET UP COMMAND
005DR  61                          43  ENRCV   DB     X'61'        ENABLE INTS, DTR, READ, COMMAND1
005ER  63                          44  ENXMT   DB     X'63'        ENABLE INTS, DTR,READ, COMMAND1
005FR                              45  BUFFER  DS     10           BUFFER - 10 CHARACTERS LONG
                    0069R          46  BUFFND1 EQU    *
0069R  0D                          47          DB     X'0D'        CARRIAGE RETURN
006AR  0A                          48          DB     X'0A'        LINE FEED
                    006BR          49  BUFFND2 EQU    *
006CR                              50          END
```

NO ERRORS

CAL/16 00-00

ABSTOP	0000
ADC	0002
BPTR	0004
BUFFER	005FR
BUFFND1	0069R
BUFFND2	006BR
CHAR	0005
ENRCV	005DR
ENXMT	005ER
FLAG	0003
IMPTOP	006CR
LADC	0001
LOOP	001CR
RADDR	0058R
RCV	0030R
RDEV	0001
SETUP	005CR
STOP	002CR
XADDR	005AR
XDEV	0002
XDONE	0054R
XMT	0040R

CHAPTER 3

Asynchronous Control Line Driver - an Assembler Version

The decision was made to use Asynchronous lines to carry control supervisor information between machine message systems within a cluster of the MIMICS network. Therefore, a driver was needed for these lines and it is called the Asynchronous Control Line Driver (ACLDR). The ACLDR acts as an interface between the Asynchronous Communication Processes and the PASLA hardware.

The ACLDR is designed to set up the PASLA and transfer the control information across asynchronous lines in the form of System Control Blocks (SCB). Figure 3.1 shows the format of an SCB that is being transmitted.

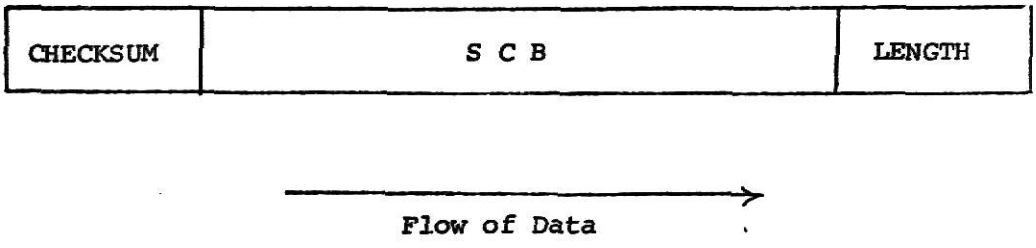

Figure 3.1

Format of SCB Being Transmitted

The length, which is transmitted first, is one byte long and is equal to the number of bytes in the SCB excluding the length and checksum bytes. Each byte of the SCB is sent

next followed by a checksum. This checksum is the Modulo 256 sum of the length and all the bytes of the SCB. This chapter describes the ACLDR and how it was implemented on the Interdata 16 bit machines, using Interdata's Common Assembler Language (CAL) [INT02], to transfer SCB's across asynchronous lines.

3.1 Interrupt Structure

Before getting into the details of the ACLDR, one must understand how the Interdata machines handle interrupts. The Interdata has several types of interrupts, such as External, Machine Malfunction, and Fixed Point Fault. These are explained in reference INT02. This section is concerned only with Automatic I/O interrupts which are used for the ACLDR.

When the processor detects an interrupt, it saves the current state of the machine and transfers control to a software routine to handle the interrupt. Automatic I/O interrupts can be enabled or disabled by setting or resetting bits in the Program Status Word. The Program Status Word (PSW) [INT02] is a register in the machine which defines the state of the processor at any given time. If both bits 1 and 4 in the PSW are set then Automatic I/O interrupts are enabled. If bit 1 or both bits 1 and 4 are reset then they are disabled.

When an interrupt occurs and both bits 1 and 4 of the PSW are set, the microcode of the machine takes the interrupting device address, multiplies it by two, and uses

this value as a displacement into the Interrupt Service Table (IST). The IST starts at memory location X'D0' and contains the addresses of the Interrupt Service Routines (ISR) for 255 devices that could be in the system. Referring to Figure 3.2, if device X'11' gave an interrupt the microcode would add 2 * X'11' to X'D0' to obtain X'F2' which would be the memory location that contains the address of the ISR for device X'11'. When the ISR is entered, the PSW at the time of the interrupt is stored in the first two halfwords of the ISR and the new status is put into the PSW from the third halfword. The new instruction counter in the PSW points to the fourth halfword and execution of the ISR begins there. The Load PSW instruction loads a new state of the processor from a location in memory into the PSW register and when the ISR is to be exited, this instruction is executed with the OLD PSW that was stored in the first two halfwords of the ISR. In doing so the routine that was interrupted regains control at the point where the interrupt occurred.

3.2 Relationship to Asynchronous Processes

As shown in Figure 3.3, the Asynchronous Communication Processes call the ACLDR via one of its three entry points, ACL.SETUP, ACL.RCV, or ACL.XMT. These entry points communicate with the PASLA and the interrupt service routines to set up the PASLA and transfer bytes of data to and from it.

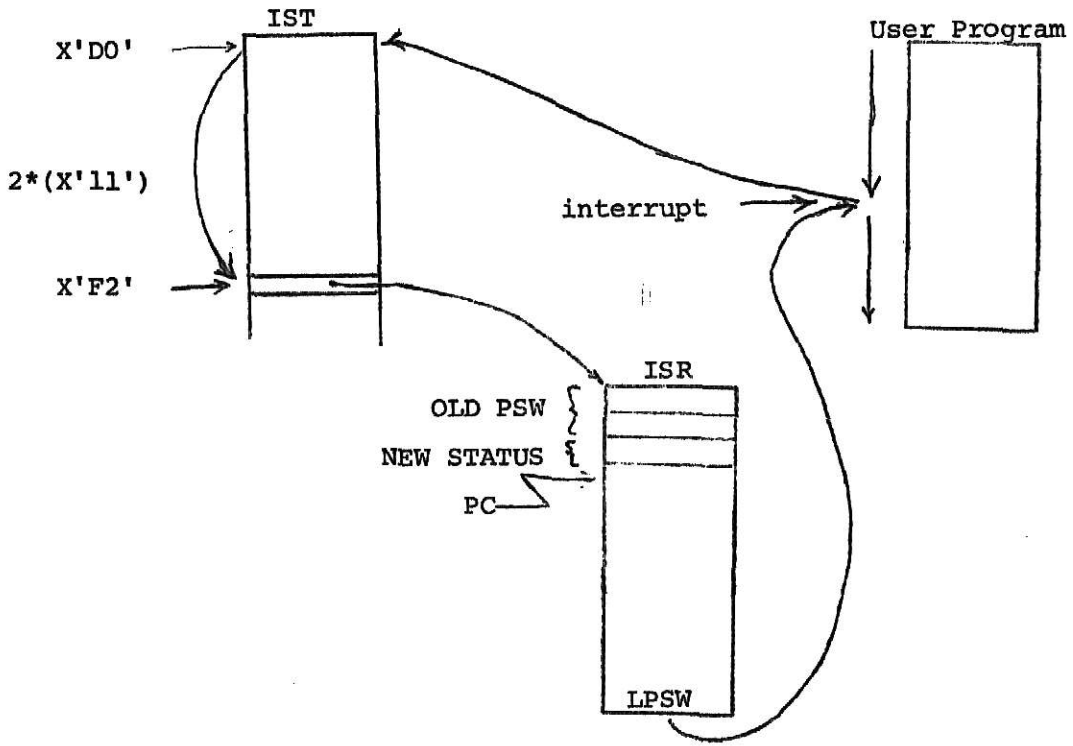

Figure 3.2

Transfer of control caused by an interrupt
from device X'11'

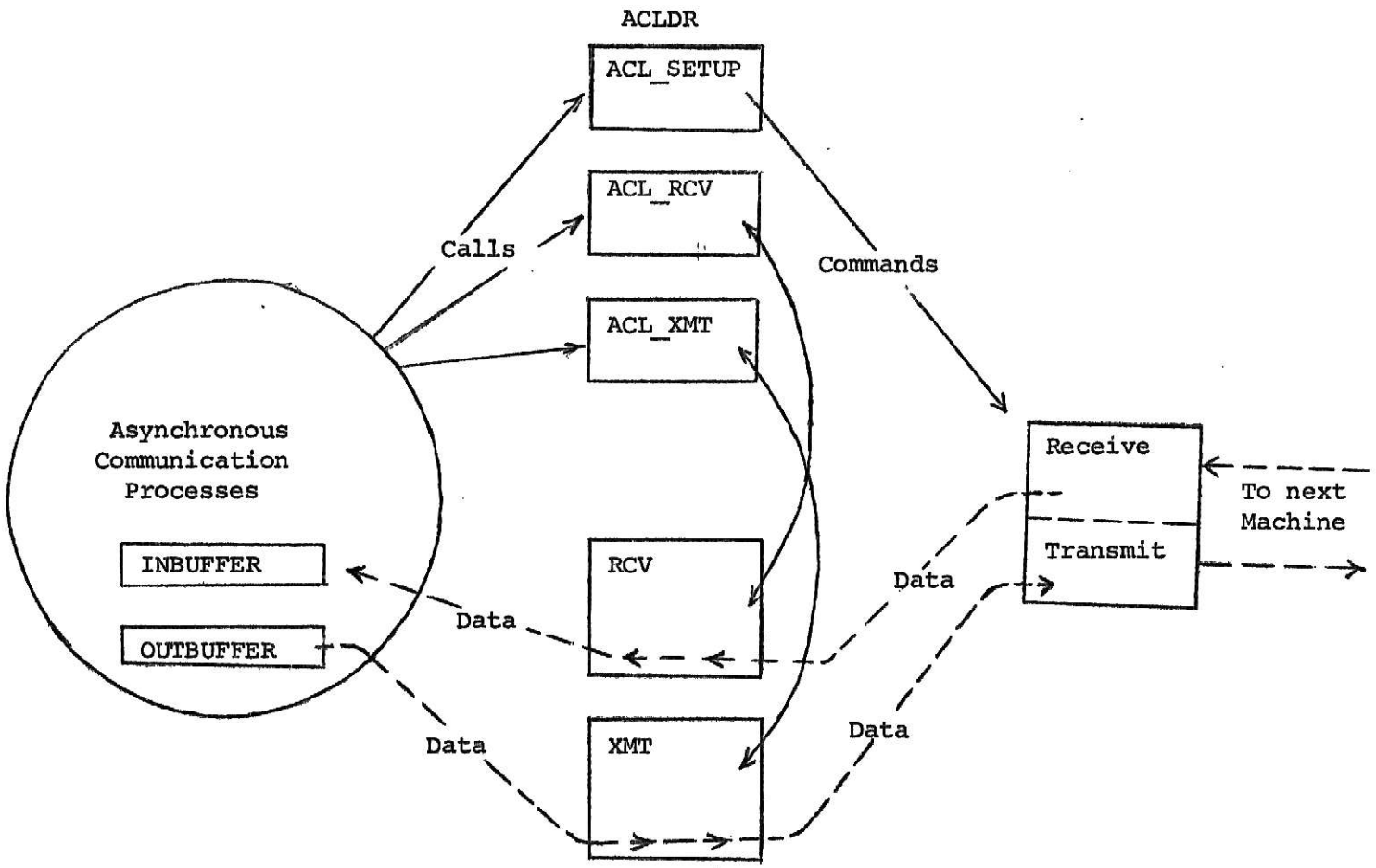

Figure 3.3

Relationship of ACLDR to Asynchronous Processes

3.3 Module Layout

In Figure 3.4 the modules of the ACLDR are presented. The ACLDR is reentrant so a block of memory is allocatted for all the variables used for each PASLA (Receive/Transmit pair). This block of memory is called the Device Variable Table (DVT). The address of the DVT for a particular device is found in the Device Variable Table Map (DVTMAP). Each entry point in the ACLDR uses the physical line address passed to it as a displacement from the beginning of the DVTMAP to obtain the address of the DVT for that particular device.

When an interrupt occurs on a PASLA, the Device Map (DEVMAP), corresponding to the device which caused that interrupt, is entered. The DEVMAP is similar to Interdata's DCB [IDAT03]. It saves the current state of the machine, sets up the new state for handling the interrupt, and then branches to the Interrupt Service Routine. In the new machine state, interrupts to the processor are disabled and one register contains the address of the DVT. All the variables used by the ISR are obtained by a displacement from this address. The ISR is exited by a branch back to the DEVMAP where the previous state of the machine is restored and the processor resumes execution of the code which was interrupted.

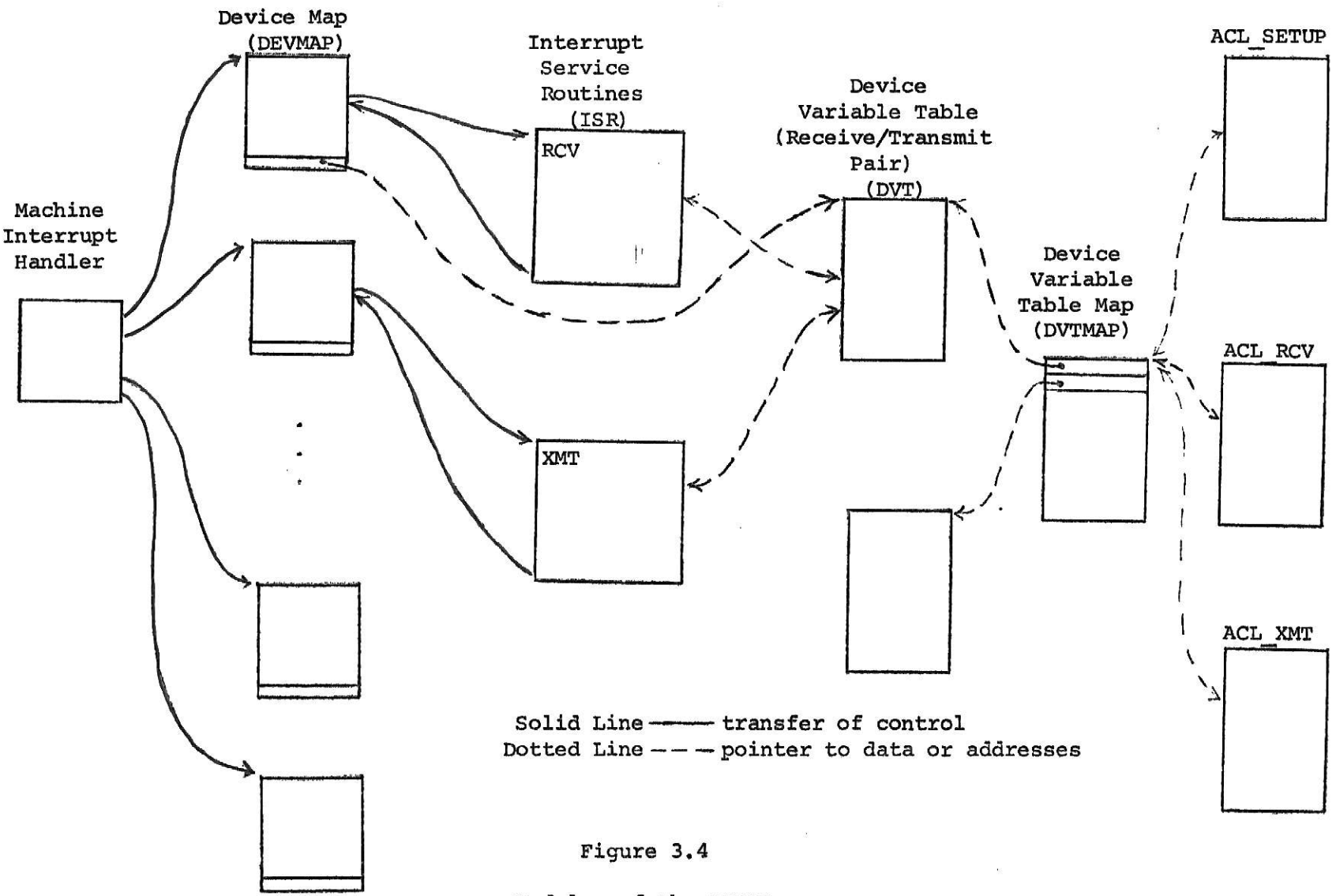

Figure 3.4

Modules of the ACLDR

3.4 ACLDR entry Points

The Asynchronous Communication Processes can call any one of the three entry points in the ACLDR, ACL.SETUP, ACL.RCV, or ACL.XMT. Before doing so, the Asynchronous Process must store the parameters or addresses of the parameters to be passed in an array of contiguous halfwords (two bytes) in memory. The address of this array is stored in General Purpose Register 1 and the return address from the ACLDR is in General Purpose Register 15. Each entry point's functional specifications, the parameters passed to them, and high level algorithms of their code are given below.

ACL_SETUP(line_addr,return_code)

Functional Specifications:

ACL_SETUP is called by the Asynchronous Processes to set up the PASLA corresponding to the line_address passed as a parameter to allow transmission of an SCB.

Parameters:

line_addr	- entry parameter - Physical line address of the PASLA being initialized.
return_code	- return parameter - The return code to the calling routine. There are two conditions that can be returned: 0 - OK 1 - Instruction timeout

Algorithm:

```
ACL_SETUP(line_addr,return_code)
     DISABLE INTERRUPTS
     SET UP THE PASLA
     IF "INSTRUCTION TIMEOUT" THEN
          RETURN_CODE = "INSTRUCTION TIMEOUT"
     ELSE RETURN_CODE = "OK"
     ENABLE INTERRUPTS
     RETURN.
```

ACL_RCV(line_addr,in_buff_addr,length,max_size,return_code)

Functional Specifications:

ACL_RCV is called by the Asynchronous Processes when an SCB is expected to be received. The incoming SCB is placed in the input buffer. If an SCB was not received after a certain length of time, then the return code indicates this condition and control returns to the calling routine.

Parameters:

line_addr	- entry parameter - Physical line address of the PASLA used to receive the SCB.
in_buff_addr	- entry parameter - The address of the buffer where the SCB to be received will be stored.
length	- return parameter - The length of the SCB that was received.
max_size	- entry parameter - The maximum size of the input buffer.
return_code	- return parameter - The return code to the calling routine. There are five conditions that can be returned: 0 - Reception OK 1 - Loss of carrier indicating bad line condition 2 - Bad Checksum; error in transmission 3 - Timeout; SCB was not received in allotted amount of time 4 - Length to large; length received was larger than buffer length

Algorithm:

```
ACL_RCV(line_addr,in_buff_addr,length,maxsize,return_code)
     SET RECEIVE_MODE TO ALLOW RECEIVING OF CHARACTERS
     DO WHILE TIMEOUT HAS NOT OCCURRED
     IF CARRIER WAS LOST THEN
          SET RETURN_CODE = "LOSS OF CARRIER"
          CLEAR RECEIVE_MODE
          RETURN
     IF RECEPTION OF SCB IS DONE THEN
          IF CHECKSUM IS BAD THEN
               SET RETURN_CODE = "BAD CHECKSUM"
          ELSE SET RETURN_CODE = "OK"
          CLEAR RECEIVE_MODE
          RETURN
     END WHILE
     CLEAR RECEIVE_MODE
     SET RETURN_CODE = "TIMEOUT"
     RETURN.
```

ACL_XMT(line_addr,out_buff_addr,length,return_code)

Functional Specifications:

ACL_XMT is called by the Asynchronous Processes for transmission of an SCB from the output buffer. The length of this SCB is transmitted just before, and a checksum right after, the SCB itself. If the loss of carrier is detected during transmission then the return code indicates this fact, otherwise OK is returned.

Parameters:

line_addr	- entry parameter - The physical line address of the PASLA used to transmit the SCB
out_buff_addr	- entry parameter - The address of the SCB to be transmitted
length	- entry parameter - The length of the SCB being transmitted. This length is attached to the beginning of the SCB
return_code	- return parameter - The return code to the calling routine. There are two conditions that can be returned: 0 - OK 1 - Loss of carrier

Algorithm:

```
ACL_XMT(line_addr,out_buff_addr,length,return_code)
     SIMULATE AN INTERRUPT TO START TRANSMISSION
     DO WHILE TRANSMISSION TAKING PLACE
     IF LOSS OF CARRIER THEN
          SET RETURN_CODE = "LOSS OF CARRIER"
          RETURN
     END WHILE
     SET RETURN_CODE = "OK"
     RETURN.
```

3.5 Interrupt Service Routine:

RCV

Functional Specifications:

RCV is a reentrant procedure used to service interrupts coming from the receive side of a PASLA. The RCV routine ignores all interrupts until ACL_RCV sets the Receive Mode flag on. It then receives the length of the incoming SCB, the SCB itself, and a checksum. If the length is longer than the buffer size, then only enough characters to fill up the buffer will be read in. The excess will be ignored, resulting in a checksum error. If the length is correct but the checksum is wrong then the return code from ACL_RCV will indicate so.

Algorithm:

```
IF LOSS OF CARRIER IS DETECTED THEN
     SET CARRIER FLAG
     RETURN
IF THE RECEIVE MODE IS NOT ACTIVE THEN RETURN
READ THE CHARACTER
IF IT IS THE FIRST CHARACTER THEN
     SET THE BUFFER POINTER
     SET LENGTH = VALUE OF THE CHARACTER
     RETURN
IF THE BUFFER IS FULL THEN
     IF THE CHECKSUM = THE CHARACTER
          THEN CHECKSUMFLAG = "OK"
     ELSE CHECKSUMFLAG = "BAD CHECKSUM"
     SET THE DONE FLAG
     DISABLE RECEIVE MODE
     RETURN
ELSE
     STORE THE CHARACTER IN THE BUFFER
     ADD IT TO THE CHECKSUM
     INCREMENT THE BUFFER POINTER
     RETURN.
```

XMT

Functional Specifications

XMT is a reentrant procedure used to service interrupts comming from the transmit side of a PASLA. When a process wishes to send a message across an asynchronous line it calls ACL_XMT. ACL_XMT sets up the necessary variables to transmit a buffer and then simulates an interrupt to send control to XMT. XMT then transmits a character from the buffer. Each time a character leaves the PASLA another interrupt is generated which informs XMT that it may send another character. XMT first transmits the length of the buffer, then each character in the buffer, and then a checksum.

Algorithm:

```
IF DATA TERMINAL READY INTERRUPT THEN RETURN
IF LOSS OF CARRIER DETECTED THEN
     SET CARRIER FLAG
     RETURN
IF BEGINNING OF BUFFER THEN
     TRANSMIT LENGTH
     RETURN
IF BUFFER EMPTY THEN
     IF CHECKSUM HAS BEEN SENT THEN
          RETURN
     ELSE TRANSMIT CHECKSUM
     SET THE DONE FLAG
     RETURN
ELSE
     ADD CHARACTER TO CHECKSUM
     TRANSMIT THE CHARACTER
     INCREMENT THE BUFFER POINTER
     RETURN.
```

3.6 Notes on ACLDR Operation

The user of the ACLDR must know about certain assertions and conditions necessary for its correct operation. The ACLDR assumes that any parameter that is sent to it is a legitimate variable, i.e. the line_addresses are actually in the system, the buffer_addresses do not exceed the top of memory, etc. Also, when loss of carrier occurs because of a line break, hardware malfunction, etc., this condition is hidden from the Asynchronous Processes until the ACLDR is called to transmit or receive an SCB.

CHAPTER 4

Asynchronous Control Line Driver - a Concurrent PASCAL Version

Because of the difficulty in programming and debugging in assembler language, the ACLDR was also written in Concurrent PASCAL. This high level language was designed and implemented by Per Brinch Hansen [BRH75a] on a PDP 11/45. It has also been "ported" to an Interdata 8/32 by KSU personnel [NAR76]. The NAVY has developed a KERNEL [COZ76] which runs on Interdata 16-bit machines and Concurrent PASCAL programs compiled on the 8/32 run under this KERNEL. The remainder of this chapter first describes the input/output machine of the NAVY's KERNEL and then the implementation of the ACLDR in Concurrent PASCAL.

It is assumed that the reader is familiar with Concurrent PASCAL and the compilation of programs under the SOLO operating system [BRH76] [NAR76]. Since the 16-bit Interdata machines' amount of memory is too small to run a PASCAL compiler, the KERNEL and any Concurrent PASCAL programs running under that KERNEL must be compiled on the 8/32 using the SOLO system and transferred via mag tape, disk, etc. to the 16-bit machines.

4.1 IO_MACHINE

The KERNEL which runs on the 16-bit machines contains an entry point called the IO_MACHINE [COZ76]. This entry

point was designed to allow the user to control the operation of a device, independent of a particular machine or device. This eliminates the need for the user to handle the interrupts at the hardware level and also hides from him the actual method of transferring data to and from the device. This however, does not relinquish his control of the device. The IO_MACHINE instructions handle the usual I/O functions of issuing commands, sensing status, and reading or writing of data for each device. Although a lot of the work is done by the IO_MACHINE, the user must still understand the functional operation of the device in order to know which output commands control it and to understand the status that is returned. A device, in this context, is the actual hardware interface which handles the transfer of data to and from a terminal, disk, etc. or in the case of this report, to and from another hardware interface, i.e. a PASLA.

There are 21 instructions available for the user of the IO_MACHINE. These instructions can be divided into three categories. First, the device control instructions consist of CHANNEL, COMMAND, SENSE, WAIT_INTERRUPT, PREEMPT, and FREE. Second, the flow of execution is controlled by the COMPARE, JUMP, and RETURN instructions. The final category, data transfer instructions, are DATA, START_INPUT, and START_OUTPUT. The explanation of each individual instruction can be found in Appendix A. This explanation is contained in reference COZ76; It was included because it was not a permanent record at KSU as of April 1977, and was

found to be concise. A description of the instructions by this author would be redundant so the remainder of this section is dedicated to an explanation of how the IO_MACHINE is used followed by a simple example.

When the IO_MACHINE is called, the parameter passed to it is a record in the form shown below:

```
io_record = RECORD
     instruction_counter: integer;
     status: integer;
     bytes_transferred: integer;
     internal_use: ARRAY [0..12] OF integer;
     inst: ARRAY [0..max_instruction] OF integer
     END "record";
```

The instruction_counter is the index of which instruction will be executed next in the instruction array. The status contains the device status in the lower byte of the integer and IO_MACHINE status in the upper byte upon exit from the IO_MACHINE. The number of bytes transferred is given next. The internal_use array is for the IO_MACHINE to make the code reentrant and is of no use to the user. The "inst" variable is an array of integers specifying the IO_MACHINE instructions and must have the starting index at 0. These integers are the instructions interpreted by the IO_MACHINE. Each instruction will be referenced in this report by name (e.g. IO_COMMAND or IO_RETURN) and not by its integer value.

The instruction array can be set up at either initialization time or at run time. Particular instructions can also be added or changed at run time as long as the instruction array index does not exceed the maximum number

of instructions. If any instruction has an argument, this argument will be in the next successive element or elements of the instruction array. In the case of an immediate instruction, the data is the argument itself; otherwise the argument is the address of the data.

The instruction_counter in the io_record points to the next instruction to be executed. When the IO_MACHINE is entered, the first instruction must be an IO_CHANNEL or an IO_CHANNEL_IMMEDIATE instruction. These instructions indicate to the IO_MACHINE which device to use. Each instruction is executed in order unless a jump is encountered. A normal exit from the IO_MACHINE occurs when an IO_RETURN is executed. Abnormal exits are caused by errors which occur during the execution of the IO_MACHINE instructions. These errors are also explained in Appendix A. The error conditions are returned in the STATUS word of the io_record, as shown in Figure 4.1, with the value of the device status in the lower byte and the value of the IO_MACHINE status in the upper byte. This feature gives the user the ability to look at the condition of the device and take appropriate action to control the device in the manner which best fits his needs.

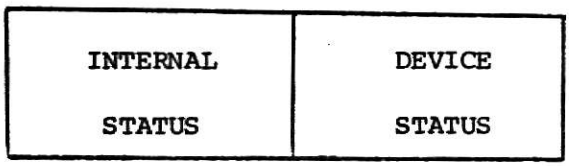

Figure 4.1

IO_MACHINE STATUS WORD

4.2 Example of a CRT Driver Using a PASLA

The use of the IO_MACHINE is best described by a short example. Section 2.4 gave the assembler version for a CRT driver which reads in 10 characters and then echoes them back to the CRT. The following example does the same function, but it is written in Concurrent PASCAL and uses the IO_MACHINE. The Command words for the Output Commands are the same in both examples except that the assembler version uses a hexadecimal value and the Concurrent PASCAL version uses decimal integers. In other words, the COMMAND 2 byte for the example in Section 2.4 was hexadecimal '6C' and in this it is decimal '108'. The Concurrent PASCAL code for this example follows:

```
"*****************************************************************
/              CRT DRIVER USING A PASLA                           /
 *****************************************************************"
CONST
    CRT       = 9; "CRT - PASLA X'12'"


TYPE EXAMPLE_CLASS = CLASS;
    CONST
       MAX_INSTRUCTION = 29;
       BUSY_STAT = 8;
    "COMMAND 1 BITS"
       TENABLE    = 64; "INTERRUPTS ON"
       TDTR       = 32; "DATA TERMINAL READY"
       TWRT       = 2;  "WRITE MODE"
       TRD        = 0;  "READ MODE"
       TCMND1     = 1;  "COMMAND 1"
    "COMMAND 2 BITS"
       THIGH      = 64; "HIGH BAUD RATE"
       T7BITS     = 32; "7 BITS OF DATA"
       T2STOPS    = 8;  "2 STOP BITS"
       TODD       = 4;  "ODD PARITY"

TYPE
    IO_TYPE = RECORD
       IC,                          "IO INSTRUCTION COUNTER"
       STATUS,                      "IO MACHINE STATUS ON RETURN"
       N_XFER: INTEGER;
       INTERNAL: ARRAY [0..13] OF INTEGER;
       INST: ARRAY [0..MAX_INSTRUCTION] OF INTEGER
       END "IO_TYPE";

TYPE LINE12 = ARRAY [1..12] OF CHAR;

VAR IO_RECORD: IO_TYPE; BUFFER: LINE12;

PROCEDURE ENTRY EXAMPLE;
    BEGIN
    WITH IO_RECORD DO
       BEGIN
       IC := 0; STATUS := 0;
       IO(IO_RECORD);  "READ IN 10 CHARACTERS"
       BUFFER[11] := '(:13:)'; "CARRIAGE RETURN"
       BUFFER[12] := '(:10:)'; "LINE FEED"
       IC := 13; STATUS := 0;
       IO(IO_RECORD); "WRITE OUT THE BUFFER"
       END;
    END;

BEGIN "INIT"
    WITH IO_RECORD DO
       BEGIN
       INST[0] := IO_CHANNEL_IMMEDIATE; INST[1] := CRT;
       INST[2] := IO_DATA; INST[3] := 10; INST[4] := ADDRESS(BUFFER);
       INST[5] := IO_COMMAND_IMMEDIATE; INST[6] := THIGH+T7BITS+T2STOPS+TODD;
       INST[7] := IO_COMMAND_IMMEDITATE; INST[8] := TENABLE+TDTR+TRD+TCMND1;
       INST[9] := IO_WAIT_INTERRUPT;
       INST[10] := IO_START_INPUT; INST[11] := NO_COMMAND;
       INST[12] := IO_RETURN;
       INST[13] := IO_CHANNEL_IMMEDIATE; INST[14] := CRT+1;
```

```
        INST[15] := IO_COMMAND_IMMEDIATE;
        INST[16] := TENABLE+TDTR+TWRT+TCMND1;
        INST[17] := IO_DATA; INST[18] := 12; INST[19] := ADDRESS(BUFFER
        INST[20] := IO_SENSE;
        INST[21] := IO_COMPARE_IMMEDIATE; INST[22] := BUSY_STAT;
        INST[23] := IO_JUMP_FALSE; INST[24] := 26;
        INST[25] := IO_WAIT_INTERRUPT;
        INST[26] := IO_START_OUTPUT; INST[27] := NO_COMMAND;
        INST[28] := IO_WAIT_INTERRUPT;
        INST[29] := IO_RETURN;
    END;
END; "EXAMPLE CLASS"
END.
```

4.3 Relationship to Asynchronous Processes

The Asynchronous Control Line Driver is functionally the same for both the assembler version and the Concurrent PASCAL version. The Asynchronous Communication Processes call the three entry points of the ACLDR to set up the PASLA (CALL ACL_SETUP), receive an SCB (CALL ACL_RCV), or transmit an SCB (CALL ACL_XMT). Each entry point then communicates with the PASLA to perform the desired operations. In the assembler version, it is the job of the programmer to write the code which communicates with the PASLA and handles the interrupts coming from that PASLA. The PASCAL process (or a class being executed by a process) needs only to call the IO_MACHINE in the KERNEL, with a set of instructions as parameters, and the IO_MACHINE does the rest.

In Figure 3.3 we show several modules needed for the implementation of the ACLDR in assembler language. The use of Concurrent PASCAL and the IO_MACHINE eliminates the need for the user to program these modules. Concurrent PASCAL allows the entry points, defined to be classes, to be reentrant so the user does not have to be concerned with special variable tables for each device, nor does he need the mapping of addresses to gain access to these tables. Because of the features of the IO_MACHINE, Figure 3.3 for the assembler version has been reduced to Figure 4.2 for the Concurrent PASCAL version. The ACLDR now contains only three classes, each being an entry point called by the Asynchronous Processes.

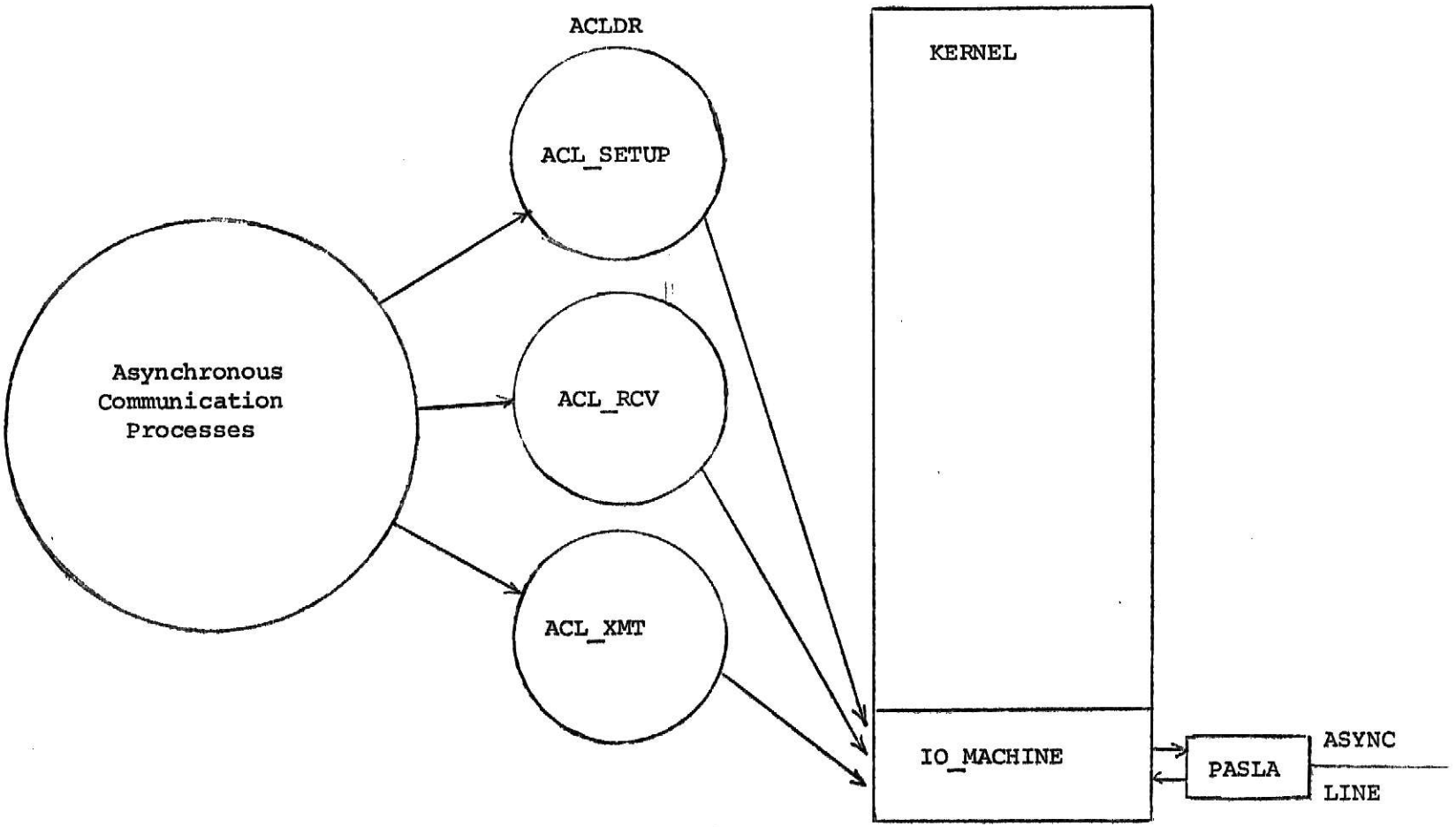

Figure 4.2

Relationship of ACLDR to Asynchronous Processes

4.4 Entry Points

The purpose of each entry point call and the parameters passed with the call are identical for both the assembler version and the Concurrent PASCAL version except for one slight change. Section 3.4 presented the functional specifications, the parameters passed, and high level algorithms for each entry point. To review, the calls and the parameters passed were as follows:

ACL_SETUP(line_addr,return_code)

ACL_RCV(line_addr,in_buff_addr,length,max_size,return_code)

ACL_XMT(line_addr,out_buff_addr,length,return_code)

The only difference between the two versions is that the line_addr parameter is the actual hardware device address in the assembler version where as it is a device index in the Concurrent PASCAL version. The KERNEL uses this index as a displacement into a table of the devices in the system.

CHAPTER 5

Summary

In this paper we presented the design and implementation of an Asynchronous Control Line Driver for the MIMICS network. We have shown how the ACLDR is used in the network and why asynchronous lines were chosen to carry control information in MIMICS. Implementation was accomplished on an Interdata 7/16 and an Interdata 85 using PASLA's to connect the two machines. Two different languages were used for this implementation. One version of the ACLDR is written in Assembler language and the other in Concurrent PASCAL. This summary discusses the advantages and disadvantages of using assembler language versus Concurrent PASCAL in the implementation of the ACLDR, the work completed, and a few possible extensions of the ACLDR to other machines.

5.1 Comparison of Assembler and PASCAL versions

The ACLDR was first written in assembler language on the Interdata 85. Since the operation of the PASLA for this project is somewhat different than most Input/Output devices, assembler language provided a low enough level of programming to control the hardware in the manner needed. It also allowed the programmer to minimize the amount of memory used by the ACLDR which is necessary because the ACLDR will eventually be used in micro-computers where memory use is critical.

Programming in assembler language has its advantages but it is very time consumming and often difficult to debug. After the assembler version of the ACLDR was written, Concurrent PASCAL became available for the Interdata 16-bit machines. This allowed the programming of the driver with a high level language. Concurrent PASCAL eliminates the need for the programmer to manipulate addresses or registers or to make the code reentrant. Along with these, the major advantage of using Concurrent PASCAL and the IO_MACHINE is that interrupts are handled by the IO_MACHINE. The time involved in writing and debugging the ACLDR in assembler language was approximately three times as long as for the Concurrent PASCAL version.

The only disadvantage in using Concurrent PASCAL is the amount of memory used. The assembler version of the ACLDR used approximately 1K bytes of memory with one asynchronous line. Each additional asynchronous line adds 134 bytes for the DEVMAP and the DVT of that line. The IO_MACHINE, on the other hand, eliminates the need for the Interrupt Service Routines but it also uses approximately 6K bytes of memory. This code however, is utilized by all peripheral devices. The three classes that make up the ACLDR take nearly 2K bytes of memory. On this basis, if the amount of memory used by the ACLDR is not as critical as the time and effort of implementation, then development using Concurrent PASCAL is worth the use of more memory.

5.2 Worked completed

Both the Assembler version and the Concurrent PASCAL version of the ACLDR run on either the Interdata 85 or the 7/16. The Concurrent PASCAL KERNEL is different for the two machines but this has no effect on the ACLDR running under the KERNEL.

5.3 Extensions

One of the major reasons for the use of Asynchronous lines to carry control information between computers in MIMICS is that they are fairly standard throughout the computer industry. This allows the connection of any two machines that support asynchronous lines. This project involved linking two Interdata 16-bit machines together. Some other possible connections would be to an Interdata 8/32, an ALTAIR 8800, or a Data General NOVA. The 8/32 would use the PASLA just like the Interdata 16-bit machines. The ALTAIR is a microcomputer which uses a Motorola Asynchronous Communication Interface Adapter (ACIA). The ACLDR would then have to be written in Intel 8080 code which is the assembler code for the ALTAIR 8800. If the NOVA were to be used the ACLDR would have to be converted to NOVA code and hardware interface would be NOVA's 4029 modem interface. The control signals used for the link between these other machines would be the same for the links to the two Interdata machines.

BIBLIOGRAPHY

[BRH75a] Brinch Hansen,P. "The Programming Language Concurrent PASCAL." IEEE Transactions on Software Engineering, Vol.1, No.2 (June,1975), pp.199-207.

[BRH75b] Brinch Hansen,P. "Concurrent PASCAL Report." Information Science, California Institute of Technology, June, 1975.

[BRH76] Brinch Hansen,P. "The SOLO Operating System Software - Practice and Experience." Vol 6, No.2, (April-June, 1976) pp. 141-206.

[COZ76] Cottel,D. and Zaun,J. "Concurrent PASCAL for the Interdata 7/16." Preliminary release, December,1976.

[INT01] INTERDATA INC. "Programmable Asynchronous Single Line Adapter Programming Manual" Pub. No. 29-446.

[INT02] INTERDATA INC. "16 Bit Series Reference Manual." Pub. No. 29-398R03.

[INT03] INTERDATA INC. "OS/16 MT2 Guide To Writing Drivers." Pub. No. B29-517.

[INT04] INTERDATA INC. "Common Assembler Language (CAL) User's Manual." Pub.No. 29-375R05.

[INT05] INTERDATA INC. "Data Communications Applications Guide." Pub. No. 29-468.

[NAR76] Neal, D., Anderson, G., Ratliff, J., Wallentine, V. "KSU Implementation of Concurrent PASCAL - A Reference Manual." KSU Department of Computer Science CS76-16.

[NLD76] Neal, D., "An Architectual Base For Concurrent PASCAL." (M.S. Theses), KSU Department of Computer Science (in preparation), Tech. Rpt. CS76-19, Nov.,1976.

[WHA76] Wallentine, V., Hankley, W., Anderson, G., Calhoun, M., and Maryanski, F., "Functionally Distributed Computer Systems Development: Software and Systems Structure." Technical Report, CS77-4, December 1976.

APPENDIX A

IO_MACHINE Instructions

Channel_Immediate (device index)

On every entry to the IO_MACHINE, the device index must be registered before any other instructions are executed. This allows the IO_MACHINE to make sure that the device is known to the system, that there are no errors associated with the device, and that the device has not already been preempted by another process. The device index is an integer established to reference the peripheral without needing to know its hardware address.

Channel (address of device index)

This is the same as 'Channel_Immediate' except for the argument. For this instruction, the next location in the instruction array contains the address where the device index may be found.

Command_Immediate (command data)

This instruction sends 'command data' to the currently addressed device as a command word. Since the Interdata commands are bytes, only the eight least significant bits of the argument are interpreted as a command. The rest of the integer is ignored. The status of the device is checked here. If the device does not respond (timeout status), then execution in the IO_MACHINE is terminated.

Command (address of command data)

This is similar to 'Command_Immediate' above.

Sense

This instruction simply senses the status of the presently addressed device.

Sense_External (address of status location)

The device status is read and stored into the location provided by the argument. The status word in the io record is also updated.

Compare_Immediate (mask)

The contents of the status word in the io record is logically AND'ed with 'mask'. If the result is zero, an internal flag is set to FALSE, otherwise it is set to TRUE, This flag is used by the conditional jump and return instructions.

Compare (address of mask)

This is similar to 'Compare_Immediate', of course.

Jump (instruction index)

Set the instruction counter to the value of the argument. This instruction causes a jump to continue execution at another part of the instruction array.

Jump_True (instruction index)

If the result of the last comparison instruction was TRUE, then set the instruction index to the argument value. Otherwise, execution continues with the following instruction.

Jump_False (instruction index)

If the result of the last comparison was FALSE, then jump to the instruction specified by the argument to continue execution. Otherwise, execution continues at the following instruction.

Return

Execution of instructions in the IO_MACHINE is terminated, and control of the program returns to the calling process. The instruction counter is left positioned at the next instruction so that another call to the IO_MACHINE with this io record would resume execution of the io instructions following the 'Return'.

Return_True

If the result of the last comparison was TRUE, then control returns to the calling process as in the

'Return' instruction. Otherwise, execution continues with the following instruction.

Return_False

If the result of the last comparison was FALSE, then the IO_MACHINE returns to the calling process. Otherwise, execution continues with the following instruction.

Data_Immediate (n, byte 1, byte 2, ..., byte n)

The purpose of this instruction is to establish a data buffer for a subsequent transfer to the addressed device. In the case of the 'immediate' version, the data buffer is actually in the instruction array following the 'Data_Immediate' instruction. The first argument gives the number of bytes to be transferred; the remaining arguments are the data. The form of data transfer depends upon the device - see the description of the 'Start' instruction for more details. Note that no direct buffer linking is possible with the IO_MACHINE. This is a very insecure method of handling the devices and was avoided. If a 'Data' instruction is executed for some device before the previous buffer was used, the description of the previous buffer will be lost.

Data (n, address of buffer)

This is similar to 'Data_Immediate' above, but the address of the data buffer is the second argument.

Wait_Interrupt

On executing this instruction, the calling process is suspended from execution on an interrupt queue associated with the currently addressed device. When an interrupt from that device is detected, execution of the process resumes immediately inside the IO_MACHINE. Remember that the interrupts are always off while the executing IO_MACHINE instructions. The user must insure that only one process is allowed to wait for an interrupt for any given device at a time. Clearly, this is an opportunity for messing up the io, since a process could wait for an interrupt without having previously caused the device to generate one. It could be a long wait!

Start_Input (command)

Data is transferred from the currently addressed device to the data buffer which was established earlier with a 'Data' or 'Data_Immediate' instruction. When execution of io instructions resumes, the entire data buffer has been transferred. The status word in the io record is set to indicate the final status of the device. The number of bytes actually transferred is returned in the io record, so in case of an incomplete operation, the calling process can determine the state of things.

The argument (command) is an output command that is sent immediately to the device and is included mainly for disk operations.

In the IO_MACHINE, devices are configured to transfer data in one of four distinct ways. The configuration of the current devices is given in the device table below. For general reference, the four types of transfer are:

Direct IO: The first and slowest of these methods uses direct io instructions to transfer data to or from the device. This may be by single byte or by halfword, and may be done with or without interrupts. If interrupts are not used, then the device must send or accept data as fast as the IO_MACHINE can transfer it. If the device transfer is to use interrupts, then if more than one io instruction must be executed, the IO_MACHINE waits for an interrupt from the device and checks that the status is zero before proceeding. If interrupts are being used, the user will probably want to use an explicit 'Wait_Interrupt' instruction before starting a read and after finishing a write.

Autochannel: Interdata provides an 'automatic' method for sending one byte per interrupt called the Automatic IO Channel. In this scheme, the entire block is transferred using interrupts without the user having to handle them explicitly with IO_MACHINE instructions.

Selector Channel: This is essentially a single channel DMA in series with the device. The Selector Channel hardware handles the interrupts and transfers the data to memory without disturbing the CPU.

Direct Memory Access: Our DMA will transfer data

directly to or from the device and memory. There are eight channels available.

Start_Output (command)

This is similar to 'Start_Input'. For some devices, the handling of interrupts is different for output than for input so the user must be careful.

Preempt

If the calling process wants to be sure that no other device will interfere with its device operation, it may execute the 'Preempt' instruction. This sets the state of the currently addressed device so that no other process may address the device with the 'Channel' instructions until the preempting process decides to release it. If a process is already waiting for an interrupt, it is returned to execution with the status indicating that it was preempted. Note that the preempting process cannot preempt an already preempted device, because the initial 'Channel' instruction would have failed. It is also important for the user to be sure that preempting a device doesn't leave it in some sort of state where errors could be produced.

Free

When a process which has preempted a device is through with it, this instruction is executed so that other processes may have the use of the device. A preempted device stays preempted through multiple calls to the IO_MACHINE - only execution of the 'free' will release it.

IO_MACHINE ERRORS

When the IO_MACHINE detects an error condition, it sets the status word in the io record to indicate the problem. In the In the Interdata, the status of a device is represented by one byte. The IO_MACHINE returns device status in the least significant byte, internal status in the next byte. Since any internal error causes the IO_MACHINE to terminate execution of the instruction, take care to leave the device in a proper state. In particular, an error may cause a process to be removed from the IO_MACHINE before a preempted device can be freed. The various error states are mutually exclusive and are represented by different integers. The values of the IO_MACHINE errors and their explanation follow. Some have been described earlier in the discussion of the IO_MACHINE instructions.

Number	Description
1	Instruction Counter Range - The instruction counter has been incremented past the end of the instruction array.
2	Device Index Range - The device index as given by the argument to the 'channel' instruction is out of range.
3	Device Interrupt - An interrupt occurred for this device while there was no process waiting to handle it. This status may be set upon execution of the 'channel' instruction.
4	Instruction Range - The integer indicated by the instruction counter is not a valid IO_MACHINE instruction.
5	Command Timeout - Sending the command to the addressed device gave a timeout status. That status will be in the lower byte of the io record status word.
6	Channel Timeout - Block transfer attempted on a channel which gives a timeout status.
7	Autochannel Timeout - A device using the autochannel returned a timeout status.
8	Selch Timeout - A selector channel returned a timeout status.
9	DMA Timeout - A DMA channel returned a timeout status.
10	Preempted - The device referenced by a 'channel'

instruction is already preempted, or else this process was preempted from a device interrupt queue.

11 Device Setup - The first instruction is not either a 'channel' or 'channel immediate' instruction.

APPENDIX B

ACLDR Assembler Code

The following listing is the Assembler version code for the ACLDR

ASYNCHRONOUS CONTROL LINE DRIVER

```
                         4  *
                         5  *
                         6  *-------------------------------------------------------------*
                         7  *                                                             *
                         8  *     THE ASYNCHRONOUS CONTRL LINE DRIVER CONSISTS OF         *
                         9  *     THREE ENTRY POINTS, ACL.SET, ACL.RCV, ACL.XMT,          *
                        10  *     PLUS TWO INTERRUPT SERVICE ROUTINES, RCV AND XMT.        *
                        11  *     EACH ENTRY POINT CAN BE CALLED BY THE ACLDR              *
                        12  *     TO SET UP A PASLA AND TRANSMIT AND RECEIVE              *
                        13  *     CONTROL MESSAGES THROUGH THAT PASLA                     *
                        14  *                                                             *
                        15  *-------------------------------------------------------------*
                        16  *
                        17  *
                        18  *-------------------------------------------------------------*
                        19  *                                                             *
                        20  *                  REGISTER EQUATES                           *
                        21  *                                                             *
                        22  *-------------------------------------------------------------*
                        23  *
                        24  *
0000                    25  R0        EQU   0
0001                    26  R1        EQU   1
0002                    27  R2        EQU   2
0003                    28  R3        EQU   3
0004                    29  R4        EQU   4
0005                    30  R5        EQU   5
0006                    31  R6        EQU   6
0007                    32  R7        EQU   7
0008                    33  R8        EQU   8
0009                    34  R9        EQU   9
000A                    35  R10       EQU   10
000B                    36  R11       EQU   11
000C                    37  R12       EQU   12
000D                    38  R13       EQU   13
000E                    39  R14       EQU   14
000F                    40  R15       EQU   15
```

ASYNCHRONOUS CONTROL LINE DRIVER

```
                        42  *
                        43  *
                        44  *-------------------------------------------------------*
                        45  *                                                       *
                        46  *          DEVICE VARIABLE TABLE DISPLACEMENTS          *
                        47  *                                                       *
                        48  *-------------------------------------------------------*
                        49  *
                        50  *
0000                    51  RCVADDR  EQU   0          ADDRESS OF RECEVIE SIDE
0002                    52  XMTADDR  EQU   2          ADDRESS OF TRANSMIT SIDE
0004                    53  INPTR    EQU   4          POINTER TO BYTE BEING RECEIVED
0006                    54  INSCB    EQU   6          ADDRESS OF INPUT BUFFER
0008                    55  ISCBND   EQU   8          POINTER OF END OF INPUT BUFFER
000A                    56  OUTPTR   EQU   10         POINTER TO BYTE BEING TRANSMITTED
000C                    57  OUTSCB   EQU   12         ADDRESS OF OUTPUT BUFFER
000E                    58  OSCBND   EQU   14         POINTER TO END OF OUTPUT BUFFER
0010                    59  MAXSIZE  EQU   16         MAXIMUM SIZE AN SCB CAN BE
0012                    60  LENGTH   EQU   18         LENGTH OF SCB
0014                    61  MLENFLG  EQU   20         MAXIMUM LENGTH FLAG
0016                    62  CARRFLG  EQU   22         CARRIER FLAG
0018                    63  LENGFLG  EQU   24         LENGTH FLAG
001A                    64  CKSMFLG  EQU   26         CHECKSUM FLAG
001C                    65  DONEFLG  EQU   28         DONE FLAG
001E                    66  INITFLG  EQU   30         INITIALIZATION FLAG
0020                    67  MODE     EQU   32         RECEIVE OR SEND MODE
0022                    68  CHECKSUM EQU   34         CHECK SUM
0024                    69  PASLA    EQU   36         PASLA COMMAND BYTE
0025                    70  ENRCV    EQU   37         RECEIVE SIDE COMMAND BYTE
0026                    71  ENXMT    EQU   38         TRANSMIT SIDE COMMAND BYTE
0028                    72  REGSAVE  EQU   40         REGISTER SAVE AREA
                        73  *
                        74  *
                        75  *-------------------------------------------------------*
                        76  *                                                       *
                        77  *              PASLA ADDRESSES                          *
                        78  *                                                       *
                        79  *-------------------------------------------------------*
                        80  *
                        81  *
0014                    82  RADDR    EQU   X'14'
0015                    83  XADDR    EQU   X'15'
```

ASYNCHRONOUS CONTROL LINE DRIVER

```
                                    *
                                    *
                                    *----------------------------------------------------------------*
                                    *                                                                *
                                    *          DEVICE MAP FOR RECEIVE SIDE OF PASLA                  *
                                    *                                                                *
                                    *----------------------------------------------------------------*
                                    *
                                    *
0000R  0000                         DEVMAP14 DC      0,0,X'3000'
0002R  0000
0004R  3000
0006R  D000 003CR                            STM     0,IORSAVE        SAVE REGISTERS
000AR  D1D0 001AR                            LM      13,DVB14         REG 13 = ADDR OF SERVICE ROUTINE
                                    *                                REG 14 = DEVICE VARIABLE TABLE
                                    *                                AND REG 15 = RETURN ADDRESS
000ER  01FD                                  BALR    15,13            GO TO ISR
0010R  D100 003CR                            LM      0,IORSAVE        RESTORE REGISTERS
0014R  C200 000AR                            LPSW    DEVMAP14         RETURN
                                    *
0018R  0240R                        DVB14    DC      Z(RCV)
001AR  005CR                                 DC      Z(DVT14)
001CR  0000                                  DC      0
                                    *
                                    *
                                    *----------------------------------------------------------------*
                                    *                                                                *
                                    *          DEVICE MAP FOR TRANSMIT SIDE OF PASLA                 *
                                    *                                                                *
                                    *----------------------------------------------------------------*
                                    *
                                    *
001ER  0000                         DEVMAP15 DC      0,0,X'3000'
0020R  0000
0022R  3000
0024R  D000 003CR                            STM     0,IORSAVE        SAVE REGISTERS
0028R  D1D0 003AR                            LM      13,DVB15         REG 13 = ADDR OF SERVICE ROUTINE
                                    *                                REG = 14 DEVICE VARIABLE TABLE
                                    *                                AND REG 15 = RETURN ADDRESS
002CR  01FD                                  BALR    15,13            GO TO ISR
002ER  D100 003CR                            LM      0,IORSAVE        RESTORE REGISTERS
0032R  C200 001ER                            LPSW    DEVMAP15         RETURN
                                    *
0036R  02EER                        DVB15    DC      Z(XMT)
0038R  005CR                                 DC      Z(DVT14)
003AR  0000                                  DC      0
                                    *
                                    *
003CR                               IORSAVE  DSH     16               SAVE AREA
                                    *
                                    *
```

ASYNCHRONOUS CONTROL LINE DRIVER

```
                                134  *
                                135  *
                                136  *-------------------------------------------------------------*
                                137  *                                                             *
                                138  *          DEVICE VARIABLE TABLE FOR PASLA X'14'              *
                                139  *                                                             *
                                140  *-------------------------------------------------------------*
                                141  *
                                142  *
005CR  0014                     143  DVT14    DC     RADDR            RCVADDR
005ER  0015                     144           DC     XADDR            XMTADDR
0060R  0000                     145           DC     0                INPTR
0062R  0000                     146           DC     0                INSCB
0064R  0000                     147           DC     0                ISCBND
0066R  0000                     148           DC     0                OUTPTR
0068R  0000                     149           DC     0                OUTSCB
006AR  0000                     150           DC     0                OSCBND
006CR  0000                     151           DC     0,0              MAXSIZE,LENGTH
006ER  0000
0070R  0000                     152           DC     0,0              MLENFLG,CARRFLG
0072R  0000
0074R  0000                     153           DC     0,0              LENGFLG,CKSMFLG
0076R  0000
0078R  0000                     154           DC     0,0              DONEFLG,INITFLG
007AR  0000
007CR  0000                     155           DC     0                MODE
007ER  0000                     156           DC     0                CHECKSUM
0080R  78                       157           DB     X'78'            PASLA SET UP
0081R  61                       158           DB     X'61'            ENRCV
0082R  63                       159           DB     X'63'            ENXMT
0083R  00                       160           DB     0
0084R                           161           DSH    16               REGSAVE
```

ASYNCHRONOUS CONTROL LINE DRIVER

```
                        163  *
                        164  *
                        165  *-------------------------------------------------------------*
                        166  *                                                             *
                        167  *              DEVICE VARIABLE TABLE MAP                      *
                        168  *                                                             *
                        169  *-------------------------------------------------------------*
                        170  *
                        171  *
                        172  *
00A4R                   173  DVTMAP   DSH    2
00A8R  005CR            174          DC     Z(DVT14)
```

```
                                176  *
                                177  *
                                178  *-------------------------------------------------------------------*
                                179  *                                                                   *
                                180  *                        ACL.SET                                    *
                                181  *                                                                   *
                                182  *      ACL.SET IS USED TO SET UP A PASLA AND ENABLE                  *
                                183  *      INTERRUPTS FOR THE RECEIVE AND TRANSMIT SIDES                *
                                184  *      OF THE PASLA.                                                *
                                185  *                                                                   *
                                186  *-------------------------------------------------------------------*
                                187  *
                                188  *
          00AAR                 189  ACL.SET  EQU   *
00AAR  48E1 0000                190           LH    R14,0(R1)          GET LINE NUMBER
00AER  08DE                     191           LHR   R13,R14            SAVE RECEIVE LINE NUMBER
00B0R  08CD                     192           LHR   R12,R13            SAVE TRANSMIT LINE NUMBER
00B2R  26C1                     193           AIS   R12,1              XMT LINE NUMBER
00B4R  C4E0 FFEF                194           NHI   R14,X'FFEE'        SUBTRACT X'10' AND MAKE IT EVEN
00B8R  C8B0 00A4R               195           LHI   R11,DVTMAP         GET DEVICE VARIABLE MAP
00BCR  0AEB                     196           AHR   R14,R11            ADD DISPLACEMENT
00BER  48EE 0000                197           LH    R14,0(R14)         GET ADDR OF DEVICE VARIABLE TABLE
00C2R  D00E 002A                198           STM   0,REGSAVE(R14)     SAVE REGISTERS
00C6R  C200 0102R               199  I1       LPSW  DISABLE            DISABLE INTERRUPTS
00CAR  DEDE 0024                200  I2       OC    R13,PASLA(R14)     TURN PASLA ON
00CER  4340 00DCR               201           BFC   4,I3               NO BRANCH IF INST TIME-OUT
00D2R  24A1                     202           LIS   R10,1
00D4R  40A1 0002                203           STH   R10,2(R1)          SET RCODE
00D8R  C200 010AR               204           LPSW  SPSW
00DCR  DEDE 0025                205  I3       OC    R13,ENRCV(R14)     ENABLE RECEIVE INTERRUPTS
00E0R  DECE 0026                206           OC    R12,ENXMT(R14)     ENABLE TRANSMIT INTERRUPTS
00E4R  07AA                     207           XHR   R10,R10
00E6R  40AE 001E                208           STH   R10,INITFLG(R14)   CLEAR INITIALIZATION FLAG
00EAR  9BD9                     209           RDR   R13,R9             INSURE THAT BUSY IS SET
00ECR  C200 0106R               210           LPSW  ENABLE             ENABLE INTERRUPTS
00F0R  40A1 0002                211  I4       STH   R10,2(R1)          SET RCODE
00F4R  2461                     212           LIS   R6,1
00F6R  406E 0016                213           STH   R6,CARRFLG(R14)    SET CARRIER FLAG
00FAR  D10E 002A                214  SRTN     LM    0,REGSAVE(R14)     RESTORE REGISTERS
00FER  430F 0000                215           B     0(R15)             RETURN
                                216  *
                                217  *
0102R  3000                     218  DISABLE  DC    X'3000',Z(I2)
0104R  00CAR
0106R  7C00                     219  ENABLE   DC    X'7C00',Z(I4)
0108R  00F0R
010AR  7C00                     220  SPSW     DC    X'7C00',Z(SRTN)
010CR  00FAR
```

```
                                  222  *
                                  223  *
                                  224  *-------------------------------------------------------------*
                                  225  *                                                             *
                                  226  *                          ACL.RCV                            *
                                  227  *                                                             *
                                  228  *     ACL.RCV IS USED TO SET THE ACLDR IN THE RECEIVE          *
                                  229  *     MODE AND WAIT FOR A CONTROL MESSAGE TO BE RECEIVED      *
                                  230  *     OR A TIMEOUT TO OCCUR.                                  *
                                  231  *                                                             *
                                  232  *-------------------------------------------------------------*
                                  233  *
                                  234  *
                   010ER          235  ACL.RCV  EQU   *
010ER  48E1 0000                  236          LH    R14,0(R1)          GET LINE NUMBER
0112R  08CE                       237          LHR   R12,R14            SAVE LINE NUMBER
0114R  C4E0 FFEE                  238          NHI   R14,X'FFEE'        SUBTRACT X'10' AND MAKE IT EVEN
0118R  C8D0 00A4R                 239          LHI   R13,DVTMAP         GET DEVICE MAP ADDRESS
011CR  0AED                       240          AHR   R14,R13            ADD DISPLACEMENT
011ER  48EE 0000                  241          LH    R14,0(R14)         GET ADDR OF DEVICE VARIABLE TABLE
0122R  D00E 002A                  242          STM   0,REGSAVE(R14)     SAVE REGISTERS
0126R  4801 0002                  243          LH    R0,2(R1)           GET INPUT BUFFER ADDRESS
012AR  400E 0006                  244          STH   R0,INSCB(R14)      STORE IT
012ER  4801 0006                  245          LH    R0,6(R1)           GET MAXIMUM SIZE OF BUFFER
0132R  400E 0010                  246          STH   R0,MAXSIZE(R14)    STORE IT
0136R  C800 0001                  247          LHI   R0,1
013AR  400E 002D                  248          STH   R0,MODE(R14)       SET RECEIVE MODE
013ER  0700                       249          XHR   R0,R0
0140R  400E 0014                  250          STH   R0,MLENFLG(R14)    CLEAR LENGTH FLAG
0144R  400E 001C                  251          STH   R0,DONEFLG(R14)    CLEAR DONE FLAG
0148R  400E 0012                  252          STH   R0,LENGTH(R14)     CLEAR LENGTH
014CR  48CE 0000                  253          LH    R12,RCVADDR(R14)   GET RECEIVE ADDRESS
0150R  98C7                       254          RDR   R12,R7             DUMMY READ TO SET BUSY
0152R  C870 7FFF                  255          LHI   R7,X'7FFF'         TIMEOUT COUNTER
                   0156R          256  AR0      EQU   *
0156R  90C3                       257          SSR   R12,R3             LOSS OF CARRIER ?
0158R  4320 016AR                 258          BFC   2,AR1              BRANCH IF NOT
015CR  2431                       259          LIS   R3,1
015ER  4031 000A                  260          STH   R3,8(R1)           PUT RCODE IN PARM LIST
0162R  403E 0016                  261          STH   R3,CARRFLG(R14)    SET CARRIER FLAG
0166R  4300 01C8R                 262          B     ARRTN
016AR  484E 001C                  263  AR1      LH    R4,DONEFLG(R14)    GET DONE FLAG
016ER  C540 0000                  264          CLHI  R4,0               DONE ?
0172R  4330 019ER                 265          BE    AR4                BRANCH IF NOT
0176R  485E 001A                  266          LH    R5,CKSMFLG(R14)    GET CHECKSUM FLAG
017AR  C550 0001                  267          CLHI  R5,1               BAD CHECKSUM ?
017ER  4330 018AR                 268          BE    AR2                BRANCH IF YES
0182R  4051 000A                  269          STH   R5,8(R1)           RCODE = OK
0186R  4300 0192R                 270          B     AR3
018AR  C850 0002                  271  AR2      LHI   R5,2
018ER  4051 000A                  272          STH   R5,8(R1)           RCODE = BAD CHECKSUM
0192R  486E 0012                  273  AR3      LH    R6,LENGTH(R14)     GET LENGTH OF SCB
0196R  4061 0004                  274          STH   R6,4(R1)           PUT LENGTH IN PARM LIST
019AR  4300 01C8R                 275          B     ARRTN
```

ASYNCHRONOUS CONTROL LINE DRIVER

```
                019ER           276  AR4      EQU   *
019ER  488E 0014                277           LH    R8,MLENFLG(R14)     GET MAX LENGTH FLAG
01A2R  C580 0001                278           CLHI  R8,1                LENGTH TO LARGE?
01A6R  4230 01B0R               279           BNE   AR5                  BRANCH IF NOT
01AAR  2454                     280           LIS   R5,4
01ACR  4300 01C4R               281           B     AR7
01B0R  C8A0 0064                282  AR5      LHI   R8,100              DELAY LOOP
01B4R  2781                     283  AR6      SIS   R8,1                FOR 5 SECOND
01B6R  4220 01B4R               284           BP    AR6                  TIMEOUT
01BAR  2771                     285           SIS   R7,1                DECREMENT TIME COUNTER
01BCR  4310 0156R               286           BNM   AR0                  NO BRANCH IF TIMEOUT
01C0R  C850 0003                287           LHI   R5,3                RCODE = TIMEOUT
01C4R  4051 000A                288  AR7      STH   R5,8(R1)            PUT RCODE IN PARM LIST
01C8R  0755                     289  ARRTN    XHR   R5,R5
01CAR  405E 0020                290           STH   R5,MODE(R14)        CLEAR RECEIVE MODE
01CER  D10E 002A                291           LM    0,REGSAVE(R14)      RESTORE REGISTERS
01D2R  430F 0000                292           B     0(R15)              RETURN
```

```
                                294  *
                                295  *
                                296  *-------------------------------------------------------------*
                                297  *                                                             *
                                298  *                          ACL.XMT                            *
                                299  *                                                             *
                                300  *     ACL.XMT IS USED TO START THE TRANSMISSION OF A           *
                                301  *     CONTROL MESSAGE AND SEE TO ITS COMPLETION.              *
                                302  *                                                             *
                                303  *-------------------------------------------------------------*
                                304  *
                                305  *
                    01D6R       306  ACL.XMT  EQU     *
01D6R  48E1 0000                307           LH      R14,0(R1)           GET LINE NUMBER
01DAR  08CE                     308           LHR     R12,R14             SAVE LINE NUMBER
01DCR  C4E0 FFEE                309           NHI     R14,X'FFEE'         SUBTRACT X'10' AND MAKE IT EVEN
01E0R  C8D0 00A4R               310           LHI     R13,DVTMAP          GET DEVICE VARIABLE MAP
01E4R  0AED                     311           AHR     R14,R13             ADD DISPLACEMENT
01E6R  48EE 0000                312           LH      R14,0(R14)          GET ADDRESS OF DVT
01EAR  D0AE 002A                313           STM     R10,REGSAVE(R14)    SAVE REGISTERS
01EER  07DD                     314           XHR     R13,R13
01F0R  40DE 001C                315           STH     R13,DONEFLG(R14)    CLEAR DONE FLAG
01F4R  40DE 001A                316           STH     R13,CKSMFLG(R14)    CLEAR CHECKSUM FLAG
01F8R  48B1 0004                317           LH      R11,4(R1)           GET LENGTH OF SCB
01FCR  40BE 0012                318           STH     R11,LENGTH(R14)     STORE IT
0200R  48A1 0002                319           LH      R10,2(R1)           GET OUTPUT BUFFER ADDRESS
0204R  40AE 000C                320           STH     R10,OUTSCB(R14)     STORE IT
0208R  0ABA                     321           AHR     R11,R10             ADD LENGTH
020AR  40BE 000E                322           STH     R11,OSCBND(R14)     STORE ENDING ADDRESS OF SCB
020ER  E20C 0000                323           SINT    0(R12)              START TRANSMISSION OF SCB
0212R  48DE 0016                324  AX1      LH      R13,CARRFLG(R14)    GET CARRIER FLAG
0216R  C8A0 0001                325           LHI     R10,1
021AR  40AE 0016                326           STH     R10,CARRFLG(R14)    SET CARRIER FLAG
021ER  C5D0 0000                327           CLHI    R13,0               LOST CARRIER ?
0222R  4330 0234R               328           BE      AXRTN
0226R  43AE 001C                329  AX2      LH      R10,DONEFLG(R14)    GET DONE FLAG
022AR  C5A0 0001                330           CLHI    R10,1               DONE ?
022ER  4230 0212R               331           BNE     AX1                 BRANCH IF NOT
0232R  07AA                     332           XHR     R10,R10             RCODE = OK
0234R  40A1 0006                333  AXRTN    STH     R10,6(R1)           PUT RCODE IN PARM LIST
0238R  D1AE 002A                334           LM      R10,REGSAVE(R14)    RESTORE REGISTERS
023CR  430F 0000                335           B       0(R15)              RETURN
```

```
                        337  *
                        338  *
                        339  *-------------------------------------------------------------------*
                        340  *                                                                   *
                        341  *           RECEIVE INTERRUPT SERVICE ROUTINE                       *
                        342  *                                                                   *
                        343  *     RCV HANDLES ALL INTERRUPTS COMING FROM THE RECEIVE             *
                        344  *     SIDE OF THE PASLA.  IF THE ACLDR IS IN THE RECEIVE            *
                        345  *     MODE THEN THE CHARACTERS ARE READ IN OTHERWISE                 *
                        346  *     THE INTERRUPTS ARE IGNORED.                                   *
                        347  *                                                                   *
                        348  *-------------------------------------------------------------------*
                        349  *
                        350  *
0240R 48DE 0000         351  RCV     LH      R13,RCVADDR(R14)    GET RECEIVE ADDRESS
0244R 9DD4              352  RC1     SSR     R13,R4              GET STATUS
0246R 4320 0254R        353          BFC     2,RC2               BRANCH IF CARRIER IS ON
024AR 0744              354          XHR     R4,R4
024CR 404E 0016         355          STH     R4,CARRFLG(R14)     CLEAR CARRIER FLAG
0250R 430F 0000         356          B       0(R15)              RETURN
0254R 0744              357  RC2     XHR     R4,R4
0256R 454E 0020         358          CLH     R4,MODE(R14)        IN RECEIVE MODE ?
025AR 433F 0000         359          BE      0(R15)              IGNORE INTERRUPT IF NOT
025ER 9BD3              360          RDR     R13,R3              READ CHARACTER
0260R 485E 0012         361          LH      R5,LENGTH(R14)      LENGTH = 0 IF FIRST CHAR
0264R C550 0000         362          CLHI    R5,0
0268R 4230 029ER        363          BNE     RC4                 BRANCH IF NOT FIRST CHAR
026CR 485E 0010         364          LH      R5,MAXSIZE(R14)     GET MAXIMUM SIZE OF BUFFER
0270R 48AE 0006         365          LH      R10,INSCB(R14)      GET SCB ADDRESS
0274R 40AE 0004         366          STH     R10,INPTR(R14)      SET INPUT POINTER
0278R D535              367          CLHR    R3,R5               LENGTH : MAXSIZE
027AR 4320 028CR        368          BNP     RC3                 BRANCH IF LENGTH <= MAXSIZE
027ER 403E 0012         369          STH     R3,LENGTH(R14)      STORE LENGTH
0282R 2451              370          LIS     R5,1
0284R 405E 0014         371          STH     R5,MLENFLG(R14)     SET MAX LENGTH FLAG
0288R 430F 0000         372          B       0(R15)              RETURN
028CR 403E 0012         373  RC3     STH     R3,LENGTH(R14)      LENGTH = CHARACTER
0290R 403E 0022         374          STH     R3,CHECKSUM(R14)    CHECKSUM = LENGTH
0294R 0AA3              375          AHR     R10,R3              ADD SCB ADDRESS AND LENGTH
0296R 40AE 000A         376          STH     R10,ISCBND(R14)     STORE SCB ENDING ADDRESS
029AR 430F 0000         377          B       0(R15)              RETURN
029ER 486E 0004         378  RC4     LH      R6,INPTR(R14)       GET SCB POINTER
02A2R 456E 0008         379          CLH     R6,ISCBND(R14)      BUFFER FULL ?
02A6R 4210 02DCR        380          BH      RC7                 BRANCH IF NOT
02AAR 487E 0022         381          LH      R7,CHECKSUM(R14)    GET CHECKSUM
02AER C470 00FF         382          NHI     R7,X'00FF'
02B2R 0537              383          CLHR    R3,R7               CHECKSUM OK ?
02B4R 4230 02C2R        384          BNE     RC5                 BRANCH IF NOT
02B8R 0777              385          XHR     R7,R7
02BAR 407E 001A         386          STH     R7,CKSMFLG(R14)     CHECKSUM FLAG = OK
02BER 4300 02CAR        387          B       RC6
02C2R C870 0001         388  RC5     LHI     R7,1
02C6R 407E 001A         389          STH     R7,CKSMFLG(R14)     CHECKSUM FLAG = ERROR
02CAR C870 0001         390  RC6     LHI     R7,1
```

ASYNCHRONOUS CONTROL LINE DRIVER

```
02CER  407E 001C          391          STH   R7,DONEFLG(R14)     SET DONE FLAG
02D2R  0777               392          XHR   R7,R7
02D4R  407E 0020          393          STH   R7,MODE(R14)        GET OUT OF RECEIVE MODE
02D8R  430F 0000          394          B     0(R15)              RETURN
02DCR  D236 0000          395  RC7     STB   R3,0(R6)            STORE BYTE IN SCB
02E0R  613E 0022          396          AHM   R3,CHECKSUM(R14)    ADD BYTE TO CHECKSUM
02E4R  2661               397          AIS   R6,1                INCREMENT INPTR
02E6R  406E 0004          398          STH   R6,INPTR(R14)
02EAR  430F 0000          399  RRTN    B     0(R15)              RETURN
```

ASYNCHRONOUS CONTROL LINE DRIVER

```
                                401  *
                                402  *
                                403  *-------------------------------------------------------------*
                                404  *                                                             *
                                405  *           TRANSMIT INTERRUPT SERVICE ROUTINE                *
                                406  *                                                             *
                                407  *     XMT HANDLES ALL INTERRUPTS COMING FROM THE TRANSMIT      *
                                408  *     SIDE OF THE PASLA.  THE INITIAL INTERRUPT CAUSED BY     *
                                409  *     DATA TERMINAL READY BECOMING ACTIVE IS IGNORED AND THE  *
                                410  *     REST ARE USED TO TRANSMIT THE CHARACTERS.               *
                                411  *                                                             *
                                412  *-------------------------------------------------------------*
                                413  *
                                414  *
02EER  480E 001E                415  XMT      LH     R13,INITFLG(R14)    GET INITIALIZATION FLAG
02F2R  C500 0001                416           CLHI   R13,1               FIRST INTERRUPT ?
02F6R  4330 0304R               417           BE     X00                 BRANCH IF NOT
02FAR  2401                     418           LIS    R13,1
02FCR  40DE 001E                419           STH    R13,INITFLG(R14)    SET INITIALIZATION FLAG
0300R  430F 0000                420           B      0(R15)              RETURN
0304R  48DE 0000                421  X00      LH     R13,RCVADDR(R14)    GET PASLA ADDRESS
0308R  48CE 0002                422           LH     R12,XMTADDR(R14)    GET TRANSMIT ADDRESS
030CR  9DD2                     423           SSR    R13,R2              GET STATUS
030ER  4320 031CR               424           BFC    2,X0                BRANCH IF CARRIER IS ON
0312R  0722                     425           XHR    R2,R2
0314R  402E 0016                426           STH    R2,CARRFLG(R14)     CLEAR CARRIER FLAG
0318R  430F 000D                427           B      0(R15)              RETURN
031CR  480E 0018                428  X0       LH     R0,LENGFLG(R14)     GET LENGTH FLAG
0320R  C500 0001                429           CLHI   R0,1                HAS LENGTH BEEN SENT
0324R  4330 0342R               430           BE     X1                  BRANCH IF YES
0328R  480E 0012                431           LH     R0,LENGTH(R14)      GET LENGTH
032CR  400E 0022                432           STH    R0,CHECKSUM(R14)    STORE IN CHECKSUM
0330R  9AC0                     433           WOR    R12,R0              TRANSMIT LENGTH
0332R  C800 0001                434           LHI    R0,1
0336R  400E 0018                435           STH    R0,LENGFLG(R14)     SET LENGFLG =1
033AR  48AE 000C                436           LH     R10,OUTSCB(R14)     GET ADDRESS OF OUTPUT BUFFER
033ER  40AE 000A                437           STH    R10,OUTPTR(R14)     RESET OUTPUT POINTER
0342R  48AE 000A                438  X1       LH     R10,OUTPTR(R14)     GET OUTPUT BUFFER POINTER
0346R  45AE 000E                439           CLH    R10,OSCBND(R14)     BUFFER EMPTY ?
034AR  4210 0380R               440           BM     X3                  BRANCH IF NOT
034ER  0788                     441           XHR    R8,R8
0350R  458E 001A                442           CLH    R8,CKSMFLG(R14)     HAS CHECKSUM BEEN SENT ?
0354R  4330 036ER               443           BE     X2                  BRANCH IF NOT
0358R  0788                     444           XHR    R8,R8
035AR  408E 001A                445           STH    R8,CKSMFLG(R14)     CLEAR CHECKSUM FLAG
035ER  408E 0018                446           STH    R8,LENGFLG(R14)     CLEAR LENGTH FLAG
0362R  C870 0001                447           LHI    R7,1
0366R  407E 001C                448           STH    R7,DONEFLG(R14)     SET DONE FLAG
036AR  430F 000D                449           B      0(R15)              RETURN
036ER  486E 0022                450  X2       LH     R6,CHECKSUM(R14)    GET CHECKSUM
0372R  9AC6                     451           WOR    R12,R6              TRANSMIT CHECKSUM
0374R  C860 0001                452           LHI    R6,1
0378R  406E 001A                453           STH    R6,CKSMFLG(R14)     SET CHECKSUM FLAG
037CR  430F 0000                454           B      0(R15)              RETURN
```

ASYNCHRONOUS CONTROL LINE DRIVER

```
0380R  D35A 0000          455  X3      LB      R5,0(R10)            GET CHARACTER
0384R  615E 0022          456           AHM     R5,CHECKSUM(R14)     ADD TO CHECKSUM
0388R  9AC5               457           WDR     R12,R5               TRANSMIT CHARACTER
038AR  26A1               458           AIS     R10,1                INCREMENT OUTPTR
038CR  40AE 000A          459           STH     R10,OUTPTR(R14)
0390R  430F 0000          460  XRTN     B       0(R15)               RETURN
0394R                     461           END
```

APPENDIX C

ACLDR Concurrent PASCAL Code

The following listing is the Concurrent PASCAL version for the ACLDR

```
"   *-------------------------------------------------------------*
    [        **** ASYNCHRONOUS CONTROL LINE DRIVER *****          ]
    [                                                             ]
    [     THIS DRIVER CONSISTS OF THREE CLASSES, ACL_SETUP,       ]
    [     ACL_RCV, AND ACL_XMT.  EACH CLASS CAN BE CALLED BY      ]
    [     THE ASNCHRONOUS COMMUNICATION PROCESSES TO SET UP A     ]
    [     PASLA AND TRANSMIT AND RECEIVE CONTROL MESSAGES         ]
    [     THROUGH THAT PASLA.                                     ]
    [                                                             ]
    *-------------------------------------------------------------*"


"   *-------------------------------------------------------------*
    [                                                             ]
    [          IO_MACHINE COMMANDS USED BY THE INTERPRETER        ]
    [                                                             ]
    *-------------------------------------------------------------*"

CONST

  IO_CHANNEL_IMMEDIATE"(DEVICE_INDEX_VALUE)"                  =  0;
  IO_CHANNEL"(DEVICE_INDEX_ADDRESS)"                          =  1;

  IO_COMMAND_IMMEDIATE"(COMMAND_BYTE_VALUE)"                  =  2;
  IO_COMMAND"(COMMAND_BYTE_ADDRESS)"                          =  3;

  IO_SENSE  "-RETURN DEVICE STATUS BYTE-"                     =  4;
  IO_SENSE_EXTERNAL"(STATUS_BYTE_ADDRESS)"                    =  5;

  IO_COMPARE_IMMEDIATE"(STATUS_MASK_VALUE)"                   =  6;
  IO_COMPARE"(STATUS_MASK_ADDRESS)"                           =  7;

  IO_JUMP"(IO_INSTRUCTION_INDEX_VALUE)"                       =  8;
  IO_JUMP_TRUE"(IO_INSTR_INDEX_VALUE)  -COMPARE SETS COND-"   =  9;
  IO_JUMP_FALSE"(IO_INSTR_INDEX_VALUE) -COMPARE SETS COND-"   = 10;

  IO_RETURN      "-EXITS IO_INTERPRETER-"                     = 11;
  IO_RETURN_TRUE                                              = 12;
  IO_RETURN_FALSE                                             = 13;

  IO_DATA_IMMEDIATE"(N, BYTE1, BYTE2,.BYTEN)"                 = 14;
  IO_DATA"(BUFFER_BYTE_LENGTH, BUFFER_ADDRESS)"               = 15;

  IO_WAIT_INTERRUPT                                           = 16;

  IO_START_INPUT  "-SETS UP & STARTS AUTOCH, MUX, SELCH-"     = 17;
  IO_START_OUTPUT                                             = 18;

  IO_PREEMPT  "-EMERGENCY DEVICE CONTROL-"                    = 19;
  IO_FREE     "-EMERGENCY OVER, GIVE DEVICE BACK-"            = 20;

CONST
  ASYNC_LINE1_IN  = 7; "X'10' - ASYNCHRONOUS LINE PASLA 10"
  ASYNC_LINE1_OUT = 8; "X'11' - ASYNCHRONOUS LINE PASLA 11"
TYPE
```

```
    S_RETURN_CODE = (S0,S1,S2);
TYPE
    RCV_RETURN_CODE = (R0,R1,R2,R3,R4);
TYPE
    XMT_RETURN_CODE = (X0,X1);
TYPE
    CH_WORD = ARRAY [1..2] OF CHAR;

CONST
    UNEX_INTERRUPT = 768;
    COMMAND_TIMEOUT = 1280;
    CHANNEL_TIMEOUT = 1536;
     NO_COMMAND = 256;  "NO OP FOR START I/O"

TYPE STRING76 = ARRAY [1..76] OF CHAR;

 "   *-------------------------------------------------------------*
     [                                                             ]
     [                  ***** COMMENTS *****                       ]
     [                                                             ]
     [      WHEN EVER AN INTERRUPT IS GENERATED BY A DEVICE AND    ]
     [      NO PROCESS IS WAITING FOR THAT INTERRUPT THEN AN       ]
     [      INTERNAL FLAG IS SET.  WHEN A PROCESS CALLS THE        ]
     [      IO_MACHINE WITH THAT DEVICE INDEX THE IO_MACHINE       ]
     [      IMMEDIATELY RETURNS CONTROL AFTER THE IO_CHANNEL       ]
     [      INSTRUCTION WITH AN INTERNAL STATUS OF 3 IN THE        ]
     [      STATUS WORD.  THESE TYPES OF INTERRUPTS ARE IGNORED    ]
     [      BY THE ACLDR SO THE IO_MACHINE IS REENTERED AFTER      ]
     [      DETECTION OF THIS STATUS.                              ]
     [                                                             ]
     *-------------------------------------------------------------*"
```

```
"   *-----------------------------------------------------------------*
    [                                                                 ]
    [                ***** ACL_SETUP *****                            ]
    [                                                                 ]
    [      ACL_SETUP IS CALLED TO SET UP A PASLA AND ENABLE           ]
    [      INTERRUPTS FOR THE RECEIVE AND TRANSMIT SIDES OF           ]
    [      THAT PASLA.                                                ]
    [                                                                 ]
    *-----------------------------------------------------------------*"

TYPE SET_UP_CLASS = CLASS;
    CONST
       MAX_INSTRUCTION = 7;
    "COMMAND 1 BITS"
          TENABLE   = 64; "INTERRUPTS ON"
          TDTR      = 32; "DATA TERMINAL READY"
          TWRT      = 2;  "WRITE MODE"
          TRD       = 0;  "READ MODE"
          TCMND1    = 1;  "COMMAND 1"
       "COMMAND 2 BITS"
          THIGH       = 64; "HIGH BAUD RATE"
          T8BITS      = 48; "8 BIT DATA"
          T2STOPS     = 8;  "2 STOP BITS"
       TYPE
          IO_TYPE = RECORD
             IC.                  "IO INSTRUCTION COUNTER"
             STATUS.              "IO MACHINE STATUS ON RETURN"
             N_XFER: INTEGER;
             INTERNAL: ARRAY[1..13] OF INTEGER;
             INST: ARRAY [0..MAX_INSTRUCTION] OF INTEGER
             END "IO_TYPE";
VAR IO_RECORD: IO_TYPE;
       COMMAND_FLAG: BOOLEAN;

    PROCEDURE ENTRY ACL_SETUP(LINE_ADDRESS: INTEGER;
          VAR RCODE: S_RETURN_CODE);

       BEGIN WITH IO_RECORD DO
          BEGIN
          COMMAND_FLAG := TRUE;
       REPEAT "IGNORE ANY PREVIOUS INTERRUPTS"
          IC := 0; STATUS := 0;
          INST[1] := LINE_ADDRESS;
          INST[5] := TENABLE+TDTR+TRD+TCMND1;
          INST[6] := IO_RETURN;
          IO(IO_RECORD); "TURN ON PASLA"
          IF (STATUS >= COMMAND_TIMEOUT) AND
             (STATUS < CHANNEL_TIMEOUT) THEN
             COMMAND_FLAG := FALSE;
       UNTIL (STATUS <> UNEX_INTERRUPT);
       REPEAT "IGNORE ANY PREVIOUS INTERRUPTS"
          IC := 0; STATUS := 0;
          INST[1] := LINE_ADDRESS + 1;
          INST[5] := TENABLE+TDTR+TWRT+TCMND1;
          INST[6] := IO_WAIT_INTERRUPT;
          IO(IO_RECORD);
       UNTIL (STATUS <> UNEX_INTERRUPT);
          IF COMMAND_FLAG THEN RCODE := S0 ELSE RCODE := S1;
          END;
       END;
```

```
BEGIN "INIT"
   WITH IO_RECORD DO
      BEGIN
      INST[0] := IO_CHANNEL_IMMEDIATE;
      INST[2] := IO_COMMAND_IMMEDIATE;
      INST[3] := THIGH+T8BITS+T2STOPS;
      INST[4] := IO_COMMAND_IMMEDIATE;
      INST[7] := IO_RETURN;
      END "WITH IO_RECORD";
END "SET UP CLASS";
```

```
"   *-----------------------------------------------------------------*
    [                                                                 ]
    [                ***** ACL_RCV *****                              ]
    [                                                                 ]
    [      ACL_RCV IS CALLED TO SET THE ACLDR IN THE RECEIVE          ]
    [      MODE AND WAIT FOR A CONTROL LINE MESSAGE TO BE             ]
    [      RECEIVED.                                                  ]
    [                                                                 ]
    *-----------------------------------------------------------------*"

TYPE RCV_CLASS = CLASS;
    CONST
       MAX_INSTRUCTION = 8;
    TYPE
       IO_TYPE = RECORD
          IC,                          "IO INSTRUCTION COUNTER"
          STATUS,                      "IO MACHINE STATUS ON RETURN"
          N_XFER: INTEGER;                "NUMBER BYTES IN LAST TRANSFER"
          INTERNAL: ARRAY [1..13] OF INTEGER;
          INST: ARRAY [0..MAX_INSTRUCTION] OF INTEGER
          END "IO_TYPE";
    VAR IO_RECORD: IO_TYPE; POS: INTEGER; CHECKSUM: INTEGER;
       IN_LENGTH: INTEGER; IN_CHECKSUM: INTEGER;
       TEMP_CHECKSUM: INTEGER;

PROCEDURE RBYTE(IN_W: UNIV CH_WORD; VAR OUT_W: UNIV CH_WORD);
          "RBYTE IS USED TO PLACE THE UPPER BYTE OF AN INTEGER INTO
           THE LOWER ONE AND ZERO OUT THE UPPER BYTE."
    BEGIN
    OUT_WC1] := '(:0:)';
    OUT_WC2] := IN_WC1];
    END;

PROCEDURE ENTRY ACL_RCV(LINE_ADDRESS: INTEGER; VAR BUFFER: STRING76;
    VAR LENGTH: INTEGER; MAX: INTEGER; VAR RCODE: RCV_RETURN_CODE);
    BEGIN WITH IO_RECORD DO
       BEGIN
       REPEAT "IGNORE ANY PREVIOUS INTERRUPTS"
          IC := 0; STATUS := 0;
          INSTC1] := LINE_ADDRESS;
          INSTC2] := IO_SENSE;
          INSTC3] := IO_RETURN;
          IO(IO_RECORD);
       UNTIL(STATUS <> UNEX_INTERRUPT);
          INSTC2] := IO_DATA;
          IN_LENGTH := 0;
       IF NOT ODD(STATUS DIV 2) THEN
          BEGIN
       REPEAT "IGNORE ANY PREVIOUS INTERRUPTS"
          IC := 0; STATUS := 0;
          INSTC3] := 1;
          INSTC4] := ADDRESS(IN_LENGTH);
          IO(IO_RECORD); "RECEIVE LENGTH OF INCOMING SCB"
        " THE LENGTH IS RECEIVED IN THE UPPER BYTE
          OF THE INTEGER SO RBYTE IS CALLED TO PLACE
          IT IN THE LOWER BYTE."
          RBYTE(IN_LENGTH,LENGTH);
       UNTIL (STATUS <> UNEX_INTERRUPT);
          END;
       IF LENGTH <= MAX THEN
```

```
            BEGIN
            TEMP_CHECKSUM := LENGTH;
         IF NOT ODD(STATUS DIV 2) THEN
            BEGIN
            REPEAT "IGNORE ANY PREVIOUS INTERRUPTS"
               IC := 0; STATUS := 0;
               INSTR[3] := LENGTH;
               INSTR[4] := ADDRESS(BUFFER);
               IO(IO_RECORD);   "RECEIVE SCB"
            UNTIL (STATUS <> UNEX_INTERRUPT);
               END;
            IF NOT ODD(STATUS DIV 2) THEN
               BEGIN
            REPEAT "IGNORE ANY PREVIOUS INTERRUPTS"
               IC := 0; STATUS := 0;
                INSTR[3] := 1;
               INSTR[4] := ADDRESS(IN_CHECKSUM);
               IO(IO_RECORD);  "RECEIVE CHECKSUM"
             " THE CHECKSUM IS RECEIVED IN THE UPPER BYTE
               OF THE INTEGER SO RBYTE IS CALLED TO PLACE
               IT IN THE LOWER BYTE."
                RBYTE(IN_CHECKSUM,CHECKSUM);
            UNTIL (STATUS <> UNEX_INTERRUPT);
               TEMP_CHECKSUM := 0;
               POS := 1;
                  WHILE (POS <= LENGTH) DO
                     BEGIN
                     TEMP_CHECKSUM := TEMP_CHECKSUM + ORD(BUFFER[POS]);
                     POS := POS + 1;
                  END;
               TEMP_CHECKSUM := TEMP_CHECKSUM + LENGTH;
               TEMP_CHECKSUM := TEMP_CHECKSUM MOD 256;
               IF TEMP_CHECKSUM = CHECKSUM THEN
                  RCODE := R0
               ELSE RCODE := R2;
               END;
            IF ODD(STATUS DIV 2) THEN RCODE := R1;
          END
          ELSE RCODE := R4;
         END;
      END;

BEGIN "INIT"
      WITH IO_RECORD DO
         BEGIN
         INSTR[0] := IO_CHANNEL_IMMEDIATE;
         INSTR[5] := IO_WAIT_INTERRUPT;
         INSTR[6] := IO_START_INPUT;
         INSTR[7] := NO_COMMAND;
         INSTR[8] := IO_RETURN;
         END "WITH IO_RECORD";
END "RCV_CLASS";
```

```
"   *-----------------------------------------------------------------*
    [                ***** ACL_XMT *****                             ]
    [                                                                ]
    [      ACL_XMT IS CALLED TO START THE TRANSMISSION OF A          ]
    [      CONTROL MESSAGE AND SEE TO ITS COMPLETION.                ]
    [                                                                ]
    *-----------------------------------------------------------------*"

    TYPE XMT_CLASS =  LASS;
    CONST
       BSY_STAT = 08;      "PASLA BUSY STATUS"
       MAX_INSTRUCTION = 14;  "MAXIMUM IO INSTRUCTIONS"
    TYPE
       IO_TYPE = RECORD
       IC,                    "IO INSTRUCTION COUNTER"
       STATUS,                "IO MACHINE STATUS ON RETURN"
       N_XFER: INTEGER;       "NUMBER OF BYTES TRANSFERED"
       INTERNAL: ARRAY [1..13] OF INTEGER;
       INST: ARRAY [0..MAX_INSTRUCTION] OF INTEGER
       END "IO_TYPE";
VAR IO_RECORD: IO_TYPE; POS: INTEGER; CHECKSUM: INTEGER;
       OUT_CHECKSUM: INTEGER; OUT_LENGTH: INTEGER;

PROCEDURE XBYTE(IN_W: UNIV CH_WORD; VAR OUT_W: UNIV CH_WORD);
        " XBYTE IS USED TO PLACE THE LOWER BYTE OF AN
          INTEGER INTO THE UPPER BYTE."
    BEGIN
    OUT_W[2] := '(:0:)';
    OUT_W[1] := IN_W[2];
    END;

PROCEDURE ENTRY ACL_XMT(LINE_ADDRESS: INTEGER; VAR BUFFER: STRING76;
       VAR LENGTH: INTEGER; VAR RCODE: XMT_RETURN_CODE);
    BEGIN WITH IO_RECORD DO
       BEGIN
       "CHECK STATUS FOR LOSS OF CARRIER"
       REPEAT "IGNORE ANY PREVIOUS INTERRUPTS"
          IC := 0; STATUS := 0;
          INST[1] := LINE_ADDRESS;
          INST[2] := IO_SENSE;
          INST[3] := IO_RETURN;
          IO(IO_RECORD);
       UNTIL(STATUS <> UNEX_INTERRUPT);
       IF NOT ODD(STATUS DIV 2) THEN
          BEGIN
          INST[1] := LINE_ADDRESS + 1;
          INST[2] := IO_DATA;
          CHECKSUM := LENGTH;
          POS := 1;
          "  FIGURE CHECKSUM   "
          WHILE (POS <= LENGTH) DO
             BEGIN
             CHECKSUM := CHECKSUM + ORD(BUFFER[POS]);
             POS := POS + 1;
             END;
        " XBYTE IS CALLED SO THAT THE LENGTH WHICH IS ONE
          BYTE LONG CAN BE SENT OUT OF THE UPPER BYTE."
          XBYTE(CHECKSUM,OUT_CHECKSUM);
          " TRANSMIT LENGTH "
       REPEAT "IGNORE ANY PREVIOUS INTERRUPTS"
```

```
            IC := 0; STATUS := 0;
            INSTC3] := 1;
            INSTC4] :=  ADDRESS(OUT_LENGTH);
          " XBYTE IS CALLED SO THAT THE LENGTH WHICH IS ONE
            BYTE LONG CAN BE SENT OUT OF THE UPPER BYTE."
            XBYTE(LENGTH,OUT_LENGTH);
            IO(IO_RECORD);  "TRANSMIT LENGTH OF SCB"
         UNTIL (STATUS <> UNEX_INTERRUPT);
         REPEAT "IGNORE ANY PREVIOUS INTERRUPTS"
            IC := 0; STATUS := 0;
            INSTC3] := LENGTH;
            INSTC4] := ADDRESS(BUFFER);
            IO(IO_RECORD);   "TRANSMIT SCB"
         UNTIL(STATUS <> UNEX_INTERRUPT);
         REPEAT "IGNORE ANY PREVIOUS INTERRUPTS"
            IC := 0; STATUS := 0;
            INSTC3] := 1;
            INSTC4] := ADDRESS(OUT_CHECKSUM);
            IO(IO_RECORD);  "TRANSMIT CHECKSUM OF SCB"
         UNTIL(STATUS <> UNEX_INTERRUPT);
            RCODE := X0
            END
         ELSE RCODE := X1;
         END;
      END;

BEGIN "INIT"
      WITH IO_RECORD DO
         BEGIN
         INSTC0] := IO_CHANNEL_IMMEDIATE;
      "IF BUSY THEN WAIT_INTERRUPT"
            INSTC5] := IO_SENSE;
            INSTC6] := IO_COMPARE_IMMEDIATE;
            INSTC7] := BSY_STAT;
            INSTC8] := IO_JUMP_FALSE;
            INSTC9] := 11;
            INSTC10] := IO_WAIT_INTERRUPT;
         INSTC11] := IO_START_OUTPUT;
         INSTC12] := NO_COMMAND;
         INSTC13] := IO_WAIT_INTERRUPT;
         INSTC14] := IO_RETURN;
         END "WITH IO_RECORD";
END "XMT_CLASS";
END.
```

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

by

ERWIN LYNN REHME

B.S., Kansas State University, 1975

AN ABSTRACT OF 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

Abstract

This report contains a description of the design and implementation of an asynchronous control line driver in the MIMICS network. The driver handles the functions necessary for transmitting and receiving of control information between computers within a cluster of the network. In the report we give a brief description of the MIMICS network and how the driver is used in that network. We then describe the use of asynchronous lines for communication, why they were chosen for this particular project, and how they are programmed on the Interdata 85 and the Interdata 7/16. It also tells how the computers were wired together to insure that the interface boards could detect abnormal conditions of the line. The implementation of the driver on the Interdata machines using assembler language and PASCAL is then presented, followed by a summary of the work completed and some extensions to conclude the report.