IN-04

61932

P-43

NASA Technical Memorandum 104175

A PC-BASED BUS MONITOR PROGRAM FOR USE WITH THE TRANSPORT SYSTEMS RESEARCH VEHICLE RS-232 COMMUNICATION INTERFACES

Wesley C. Easley

December 1991

National Aeronautics and Space Administration

Langley Research Center
Hampton, Virginia 23665-5225

Table of Contents

SUMMARY

Utilization of RS-232 serial data busses in the The Transport Systems Research Vehicle (TSRV) at the NASA Langley Research Center has increased beyond normal terminal interfaces. The Data Link and Differential Global Positioning Satellite (DGPS) flight programs conducted during 1990 made extensive use of RS-232 as a real-time experiment critical data transfer medium. At least two windshear experiments will also require this data bus for real-time transfer of critical data.

Various software modules, all task specific, are required for interfacing the TSRV computers and other peripherals connected via RS-232 busses. To enhance such development tasks, a PC-based RS-232 monitoring system has been developed. An important feature of the system is the capability for simultaneous real-time monitoring of two data lines with each line's input displayed in a separate window on the PC screen. A number of online selectable features such as binary display, transmission to either or both ports, and binary log files are incorporated.

The RS-232 bus monitoring system was developed by writing communication software for an IBM PC or compatible using two of the four normal communication ports (COM 1 through COM 4). Also an added advantage for TSRV use was gained by development of a special configuration for the TSRV Grid laptop computers, a task requiring software adaptation to a Grid nonstandard serial port.

User documentation and discussions of the algorithms used to develop the RS-232 bus monitor and control program are contained in this document. Detailed technical and tutorial discussions of C and assembly language programming are not included, but numerous references containing that type of information are cited.

LIST OF FIGURES

CREDITS

Technical information regarding Grid Input/Output design and hardware interrupt configuration was provided by Jim Kuhfeld of the Grid staff.

1.0 INTRODUCTION

The Transport Systems Research Vehicle (TSRV) is a research flight system operated by the Advanced Transport Operating Systems Program Office (ATOPSPO) at the NASA Langley Research Center. Real-time data transfer among experimental system computers and other equipment units in the TSRV system is accomplished using a variety of data bus types. ARINC 429, Digital Autonomous Terminal Access Communication (DATAC), high-speed parallel Direct Memory Access (DMA), and specialized high bit rate serial busses are prominent examples.

RS-232 communication links have traditionally been used only for interfacing terminals to the host VAX minicomputers in the TSRV system. However, during 1990 this data bus became a critical link in two highly successful flight research programs--the Data Link and the Differential Global Positioning Satellite (DGPS) experiments. Up to four separate RS-232 links operating simultaneously were required for support during these flight test efforts with the major RS-232 subsystem in both programs being a Packet Radio link used for ground to air data transfer.

For support of the TSRV windshear detection flight program, significant real-time use of RS-232 busses for experiment-critical data will be required. The same Packet Radio system mentioned above will be used for data uplink in the Terminal Doppler Weather Radar (TDWR) flight tests. Also, the windshear infrared detector requires an RS-232 data bus for interface to airborne data recording and display systems.

This extensive use of RS-232 data busses requires generation of task-specific software for configuration of computer interfaces to a number of peripherals, both programmable and non-programmable. Serial communication hardware and firmware design variations among these peripherals have presented difficulties in implementing the required interfaces. For example some units, on initial power-up, transmit character strings containing bytes that can be interpreted as control codes by another connected unit. Troublesome results such as communication hang-up and mode changes often result. These extraneous characters can be difficult to identify using only the connected systems, especially if normal communication functions have been disrupted. However, once such problems are identified, software can usually be readily configured to avoid conflicts.

Thus, for efficient development of the TSRV requirements, RS-232 bus diagnostic and monitoring aids are needed for troubleshooting and debugging. This document describes one such troubleshooting tool which was developed to utilize the serial communication features of IBM PC's and compatibles, including the TSRV Grid laptops. The resulting system can function as a simultaneous on-line monitor for two RS-232 busses with input from each bus being displayed in a separate window on a PC screen. A third window displays data from a local device such as the keyboard or a disk

file. Text or binary display of the received data can be selected. In the binary mode all received data bytes, printable or not, are displayed in Hex form permitting detection of any potentially troublesome control characters which might be missed by normal text display. Log files to save (in binary form) all received bytes and instantaneous screen capture to a file are online selectable features. Also, transmission from a local keyboard or disk file to either or both monitored ports is selectable online in case output of test or configuration information is needed.

2.0 LIST OF ABBREVIATIONS

ALT	Alternate Key on PC Keyboard
ARINC	Aeronautical Radio, Inc.
ASCII	American Standard Code for Information Interchange
ATOPSPO	Advanced Transport Operating Systems Program Office
AX	General Purpose 8086 Family 16-Bit CPU Register
AH	Upper Eight Bits of AX
AL	Lower Eight Bits of AX
BIN	Binary
BIOS	Basic Input/Output System
BIT	Binary Digit, One or Zero
BYTE	Eight Bit Data Unit
BX	General Purpose 8086 Family 16-Bit CPU Register
BH	Upper Eight Bits of BX
BL	Lower Eight Bits of BX
C	Name of a Computer Programming Language
CHAR	Character
COM	Communications Port of PC
CPU	Central Processor Unit
CR	Carriage Return
CRLF	Carriage Return/Line Feed
CX	General Purpose 8086 Family 16-Bit CPU Register
CL	Lower Eight Bits of CX
DATAC	Digital Autonomous Access Terminal Communication
DGPS	Differential Global Positioning Satellite

DMA	Direct Memory Access
DOS	Disk Operating System
DX	General Purpose 8086 Family 16-Bit CPU Register
F1-F10	PC Function Keys
FDX	Full Duplex
FIFO	First-In, First-Out
HDX	Half Duplex
Hex	Hexadecimal Number
IBM	International Business Machines
INOP	Inoperative
INT_NO	Interrupt Number
I/O	Input/Output
ISR	Interrupt Service Routine
MS-DOS	Microsoft Disk Operating System
NASA	National Aeronautics and Space Administration
PC	Personal Computer
RS-232	Serial Communication Standard
TDWR	Terminal Doppler Weather Radar
TSRV	Transport Systems Research Vehicle
XMIT	Transmit

3.0 USER'S GUIDE

Documentation describing the functions and use of the various features of the RS-232 bus monitor system are contained in this section.

3.1 General Description and Initialization

The RS-232 monitor system consists of an MS-DOS executable program, with filename "DUALCOM.EXE", written to run on MS-DOS computers. Figure 1 illustrates the screen appearance after the system is initialized by entering that filename from the DOS prompt. Three distinct windows will appear on the screen separated by horizontal reverse video bars containing various messages to indicate the status of the various operational modes. Identification information for each message is also shown in figure 1. The top window displays input from COM 1, the center window displays input from COM 2, and the lower window displays local input from the keyboard or a disk file.

As part of initial loading, the software checks internally for availability of communication port hardware and initializes the first two ports found. In the example of figure 1, it is seen from messages at the left of the top and center bars that hardware for communication ports one and two (COM 1 and COM 2) were found. Immediately to the right of the port indications, messages appear showing the communication parameters for each port. The default values of these parameters are contained in program code but online changes can be easily made for matching the parameters to those of devices connected to the PC serial ports. Port initialization also includes setup for interrupt driven two-way data communication. Techniques used to determine port availability and setup for interrupt driven operation are discussed in later sections of this document.

In normal operation, the PC will contain one or two serial communication ports connected to external RS-232 data lines. An example application is illustrated in Figure 2 wherein the receive lines of two PC serial ports are connected in parallel with the transmit and receive lines of an RS-232 bus which is actively passing data between two devices. The system is then monitoring and displaying data simultaneously from each line in separate windows. This creates a real time monitor of both input and output data for each operating RS-232 device. Figure 3 illustrates the PC screen for this case.

3.2 Communication Port Hardware Variations

Failure to find PC hardware communication ports at startup will be indicated by messages displayed on the screen. An example of this is shown in figure 4 where COM 2 hardware was not found, and the message "COM 2 INOP" appears at the left in the center bar where

port identification is normally displayed. The COM 2 window will then remain blank throughout the session.

3.3 Mode Selection Features

Selection and use of all operational modes are discussed in the following subsections. Keyboard entries for mode selection are shown in the help screen which, as shown in the center of the lower bar in figure 1, is displayed by entering ALT-Z (ALT and "Z" keys simultaneously). Figure 6 illustrates the help screen display wherein a window containing a help list is superimposed on the PC screen. In the following discussion all specified keyboard selection combinations, such as ALT-F1, means simultaneously pressing the specified keys.

3.3.1 Log File Toggle

Log files for saving received data are toggled on and off by entry of ALT-F1 from the keyboard. Activating this feature will result in opening two files in the default directory for separate storage of data bytes received from each communication port. The filenames are "COM1IN.DAT" and COM2IN.DAT". Storage is in binary form for exact preservation of all received bytes, printable or not. The log file status is indicated on the screen by the message at the right of the lower reverse video bar.

Normal program exit will result in proper closing of all open files after associated nonempty buffers are flushed. However, the monitor program uses its own intermediate 512-byte buffer to collect data for file storage. When this buffer fills, the appropriate file is opened, the buffer contents are written to it, and the file is once again closed. Thus, no more than the last 512 bytes of saved data from any session should be lost by improperly closed files; i.e. abnormal termination of program.

Once log files are created, all saved data will be appended to them. With each program startup, a date and time stamp is placed in each file to aid in correlation of operating sessions with various portions of appended files.

Keyboard entries displayed in the lower window are not stored in a file since the prime purpose of this program is accurate detection and storage of input data from the RS-232 lines it monitors.

3.3.2 Full/Half Duplex Mode

This mode, which determines whether keyboard entries are reflected on the screen, is toggled on and off by entering ALT-F3 from the keyboard. Half duplex (HDX) results in local screen reflection of keyboard entries while full duplex (FDX) does not. Normally full duplex mode is used when the remote system echoes locally transmitted keyboard entries which are then displayed upon return. Use of half duplex in that case will result in double display of

keyboard entries. Full duplex will almost never be applicable for this monitor program.

3.3.3 Exit to DOS

Entry of ALT-X from the keyboard executes normal program termination. A message will appear on the screen asking the user to confirm the request for exit. Entering upper or lower case "Y" or "RETURN" will result in exit to DOS, and the ESCAPE key or "N" (upper or lower case) will result in return to normal program operation. Before returning to DOS, the program will flush any buffers containing data for file saving and close all open files. Normal program termination will also reset interrupt vectors and communication port hardware to the configuration found at startup.

3.3.4 Carriage Return/Line Feed Toggle

This mode, controlled by entry of ALT-F3, determines whether a line feed (0A Hex) is appended in the screen display when the program receives a carriage return (0D HEX). Its setting affects all three windows. If a carriage return is displayed without a line feed, each new line will be written over the previous one resulting in display of only the last line received. Real time visual observation of displayed information is thus difficult or impossible. Appending the new line character to the carriage return will move all displayed information up one row before printing another line, thus preserving previous information until scrolling past the top of a window occurs.

If the incoming data stream already contains a new line character with each carriage return, then local appending will result in double spacing on the screen.

3.3.5 Moving the Cursor Among Screen Windows

Keyboard entry of ALT-W will move the cursor to another of the three screen windows. Repeated entry of this key combination will cycle the cursor position through all windows in turn. This feature can be useful for clearing the contents of a selected window, a feature described in the next subsection. Cursor positioning to print input data from a serial port or local device will not be affected by this entry. Automatic window positioning of the cursor for display of received data is done by program software based on the data source.

3.3.6 Clearing the Window Containing the Cursor

Pressing ALT-C will clear the contents of the window containing the cursor. Use of this feature with the cursor cycling described above will permit total screen clearing.

3.3.7 Transmission Modes

The primary purpose of this system is line monitoring which, as illustrated in figure 2, does not include hardware connection of the RS-232 transmit lines from the monitoring PC. For increased versatility, however, transmit modes are allowed and are online selectable from the keyboard. As shown in figures 1, 3, 4, and 5, transmit status is indicated by the "XMIT ON" or "XMIT OFF" messages in the right center of the COM 1 and COM 2 status bars. Entry of ALT-F2 controls the state of this mode which is individually selectable for each communication port allowing locally entered keyboard data to be transmitted to either, neither, or both ports. Repeated entry of ALT-F2 will cycle through this mode for both ports and the messages will change to properly indicate the status.

Ability to transmit from the monitoring system can be convenient if test or configuration data needs to be sent to a device being monitored. Figure 7 shows an example implementation which has been used for TSRV packet radio operations. The hardware switching arrangement permits toggling the receive line of each RS-232 device between its normal operating data source and the transmit line of the monitoring PC. This operating mode eliminates any requirement for hardware or software reconfiguration if new control parameters need to be sent to either RS-232 device. The default state of the transmit mode is off and care should always be exercised in its use to prevent parallel connection of two transmitters to the same receiver.

3.3.8 Upload of a File

A typical use of the transmit mode described in section 3.3.7 is sending a file containing new configuration parameters to a device such as a packet radio modem. Pressing the PAGE UP key will prompt the user for a filename to send. The filename must then be entered complete with MS-DOS path information. Only ASCII (American Standard Code for Information Interchange) files can be used. The ENTER key starts file transfer and a beep will signal its completion. Transmitted data characters scroll past in the lower screen window if half duplex is active.

3.3.9 Character or Binary Display Mode

By default all received data bytes are printed on the screen in normal character (text) mode. Figures 3 and 4 illustrate the resulting screen appearance. For reliable detection and identification of non-printable control characters, however, a binary display mode is available in which the Hex representation of all received bytes will be written to the screen. Figure 5 illustrates the resulting screen for this mode with 16 bytes displayed on each line. The right side of the display contains the normal text representation of all printable bytes with a period appearing for any that are non-printable.

The character/binary display mode can be individually toggled for each of the two upper windows. As shown by the help screen in figure 6, this mode for the upper and center windows is controlled by ALT-F9 and ALT-F10 respectively. Status of this mode is indicated by the messages "CHAR" or "BIN" at the right of the two upper bars. Binary display mode does not exist for the lower window.

3.3.10 Communication Port Parameter Select

Capability for online communication parameter selection and change are important features of any versatile serial communication program. This monitor system permits individual configuration of the ports associated with each of the two upper windows. As can be seen from the help screen in Figure 6, this feature in the upper and center windows is controlled by ALT-F7 and ALT-F8 respectively

Figure 8 illustrates the screen appearance when online parameter selection is requested. Two small menu windows are superimposed on the screen. One contains the baud rate menu while the other contains a menu of parity, word length, and stop bit parameters. Each menu window identifies the port in question and lists the currently active parameter value. When these menus are first displayed, the blinking cursor appears at the "SELECTION" prompt in the baud rate window indicating that this parameter is to be selected first. Valid selections are a menu number or carriage return to retain the present value. After a valid baud rate selection, the cursor moves to the other menu window for similar selection of remaining parameters. Then the menu windows disappear, the ports are reconfigured per the selections, and the parameter messages at the left of the two upper bars reflect the new values.

3.3.11 Saving Screen Contents to a File

Instantaneous snapshots of screen contents can be saved to a file by entry of ALT-G from the keyboard. A file "SCREEN.DAT" is created in the default directory for this storage and each captured screen is appended to this file. With each screen capture, the file is opened, screen data are written to it, and it is closed. Thus, abnormal program termination should result in loss of no captured screen data.

4.0 PROGRAMMING TECHNIQUES AND ALGORITHMS

Section 4.0 contains descriptions of many of the programming techniques and algorithms used to develop the RS-232 bus monitor system. Tutorial discussions of C and assembly language programming are not included, but numerous references to publications containing this type of information are provided.

4.1 General Description

The RS-232 monitor development effort consisted in large part of communication software development. Mixed language programming using Turbo C and Turbo Assembler for MS-DOS, both products of Borland International, was used. Prominent programming tasks included initialization of IBM PC serial communication hardware for interrupt driven data transfer, interrupt vector redirection, direct writing to video memory for rapid screen display, detection and handling of keyboard input, and file handling.
Figures 9 and 10 are flowcharts outlining total program operation. Functions for initial setup and keyboard input handling are shown in figure 9, and actions dealing with serial port input are shown in figure 10. Discussions in the following sections will refer to the labeled symbols shown in these two figures.

4.2 Check for Serial Communication Port Hardware

A test for availability of serial port hardware, element 9A in figure 9, is the first setup operation performed. During boot-up, MS-DOS and the IBM BIOS (Basic Input/Output System) check the status of hardware peripheral configuration and accordingly initialize memory at specified segment and offset locations. References 3, 6, and 9 are examples of a number of publications which provide information on the segmented architecture and mapping of IBM PC memory. Information in these publications shows serial communication port status information appearing at offset zero of segment 0040 Hex, or absolute memory location 400000 Hex. The first four 16-bit locations starting at this memory cell contain I/O (Input/Output) port addresses, beginning with COM 1, for all communication ports found by the BIOS at boot-up.

To retrieve the required port information, a C language pointer to an integer data type (16 bit word in Turbo C) is defined and initialized to point to memory location 400000 Hex. The following statement can be used to define "COM_IO" as such a pointer.

```
#define COM_IO ((unsigned int far *)(0x400000))
```

Pointers are, in fact, addresses. When locked to specific locations, pointers should be declared as "far" to ensure inclusion of both segment and offset address components. This allows the selected location to be accessed no matter where it resides in physical memory (See reference 1.). Using "COM_IO", the contents of the required four 16-bit memory elements can be acquired and placed into an array with the following C code:

```
index = 0;
while( index <= 3 )
   cm_port[index] = *(COM_IO + index++ );
```

Here "index" must be declared as an integer and "cm_port[4]" must be declared as an array of unsigned integers. The expression

*(COM_IO + index) means fetch the contents of the 16-bit location pointed to by (COM_IO + index).

The notation "index++" in the while loop causes the integer variable "index" to be incremented after each of four iterations. This, in effect, increments an integer pointer which causes a 16-bit value to be read from each of four successive locations. Each 16-bit value read is a port address which is placed in successive elements of the array "cm_port". Array elements must then be checked against I/O addresses in the following table to determine which ports were found.

Port	I/O Address
COM 1:	03F8 Hex
COM 2:	02F8 Hex
COM 3:	03E8 Hex
COM 4:	02E8 Hex

As depicted in elements 9B and 9E of figure 9, checks are first made in turn for COM 1 and COM 3. If COM 1 is located then no check is made for COM 3 since these ports, while using different I/O port addresses, share a common interrupt number and the monitor program will not resolve the resulting conflicts. If either is found, it is initialized (elements 9C and 9F) and a flag is set to so indicate (element 9D).

The same procedures are then performed to check for the existence of COM 2 or COM 4, actions represented by elements 9G, 9J, 9H, and 9L in figure 9. Again only the first of these two ports found will be used because they also share a common interrupt number. Element 9I shows a flag being set if either of these two ports have been initialized for use.

All supported ports found will have been setup for use after completion of the action represented by element 9I. Possibilities are:

1. Any one of the four ports
2. COM 1 and COM 2
3. COM 1 and COM 4
4. COM 2 and COM 3
5. COM 3 and COM 4

If no ports at all are detected, exit to DOS will occur as shown in elements 9K and 9Y.

4.3 Communication Port Initialization

Initialization of the communication ports selected for use is represented by elements 9C, 9F, 9H, and 9L of figure 9. Tasks involved are setting communication parameters, changing interrupt

vectors, and writing interrupt service routines (ISR's) specific to program requirements.

4.3.1 Interrupt Interception and Redirection

Occurrence of either a hardware or software interrupt in the IBM PC results in immediate redirection of processing to the address of a corresponding ISR. Resulting action will be determined by processor instructions placed in the ISR by the programmer. All ISR addresses for the IBM PC are contained in interrupt vector tables which comprise the lowest 1024 memory locations. Each interrupt vector consists of a four-byte block with each block having an associated interrupt number, or type, starting at zero and continuing through 255.

Application programs can intercept an interrupt by changing the contents of its vector thus directing processing to a different address when the interrupt occurs. This new address must contain an ISR written by the programmer to perform the specific actions desired. Internal MS-DOS software services accessible via assembly instructions are provided for this vector redirection; but, most C compilers now contain higher level functions to accomplish this. In Turbo C the interrupt vector redirection functions are "getvect()" and "setvect()" which are used as follows:

```
entry_handler = getvect(INT_NO);
```
Fetch the address of the handler for interrupt type INT_NO and place it into the variable "entry_handler". This permits reset at program exit.

```
setvect(INT_NO, prog_handler);
```
Replace the interrupt vector for INT_NO with the address of the function "prog_handler" which points to a user-written ISR.

The variable INT_NO represents the number associated with the hardware or software interrupt being redirected. These Turbo C library functions were used in the monitor program described herein to intercept hardware interrupt numbers 0C Hex and 0B Hex. These numbers correspond to COM 1 and COM 2 interrupts respectively from the PC communication device, an 8250 serial interface controller. Reference 6 contains complete technical data for the 8250 serial controller including hardware register structure, associated interrupt types, and control bits required for register configuration.

4.3.2 Communication Parameter Setup

Communication parameter setup is accomplished by writing control bits into the various eight-bit registers of the 8250 serial controller. Parameters requiring attention are baud rate (bits per second of data to be transferred through the port), number of

stop bits transmitted after each data byte, parity (even, odd, or none), and word length for each byte (seven or eight bits). Values of 1200 baud, no parity, one stop bit, and eight bit word length are default settings for initial boot-up of the monitor program. Other parameter configurations are available for online selection and are listed in the menus shown in figure 8. The monitor program contains tables of eight-bit binary values required to configure the 8250 registers for each of these supported configurations. Software functions obtain proper binary values for required register configuration by using menu selections from figure 8 as pointers into these tables.

Reference 6 contains a technical description of the 8250 register structure and control bit configurations required for all supported functions.

At startup, the existent contents of the 8250 registers are read and stored before writing any new setup information to them. These entry parameters are then used at program exit to restore each port status to its original condition.

After completion of communication port setup, program flow will have progressed through element 9L of figure 9.

4.4 Screen Layout Implementation

The final setup task involves creation of the screen layout illustrated in figure 1 and represented by element 9N in figure 9. Screen configuration is accomplished by writing text characters directly into video memory, a technique which enhances program performance by very rapidly creating a screen display.

4.4.1 Identification of Video Memory Locations

Before any direct memory writing is done, proper memory locations must be identified to prevent overwriting critical system information. Determination of the location of the active video page is required since some graphics hardware adapters can simultaneously store several screens, or pages, of video memory. Use of Read-Only Memory (ROM) Basic Input/Output System (BIOS) functions is an effective method for obtaining current values of video mode and video page which can, in turn, be used to identify the current video memory starting location for the system currently in use. Reference 2 provides a description of this technique.

An integer pointer initialized to the starting location of current video memory can then be used to write directly into memory cells corresponding to any row and column screen position. A pointer to an integer (16 bits) must be used, since two bytes are required to control the display at each screen position, one byte being the character to print and the other its display attribute. Examples of attributes for a monochrome display are normal video (white on

black), reverse video (black on white), and blinking video. A 25 row by 80 column text screen will contain two thousand character locations. Thus two thousand integer values in memory (4K bytes) are required for a full screen of text display.

4.4.2 Accessing Video Memory

A pointer variable "vidpointer" for use in selecting video memory locations can be declared in the following manner:

```
unsigned int far *vidpointer;
```

As was the case in section 4.2 above, declaration of a "far" pointer is needed to allow access to locations anywhere in physical memory. The value assigned to this pointer will be determined by the results of ROM BIOS checks for graphics hardware configuration (See reference 2.).

A general C function of the following form was written for directly writing one or more identical characters to video memory:

```
1. void vid_write(char ch, unsigned char attr,
                  int first_cell, int cell_ct)

{
  2. int v_off, last_cell; unsigned int vid_cell;
  3. last_cell = first_cell + cell_ct;
  4. vid_cell = bytes_to_int(attr, ch);

  5. for( v_off = first_cell; v_off < last_cell; v_off++);
  6.      *(vidpointer + v_off) = vid_cell;
  7. return;

}   /* End of Function /*
```

Statement 1 defines the function and the parameters that must be passed to it. A variable "ch" is the character to be written to video memory and thus immediately displayed on the screen. The display attribute is passed to the function through the parameter "attr". Another parameter "first_cell" is the starting location, offset from the start of video text memory, where writing is to occur. The value "cell_ct" is the number of consecutive locations into which the character is to be written. Statement 4 combines the two byte values, "ch" and "attr", into the unsigned integer "vid_cell" for fitting into a 16-bit video memory data element. Another function "bytes_to_int()" is called to form the 16-bit value. Statements 5 and 6 constitute a loop to actually place the variable "vid_cell" into the requested number of consecutive video memory locations. The expression in statement 6 means place the value of "vid_cell" into the memory location pointed to by "(vidpointer + v_off)" where the integer "v_off" is an offset from

the beginning location of video text memory. Looping with "v_off" being incremented in each iteration will result in the appearance of a string of identical characters on the screen.

4.4.3 Printing Reverse Video Bars and Status Messages

Passing a space character (20 Hex) to the function "vid_write()" with reverse video attribute and a cell count of 80 is used to produce each of the three horizontal bars shown in figure 1.

Status messages in the reverse video bars are also produced by writing character strings directly into video memory. A general C function of the following form was written for this purpose:

```
1. void string_to_vmem( int vm_off, unsigned char attr,
                        char *c_ptr )

   {
     2. while( *(c_ptr) != '\0')
     3. *(vidpointer + vm_off++) =
          bytes_to_int(attr, *c_ptr++);
     4. return;

   {      /* End of Function */
```

This function is similar to the function "vid_write()" shown in the previous section except that a "while" loop replaces the "for" loop. A pointer to the character string to be written is passed to this function through the variable "char *c_ptr" in statement 1. A single display attribute applies to all characters in the string and is given to the function by the expression "unsigned char attr" in statement 1. Each pass through the while loop in statements 2 and 3 increments both the video memory offset and the string's character position. Also, with each pass the character and attribute bytes are combined into a 16-bit word and written into the cell pointed to by "*(vidpointer + vm_off)".

An example definition of a string constant that can be passed to this function is:

```
#define PORT_ID_STR "COMM 1";
```

The name of the string, "PORT_ID_STR", is a pointer to its first character. Printing this string on the screen is accomplished by passing its name ("PORT_ID_STR"), its desired attribute, and its starting video memory offset to the above function. A video memory offset value corresponding to a row and column position on the screen is obtained by multiplying the row number by 80 and adding the column number.

All the messages shown in figure 1 are defined as string constants and printed to the screen using the above-described function "string_to_vmem()". Default messages are printed at boot-up, but

online changes are made in response to the keyboard entries listed in the help screen of figure 6.

4.5 Main Program Operation

After screen initialization, element 90 of figure 9 has been reached and the monitor program enters a loop of indefinite duration which can be terminated by entry of ALT-X by the user. Program looping repeatedly checks, in turn, the keyboard buffer and two buffers configured to separately collect input bytes from each communication port. Various resulting actions occur as a function of program operational modes and values of the data bytes fetched from these sources.

4.5.1 Checking for Keyboard Entry

Element 90 in figure 9 represents the monitor program's check for any keyboard entries which would place characters in the PC's internal keyboard buffer. The Turbo C library function "kbhit()" provides a convenient means of fetching a waiting keyboard entry. An example of its use is:

```
int key_in;
if( kbhit() )
  key_in = getch();
else
  { No keyboard entry waiting. Execute next program
     statement. }
```

If a key has been pressed, its value will be placed in the variable "key_in". Otherwise, program flow continues to the next instructions which check the buffers holding any data received from the serial ports, (operations illustrated in figure 10 and described later.)

4.5.2 Keyboard Mode Control Entries

Continuing with the flow chart in figure 9, it is seen that if keyboard activity is found then a check is made to determine whether it is one of the mode control functions listed in the help screen of figure 6. If so, and if ALT-X was entered, the program will terminate with normal exit to DOS. Flow chart elements 9P, 9Q, and 9Z illustrate this path. Normal exit will result in closing all open files and resetting both the interrupt and communication parameter status of each serial port to the conditions found at program start-up. All the necessary entry information was saved during the port initialization procedures described earlier in section 4.3.2. Other appropriate mode control actions, such as opening log files or changing communication parameters, will be performed if the requested mode control was not the exit command.

Figures 6 and 8 illustrate mode control actions which utilize menu windows superimposed on the screen. Each of these menus is displayed by writing an array of string constants to video memory. The following baud rate selection menu is an example of such an array of string constants:

```
char *baud_lst[11] =
        {    " BAUD RATE SELECT   ",
             "====================",
             "     <1> 300        ",
             "     <2> 1200       ",
             "     <3> 2400       ",
             "     <4> 4800       ",
             "     <5> 9600       ",
             "     <6> 19200      ",
             "     <7> 38400      ",
             "                    ",
             "     SELECTION      " };
```

Each array element is a string constant which can be written to video memory through a call to the function "string_to_vmem()" described earlier in section 4.4. A loop of 11 iterations with each iteration calling this function will then display the entire table one string at a time. With each iteration, the offset into video memory for writing is increased by 80 integer elements to place each successive string constant in the same column of a new line. This screen writing method is very fast and results in a menu window that pops up instantly with keyboard selection.

Underlying screen contents must not be destroyed by any superimposed menu window. A convenient method of accomplishing this task is reading the contents of the screen's video memory into a buffer, displaying a menu such as shown above, making a menu selection, and then writing the saved buffer back into video memory. Restoring the screen from the buffer will overwrite and thus erase the temporary menu. Turbo C compiler packages contain library functions "gettext()" and "puttext()" to respectively capture and restore the screen contents using a four kilobyte buffer defined by the programmer.

Capture of the screen contents to a file also involves reading the contents of video memory. When this operation is activated, by entry of ALT-G from the keyboard, 2000 integer locations are read beginning with the location pointed to by "vidpointer" (See subsection 4.4.2.). Then the attribute byte of each resulting integer is discarded and the remaining character byte is written into the file "SCREEN.DAT" in the default directory. Thus, the file will contain only printable ASCII characters.

Reference 1 contains detailed treatments of file handling and all other programming techniques used to implement the actions described in this subsection.

4.5.3 Keyboard Entries Not Requiring Mode Control Actions

Elements 9S, 9T, 9U, and 9V of figure 9 illustrate the path taken by the monitor program if the keyboard entry found in element 9O does not require a mode control function. If half duplex is active, the keyboard entry will be printed to the screen in the lower window as shown in figures 1, 4, and 5. Programming techniques used for this screen printing operation are described in subsection 4.5.3.1 below.

Finally the need for transmission of the keyboard entry to one or both communication ports is checked as shown in element 9U of figure 9. If transmit mode is active for either or both ports, the keyboard entry is sent out accordingly using methods described in subsection 4.5.3.2 below. If transmit is not active, program flow proceeds to elements 9W and 9X for checking of the communication ports for any waiting input.

4.5.3.1 Printing to the Screen

For enhanced performance, low-level ROM BIOS functions called by assembly language routines are used both to position the cursor to the screen position where a character is to be printed and to actually print the character. As described in detail in references 2 and 4, BIOS interrupt 10 Hex provides a number of video services including cursor positioning, setting the cursor size, scrolling up or down in a window, clearing a screen window, obtaining the current video mode, and screen printing.

Each of the three screen windows is defined by selecting row and column values for its corners. The first character printed in a window will be placed at the lower left corner, or the zero window position. Current position for writing the next character in each window is maintained by a counter which is incremented after every write operation. A flag, unique to each of the three input data sources, directs each screen write operation to the correct window.

An example assembly language routine to position the cursor at row 10 and column 15 follows:

```
_CUR_POSITION  PROC NEAR ;Name of Assembly Procedure
  MOV   DH,10            ;Row number in DH register
  MOV   DL,15            ;Column number in DL register
  MOV   AH,2             ;BIOS cursor position service
  INT   10H              ;Transfer control to BIOS
  RET                    ;Return to calling routine
_CUR_POSITION  ENDP      ;End of Assembly Procedure
```

All calls to the BIOS follow this general method. Register AH must contain the BIOS service number, in this case 2, associated with the desired task. Printing a character to the screen at the

present cursor position using BIOS interrupt 10 Hex can be done with the following procedure:

```
_SCREEN_PRINT  PROC  NEAR      ;Name of Assembly Procedure
  PUSH    BX                ;Save BX register contents
  PUSH    CX                ;Save CX register contents
  MOV     AH,9              ;BIOS screen print service.
  MOV     BH,VID_PAGE       ;Current video page.
  MOV     BL,VID_ATTR       ;Current display attribute.
  MOV     CX,1              ;Number of characters to print
  INT     10H               ;Transfer control to BIOS.
  POP     CX                ;Restore CX register contents
  POP     BX                ;Restore BX register contents
  RET                       ;Return to calling routine.
_SCREEN_PRINT  ENDP         ;End of Assembly Procedure
```

Notice here that register AH contains the value nine which is the screen-print service for BIOS interrupt 10 Hex. The current video page for use in register BH above can be determined with another call to BIOS interrupt 10 Hex using service number 15 in register AH. Examples of additional uses made by the RS-232 monitor program of BIOS interrupt 10 Hex include completely clearing a window and scrolling up one line in a window when printing reaches the maximum right screen column. Most assembly procedures such as that listed above are written to be called from C functions.

Reference 4 contains complete descriptions of all the Central Processor Unit (CPU) registers in the IBM PC as well as technical descriptions and example uses of all DOS and BIOS interrupt services.

4.5.3.2 Transmission to COM Ports

Transmission of a byte to a COM port is accomplished by writing the byte to the data register of the 8250 serial communication device. The Turbo C library function "outportb()" can be used for this task. Use of this function takes the form:

```
char out_byte;
if( 8250_XMIT_BUFFER_EMPTY )
  outportb( out_byte, COM_IO_ADDRESS );
```

In this example, the parameter "COM_IO_ADDRESS" will be one of the I/O port addresses listed in the table of subsection 4.2. The variable "out_byte" is the value of the byte to be transmitted.

Before sending any output byte to a COM port, a status check of the transmit buffer of the 8250 serial device is needed. A nonempty buffer indicates that a previous transmission is not complete, and data loss will likely occur if a new byte is written. This test consists of checking a bit in one of the 8250

registers, a task performed by the second line of the above example.

Technical data needed for all 8250 register programming used in this subsection are provided in reference 6. That reference also describes and illustrates the I/O programming methods used in the above example.

4.5.4 Communication Port Input Handling

Any data byte received by the 8250 serial device at either COM port will generate a hardware interrupt which immediately calls a corresponding ISR. As its first step, the ISR reads the input byte from the 8250 data register, an operation which can be conveniently performed using the Turbo C library function "inportb()". An example use of this function is:

```
char in_byte;
in_byte = inportb( COM_IO_ADDRESS );
```

Here the variable "in_byte" assumes the value of a byte fetched from the port with I/O address "COM_IO_ADDRESS" which will be one of the addresses listed in the table of subsection 4.2 above. Next, the ISR will place "in_byte" into a software circular First-in, First-out (FIFO) buffer, the functioning of which is illustrated in figure 11 and described below.

4.5.4.1 Software First In First Out (FIFO) Circular Buffer

A circular FIFO buffer is created in software and used to temporarily store input data bytes received from an interrupt driven serial communication port. Reference 6 contains a detailed description of this commonly used buffering technique.

As can be seen from figure 11 the buffer functions in conjunction with two independent operations, one to deposit data and another to retrieve it. In the monitor program described herein, the ISR, which is activated when a byte is received from a communication port, is the buffer's writing operation; while the tasks depicted by elements 10A and 10C in figure 10 perform data retrieval functions. Separate input and output pointer variables (see figure 11) control buffer locations where data are written into and read from, respectively. Both these pointers are set to zero at program startup and, thus, initially point to the bottom of the buffer. An operation depositing a data byte will write it into the cell pointed to by the input pointer. Then the input pointer variable must be incremented to select the next cell available for writing. In similar fashion, a routine fetching a byte will read it from the cell pointed to by the output pointer and then increment that pointer variable to select the next cell from which reading must occur. Both pointers will eventually reach the buffer's top at which time they are reset to zero to once again point to the bottom. Thus, the buffer is circular in nature.

Read and write operations may occur totally independently, but both begin at the bottom and separately increment pointers specific to each operation. Therefore, cells will be read from in the same order as they were written into, yielding a FIFO buffer.

Testing for COM port input, an operation depicted by element 10A of figure 10, consists of checking the FIFO buffer for any waiting bytes. This is accomplished by comparing the values of the input and output buffer pointers. If the pointers are equal, they point to the same location; and, thus, no byte is waiting to be read. This indicates that read operations have reached the last buffer location which was written into. However, if the input pointer is greater than the output pointer, then one or more bytes are available for retrieval.

4.5.5 No Communication Port Input Waiting

Tests represented by elements 10A and 10C of figure 10 fail if no bytes are available in the FIFO buffer. When this happens for both COM ports, program flow will once again loop back to element 90 of figure 9 and perform another test for keyboard entry.

4.5.6 Byte Waiting in COM 1 Input Buffer

When a waiting byte received from COM port 1 is fetched from the input buffer, a check is made in element 10H to determine the mode for printing it on the screen. If normal character mode is active, screen positioning and printing is accomplished by methods described in subsection 4.5.3.1 above. If binary display mode is active, then a hexadecimal representation of the byte is displayed by separating it into two characters which are then individually printed. This binary mode, which is described in the following subsection, is useful for display of the actual bit structure of non-printable bytes which would appear either as a graphic symbol or not at all in the character display mode. An example of this display mode is shown in figure 5.

4.5.6.1 Hexadecimal Display of Received Bytes

A commonly used algorithm for display of bytes in hexadecimal form is described in detail in reference 8. A general description of its specific use in the monitor program described herein is contained in the following paragraphs.

Every eight-bit byte can be represented in hexadecimal form by two printable ASCII characters. Individual printing of these two characters will result in display of the actual bit structure of the byte. This individual printing begins by placing each character into the lower halves of two separate bytes. For example, the byte with hexadecimal value BF is not printable and will not display an alphanumeric character when processed using normal character video. However, the two characters can be individually written as such if they are placed into separate

bytes with values adjusted to correspond to "B" and "F" in the ASCII table.

A new byte resulting from a four bit right shift of the byte BF Hex will have the value 0B Hex. This isolates the most significant character of the byte BF Hex. Also, a new byte resulting from the logical AND of BF Hex and 0F Hex will have the value 0F Hex, thus isolating the least significant character of the byte BF Hex. However, additional adjustment to the values of the two new bytes is required before their direct printing will display the ASCII characters "B" and "F". This adjustment consists of adding a constant to the bytes to make them equal to the hexadecimal representation of the ASCII values of the respective characters. Reference 8 presents an algorithm for accomplishing this. It consists of adding 48 decimal to bytes with values between zero and nine or adding 55 decimal to bytes with values between "A" (10 decimal) and "F" (15 decimal). In the example discussed here (byte BF Hex), a value of 55 decimal would be added to each of the two bytes 0B Hex and 0F Hex to yield 42 Hex and 46 Hex, respectively. Consultation of Reference 8 and a standard ASCII table shows that normal character printing of these last two bytes will result in display of "BF", the original binary byte. Each final printable ASCII character is written to the screen using the ROM BIOS method described in subsection 4.5.3.1 above.

On the right side of the binary display screen, as can be seen from figure 5, character printing also occurs for any bytes that are printable. A decimal is displayed in this section for any that are non-printable. Thus, for binary display, the screen window is divided into left and right sections with each received byte being printed in a different format in each section.

4.5.7 Log File for Received Bytes

Next a status check of the log file feature is made in the step depicted by element 10J of figure 10. If the log file feature is not active, COM Port 1 operations are complete for the current loop iteration. Program flow then proceeds to element 10C which will trigger operations for COM Port 2 identical to those described in subsections 4.5.5, 4.5.6, and 4.5.6.1 above.

Active log file status will result in each received byte being initially placed into a local intermediate buffer (element 10M of figure 10). This buffer, which is 512 bytes in size, is a circular FIFO buffer very similar to that illustrated in figure 11. A buffer pointer is initially set to zero incremented with each write operation. When the buffer's top is reached, the log file is opened and the contents of the buffer are read into it beginning with the first buffer location written. Then the file is closed and the buffer pointer is reset to zero. This permits the file to remain closed for most of an operating session since it needs to be open only when receiving the contents of a buffer.

Thus, any abnormal program termination should cause loss of at most one full buffer (512 bytes) of saved data.

Log file handling operations for COM Port 1 are represented by elements 10M, 10R, 10Q, and 10S of figure 10. High-level Turbo C library functions, which are described in reference 1, are used for file handling.

4.5.8 Input from Communication Port 2

After log file operations for communication port 1 are completed, the entire procedure described in subsections 4.5.5, 4.5.6, 4.5.6.1, and 4.5.7 above is repeated for COM Port 2. A separate FIFO circular buffer, identical in operation to that described in subsection 4.5.4.1, is used to collect incoming data bytes from COM Port 2. Depending upon whether a byte is waiting and the log file status, COM Port 2 operations will end with tasks depicted in elements 10D, 10P, or 10T in figure 10. At this time program flow will return to element 9O of figure 9 and begin another complete iteration.

4.6 TSRV Grid Applications

Terminals for use with the TSRV VAX host computers now consist of five Grid series 1530 laptop computers. These units, however, are 80386 based PC clones and are readily adaptable to numerous additional applications. Compatibility of the Grid laptops with a number of quality compiler and assembly development packages permits many extended applications through software alone.

The TSRV Grid laptops were delivered with only one externally accessible serial port. This is an acceptable configuration for terminal use but is a significant limitation for extended RS-232 applications such as the dual port monitor system described in this document. Added RS-232 capability was accomplished, however, by implementing a second communication port through adaptation of a Grid internal bus expansion feature. Development of low-level communication software, configured for compatibility with bus expansion hardware purchased from Grid, accomplished this task. A Grid version of the dual port RS-232 bus monitor software was then written which includes a user interface identical to that described in section 3.0 of this document. Significant low-level software differences exist between the Grid and normal PC versions of the monitor program, but they are transparent to the user. Development of this Grid serial expansion interface for more general RS-232 applications is described in a separate technical memorandum by the author.

5.0 CONCLUDING REMARKS

Increased use of RS-232 data busses in the TSRV system requires continuous development of task-specific software for interfacing

peripherals which have different characteristics. Software development and testing are aided by the ability for both off-line determination of peripheral interface characteristics and real-time monitoring of operating data busses. The RS-232 bus monitor system described in this document represents a response to these needs.

For increased versatility, the monitor system is configured for general use on IBM PC clone computers including the Grid laptop units in the TSRV system. Many specific features required for adaptation of the program to different applications can be readily implemented through software changes. Prominent features of the monitor system include ability for simultaneous connection to two data lines, ability for online control of transmission to these connected lines, and the ability to display and record data from both connected lines in binary form.

Three different configurations of the monitor program have been used to support the TDWR portion of the TSRV windshear flights. Specific TDWR applications include use in a Grid computer for in-flight disk file recording of computed data, in-flight monitoring of the packet radio data uplink, and interfacing of the TDWR data sources to the TSRV ground-based, packet-radio system.

REFERENCES

1. Lafore, Robert: Turbo C Programming for the PC, Howard W. Sams and Company, 1989.

2. Weiskamp, Keith: Advanced Turbo C Programming, Chapter 8, Academic Press, Inc., 1988.

3. Young, Michael J.: Inside DOS: A Programmer's Guide, Sybex, Inc., 1990.

4. Duncan, Ray: Advanced MS-DOS, Microsoft Press, 1986.

5. Dettmann, Terry: Dos Programmer's Reference, Que Corporation, 1989.

6. Barkakati, Nabajyoti: MS-DOS Developer's Guide, Second Edition, Chapter 8, Howard W. Sams and Company, 1988.

7. Barkakati, Nabajyoti, The Waite Group's Turbo C Bible, Howard W. Sams and Company, 1989.

8. Norton, Peter and Socha, John: Peter Norton's Assembly Language Book for the IBM PC, Chapter 5, A Brady Book published by Prentice Hall Press, 1986.

9. Holzner, Steven: Advanced Assembly Language on the IBM PC, A Brady Book published by Prentice Hall Press, 1987.

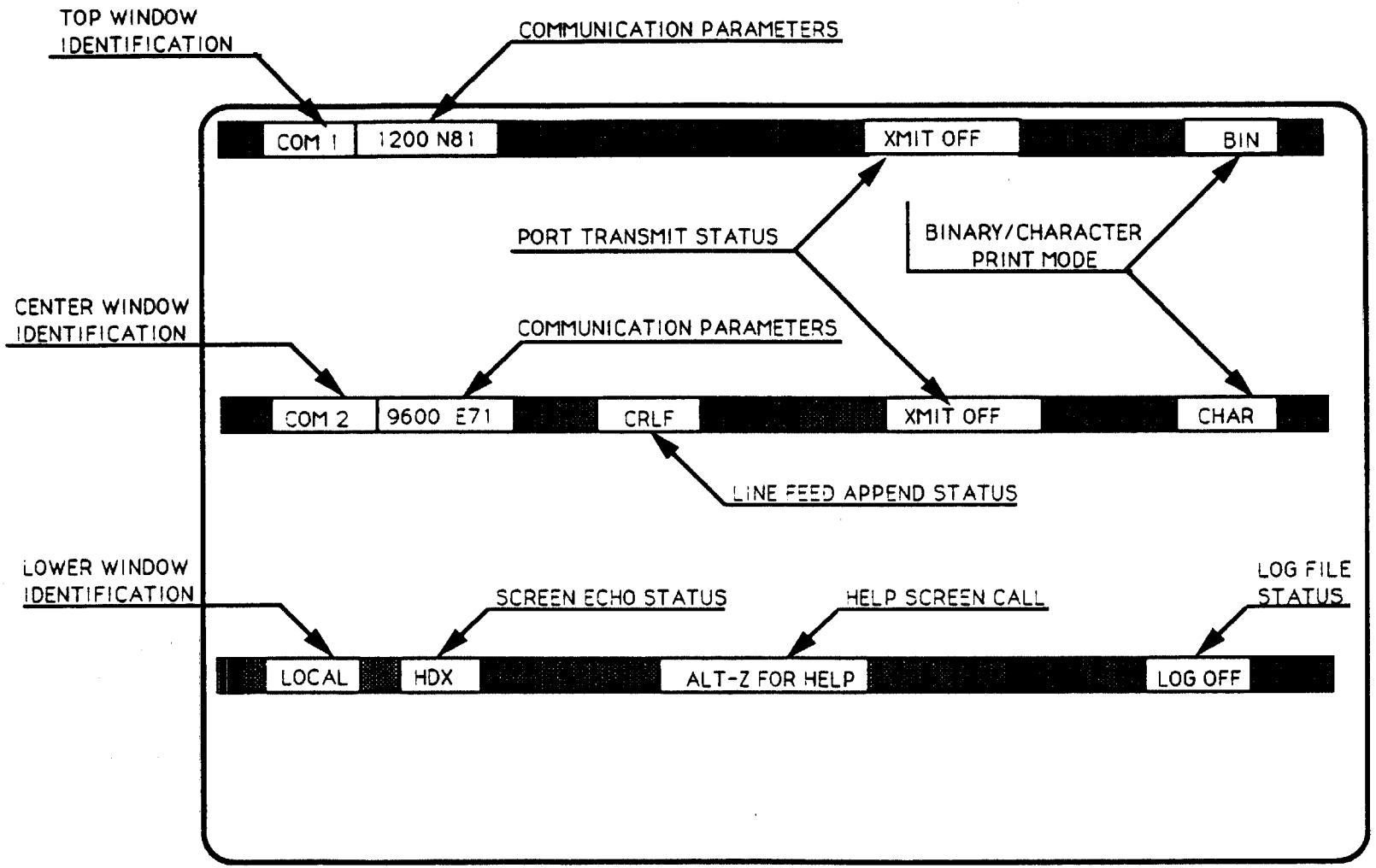

Figure 1. General Screen Layout.

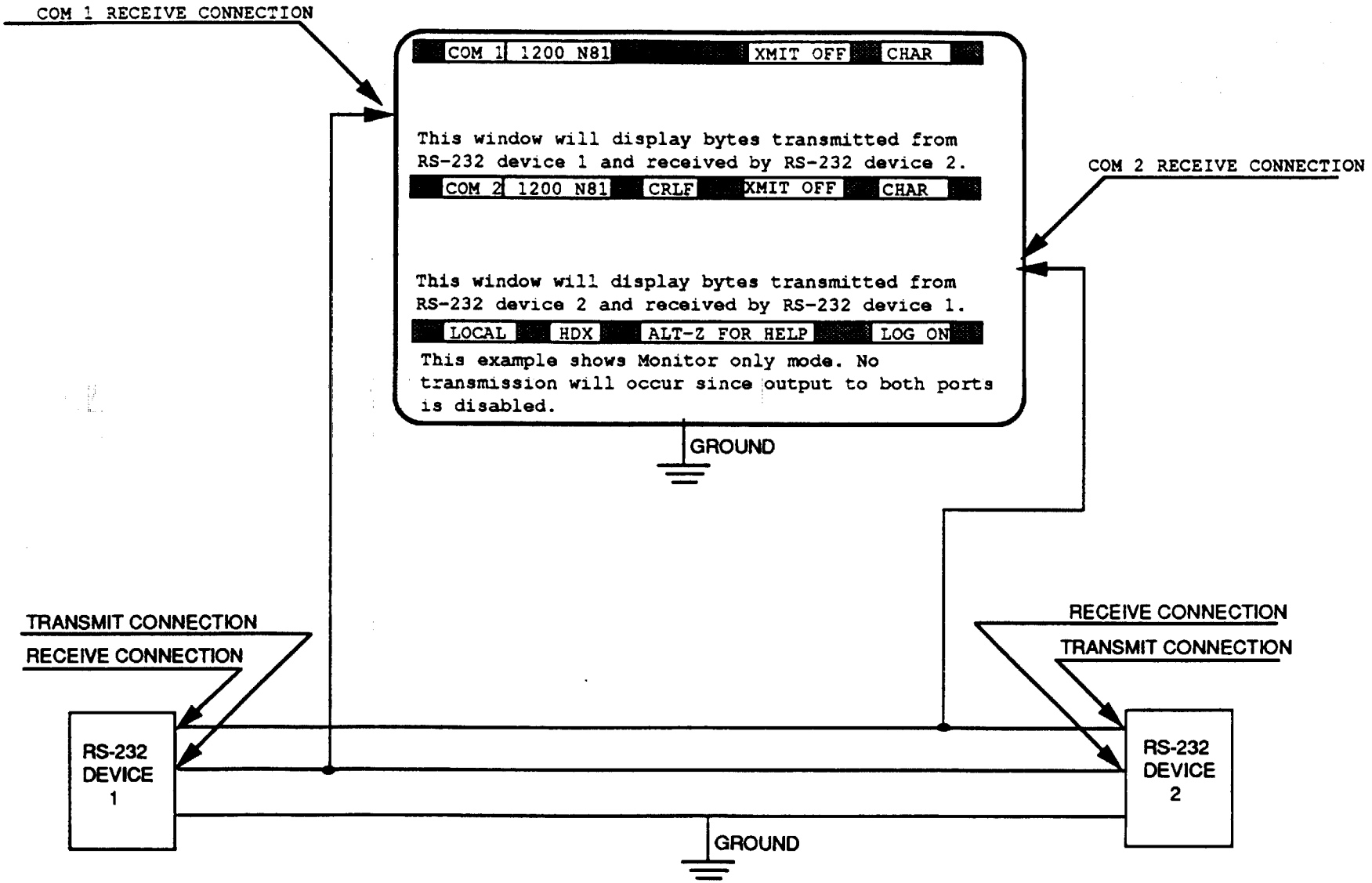

Figure 2. Typical Real Time Monitor Setup.

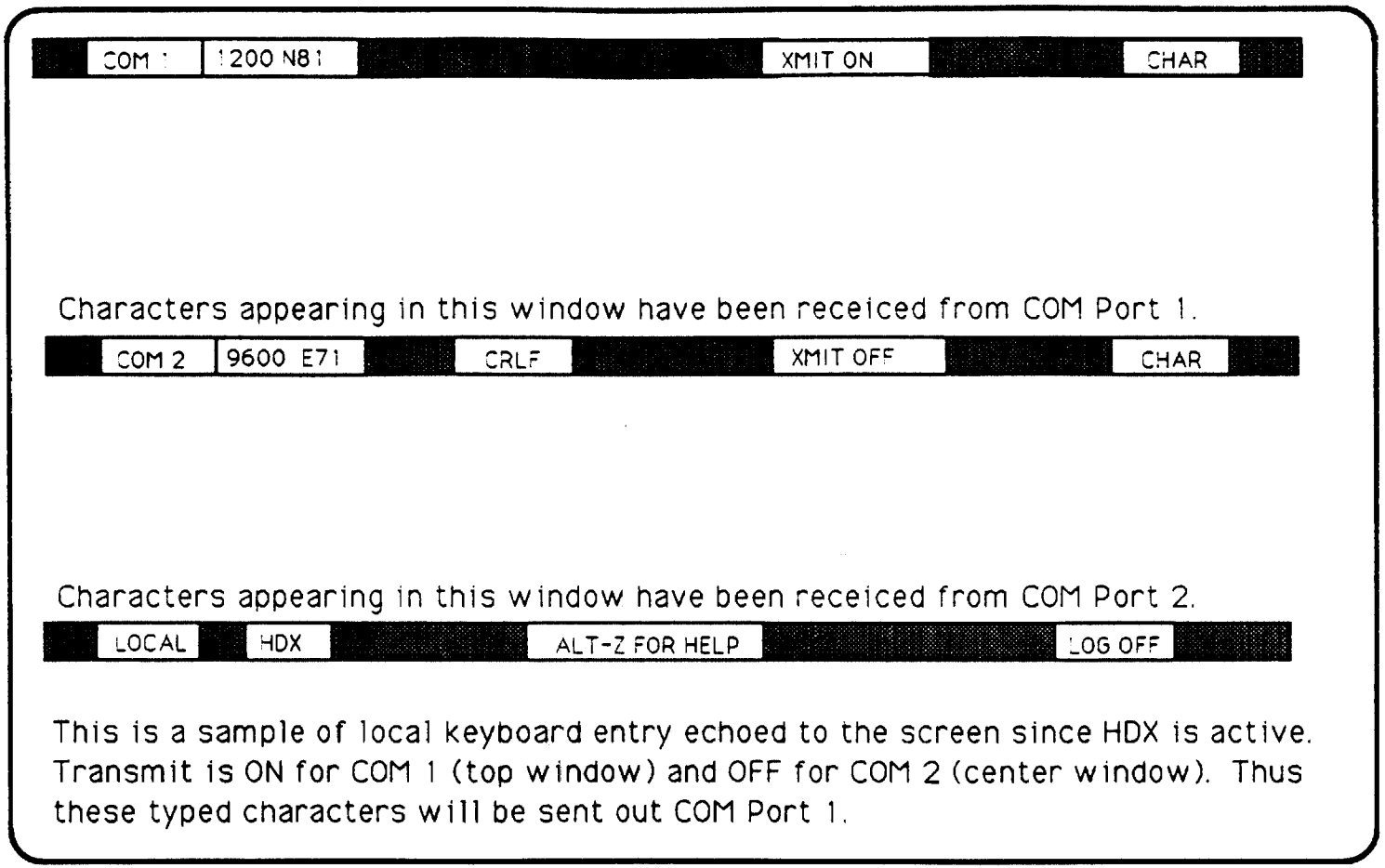

Figure 3. Example of Operation with Two COM Ports connected.

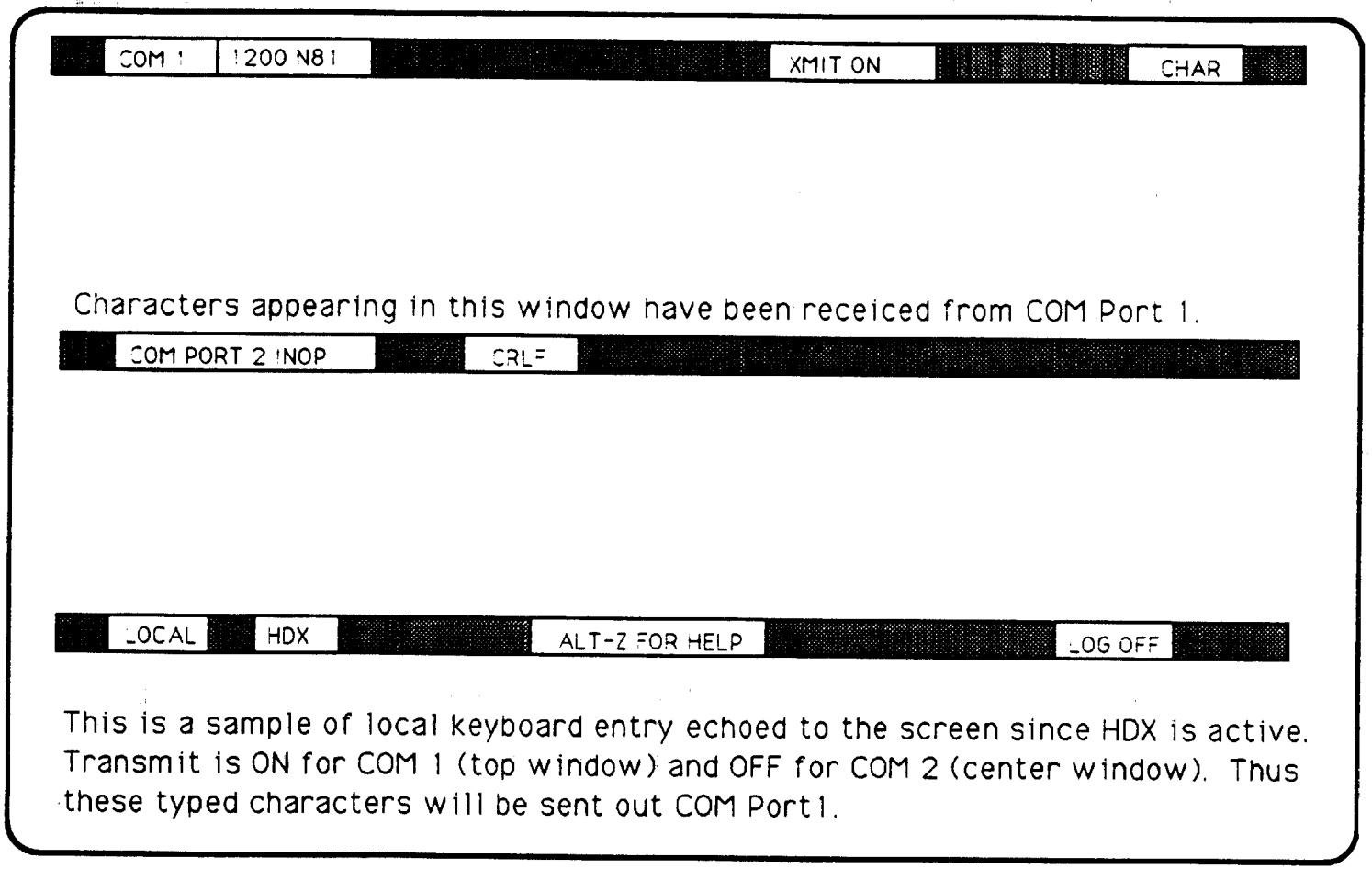

Figure 4. Example of Operation When COM Port 2 Hardware is Not Available.

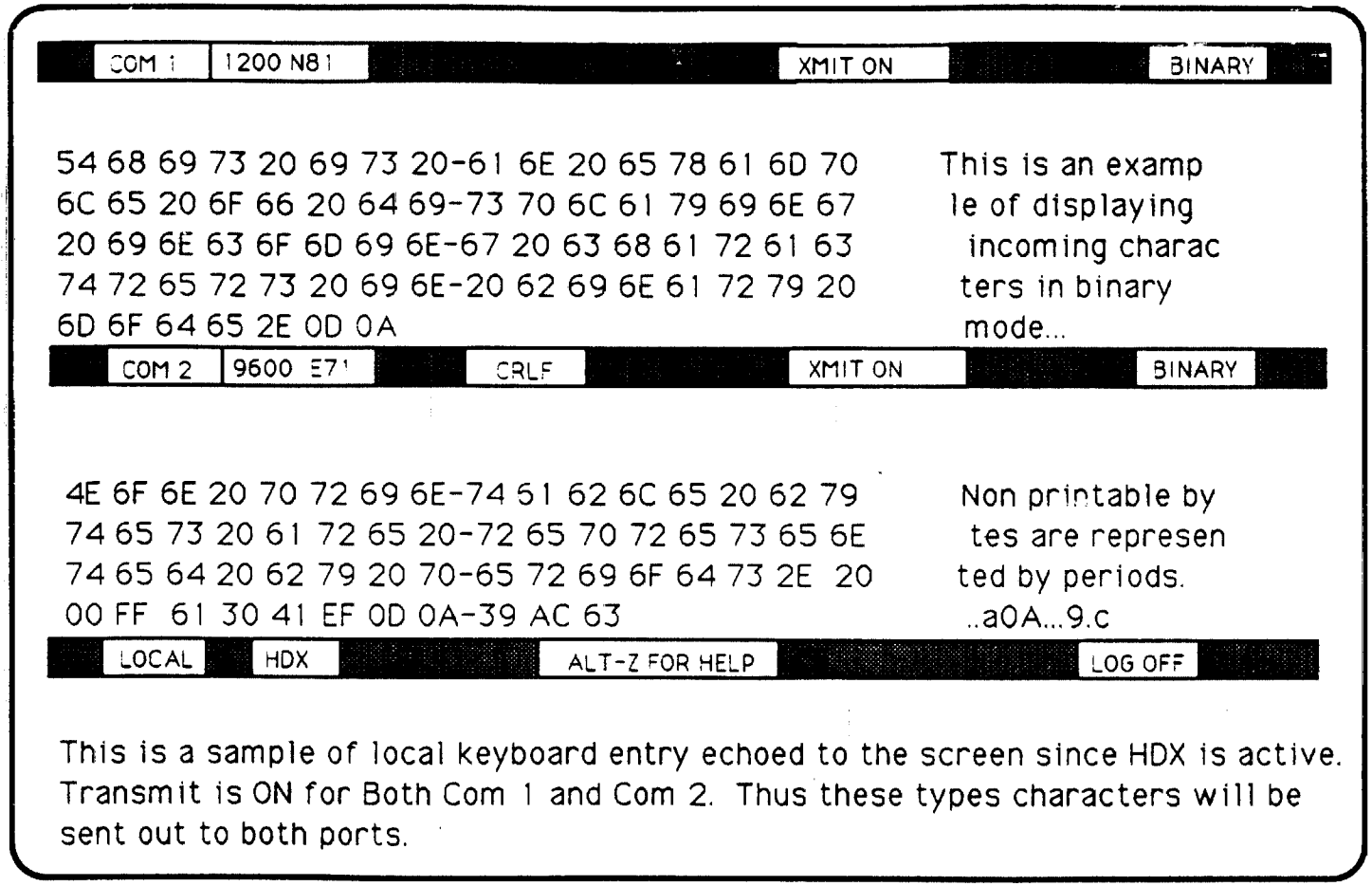

Figure 5. Example of Operation with Received Bytes Displayed in Raw Binary Form.

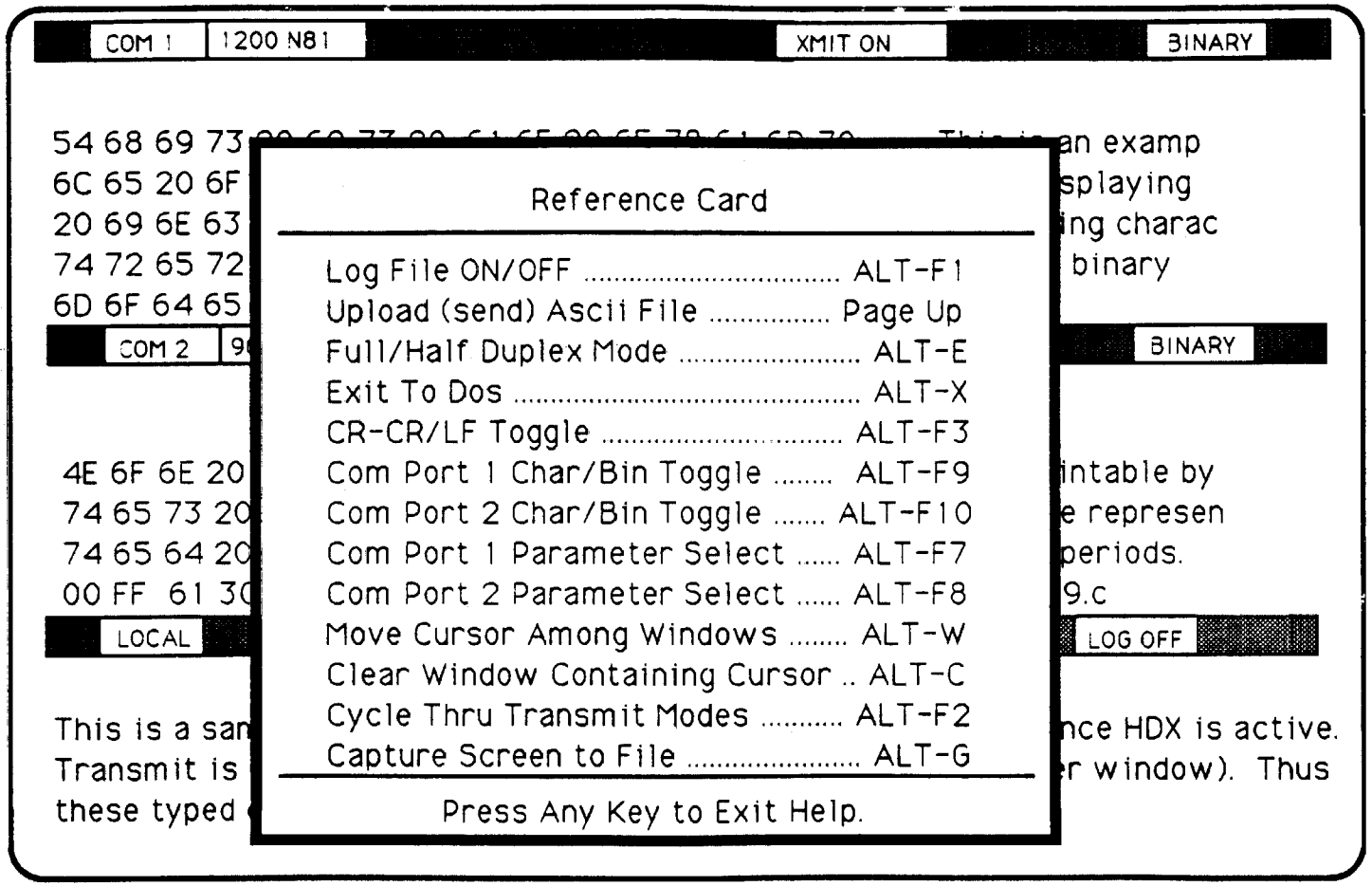

Figure 6. Display of Help Screen.

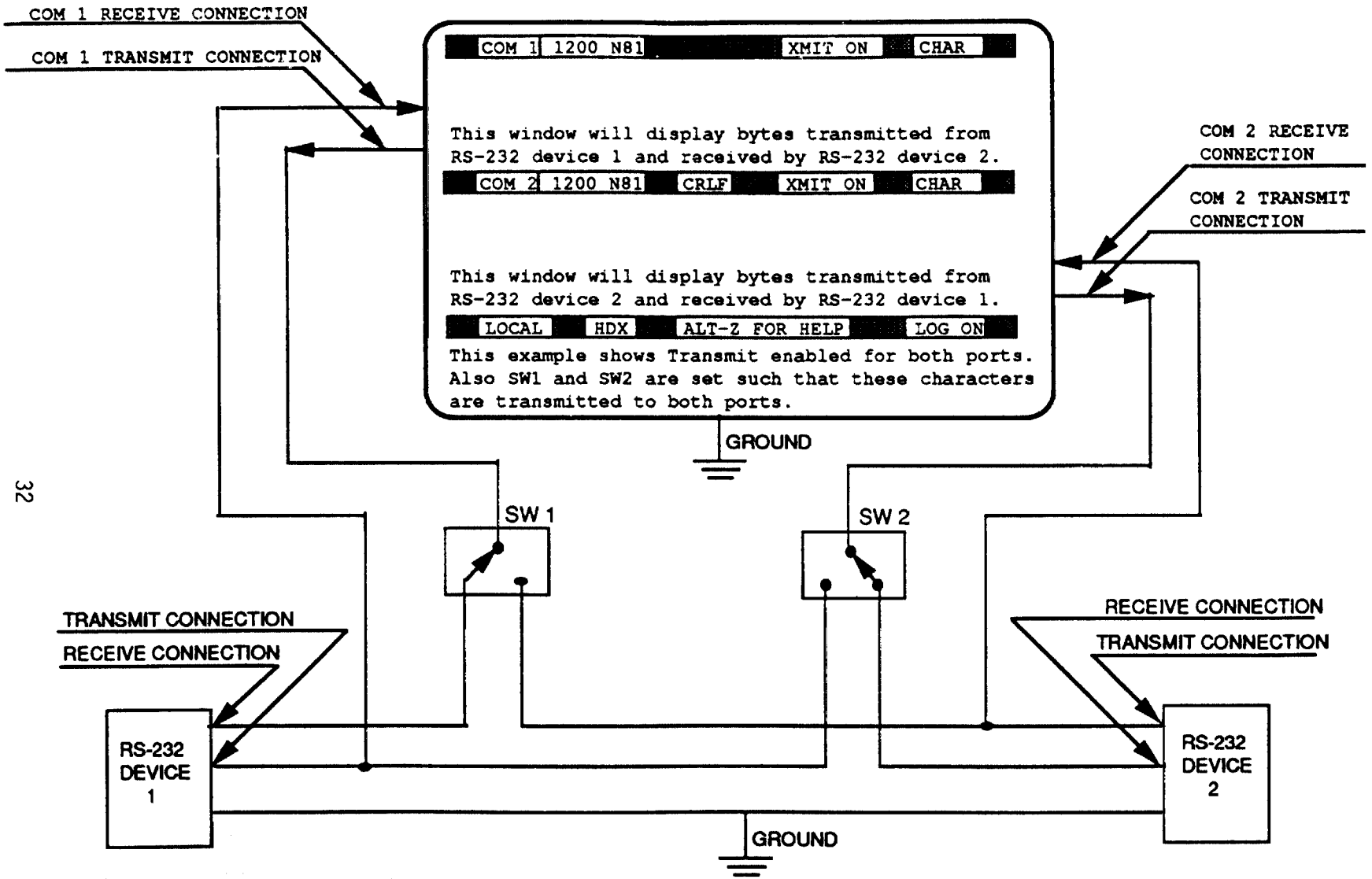

Figure 7. Typical Setup for Use of Transmit Mode.

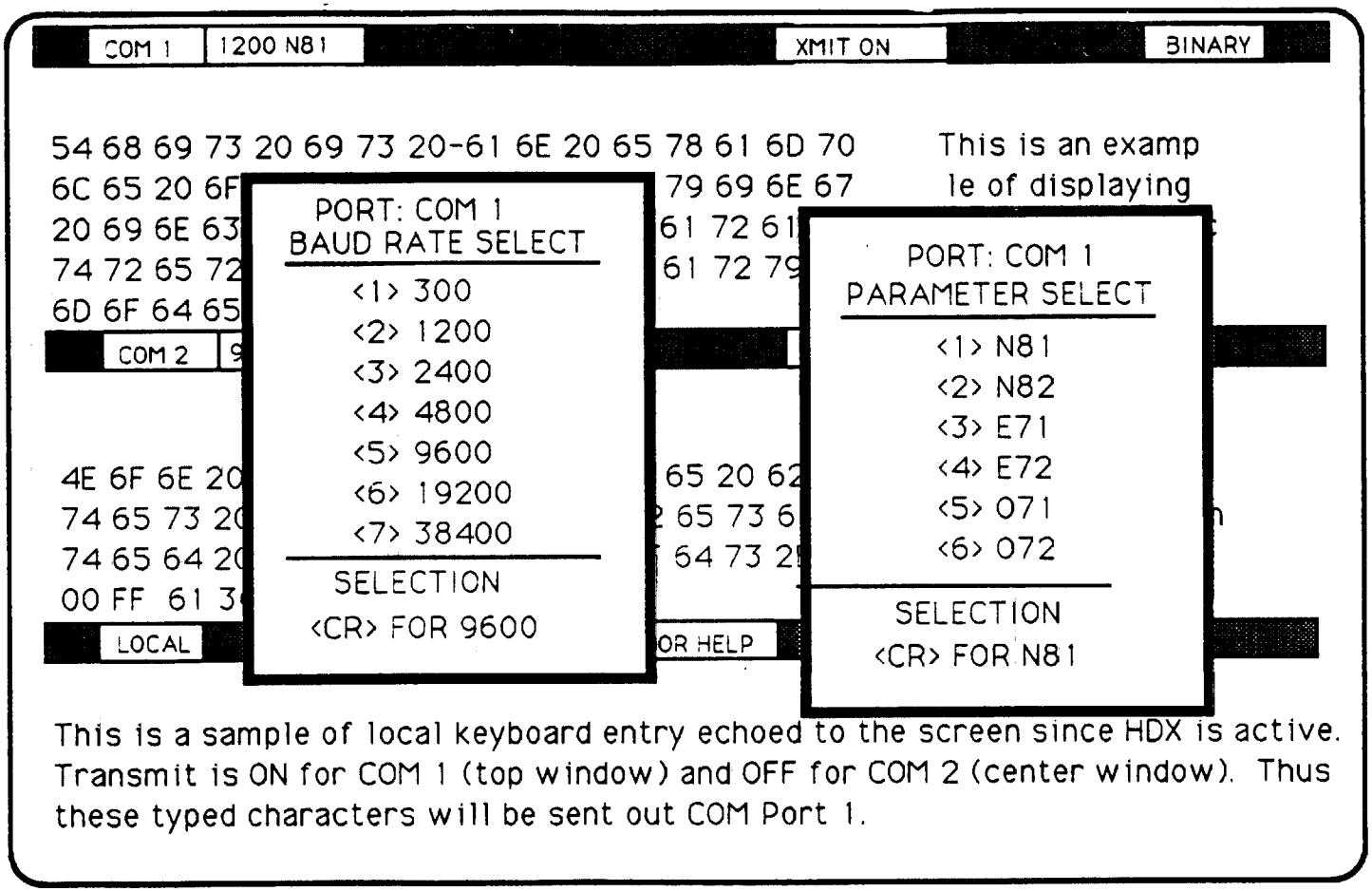

Figure 8. Display of On Line Communication Parameter Select Menus.

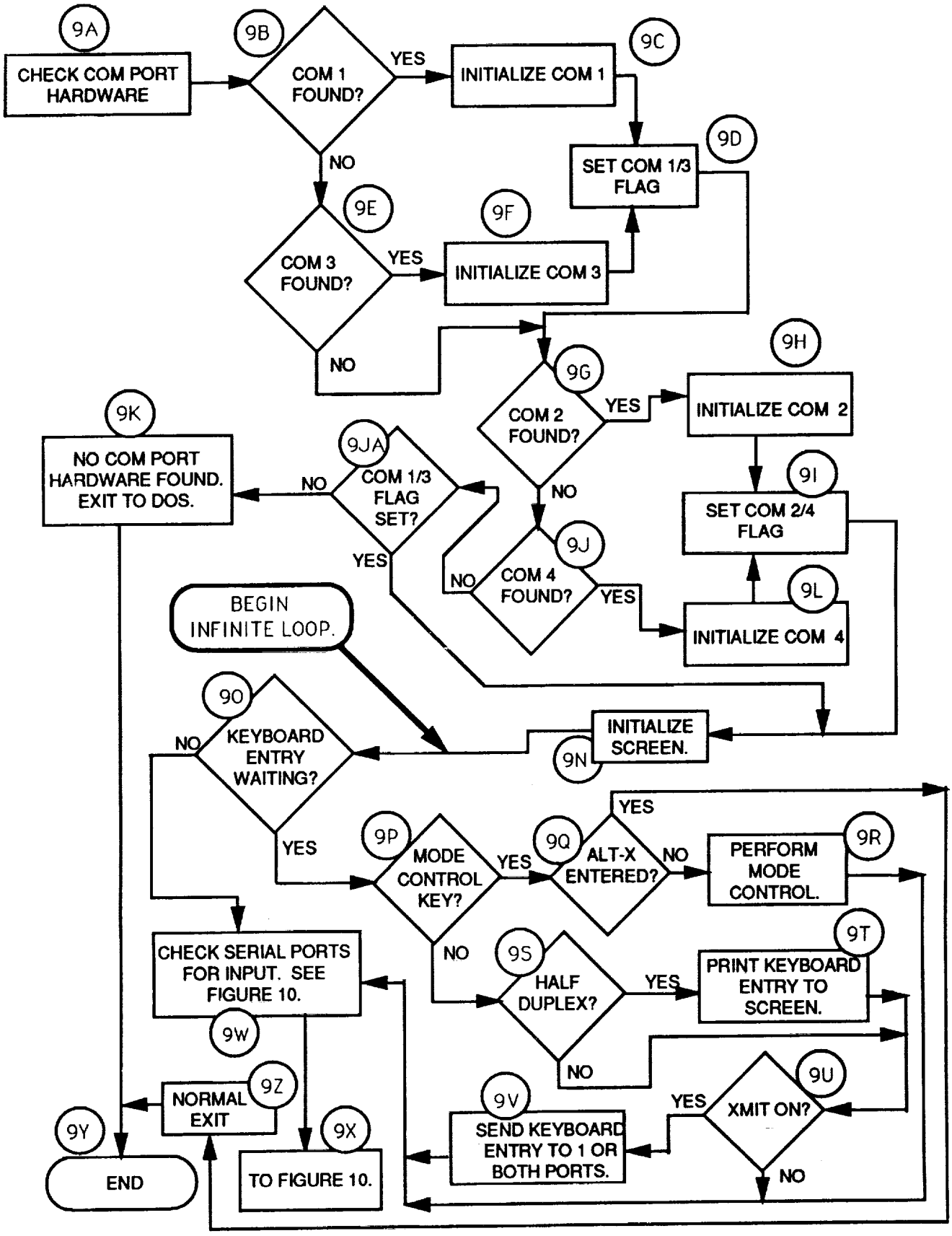

Figure 9. Flow Chart of Program Operation, Setup And Keyboard Functions.

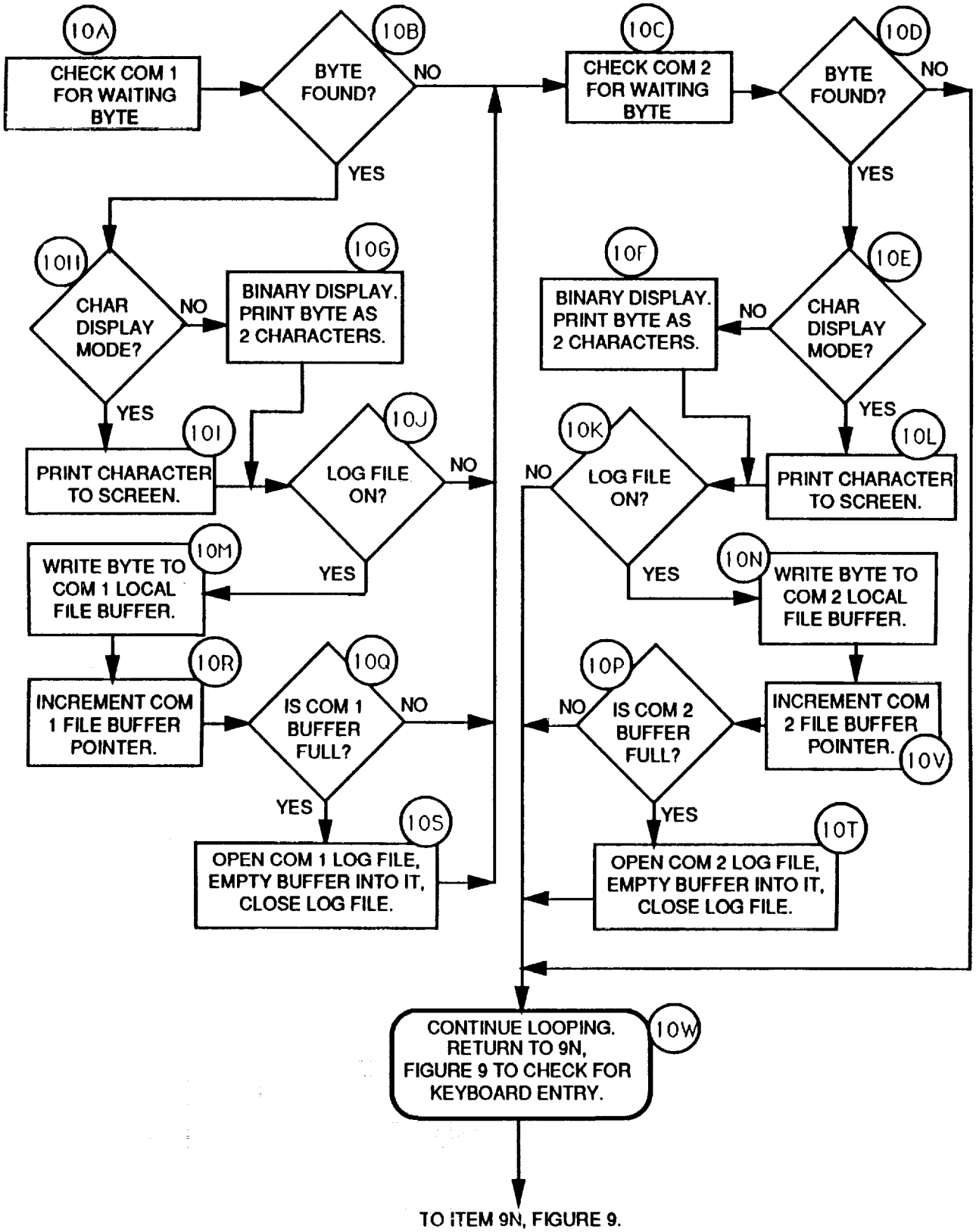

Figure 10. Flow Chart of Program Operation, Serial Port Input Handling.

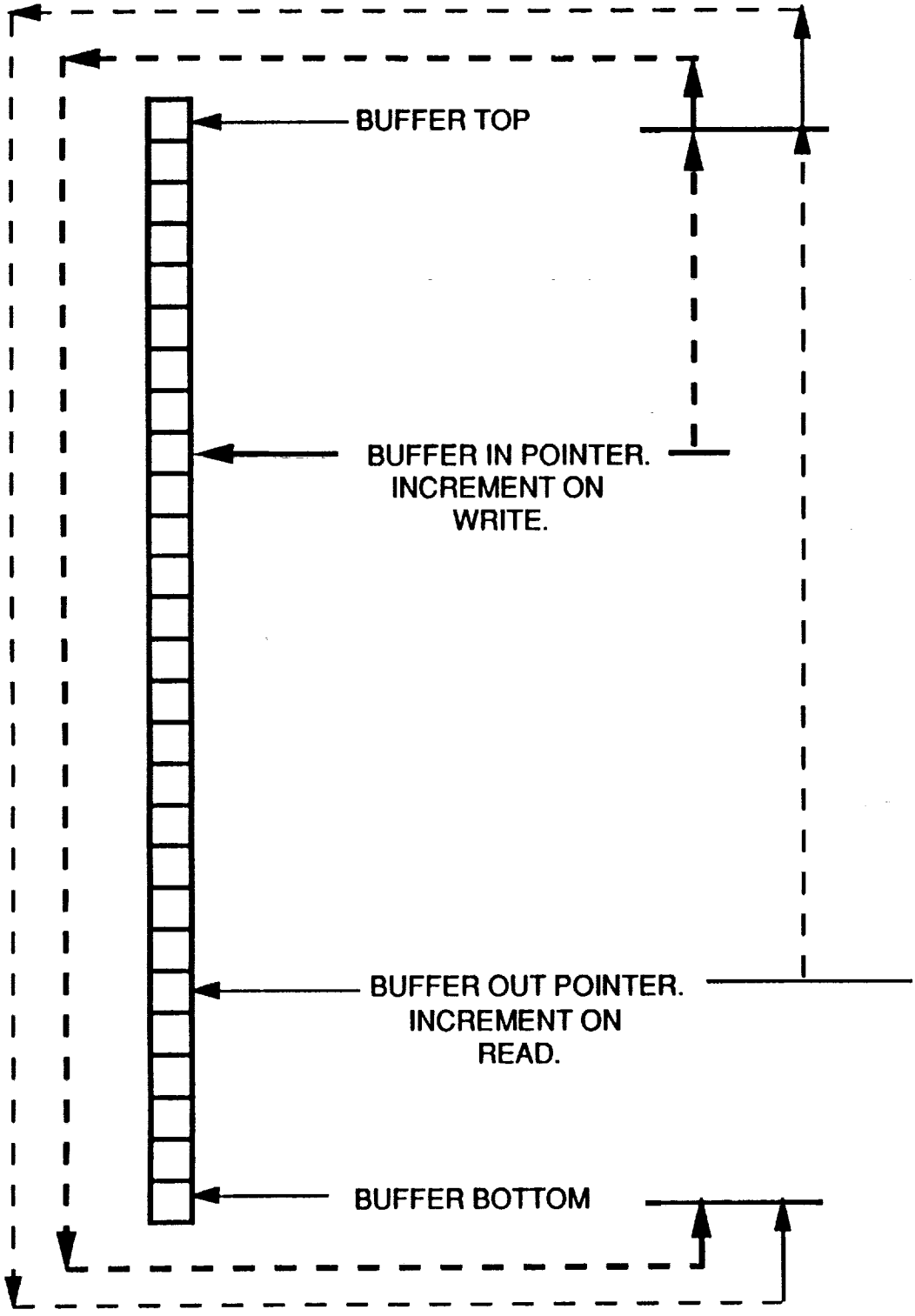

Figure 11. Circular FIFO Buffer Technique Used to Collect Input Bytes From Serial Ports.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December 1991	Technical Memorandum

4. TITLE AND SUBTITLE
A PC-Based Bus Monitor Program for Use With the Transport Systems Research Vehicle RS-232 Communication Interfaces

5. FUNDING NUMBERS
505-64-13-11

6. AUTHOR(S)
Wesley C. Easley

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
NASA Langley Research Center
Hampton, VA 23665-5225

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
National Aeronautics and Space Administration
Washington, DC 20546-0001

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
NASA TM-104175

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Unclassified-Unlimited

Subject Category 04

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Experiment critical use of RS-232 data busses in the Transport Systems Research Vehicle (TSRV) operated by the Advanced Transport Operating Systems Program Office at the NASA Langley Research Center has recently increased. Each application utilizes a number of nonidentical computer and peripheral configurations and requires task specific software development.

To aid these development tasks, an IBM PC-based RS-232 bus monitoring system has been produced. It can simultaneously monitor two communication ports of a PC or clone including the nonstandard bus expansion of the TSRV Grid laptop computers. Display occurs in a separate window for each port's input with binary display being selectable. A number of other features including binary log files, screen capture to files, and a full range of communication parameters are provided.

14. SUBJECT TERMS
Flight Operational Improvements
Experimental Flight Displays

15. NUMBER OF PAGES
43

16. PRICE CODE
A03

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED		