IN-04
61946
p-34

NASA Technical Memorandum 104176

EXPANDED SERIAL COMMUNICATION CAPABILITY FOR THE TRANSPORT SYSTEMS RESEARCH VEHICLE LAPTOP COMPUTERS

Wesley C. Easley

December 1991

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23665-5225

TABLE OF CONTENTS

SUMMARY

This document describes the results of an effort to extend the basic RS-232 communication capability of the Transport Systems Research Vehicle (TSRV) Grid computers. The effort was needed because these units (as delivered) contained only one externally accessible RS-232 serial port, IBM PC normal communication port 2 (COM 2). Significant limitation results since, in most cases, this port is needed to configure the Grid as a terminal for one of the TSRV VAX computers.

An additional serial communication link was adapted using the Grid internal bus expansion capability. Bus expansion hardware was purchased from the Grid Corporation and compatible low-level, communication handling software was written. The resulting software performs port setup and implements interrupt-driven, two-way data communication with selectable software flow control (XON/XOFF). Due to hardware design, the expansion port does not conform to any normal PC serial ports, COM 1 through COM 4, but is equally usable with user-written application software. Turbo C and Turbo Assembler were used for the software development effort. The resulting low-level, communication-handling software takes the form of several functions written to be called from Turbo C application programs developed using the small memory model. Source code has been compiled, assembled, and placed in a library module for use just like any other library functions in Turbo C development, either the command line or integrated development environment. Details of setup and use of the expansion port for interrupt-driven communication with software flow control become transparent to the application programmer who is only required to make high-level calls to a few functions.

Descriptions of the algorithms used to develop the library functions and examples of their application are contained in this document.

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. A recent upgrade of the TSRV experimental systems included installation of a number of Grid 1500 series laptop computers for use primarily as small, lightweight terminals for the Microvax minicomputers which perform flight control, navigation, and display computations. However, each Grid laptop is a 80386-based, MS-DOS compatible IBM PC clone with computational capability permitting extended applications in the flight system.

RS-232 serial data communication was important for the Data Link and Global Positioning Satellite (GPS) research conducted with the TSRV during 1990. In both of these highly successful flight research programs, RS-232 data busses served as vital, real-time data links both on the ground and aboard the TSRV. In addition, RS-232 will serve as vital data busses for at least two TSRV windshear experiments, the Infrared Sensor and Terminal Doppler Weather Radar (TDWR).

Due to their small size, low power requirements, powerful processor, and versatility resulting from IBM and MS-DOS compatibility, the Grid laptop computers are very effective tools for support of these types of applications. User-developed software is, in almost all cases, the major task required for each specific application and a number of quality MS-DOS compatible compilers and assemblers are available to aid in these development tasks.

For improved support of applications such as those mentioned above, additional serial communication capability is needed for the TSRV flight Grid computers. The task described herein has resulted in addition of a serial port to each Grid by adaptation of internal bus expansion features. Turbo C and Turbo Assembler were used for generation of low-level, communication-handling software for use with expansion hardware purchased from Grid. All resulting low-level, communication-handling software has been placed in a Turbo C library file which allows application program interfacing by means of calls to the library functions. Details of hardware programming, interrupt vector redirection, two-way data passage through the port, and software flow control become transparent to the programmer. The additional serial channel does not conform to nor disable any normal PC serial port, but it does require user-written applications--a task almost always necessary for any TSRV research effort.

2.0 LIST OF ABBREVIATIONS

ATOPSPO	Advanced Transport Operating Systems Program Office
AX	General Purpose 8086 Family 16-Bit CPU Register
AL	Lower Eight Bits of AX
BIT	Binary Digit, One or Zero
BYTE	Eight Bit Data Unit
C	Name of a Computer Programming Language
COM	Communications Port of PC
CPU	Central Processor Unit
CR	Command Register
DOS	Disk Operating System
DX	General Purpose 8086 Family 16-Bit CPU Register
FIFO	First-In, First-Out
GPS	Global Positioning Satellite
Hex	Hexadecimal Number
IBM	International Business Machines
IEEE-488	Institute of Electrical and Electronic Engineers Standard Number 488
INT_NO	Interrupt Number
I/O	Input/Output
IRQ	Interrupt Request
ISR	Interrupt Service Routine
LSB	Least Significant Bit
MPSC	Multiprotocol Serial Controller
MSB	Most Significant Bit
MS-DOS	Microsoft Disk Operating System
NASA	National Aeronautics and Space Administration

PC	Personal Computer
PIC	Programmable Interrupt Controller
PIT	Programmable Interval Timer
RS-232	Serial Communication Standard
RS-422	Serial Communication Standard
SR	Status Register
TDWR	Terminal Doppler Weather Radar
TSRV	Transport Systems Research Vehicle
VAX	Brand Name of a Minicomputer Line
XON	Clear-to-Send Software Handshake Signal for Serial Communication Link
XOFF	Negative of XON. Clear-to-Send not Granted

3.0 GENERAL SERIAL EXPANSION TASK DESCRIPTION

3.1 TSRV Grid Hardware Configuation

Grid 1500 series laptop computers contain a multipurpose internal bus expansion slot which generally functions somewhat like expansion card slots on the main board of a normal IBM PC or clone. Hardware modules are available from Grid to use this expansion feature for various interfaces such as RS-232, RS-422, and IEEE-488. For this RS-232 expansion application a serial expansion module, Grid item number 34010 which uses a 8274 multipurpose serial chip, was required.

A standard hardware configuration consisting of a Grid 1500 laptop computer with the 34010 serial module connected to the bus expansion port was used. The goal of this effort was a communication software utility package containing embedded code which allows an application written in C to call a few high-level functions and become free of all details involved in the following operations:

1. Initial setup of the port for RS-232 communication including enabling interrupts and setting data transfer parameters--baud rate, parity, data word length, and stop bits.

2. Configuration of the Grid system to acknowledge the various interrupts generated by the serial expansion module. Also, performance of system interrupt vector redirection and generation of new Interrupt Service Routines (ISR's).

3. Creation of software First In, First-Out (FIFO) buffers in memory to collect incoming and outgoing data bytes whose activity caused the serial interrupts.

4. Generation of low-level Input/Output (I/O) code to read from and write to the FIFO buffers to make the transferred data bytes available to applications programs.

5. Monitoring of FIFO buffer count and managing transmission of XON/XOFF flow control bytes as required to prevent data loss due to buffer overflow.

6. Upon termination resetting interrupt vectors and port interrupt enable status to that found at startup.

3.2 Software Development Tools

Turbo C and Turbo Assembler were used to configure software modules for accomplishing these objectives. After compiling and assembling the source files of each module, the resulting object files have been incorporated into a library file for use with the Turbo C compiler and linker, either in the integrated development environment or with the command-line compiler. Use of these Turbo C environment configurations is described in manuals supplied with the compiler as well as references 1 and 7. The library file resulting from this effort can be added to the DOS directory containing other Turbo C libraries or placed in the directory with the user's application source files. When this is done, none of the original source code used to produce the library file is needed for application development.

4.0 GRID SERIAL EXPANSION HARDWARE CONFIGURATION

The Grid 34010 Serial Expansion Module uses a 8274 Multiprotocol Serial Controller (MPSC) which handles most data transfer functions. It also contains a 8254 Programmable Interval Timer (PIT) which provides counter/timer operations and is used to control baud rate. Upon installation the 8274 becomes hardware connected to a 8259 Programmable Interrupt Controller (PIC) in the main Grid chassis, which accepts interrupt requests from several peripheral devices and issues interrupts to the Central Processor Unit (CPU) based on preprogrammed priority. Technical descriptions of these three devices are contained in reference 2. Digital information is transferred to and from registers in these devices via Input/Output (I/O) ports established by Grid hardware design. These applicable I/O port addresses are listed in table 1.

5.0 INTITIAL SETUP OF THE GRID EXPANSION PORT

5.1 Register Addressing for the 8274

Each register contained in the hardware devices mentioned in section 4.0 is one byte wide. The instruction set for the 8086 family of microprocessors (of which the 80386 is a member) contains "IN" and "OUT" instructions to transfer a byte to or from an I/O port to which a peripheral device may be connected. For example, to send a byte with value A6 Hex to the port with I/O address 778 Hex the following assembly instructions can be used:

```
MOV  DX,778h    ;Port address in CPU DX register
MOV  AL,A6h     ;Byte to send in CPU AL register
OUT  DX,AL      ;Perform the transmission
```

The "OUT" instruction sends the value in CPU register AL out the port address in CPU register DX. Details and examples of 8086 assembly language can be found in references 3 and 4.

Most C compilers written for the IBM PC and clones, including Turbo C, contain higher level functions that perform the same task. In Turbo C the operation performed by the above three assembly instructions could be accomplished with a call to the following function:

```
outportb(0x778,0xA6);
```

Additional Information on this and other functions supplied with Turbo C is contained in references 1 and 7.

Most of the setup and operational activity involved in this serial communication task involves the 8274. It is a dual-channel chip with eight write-only and three read-only registers per channel; but, not all of these registers are used for every application. Figure 1 is a functional diagram of the primary registers necessary for this task. In order to access any 8274 register for read or write, the register must be selected with an output instruction to write register zero with the number of the selected register being contained in the three least significant bits of the output byte. (See CR 0 in figure 1.) An I/O instruction immediately following will read from or write to the 8274 register selected. For example, to send the byte B5 Hex to write register 1, the following C code is needed:

```
outportb(0x77E,1);      Select register 1
outportb(0x77E,0xB5);   Send B5 Hex to write register 1
```

The following example code will read from read register 1:

```
outportb(0x77E,1);          Select register 1.
char_in = inportb(0x77E);   Read value in read register 1
                               into variable char_in.
```

Port address 77E Hex, as seen in table 1, is used to access the channel A command (write) and status (read) registers of the 8274 in the Grid Serial Expansion Module.

5.2 Main Setup Function

For this task all necessary initial setup I/O instructions have been placed the library functions "grid_setup()" and several additional functions that it, in turn, calls. Appendix A contains reference documentation for all these library functions. Complete Grid Serial Expansion setup can be accomplished from an application program written in Turbo C with a single call to the function "grid_setup()." The call must pass three integer parameters, a baud rate selection index between 0 and 6, a communication parameter selection between 0 and 5, and a 1 or 0 to turn XON/XOFF flow control ON or OFF, respectively. Table 2 provides the application programmer with the index values required for the allowed baud rates and other communication parameters. Range validity checks for all passed arguments are made by the

"grid_setup()" function, and default values are set if invalid parameters are detected.

Interrupt vector redirection is done by "grid_setup()" using calls to vendor-supplied Turbo C functions which are discussed in more detail in a later section.

5.3 Interrupt Request Enabling

Enabling of interrupt requests from the 8274 is accomplished by the function "int_install()" which writes an eight-bit mask to a control register in the 8259 PIC. Table 1 shows the Grid port used to access the PIC register and reference 2 provides detailed mask bit definitions. Each bit in this register controls an interrupt request from a peripheral device with a zero in the bit enabling its corresponding interrupt. Before setting the desired interrupt configuration, the entry value of this PIC register is fetched and stored for reset upon program exit. Grid technical documentation describes several hardware interrupt request configurations for the Grid Serial Expansion Module. For this effort, interrupt request 9 (IRQ 9) was used and is controlled by bit 1 in the above-mentioned PIC register. Use of IRQ 9 prevents interference with other lower value interrupt requests including normal PC serial ports, COM 1 through COM 4, and assures that the expansion results in an additional communication port rather than replacement of an existing one.

5.4 Word Length and General Register Setup

Setup operations specific to the 8274 are accomplished by the library function "G34_init()" which is also called by "grid_setup()." (See Appendix A.) Two bytes are passed to "G34_init()" which it writes to proper 8274 registers for setting both transmit and receive word length communication parameters to either seven or eight bits. Specific binary values for these word length bytes are obtained from reference 2 based on the second index integer passed to "grid_setup()" by the application program. In addition, binary byte values written to various other 8274 registers by the "G34_init()" function for complete setup vary with desired configuration and are covered in detail in reference 2.

5.5 Baud Rate Setup

Another function "Gset_baud_rate()" is called for setting the baud rate to the value selected by the first integer passed by the application program to the "grid_setup()" function. This integer is a baud index value which is used to extract a 16-bit binary value from a table whose elements are configured based on Grid technical documentation. The selected binary value is passed to the "Gset_baud_rate()" function which in turn writes it into baud rate control registers in the 8254 PIT contained in the Grid Expansion Module.

5.6 Parity and Stop Bit Setup

The final setup procedure involves setting parity and stop bits by calling the function "Gset_par_stop()" to which one binary byte is passed. This byte is selected from technical information in reference 2 based on the second integer index passed by the application program to "grid_setup()." The function "Gset_par_stop()" then combines this byte with two clock rate bits, and writes the result to 8274 write register 4 (CR 4 in figure 1).

The three functions "G34_init()", "Gset_baud_rate()", and "Gset_par_stop()" are not visible to the application and are reachable only through the prime interface function "grid_setup()."

6.0 SERIAL COMMUNICATION INTERRUPT HANDLING

6.1 Interrupt Versus Polling Methods

PC serial communication accomplished by simply polling the data register of the communication port is inadequate except possibly for very low transfer rates. Incoming bytes will almost certainly be overwritten while the program is processing a previously-received byte. Effective operation requires use of interrupts generated by the serial hardware device, in this case the 8274. As part of the initial setup described above, the 8274 is configured to generate an interrupt request both when a received byte is ready in its data register and when its hardware transmit buffer is empty. These are asynchronous hardware interrupts which occur totally independently of any software task in progress. The 8259 PIC arranges them according to priority and passes them to the CPU. Thus, software must be configured to permit immediate service of these interrupts to prevent data loss.

6.2 Interrupt Vectors

Proper handling of interrupts requires the programmer to write a specific routine which is activated upon interrupt occurrence. This is called an Interrupt Service Routine (ISR). Direction of processing to an ISR is accomplished by interrupt vectors which, for MS-DOS computers, are addresses contained in a table comprising the lowest 1024 memory locations. Each interrupt vector consists of a four-byte block with each block having an associated interrupt number starting at zero and continuing through 255. Interrupt numbers can be configured by hardware, as in the case of the 8259, or by DOS software. Since each vector consists of four bytes, the location of an interrupt vector in the low memory table is four times the interrupt number. Upon boot-up MS-DOS loads default interrupt vectors and ISR's, some of which merely return with no action taken.

Application programs can intercept an interrupt by changing the contents of its vector, thus directing processing to a different memory location when the interrupt occurs. This new location must contain the programmer's specific ISR which will then be executed. Internal DOS software services accessible via assembly instructions are provided for this vector redirection. However, most C compilers for MS-DOS systems now contain a higher-level function to accomplish this. In Turbo C the interrupt vector redirection functions are "getvect()" and "setvect()" and are used as follows:

```
entry_handler = getvect(INT_NO);
```
Fetch the address of the handler for interrupt INT_NO and place it into the variable "entry_handler" so it can be reset upon program exit.

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

The variable INT_NO represents the number associated with the hardware or software interrupt being redirected. For the configuration of the Grid Expansion Module used in this effort, the interrupt number obtained from Grid documentation is 71 Hex.

A number of limitations exist regarding instructions and C function calls that can be used inside an ISR. An important goal for this effort is embedding these details in the resulting library functions and rendering them transparent to any application.

More extensive technical treatment of MS-DOS interrupt handling is contained in references 3, 4, 5, and 6.

7.0 SOFTWARE BUFFERING TECHNIQUE

7.1 General Comments

First-In, First-Out (FIFO) circular buffers created in memory by software are commonly used for interrupt-driven communication tasks. Reference 6 contains a description of this technique which is used for the subject effort. The specific algorithm is illustrated in figure 2 and described below.

7.2 Buffer Pointers

Two software buffers, each two kilobytes in size and identical in operation, are created when the function "grid_setup()" is called. One is for received data and the other is for transmitted data. Each buffer functions in conjunction with two independent operations, one to deposit data and another to retrieve it. As shown in figure 2, separate input and output pointer variables are

associated with each buffer. Both pointers are set to zero at program startup and thus initially point to the buffer bottom. An operation depositing a data byte will write it into the cell pointed to by the input pointer. Then it must increment the pointer variable 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.

7.3 Buffer Byte Count

Another variable associated with each buffer is the byte count indicating the number of bytes written but not yet retrieved. A write operation adds a byte and must increment this counter while a read operation removes a byte and must decrement it. The buffer byte count, which is actually the difference between the locations pointed to by the input and output pointers, thus maintains a running total of the data bytes waiting to be fetched. If this count is zero then both pointers are selecting the same location, and the buffer is empty. If it reaches the maximum buffer size then the buffer is full, and no cells are available for writing. Since a write must occur before a read, every read operation must determine whether this count is zero before actual data capture to prevent reading past the last location which was written into.

The byte count, as will be illustrated in later sections, is used when flow control is active to trigger transmission of XON/XOFF flow control characters to prevent buffer overflow and data loss. An upper threshold of 80 percent of buffer capacity is used to trigger XOFF transmission and request the remote system to stop sending. Then, when the count is reduced to 20 percent of buffer capacity, an XON transmission is triggered signaling the remote terminal that sending is once again permitted.

8.0 IDENTIFICATION OF 8274 INTERRUPTS

When a hardware interrupt is generated by the 8274, the CPU saves its status and immediately jumps to the corresponding ISR, which was installed by the "grid_setup()" function using the "getvect()" and "setvect()" functions discussed above. Each ISR used herein is a short C function which calls an assembly function to handle details of servicing the interrupt. The first task of the ISR is identification of the interrupt type which is done by checking the three least significant bits of 8274 read register 2 in channel B (SR 2 in figure 1). The following assembly instructions will isolate these bits in the CPU AL register:

```
MOV   DX,77Fh    ;Port Address, Read Register 2, Channel B
MOV   AL,2       ;8274 Register to select placed in AL of CPU
OUT   DX,AL      ;Select Read Register 2, 8274 Channel B
IN    AL,DX      ;Fetch Register Contents into AL Register
AND   AL,07h     ;Mask to Isolate Lowest Three Bits
```

9.0 HARDWARE TRANSMIT BUFFER EMPTY INTERRUPT

9.1 Verification of Interupt Type

If the interrupt identification procedure of Section 8.0 results in a value of 04 Hex in the AL register of the CPU (see figure 1 and reference 2), a hardware transmit buffer empty interrupt has occurred signaling that the 8274 is available for transmission of a data byte. The resulting actions taken by the ISR are shown in flowchart form in figure 3 and are described in this section. This ISR is the output operation associated with the software transmit buffer which functions as illustrated in figure 2. Bytes fetched by the ISR from the buffer are written into the 8274 hardware transmit register for sending out the port.

9.2 Empty Buffer With XOFF Active

However, examination of figure 3 shows several checks made by the ISR prior to data fetch. First, if the software transmit buffer is empty (buffer byte count is zero), nothing is awaiting transmission and the ISR terminates. If the buffer is not empty and XON/XOFF flow control is active, a check for XOFF received is made. This check uses a global flow control status flag which is toggled between one and zero (on and off) by receipt of XON (11 Hex) and XOFF (13 Hex), respectively. If this flag indicates that XOFF has been received then transmission is forbidden and the ISR terminates. Notice that both these ISR terminations result in disabling the hardware transmit buffer interrupts that activated the ISR. Reasons for this and methods to re-enable the interrupts are discussed in section 10.0 which deals with received character interrupts.

If these flow control tests all fail then, as seen from figure 3, data fetch, pointer increment, and count decrement operations occur. Finally, the buffer output pointer is wrapped to zero if it has reached the buffer top.

9.3 Writing to the Software Transmit Buffer

Data bytes generated by the application program for serial transmission are deposited into the software transmit buffer by the function "grid_out_byte()" which is described in Appendix A. It is a library function written as part of this effort which must be called by the programmer. A flowchart of this function, which is written in assembly but callable from C, is shown in figure 4. It is not an ISR, requires an unsigned character argument, and

returns an unsigned integer with the MSB reset to zero, if the write attempt was successful, or with the MSB set to one, otherwise. Its prototype and use are:

```
Prototype: unsigned int grid_out_byte(unsigned char);
Use:       grid_out_byte(out_char);
Returns:   unsigned integer
```

The variable "out_char" is the byte which the application wishes to transmit. Checking the MSB of the returned integer can be used in the calling function to determine whether the output attempt was successful.

As seen from figure 4, the function "grid_out_byte()" first checks for a full software output buffer and, if this is the case, loops for a short time repeatedly checking for an available slot before returning unsuccessful and discarding the byte. If the buffer is not full a check is made to determine if it is empty, in which case, no collected bytes are awaiting transmission. An empty buffer results in a check of the global flow control status flag described above to see if an XOFF control character has been received; and, if not, the byte in question is written directly to the transmit register of the 8274, I/O address 77C Hex, and immediately transmitted. If the output buffer contains data but is not full; or, if an XOFF has been received, then the byte in question is added to the buffer for later transmission. It is written into the location selected by the buffer input pointer (see figure 2) with this pointer then being incremented to select the next location available for writing. Also, the buffer byte count is incremented to indicate that a byte has been added. The last procedure for the ISR is resetting the pointer to zero, if it has reached the buffer top.

10.0 RECEIVED CHARACTER AVAILABLE INTERRUPT

10.1 Verification of Interrupt Type

If the interrupt identification procedure of Section 8.0 results in a value of 06 Hex in the AL register of the CPU (see figure 1 and reference 2), a received character is available in the 8274 data register. The resulting actions taken by the ISR are shown in flowchart form in figure 4 and are described in this section. This ISR is the source of input data for the software-receive buffer which functions as illustrated in figure 2. Each byte it deposits is fetched from the 8274 data register in response to the received character available interrupt. The bytes are written into the software-receive buffer cell selected by the current buffer input pointer.

10.2 XOFF Received

As can be seen from figure 5, a number of checks are made on the received byte before buffer storage. If flow control is active,

the received byte is checked to determine if it is the XOFF control character, 13 Hex, or the XON control character, 11 Hex. The global flow control status flag, which was checked in the transmit operation of the previous section, is actually set in this ISR by receipt of one of the flow control characters. When an XOFF is received the status flag is reset to zero, a task represented by the "SET XOFF RECEIVE FLAG" step in figure 5. Then the interrupt is cleared and the ISR exits. The XOFF control character is not stored in the buffer when flow control is active.

10.3 XON Received

However, receiving XON means that the system might have been forbidden to transmit for a time and data bytes from the application program destined for transmission may have collected in the software output buffer described previously in section 9.0. Such collected data can originate, for example, from the local keyboard or a disk file. While clear-to-send status resulting from receiving an XON allows these buffered bytes to be transmitted, that will not automatically occur without 8274 hardware transmit buffer empty interrupts. These interrupts will have been disabled the last time one of them occurred and found an empty software transmit buffer--a scenario which was discussed in section 9.0 and is illustrated in the flowchart of figure 3. Due to 8274 design (see reference 2) one transmission needs to occur to re-enable these interrupts. As is shown in the upper right of figure 5, this is accomplished by fetching the first waiting byte, if any, from the software transmit buffer and writing it to the 8274 transmit register (I/O address 77C Hex) for immediate transmission. After transmit buffer empty interrupts are re-enabled, they will repeatedly activate the ISR illustrated in figure 3 and send collected bytes until the software transmit buffer is once again empty. Finally, interrupt reset and ISR exit will occur just as in the case of XOFF receipt. The XON control character will not be stored if flow control is active.

10.4 Buffer Storage of Received Byte

Referring again to figure 5, if flow control is not active or if a flow control character was not received, an attempt at buffer storage is made. First, the buffer byte count is checked to determine if the software receive buffer is full, in which case there is no place to write. Thus, the ISR discards the received byte and exits. Existing vacant buffer cells result in writing the received byte into the location selected by the current buffer input pointer. (See figure 2.) Then the buffer's byte count and input pointer are both incremented to respectively reflect the added entry and select the next free cell. If the pointer has reached the buffer top it is wrapped to zero.

10.5 XOFF Transmission Check

The ISR illustrated in figure 5 performs one last flow-control related operation. After writing a received byte into the software receive buffer and, if flow control is active, the subject ISR checks the byte count to determine if the XOFF threshold (80 percent of buffer capacity) has been exceeded. If so, it writes an XOFF control character to the 8274 transmit register for immediate transmission. Then,a global flag is configured to indicate that XOFF has been sent. This global flag is functionally identical to the flow control status flag described in Section 9.0 for the case of XON and XOFF reception. It is turned on (set to one) or off (set to zero) by transmission of XON or XOFF, respectively.

10.6 Data Retrieval From the Software Receive Buffer

Data bytes are fetched from the software receive buffer and made available to the application program by another library function "grid_in_byte()" which must be called by the application. A flowchart of this function, which is written in assembly but callable from C, is shown in figure 6. It is not an ISR, takes no arguments, and returns an unsigned integer. Its prototype and use are:

```
Prototype: unsigned int grid_in_byte(void);
Use:       in_char = grid_in_byte();
Returns:   MSB (bit 15) set if no byte read; otherwise
               fetched byte in lower eight bits of in_char.
```

The variable "in_char" is an unsigned integer (16 bits) with the byte fetched from the software receive buffer filling the lower eight bits. This byte was read from the buffer location selected by the receive buffer's output pointer. (See figure 2.) Then the buffer byte count is decremented to indicate that a byte has been removed, and the output pointer is incremented to select the next buffer location from which reading can occur. Again, as in previously-described cases, the pointer is wrapped to zero if it has reached the buffer top.

10.7 XON Transmission Check

Like several operations discussed in earlier sections, the function "grid_in_byte()" has flow control maintenance duties. Upon entry it checks for active XON/XOFF flow control. If this is the case and if the buffer count is below 20 percent of capacity (see section 7.0), then a check is made to see if the remote terminal's transmission has been halted by a previously sent XOFF. If so, an XON (11 Hex) is written to the 8274 transmit register for immediate transmission and the flow control status flag concerned with XON/XOFF transmission is set to one. Clear to send has thus been signaled to the remote terminal.

11.0 LIBRARY FUNCTION DESCRIPTION AND APPLICATION EXAMPLE

11.1 General Comments

Appendix A is a reference guide describing all the functions contained in the library resulting from this effort. Appendix B is a program written in Turbo C illustrating their application.

11.2 Algorithm of Application Example

Only the algorithm involved in the example of Appendix A is described. No attempt is made to explain the C language programming techniques used. References 1, 6, and 7 are examples of many widely available publications dealing with the subject of C programming. Initial setup is done via a call to "grid_setup()" with parameters passed to select a baud rate of 9600, communication parameters of N81 (no parity, 8 bits per character, and one stop bit), and to select software flow control (XON/XOFF). Interrupt enabling, interrupt vector redirection, and FIFO buffer creation are all transparently accomplished by this function.

The example program enters a loop which continues until the escape key is pressed. Inside the loop repeated checks are made for keyboard inputs and for data bytes received from the Grid serial expansion port. Keyboard inputs are fetched using the function "kb_fetch()," a function supplied in this library which checks the DOS keyboard buffer for any waiting entry and returns it in the lower byte of an unsigned integer. If no key has been pressed, the function returns an unsigned integer with its MSB set to zero. Keyboard inputs for applications using this library do not require use of "kb_fetch()." Other vendor-supplied, console-related C functions such as "kbhit()," "getch()," and "getche()" are satisfactory. Input bytes from the serial expansion module are obtained by calling "grid_in_byte()." As already described, this function reads from the software receive buffer whose locations are filled by received character interrupts from the 8274 in the Grid expansion module. These input bytes are printed to the screen and saved in the file "INBYTES.DAT." Keyboard inputs, in addition to being printed to the screen, are transmitted to the serial port using the library function "grid_out_byte()." Upon command to exit, the program closes the file and calls the library function "grid_reset()" which returns all interrupt vectors and interrupt enable settings to their entry values. These entry values were fetched and saved when "grid_setup()" was called to start the program. Any application should terminate with a call to "grid_reset()."

11.3 Comments Regarding Extended Applications

In the application example described above, all communication parameters are locked in at compile time. More sophisticated uses might include capability to change these on line, perhaps through pop-up menus. This can be accomplished by getting console input

values for parameter index and flow control flags from such menus and making a new call to "grid_setup()." Complete port reinitialization will result. However, a call must always be made to "grid_reset()" before any additional use of "grid_setup()." Otherwise ill-behaved interrupt operations may result.

11.4 Building Turbo C Executable Files

Various methods of building an executable file from source modules exist in Turbo C compiler packages. Library files such as that produced by this effort need visibility to the linker only as they have already been compiled or assembled. References 1 and 7 are examples of the wide choice of available literature concerned with the various versions of Turbo C that Borland has released. In addition, manuals supplied with the Turbo C compiler packages contain large amounts of reference documentation.

12.0 CONCLUDING REMARKS

Enhanced RS-232 serial communication capability for the Grid 1500 series laptop computers used in the TSRV experimental flight system has been accomplished by adaptation of an internal bus expansion port. Attempts were made to accomplish two goals. One, fundamental development of an effective RS-232 port, using the Grid bus expansion, to support two-way, interrupt-driven serial data transfer with software flow control. Two, development of a software interface to the port which permits a programmer to develop communication applications using the Turbo C small memory model without being concerned with any low-level, hardware-specific details. Modifications to the functions to support larger memory models of the C compiler system can be done.

A technical description of the algorithms used in attempting to accomplish the above-stated goals is contained in this document. Most software development was done using Turbo C version 2.0, but testing with Turbo C++ version 1.0 and the newly released Borland C++ indicate no compatibility problems. Minor source-code changes would be required for use with other C compiler systems.

Utilization in the TSRV of RS-232 data links is continuously increasing. Accomplishment of the desired results of this effort, combined with a small number of inexpensive hardware modules, adds at least five of these serial busses to the flight system. The results described herein apply specifically to the Grid expansion port, but similar algorithms have been used to develop corresponding utilities for normal PC serial communication ports 1 through 4.

Appendix A: Description of all library functions

I. Function Name: grid_setup()
Prototype: void grid_setup(int baud_index, int param_index, int x_flag)

Tasks:

1. Check passed parameters for valid range.
2. Fetch and store entry interrupt vector for Grid Expansion Module.
3. Enable 8274 interrupts from Grid Expansion Module.
4. Set new interrupt vector to address of new handler.
5. Call function G34_init() for complete initial setup of 8274 chip in Grid Expansion Module. Two word length parameters selected from the argument param_index are passed to G34_init().
6. Obtain the proper baud rate word from a binary table using the argument baud_index as an index into this table.
7. Call C function Gset_baud_rate() to set desired baud rate. Baud word from step 6 is passed to this function.
8. Use the argument param_index to select binary bytes for parity and stop bits. Call function Gset_par_stop() to set these parameters.
9. Set an internal XON/XOFF flow control flag to the value of argument x_flag. XON/XOFF flow control is ON if this is one, OFF if zero.

II. Function Name: kb_fetch()
Prototype: unsigned int kb_fetch(void)

Tasks:
Retrieves a keyboard entry waiting in the keyboard buffer, if any. Returns the pressed key in lower byte or returns integer MSB set to zero, if no key waiting.

III. Function Name: grid_in_byte()
Prototype: unsigned int grid_in_byte(void)

Tasks:
Fetches a byte from the serial receive FIFO buffer. Returns the byte, if any, in lower eight bits or returns MSB set to one if nothing waiting.

IV. Function Name: grid_out_byte()
Prototype: unsigned int grid_out_byte(unsigned char)

Tasks:
Write a byte into the transmit FIFO buffer. Returns with MSB reset to zero if successful or returns MSB set to one if not.

V. Function Name: grid_reset()
Prototype: void grid_reset(void)

Tasks:
Resets the interrupt vector and interrupt enable to the states found and stored at startup.

Appendix B. Application Example. Use of Turbo C small memory model is required.

```
#include "stdio.h"
#include "grxdef.h"
type def unsigned char BYTE;
FILE *f1;   /* f1 will contain the name of a file */

/* Prototypes of functions called by this module */
void grid_setup(int bdr, int parm, int flow);
unsigned int grid_out_byte(BYTE);
unsigned int grid_in_byte(void);
unsigned int kb_fetch(void);

void main() /* Main function */
{
   int twelfth_of_never = 1, today = 0; /* Loop controls */
   unsigned int gky, gci;

      /* Initialize to 9600 baud, N81, flow control ON */
   grid_setup(4,0,1);   /* Initialize Grid 34010 module */
      /* Open file for storing input bytes */
   if ( (f1 = fopen("inbytes.dat", "w")) == NULL)
   { printf("\nCan't Open Required File for Writing.");
      exit(0);  }

   /* Enter infinite while loop */
   while(twelfth_of_never)     /* Main task - Infinite loop */
   {
      if( (gky = (kb_fetch() & 0x8000)) == 0); /* No Key */
      else     /* Key was pressed so process it */
      {  putchar((BYTE)gky);        /* Write to screen */
         grid_out_byte((BYTE)gky); /* Send to port */
         if((BYTE)gky == 0x1B)       /* Escape Key Hit? */
         twelfth_of_never = today; }

      /* Is serial input byte waiting? */
      if( (gci = grid_in_byte()) & 0x8000);  /* No */
      else                            /* Yes, process it */
         {  putchar((BYTE)gci);     /* Write to screen */
            putc( (BYTE)gci,f1);    /* Write to file */
         }
   }     /* end of while */
fclose(f1);  /* Close File */
   /* Reset interrupt vector and mask */
grid_reset();
printf("\nPress Any Key to Exit ... ");
if(getch() == 0x00) getch();
exit(0);    /* Return to Dos */
}     /* End of main */
```

Port Address (Hex)	Function
778	Device Identification
779	System Interrupt Enable
77C	8274 Channel A Data Register
77D	8274 Channel B Data Register
77E	8274 Channel A Command/Status Register
77F	8274 Channel B Command/Status Register
F7A	8254 Baud Rate Generator Counter Control
F7C	Serial Port Clock Configuration Bit 0
F7D	Serial Port Clock Configuration Bit 1
F7E	Serial Port Clock Configuration Bit 2
021	8259 PIC Mask Register Port

Table 1. Port Address Information for the Grid 34010 Serial Expansion Module

Baud Rate	Required Index
300	0
1200	1
2400	2
4800	3
9600	4
19.2 K	5
38.4 K	6

Communication Parameters	Required Index
N81	0
N82	1
E71	2
E72	3
O71	4
O72	5

Table 2. Definition of index values needed for call to "grid_setup()" function for baud rate and communication parameter selection.

REFERENCES

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

2. Microsystem Components Handbook, Volumes I and II, Intel Corporation, Order No. 230843-001, 1984.

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.

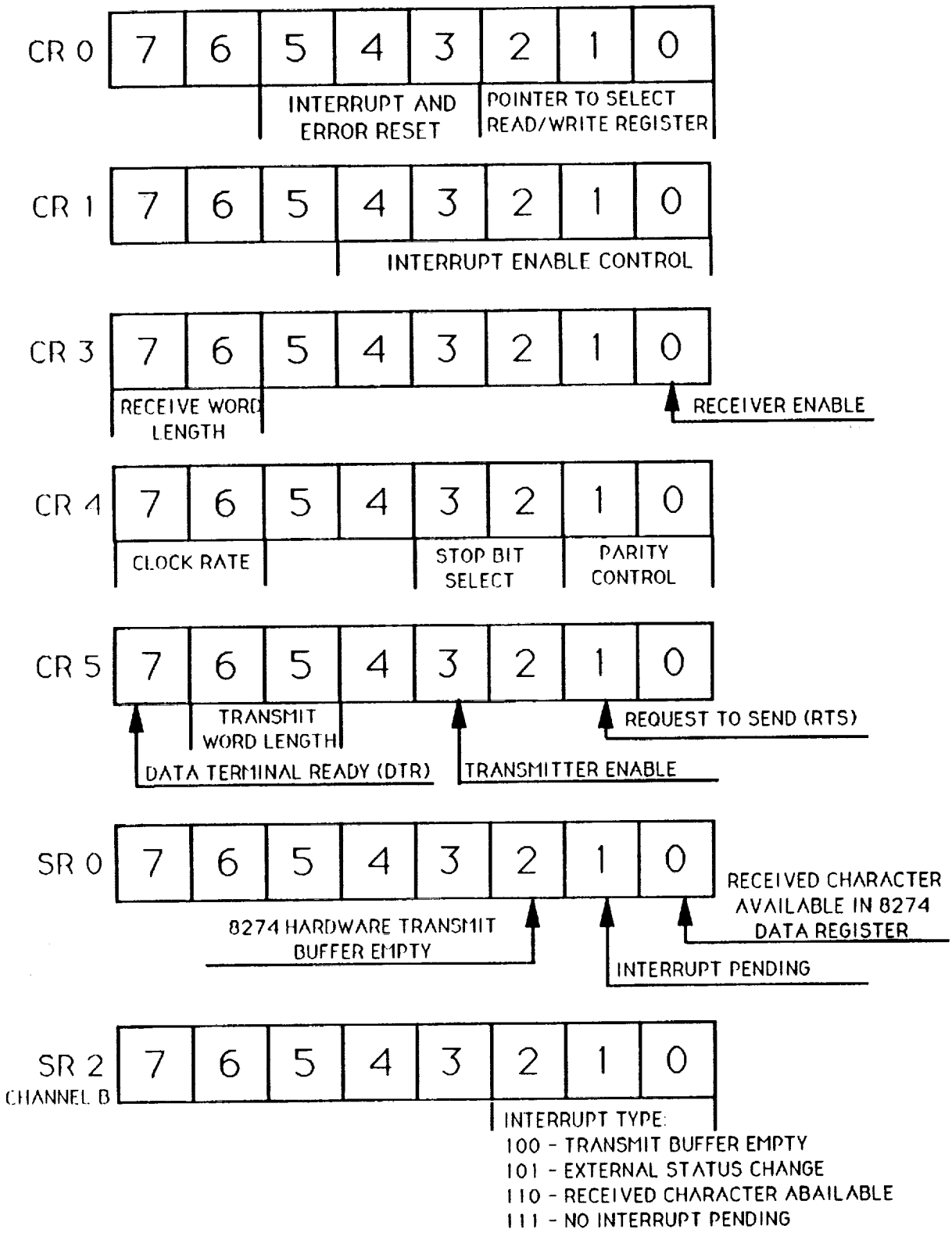

Figure 1. Functions of Prominent 8274 Hardware Registers. CR Represents Command Registers for Writing, SR Status Registers for Reading.

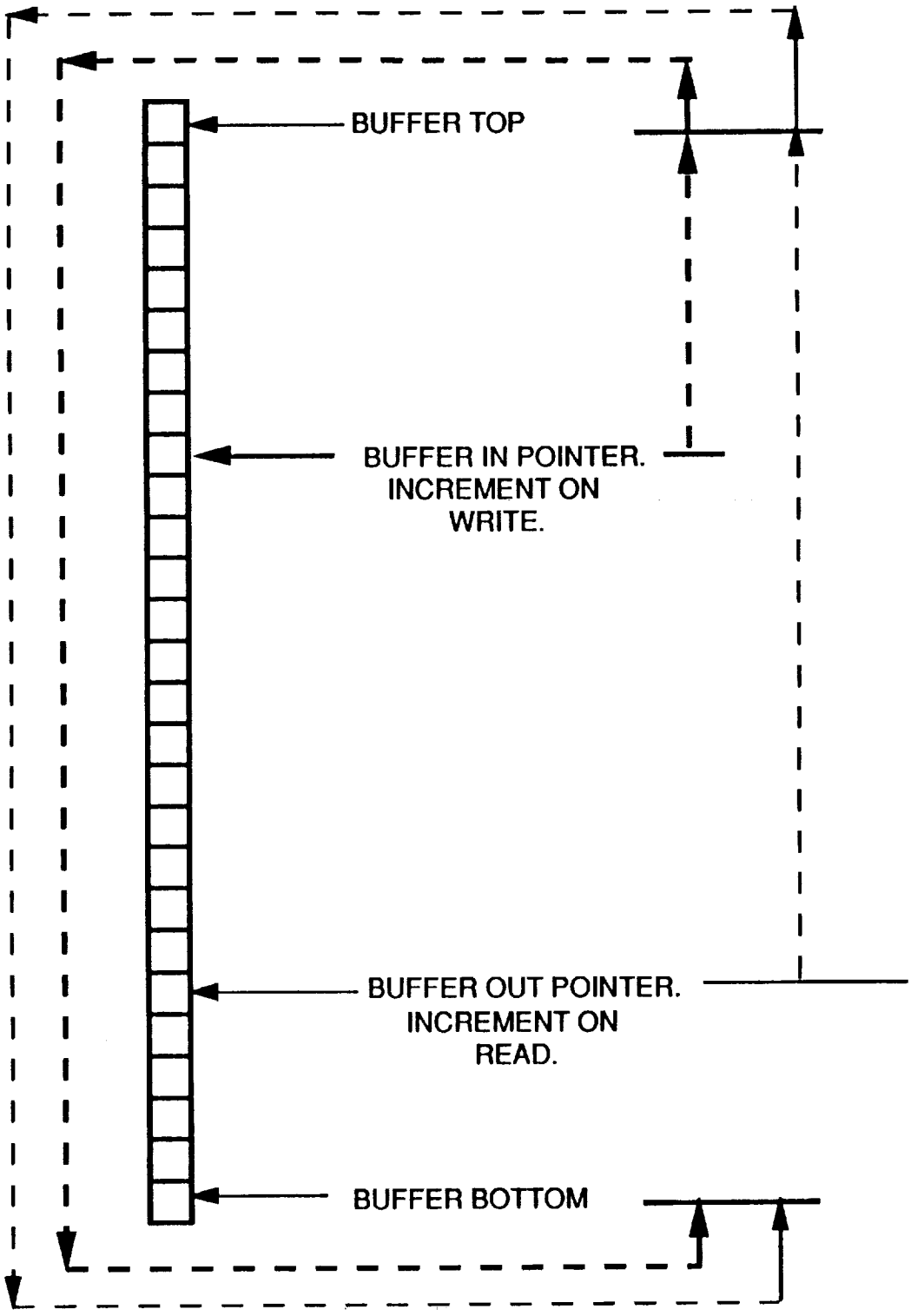

Figure 2. Circular FIFO Buffer Technique Used for Both Transmit and Receive.

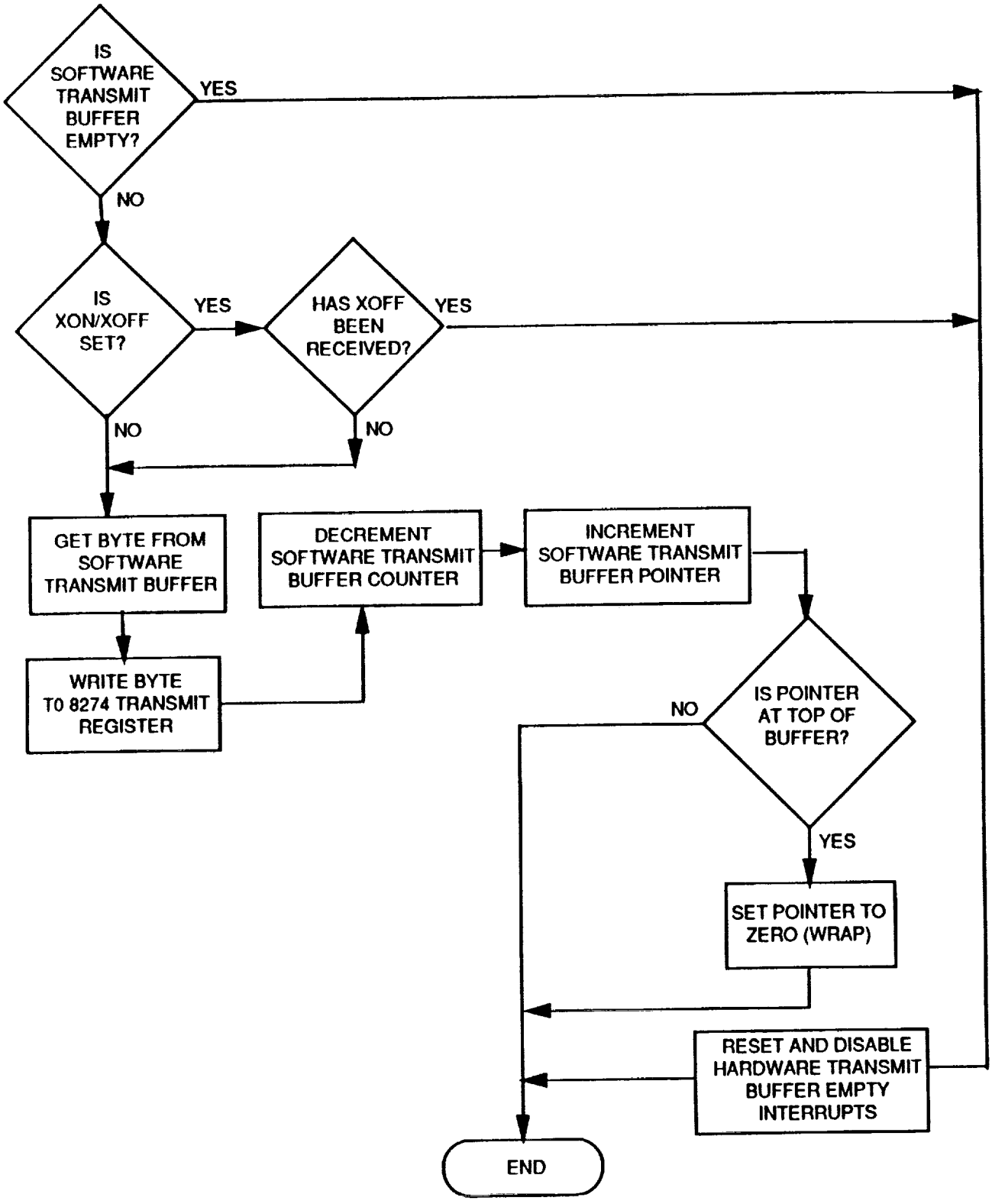

Figure 3. Flow Chart of Interrupt Service Routine For Hardware Transmit Buffer Empty Interrupt.

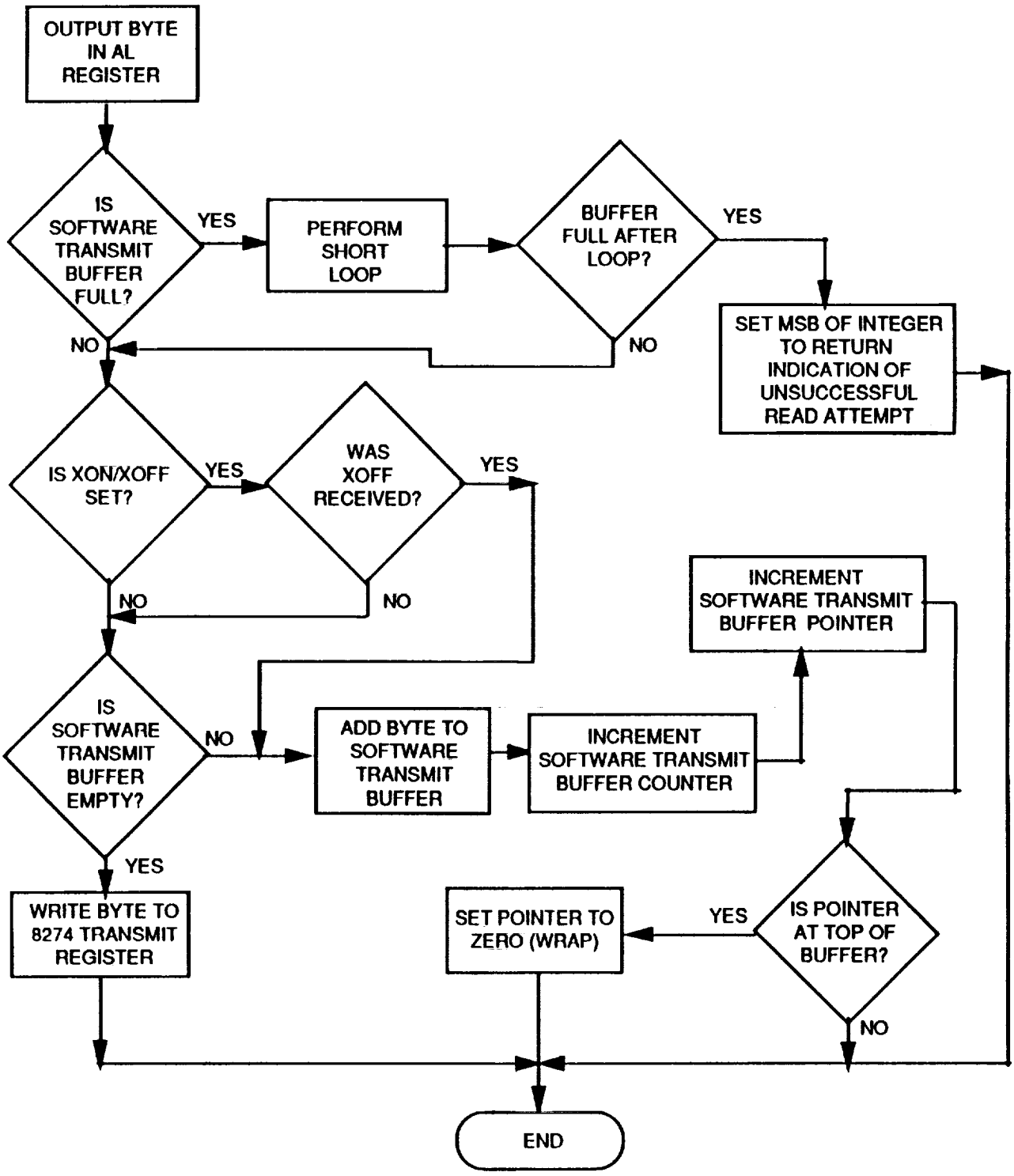

Figure 4. Flow Chart of Function Which Transmits a Byte to the Serial Port

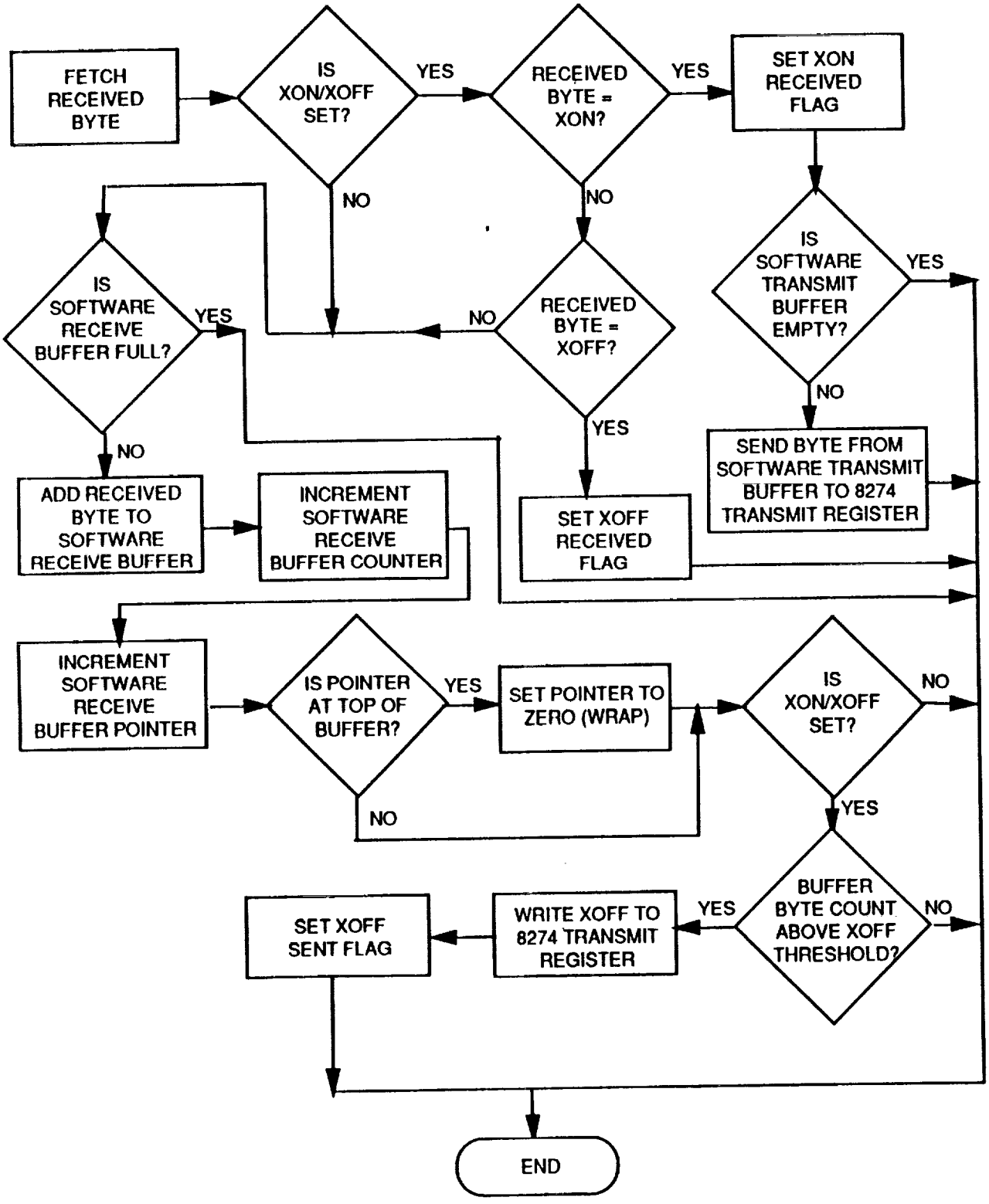

Figure 5. Flow Chart of Interrupt Service Routine For Received Character Interrupt.

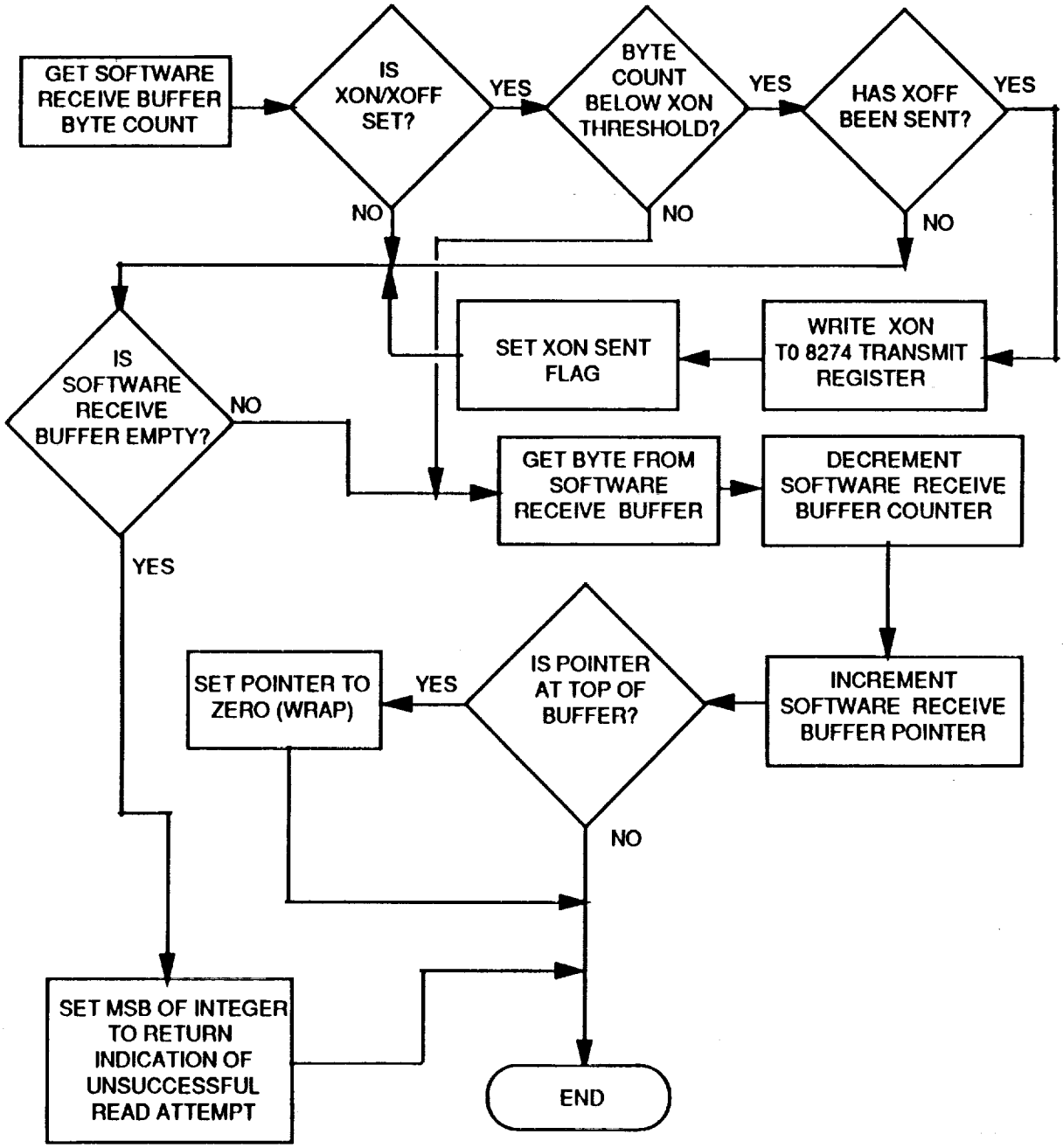

Figure 6. Flow Chart of Function Which Fetches a Byte From the Software Receive Buffer.

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

Expanded Serial Communication Capability for the Transport Systems Research Vehicle Laptop Computers

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-104176

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Unclassified-Unlimited

Subject Category 04

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

A recent upgrade of the Transport Systems Research Vehicle (TSRV) operated by the Advanced Transport Operating Systems Program Office at the NASA Langley Research Center included installation of a number of Grid 1500 series laptop computers. Each unit is a 80386-based IBM PC clone.

RS-232 data busses are needed for TSRV flight research programs, and it has been advantageous to extend the application of the Grids in this area. Use was made of the expansion features of the Grid internal bus to add a user programmable serial communication channel.

Software to allow use of the Grid bus expansion has been written and placed in a Turbo C library for incorporation into applications programs in a transparent manner via function calls. Port setup; interrupt-driven, two-way data transfer; and software flow control are built into the library functions.

14. SUBJECT TERMS

Flight Operational Improvements
Experimental Flight Displays

15. NUMBER OF PAGES

34

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		