Chapter 2
SYSTEM CONFIGURATIONS

This chapter contains general rules governing the attachment and powering of various configurations. The reasons for the procedures should be apparent from the explanations given in this chapter, but if the reasons are not apparent a more complete explanation of the principles of operation may be consulted in APPENDIX 11.

Communication between a peripheral and the computer must conform to the limitations imposed by the hardware. For example, the speed of comunication is always limited. The limitations are different for different peripherals. The computer must "know" what peripheral it is communicating with and how to compose and interpret messages to and from that peripheral. Some of the information about the peripheral that permits the computer to handle it is in computer memory, but initially comes from different sources. For example, the diskette and the disk drive are the source of information for handling communications with the disk drive. This information is passed to computer memory when the computer console is powered on, provided the disk drive is already powered on and contains an appropriate diskette.

Information about the serial ports of the ATARI 850 Interface Module is contained in the Interface Module itself and is transferred to computer memory when the computer console is powered on. Therefore, you must power on the Interface Module before you power on the console, if you intend to use a serial port.

You may unnecessarily boot the Interface Module. That is, you may turn on the Interface Module then the computer console, but not use a serial port. This does no harm, but it "wastes" RAM (1762 bytes) by loading the unused RS-232-C handler.

You may wish to use the Interface Module to interface to a non-ATARI device. In that case, you must make sure that the device is compatible with the Interface Module. You must examine the specifications of your device. You may also have to make or specify a cable to connect the Interface Module with your device, and APPENDIX 11 contains guide-lines and connector information on this subject.

Hook-up

The general rule: order of hook-up, in itself, has no importance.

Simply hooking up a peripheral to the Interface Module has no effect on its operation and is not recognized by the system. It is only when the peripheral is powered on that it has any effect. To have an effect, the peripheral must be powered on. However, an effect, once exerted by having a powered peripheral, is not necessarily canceled by turning the power off or disconnecting the hook-up. What is important, as a general rule, is the sequence of powering up the various components of the system.

Power-up

You should turn on the power to any peripheral that you intend to use before you turn on the power to the console. This general rule has exceptions which will result in your taking a pointless precaution. For example, the 825 Printer can be powered on at any time.

When the Interface Module and a Disk Drive are in your system, they will both have to boot up before they can be used, so they both have to be powered on before you power on the console. All of the ATARI Disk Operating Systems, except DOS I, are compatible with the RS-232-C handler in the Interface Module.

If you have DOS I, you cannot use the Disk Drive with the Interface Module. You should obtain DOS II (or a later DOS).

If you have DOS II (or a later DOS), power up the Disk Drive before the Interface Module.

If you have a non-ATARI DOS, there is no assurance that it will operate the Disk Drive with or without the Interface Module. You must consult the DOS documentation for guidance.

Configurations Using Only the Printer Port of the Interface Module

Turn on the computer console before the Interface Module, since the Interface Module does not need to be booted in order for the computer to access the Printer Port, but the Interface Module must be powered on in order to transfer communication between the computer and the printer. The printer handler is in the computer Operating System, not in the Interface Module ROM.

IF YOU HAVE MORE THAN ONE ATARI PRINTER

You may connect two ATARI printers to the computer at the same time. However, the printer handler built into the Operating System can control only one printer at a time. Therefore, to avoid errors, make sure you turn the power on to only one printer at a time.

You may switch printers at any time, even with the computer turned on.

If you have no printer connected to the ATARI 850 Interface Module parallel printer port, or if that printer is turned off, then the Interface Module will not act like a printer, and you may use another printer connected to the system.

Configurations Using only the Modem With the ATARI 850 Interface Module

You must have an appropriate cartridge in the console (e.g., TeleLink I) and you must power on the Interface Module before the console. With TeleLink I, you may not use a Disk Drive.

If you are using a Disk Drive as well as the Modem (with a compatible cartridge), the DOS on the diskette in the Disk Drive must be DOS II (or a later version.) DOS I is not compatible with the Interface Module. You must refer to the documentation for the version of DOS that you are using.

The ATARI 830 Modem is acoustically coupled. Both input and output signals are transformed into tones in the audible range. Consequently, the modem may respond to extraneous sounds in the environment. The rubber muffs on the modem attenuate environmental sounds, but loud sounds or nearby percussive effect (such as tapping the table) are quite likely to be received and/or transmitted as (false) signals. You should place the modem in a location to minimize unintended effects of this nature.

Table 2.1 Power-up procedure with various common configurations

System includes 400 or 800, ATARI 850, and	Cartridge	DOS*	Comments
825 Printer	Language (BASIC, Assembly/ Editor, etc.)	No	No restrictions on power-on sequence.
830 Modem	TeleLink I	No	Turn on ATARI 850, then console.
825 Printer 810 Disk Drive	Language	Yes	Turn on 810, then console, then ATARI 850. Position of 825 in sequence does not matter.
825 Printer 830 Modem	TeleLink I	No	Turn on ATARI 850, then console. Position of 825 in sequence does not matter.
830 Modem ATARI Disk Drive	Language	Yes	Turn on ATARI 850 and disk drive, then console
825 Printer 830 Modem 810 Disk Drive	Language	Yes	Make sure you have DOS II.

Note: DOS requires a machine with 16K RAM. The amount of RAM left for your BASIC program is 16K less the RAM required for OS (about 3K), Interface Module handler (1762) and BASIC (about 8K). The RAM used by DOS can be determined exactly by performing PRINT FRE(0) with and without DOS loaded--the difference in the numbers printed is the DOS size.

Chapter 3
SERIAL PORTS

The configuration and use of serial ports is a complex matter. You must keep in mind a large number of details and you must observe complicated procedures exactly. We have tried to reduce the amount of technical detail in this chapter, giving only sufficient detail to show the structure and effect of commands and how they relate to each other. The supportive detail will be found in the Appendices. Once you are familiar with capabilities of the Interface Module, you will probably make most freqeuent reference to the Appendices. \NEED 9;SPACE 5 Overview

The RS-232-C standard defines a range of values of parameters of a communication link. This is described more fully in APPENDIX 1. The ATARI ATARI 850 Interface Module is the device used in ATARI Personal Computer Systems to set the values of these parameters. The Interface Module organizes the bit stream of communication according to the software-coded intructions.

When we refer to a communication port as an RS-232-C port, we mean that signals to/from that port conform to the RS-232-C standards. We also use the adjective "RS-232-C-compatible" when the communication conforms to essential aspects of the RS-232-C standard, with the implication that the channel may lack some features of the standard and may in- corporate other features not included in the standard. Perhaps the most important aspect of the standard is the specification of voltage levels corresponding to mark and space. Accordingly, many other publications may use the term "RS-232-C-compatible" to mean "using the voltage levels in the RS-232-C standard".

Using software instructions to set the particular standard or "protocol" is called "configuring the port". In configuring the port you may set the following:

Baud
Number of bits per word sent/received
Number of stop bits per word sent
Whether the incoming control signals DSR, CTS and CRX are monitored
Whether input parity is checked
Whether output parity is set

Whether Line Feed is added after every Carriage Return sent
Translation of the word being sent or received (3 types of translation)

Whether the outgoing control signals DTR, RTS are used

These are shown as three groups, corresponding to the three configuration commands; otherwise, the division into groups is arbitrary.

Default Conditions

If you do not configure the port, the system sets default values of the port variables, as follows:

300 Baud
8 bits per word
1 stop bit per word transmitted
Input parity is not checked
Output parity bit most significant bit (bit 7) is set to zero

Linefeed is not added after every Carriage Return
Light-translation (see APPENDIX 6)

Outgoing control signals DTR, RTS are not set.

Each of these groups of conditions can be changed with a configuration command.

Limitations on Port Configurations

The ports have different signals available, as shown in Table 2.2.

Table 2.2 Available signals on ports 1, 2, 3 and 4

PORT 1	PORT 2,3	PORT 4
DTR	DTR	XMT
RTS	XMT	
XMT		
DSR	DSR	RTS
CTS		
CRX		

If you are using the modem set for Half Duplex, connected to a port of the Interface Module, then the port when outputing must be configured for Block Output and when inputing must be configured for Concurrent Mode I/O at not more than 300 Baud.

Other limitations on port configurations are imposed by using a port in the concurrent I/O mode. These limitations are described in the

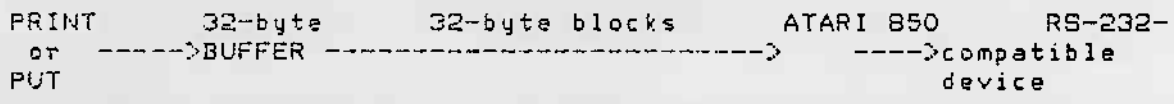
following section under the heading of Warnings and Restrictions.

Configurations with 5-, 6-, and 7-bit words are subject to limitations in other I/O parameters; these are described in the section on BASIC I/O statements, GET, INPUT, PUT and PRINT.

Mode--Block Output or Concurrent I/O

There are two different modes of using the a port, called BLOCK OUTPUT and CONCURRENT I/O.

Block Output:
Block output is used only for output from the computer to the ATARI 850 Interface Module. A block output is effected by the BASIC commands PRINT and PUT to a properly OPENed port.



The contents of the Buffer can be sent at any time (before it fills) by the Command FORCE SHORT BLOCK, described in the section called Forcing Early Transmission of Output Blocks.

Concurrent I/O Mode:
To receive information from the ATARI 850 Interface Module you must use this mode. It supports full duplex communication with the Interface Module. To use this mode, first OPEN a file for I/O then give the START CONCURRENT I/O MODE command (XIO 40), then use the BASIC commands INPUT, GET, PRINT or PUT. In this mode, BASIC is executing other instructions while I/O is proceeding. Incoming characters from a port are stored in a buffer. You may get the contents of the buffer at any time by INPUTing from that port.

Warnings and Restrictions

You must observe certain precautions when you use Concurrent Mode I/O. The only I/O opeations that are permitted with this mode are GET, INPUT, PUT, PRINT, STATUS and CLOSE to the OPENed port, and I/O to the Keyboard and Screen (which do not involve any peripheral device).

Using one Port for concurrent I/O prevents the use of any other Port of the Interface Module, including the Printer Port. The other ports remain inaccessible until you terminate the concurrent I/O mode. You terminate concurrent I/O by CLOSEing the port.

During Concurrent I/O, incoming data may overflow the conputer's buffer. In that case, data is lost. Methods for avoiding loss of data in this way are described in APPENDIX 8.

After you have started Concurrent Mode I/O, you can not configure a port. Therefore, all configuration commands (XIO 34, XIO 36 and XIO 38) must be executed before a START CONCURRENT MODE I/O command.

Once set, configured parameters will not change until you change them with an appropriate command. Pressing SYSTEM RESET on the computer console will NOT reset any parameter to its default value. Turning off the power on the Interface Module may reset some parameters but not others and may result in peculiar operation as information about some of these parameters is saved both in the computer and in the Interface Module. Turning off the power to the Interface Module during a session with the computer is not recommended.

Turning off the power to the computer also resets the Interface Module. When you turn the computer back on, and the Interface Module auto-boots (see the section on automatic bootstrapping in APPENDIX 11), all of the above parameters will have reverted to their pre-set default values.

Configuring a Port

If any default condition is to be changed, the port must be configured before it is used. Configuring a port is accomplished by one or more commands described in this section. There are three principal commands. Each of these three is concerned with several configuration variables. The parameters of the commands are coded to signify different values of the several variables. The details of the coding are presented in the Appendices.

A particular default condition is Block Output Mode in which, of course, you can not input data. To input from a port you must put it into concurrent I/O mode with the START CONCURRENT I/O command. Thus, the START CONCURRENT I/O command is a configuration command. It is different from the other configuration commands in that the port to which it applies must be OPENed first. Moreover, it is much more complex than other configuration commands. You should think of the START CONCURRENT I/O mode command as being a configuration command in one aspect, and as having more important effects on other aspects of using the configured port, and, indeed, all the ports.

Setting the Baud, word size, stop bits and ready monitoring

Format: XIO 36, #Channel, Aux1, Aux2, "Rn:"
Example: XIO 36, #1, 138, 6, "R:"

This command sets the Baud rate, word size, and number of stop bits in transmitted messages. It also controls the monitoring of incoming control signals.

Channel is the number of the Channel that BASIC commands for this port must use.

Aux1 is coded to specify 3 variables--Baud, word size and the number of stop bits. The coding is given in Tables 1, 2 and 3 in APPENDIX 5.

Aux2 is coded to specify which of the incoming control signals should be checked. These signals are DSR (Data Set Ready), CTS (Clear to Send) and CRX (Carrier Detect). The coding is given in Table 4 of APPENDIX 5.

Rn: is the port being configured. n is 1, 2, 3 or 4. R: is interpreted as R1:

Setting Translation Modes and Parity

Format: XIO 38, #Channel, Aux1, Aux2, "Rn:"
Example: XIO 38, #2, 64, 33, "R2:"

This command sets the various aspects of translation of message coding between.

38 specifies this I/O command.

Channel is the number of the Channel that BASIC commands for this port must use.

Aux1 is coded to specify the translation mode, input parity mode, output parity mode and whether a Linefeed is added after Carriage Return. The coding is given in Tables 1, 2, 3 and 4 of APPENDIX 6.

Aux2 is the number equivalent to the "won't translate" character in one translation mode. See APPENDIX 6.

"Rn:" is the port being configured. n is 1, 2, 3 or 4. "R:" is interpreted as "R1:"

Controlling the Outgoing Lines DTR, RTS and XMT

Format: XIO 34, #Channel, Aux1, Aux2, "Rn:"
Example: XIO 34, #2, 160, 0, "R1:"

This command determines the use of XMT and the outgoing control lines DTR, RTS.

34 specifies this I/O command.

Channel is the number of the Channel that BASIC commands for this port must use.

Aux1 is coded to specify control of DTR, RTS and XMT. The coding is given in Tables 1, 2 and 3 of APPENDIX 7.

Aux2 is not used by this command. It may be set to zero.

"Rn:" is the port being configured. n is 1, 2, 3 or 4. "R:" is interpreted as "R1:"

Using a Port

This section describes the use of a port, including the instructions OPEN, CLOSE, and the BASIC I/O commands GET, INPUT, PUT and PRINT, LIST, SAVE. These commands should all be familiar to you from your previous use of BASIC.

Two new commands, specific to RS-232-C ports, are described, namely, START CONCURRENT I/O (XIO 40) and FORCE SHORT BLOCK (XIO 32).

The use of the STATUS command is also described in this section. The command should be familiar from your previous use of BASIC, but the information that you can obtain about an RS-232-C port by using the STATUS command is, of course, new.

Opening an RS-232-C Port

You must OPEN a channel to an RS-232-C port before you can read from it, write to it, start concurrent mode I/O or read its status. You may configure a port without having opened it.

The OPEN command in BASIC is:

OPEN #Channel, Aux1, Aux2, "Rn:"

Channel is the number of the channel that other BASIC commands for the opened port must use. Any channel number (1 through 7) may be used. Do not use a channel if another file is already open through it.

Aux1 specifies the direction of the port:

- 5 signifies that you are going to use the port for input only (concurrent mode)
- 8 signifies that you are going to use the port for output only (block mode)

9 signifies that you are going to use the port for output only (concurrent mode)

13 signifies that you are going to use the port for input and output (concurrent mode)

Aux2 is not used in this command; make Aux2 zero.

Rn is the RS-232-C port being opened. n is 1, 2, 3, or 4. R: is interpreted as R1: For a given port no more than one channel may be open at one time.

Closing an RS-232-C Port

Having OPENed and used a port, you may disconnect the channel by closing the port with the BASIC command CLOSE, as follows:

```
CLOSE #Channel
```

Channel is the channel number previously OPENed.

CLOSE is also used to terminate concurrent mode I/O. In this case the channel number is that one through which the concurrent mode I/O is active. CLOSE is the only way to terminate concurrent mode I/O from a program.

To restart concurrent mode I/O to the port, you must first re-OPEN a channel to it.

When you CLOSE the channel, all data in the input buffer is lost, and all data in the output buffer is sent.

Closing a file does not change the configuration of the channel. You may change any configuration parameters after closing the port.

FAILURE TO TERMINATE CONCURRENT MODE I/O PROPERLY BEFORE ATTEMPTING I/O TO OTHER PERIPHERALS (OR EVEN OTHER RS-232-C PORTS) WILL PROBABLY RESULT IN PROGRAM FAILURE. THE ONLY WAY TO RECOVER FROM SUCH FAILURE IS BY TURNING THE COMPUTER OFF THEN ON AGAIN, WHICH RESULTS IN THE LOSS OF INFORMATION IN MEMORY.

Pressing SYSTEM RESET on the computer console closes all open channels and re-establishes the I/O system's registers and pointers. This method of closing files results in the loss of data being held in input and output buffers. The Interface Module may be "interrupted" by the SYSTEM RESET and so transmit only part of the character being sent at the time SYSTEM RESET was pressed. Another possible effect of SYSTEM RESET is a short burst of random data to an active concurrent mode RS-232-C port.

The exclusion of peripheral I/O to anything other than the active concurrent mode I/O port applies to the CLOSE command. If you have any other peripheral device or RS-232-C port open, you cannot CLOSE it while one open port is in the concurrent mode.

If you do not close files with CLOSE, BASIC will close them when it interprets END or comes to the end of the program. However, you do not know the order of the closing of files that BASIC will impose. BASIC will as likely as not close another channel before it closes the channel of the active concurrent mode I/O port. If it closes another channel first, your computer will "die", just as it would in response to any other attempt to perform I/O to a channel that is not the active concurrent mode I/O channel.

Therefore, ALWAYS MAKE SURE THAT AN ACTIVE CONCURRENT MODE I/O CHANNEL IS CLOSED BEFORE ANY OTHER CLOSES CAN OCCUR.

Starting Concurrent I/O Mode

```
Format:  XIO 40, #Channel, 0, 0, "Rn:"
Example: XIO 40, #1, 0, 0, "Rn:"
```

This command is used to start concurrent I/O mode.

40 specifies this I/O command.

Channel is the number of the Channel

n is the port number

With this command the input buffer is in the handler in the computer. The buffer holds 32 bytes. For some purposes a longer buffer is more convenient. APPENDIX B shows how to specify any size of buffer. The BASIC coding is more complex.

BASIC I/O Statements GET, INPUT, PUT AND PRINT

The BASIC input statements are GET and INPUT. The BASIC output statements are PUT and PRINT. Please refer to the BASIC Reference Manual for details about these statements. In this context, PRINT and INPUT must always include the proper channel number. The formats are given here as a reminder.

```
Formats:  GET #Channel,var
          INPUT #Channel{;}{avar}[{avar}...]
                         {,}{svar}[{svar}...]
```

```
PUT #Channel,aexp
PRINT #Channel{;}exp[,exp...]
                {,}
```

This section is about how these statements are used with the Serial Ports. Before reading this section you should read and understand the material on configuring the ports, including Appendices, and the sections on opening and closing ports and starting concurrent I/O.

INPUT and PRINT are line-oriented. They process a "line" of characters at a time. A line ends with an ATASCII EOL (End-of-Line) character. The translation mode you set up (or the one pre-set for you) can be used to translate the EOL character to an ASCII CR (Carriage Return) on output, and CR to EOL on input. AN EOL IS REQUIRED FOR INPUT--THAT IS, A BASIC INPUT STATEMENT WILL NOT FINISH UNTIL AN EOL IS READ IN. If your input does not have EOL, or if your translate mode will not produce it on input, you should not use INPUT; use GET instead.

Remember that if you place a comma or semicolon at the end of a PRINT command, EOL is not produced when the PRINT command is executed.

When you use a BASIC input statement, the input data must be in the proper form for BASIC. For example, if you read into numeric variables, the input must consist of digits with optional sign, decimal point, and exponent. Multiple input numbers must be separated by commas. For more details see the BASIC REFERENCE MANUAL.

GET and PUT are character-oriented. You can input or output only one character at a time. This is much slower than INPUT and PRINT, but it gives you more control over what you send and receive. You may alternate between the different types of BASIC input statements, and between the output statements, to the same port if you need to.

To do input and/or output, you must have opened a channel to the port and you must have specified the necessary in, out, and concurrent permissions when you opened the file (see the section about OPEN). To do input, you must also have started the concurrent I/O mode.

Output may be done either in BLOCK MODE or in CONCURRENT MODE. When you do output in block mode, you MUST NOT start concurrent mode I/O before doing the output. For this reason, full duplex operation is not allowed with block mode output.

Block mode output sends your data out to the Interface Module in 32-character blocks (whenever 32 characters have been collected by the handler from your PUT or PRINT statements). The Interface Module then sends the characters over the RS-232-C port. The computer waits while the Interface Module sends the block over the RS-232-C port. Between blocks the computer's I/O port is not being tied up as it is when concurrent mode I/O is active, so if you use block mode there are no restrictions on usng other I/O devices at the "same time."

Block mode is the only mode in which you can transmit 5-, 6- or 7-bit words (the word size option of the SET BAUD RATE command will not work with concurrent mode output). 8-bit words may be transmitted in either block or concurrent mode.

NOTE: On rare occasions, the Operating System may resend a block to the Interface Module. This may result in part or all of the the block being sent twice to the RS-232-C peripheral. To avoid this problem, use concurrent mode output.

Concurrent mode output does not work in blocks. Instead, whenever your program tries to output any characters to the Interface Module, they are first moved into a 32-byte buffer. As long as there are any characters in the buffer which have not been sent they will be sent as fast as possible. This "draining" of the buffer takes place concurrently with execution of other BASIC statements in your program. The only time your program will be held up is when you try to output characters faster than they can be transmitted at the Baud rate you are using and the buffer fills up. Your program will be held up until space becomes available in the buffer. If you do not want your program to be held up you may use the STATUS command to find out how much buffer space you have used and let your program use that information to decide whether or not to execute an output statement.

Concurrent mode input may be used at the same time you are using concurrent mode output (full duplex operation). Note that block mode output is NOT allowed if you do this. Also note that 5-, 6-, or 7-bit words can only be input in half-duplex mode. If you select anything other than 8-bit words you cannot output and input them at the same time. 5-, 6-, and 7-bit words can be input at speeds up to 300 Baud.

Concurrent mode input data is placed in the input buffer as it is received from the RS-232-C port. Your program must get the data out of the buffer with GET or INPUT before the buffer fills or data will be lost from the buffer. If the buffer fills, the data that has been in the buffer the longest will be replaced by the newer data. What your program will see is that characters are missing. The STATUS REQUEST command will tell you if data has been lost this way. STATUS REQUEST can be used to find out how many characters are in the input buffer, so you can program the machine to decide when to do an INPUT. STATUS can also be used to determine some kinds of errors in data reception and parity. This is fully described in the section on the STATUS command.

OTHER I/O COMMANDS FROM BASIC--LIST, SAVE, LOAD AND ENTER

There are a number of BASIC statements which perform "compound I/O" operations. That is, their operation can be thought of as consisting of combinations of the other I/O operations. These combinations are

built into BASIC so you cannot change the ways these statements work. The statements are LIST, SAVE, LOAD and ENTER. (Note that each of these statements inputs or outputs part or all of your PROGRAM, unlike, say, the PRINT statement, which outputs your program's DATA or variables. This distinction is not important here; what we are looking at in this section is how these statements work with the RS-232-C ports, not what data they transfer.

Each of these statements can be thought of as consisting of first an OPEN, then one or more input or output operations, then a CLOSE. These operations do NOT include any configuration of RS-232-C ports, and they do not include any START CONCURRENT MODE I/O action. Thus you cannot use the two input statements (LOAD and ENTER) with the RS-232-C ports. Instead, you may enter your program as data, to either a program you write in BASIC, or to the ATARI TELELINK II smart terminal cartridge, and put the program on cassette or diskette. Then you can ENTER the program to BASIC from the cassette or disk.

Since the configuration commands may be executed without opening a channel to an RS-232-C port, you can configure the Baud rate, translation modes, and so forth, before you execute a LIST or SAVE to the port. (SAVEing a program to the RS-232-C port will send the program in BASIC'S internal 8-bit tokenized format--this will probably only be useful if you are sending the program to another ATARI computer.) Since LIST has no implicit Interface Module status checking, the program will simply be sent out at the maximum rate allowed by the Baud rate you have selected. The receiving device must therefore be able to receive the data at that rate.

FORCING EARLY TRANSMISSION OF OUTPUT BLOCKS

If you are using the block output mode, your output characters will be place in a 32-byte buffer and transmitted: 1) when the buffer fills up; 2) when you close the channel to the Rs232 port; 3) when a CR (decimal 13) is placed in the buffer.

On occasion, you may want to force the sending of the information in the buffer. For example, if you have specified the Append LF translate option, the LF will be sent at a different time, later than the CR. You may want to send the LF immediately if the external device is a terminal. As another example, if you are using the DSR, CTS, or CRX monitoring feature to avoid sending more characters to a device than it can handle, you can use the FORCE SHORT BLOCK operation to send your characters one (or a few) at a time. By sending a few characters at a time, you insure that the device stays "ready" and properly receives all the characters sent to it. (See the overview section, the Baud section, and the STATUS REQUEST section for more information on the monitoring of DSR, CTS and CRX lines.)

The FORCE SHORT BLOCK operation is only valid if you are using block output mode. If you are using concurrent mode, you cannot use this command.

If you issue a FORCE SHORT BLOCK command when the buffer is empty, no action will be taken. Doing this is not an error. Since you can alternate output to two RS-232-C ports when using block output mode, you can also alternate FORCE SHORT BLOCK commands from one port to another. The ports must be opened through different channels, of course.

The BASIC form of the FORCE SHORT BLOCK command is:

```
XIO 32, #channel, Aux1, Aux2, "Rn:"
```

32 specifies the FORCE SHORT BLOCK command.

Channel is the channel through which you have OPENed the RS-232-C port.

Aux1 and Aux2 are ignored in this command. In general, you should specify zero for them.

Rn is the port whose output buffer you are forcing. n is 1, 2, 3 or 4. R: is interpreted as R1:

STATUS COMMAND

The STATUS command is useful for determining many facts about an RS-232-C port and the state of the Interface Module. You can check for certain specific error conditions, to find out why certain errors have occurred, to check parity, and so on. The STATUS command allows you to determine the amount of data in the input and output buffers while concurrent mode I/O is in effect. STATUS also allows you to check the state of the RS-232-C control lines DSR, CTS, CRX (and the state of RCV at the tme you issue the STATUS command).

The STATUS command may be issued only through a channel opened to an RS-232-C port. You may issue the command whether or not concurrent mode I/O is in effect. If this mode is in effect to a port, you cannot obtain status information (via the STATUS command) from another port.

The information returned by a STATUS command is different according to whether or not concurrent mode I/O is in effect. When concurrent mode I/O is in effect, the STATUS command allows you to see how full your input and output buffers are, but you cannot check on the state of the control lines DTR, CTS, CRX and RCV. (RCV can be directly checked, however, by PEEKing at the computer's serial I/O control register.) When concurrent mode I/O is not in effect, you get no information about buffers, of course, but the state of the control lines can be checked. There are other minor differences in the effect of the STATUS command in the two cases.

In BASIC, the STATUS REQUEST command is implemented as a "compound" command--that is, you must code multiple BASIC statements to get the status. The first, of course, is the STATUS command. This is followed by uses of the PEEK function to retrieve status which is left in a small status area by the STATUS command.

The STATUS command looks like this in BASIC:

```
STATUS #channel, avar
```

Here, #channel specifies the channel (1-7) through which you have OPENed the RS-232-C port. You may issue this statement to the port before and/or after concurrent mode I/O is started.

Avar is a variable which will get the status OF THE STATUS STATEMENT ITSELF. That is, avar will be set to the input/output system's one-byte status that is returned when BASIC calls the I/O system--since the I/O system call here is the STATUS, the value returned is the I/O system's determination about how the STATUS command went. This number is the same kind of number returned to BASIC by the I/O system after ANY I/O call, but in the other BASIC I/O statements, BASIC looks at the number itself to see if the I/O was completed without error. The STATUS command simply puts the number in the avar. This status number can be interpreted just like one of the ERROR codes--for example, you will get 130 if you neglected to OPEN the channel, since an unopen IOCB does not specify any peripheral device and error 130 means "Nonexistent Device Specified". The status number will be 1 if the STATUS call was completed without error. The status number will be some error number greater than 127 if there was some problem with the STATUS call.

If the STATUS call is successful, up to four bytes of information are stored in locations 746, 747, 748, and 749. Location 746 always contains error status bits relating to the status history of the RS-232-C port. The other three locations will contain buffer use information if concurrent mode I/O is active. If concurrent mode I/O is not active, 747 contains status bits relating to DSR, CTS, CRX, and RCV on the RS-232-C port, and locations 748 and 749 hold nothing.

Table I of APPENDIX 4 shows the definition of the error bits in location 747. The table gives each bit a decimal value which shows how that bit, if "on" or 1 (as opposed to "off" or 0), adds to the total value of the byte when interpreted as a decimal number. The meaning of each of these error bits are discussed in APPENDIX 4, but first here is a BASIC example showing how you can check one of the bits:

```
160 STATUS #1, IGNORED
```

```
170 LET ERRORBITS = PEEK(746)/128
180 IF INT(ERRORBITS) <> INT(ERRORBITS+0.5) THEN PRINT "OVERRUN!"
```

First the STATUS call is made, and the status of the STATUS goes into the variable IGNORED (which we don't check here--we assume the STATUS call itself is all right). Statement 170 takes the error bits from location 746 and divides it by twice the decimal number representing the bit being checked (as taken from Table I). In this case, we're checking for the BYTE OVERRUN error, whose number is 64, so we divide by 128. If the bit is 1, then the byte has a 64 in it, and after dividing by 128, the result has 1/2 (0.5) in it. When we add 0.5 in the next statement, we add a 1 to the result of the second INT. The INTs not being equal thus means we have found a BYTE OVERRUN. If the error bit were not there, the 0.5 would add to 0 (in the 1/2 position) and the second INT would be equal to the first.

APPENDIX 1

WHAT IS RS-232-C?

RS-232-C is a technical standard of the Electronic Industries Association (EIA). Published in August of 1969, it is titled "Interface Between Data Terminal Equipment and Data Communication Equipment Employing Serial Binary Data Interchange." The standard specifies electrical signal characteristics and names and defines the functions of the signal and control lines which make up a standard interface, called RS-232-C.

Figure 1 shows, diagrammatically, the kind of hook-up that RS-232-C was designed to standardize. A "DATA TERMINAL" is at each end of the communication link. The data terminal either generates or receives data (or does both); it could be a keyboard/screen "terminal" in the normal sense of the word; it could be a computer; and so on. The idea is that the data terminal is at the end of the communication link--hence "terminal." However, the data terminal need not really be at the end--you may really want to think of "data terminal" as just the name of one of the two ends of an RS-232-C connection.

At the other end of an RS-232-C connection is the "DATA SET." In the example of Figure 1, each data set takes data from the data terminal it is connected to and sends/receives the data over the communications link. The most familiar example of a data set is the MODEM, which takes data from a terminal and converts it for sending and receiving over a telephone line.

The ATARI Personal Computer System with the 850 Interface Module should be thought of as a unit comprising an RS-232-C Data Terminal.

FIGURE 1: COMMUNICATIONS HOOK-UP SHOWING ROLE OF RS-232-C

The data-set/data-terminal distinction should be kept in mind because the RS-232-C interface is DIRECTIONAL. That is, each line in an RS-232-C interface has a direction--one device drives the line (sends information) and the other receives the information. Each line in an RS-232-C interface is defined as being driven by either the data-set end or the data-terminal end.

The RS-232-C standard defines some 20 signalling lines or "circuits", as the standard refers to them. Most of them are optional and rarely used. Even with many omissions and deviations from the standard, a link may still be referred to as RS-232-C. It is more common to refer to the link loosely as "RS-232-C" or "RS-232-compatible".

The most commonly used RS-232-C lines are given in Table 1. The table shows the name of each line in the RS-232-C standard and the commonly used mnemonics.

TABLE I--THE MOST COMMON RS-232-C CIRCUITS

LINE NAME	DIRECTION	DESCRIPTION	ABBREVIATION (CIRCUIT)
BA	terminal-->set	Transmitted data	XMT BB
terminal	<--set	Received data	RCV CA
terminal	-->set	Request to send	RTS CB
Terminal	<--set	Clear to send	CTS CC
terminal	<--set	Data set ready	DSR AB (none)
Signal ground CF	terminal	<--set	Signal (carrier) detect
CRX CD	terminal	-->set	Data terminal ready DTR

It is most common practice to use common names or abbreviations for the RS-232-C signals, and not the two-letter names in the official standard. Thus the following happens: transmit and receive, for any given device, are RELATIVE TO THAT DEVICE. That is, data goes out of a device on XMT and comes in on RCV. Thus to connect two RS-232-C devices when given the common names of the signals, you should connect XMT to RCV (in one direction) and RCV to XMT (in the other direction). If one of the devices is wired as a data set and the other as a data terminal, then you should connect DTR to DTR, DSR to DSR, RTS to RTS, and so on. If, on the other hand, they are each wired as data terminals, you should be careful how things are connected--more on this later.

The signal ground connection must always be made. (Notice that RS-232-C requires that the ground potential of the two devices be equal, that is, their grounds are connected. Devices for which this requirement cannot be met cannot be connected via an RS-232-C interface.)

Data terminal ready is used by RS-232-C to allow the terminal to signal its readiness to send or receive data. This is a signal to auto-answer modems that they have permission to answer the ringing of the telephone line.

Data set ready is used by the data set to signal its readiness to send or receive data. This indicates that communications are established.

Request to send is used by the data terminal to tell the data set it wishes to send data. Some modems (Bell 102 for example) require this line to switch directions.

Clear to send allows the data set to signal its readiness to pass data from the data terminal.

The carrier detect line allows the data set to tell the data terminal that the communication link is established. This often differs little from data set ready, except that data set ready usually refers to "telephone off the hook" (answered) whereas carrier detect means something like "I hear the modem at the other end and we can talk now." When carrier detect goes OFF, data set ready OFF usually follows a few seconds later, indicating that the other end has "hung up."

In normal operation, DTR, DSR, and CRX are all ON. For full duplex operation RTS and CTS are also both ON. However, it is often not necessary to have all these lines be ON-- either one or the other devices on the RS-232-C connection does not have all the lines, or it is OK to ignore them (one of the properties of the RS-232-C standard is that not all of it needs to be implemented--it's perfectly OK to leave parts out). To operate the ATARI 830 Modem, for instance, none of the control lines need to be used. In fact, the ATARI 830 Modem ignores DTR and RTS, and it turns DSR, CTS and CRX on and off together (with carrier).

Note that the communication link shown in Figure 1 is not defined by RS-232-C. In particular, this link seldom has more than the "equivalent" of XMT and RCV--that is, only data lines and no control. However, as often as not this link is a full duplex link, so data can go both ways simultaneously. ASCII characters are the most common data sent, so the data sent each way can be either "control" data or "data" data.

With full duplex operation, two devices can "handshake" with data in various ways. Common terminals usually do not have an internal connection between the keyboard and display (or they have a switch, usually called half/full duplex, to make or break this internal connection) so when talking with a computer in full duplex mode (the most common mode), the computer at the other end "echoes" (sends back) each character to be displayed as it is typed. This allows you to see exactly what the computer at the other end receives. It also allows the computer at the other end to decide NOT to let you see what you have typed, as in "suppressing" the echo of a password.

Half duplex operation means that somewhere along the communications path, data may pass only one direction at a time. Not all parts of the communications path need be half duplex, but if any part is, then the whole system pretty much has to send data only one way at a time. In half duplex mode, the computer at the other end does not echo back what you type. In this case, in order to see what you type, the connection from keyboard to screen must be set locally, that is, set your terminal to "half duplex." (Note: the ATARI TeleLink I terminal emulation cartridge does not have the equivalent of a half/full switch. However, the ATARI 830 Modem does have such a switch, and when it is placed in the Half duplex position, it echoes any data sent out over the phone back to the ATARI computer console.)

A common "handshake" that requires full duplex is the XOFF/XON (transmit off/transmit on) handshake. The receiver of data can send XOFF to the sender to ask the sender to pause the data transmission, and XON to resume. This allows the user of a screen terminal to stop the data so he can read the screen, and it allows a computer which is receiving data from another computer to effectively control the rate at which it can accept data. There are many variations of XON and XOFF, including ACK/NAK and the BREAK signal, among others.

RS-232-C compatibility has come to cover many devices which are not "data sets" or "data terminals", particularly in the personal computer

world. What this usually means is that the device conforms to the ELECTRICAL RS-232-C specification, which is shown in Table II. Sometimes such devices (which include printers, plotters, digitizing pads, and many other interesting devices) also have lines which are called DSR, DTR, RTS and so on, but their use is often not covered by the RS-232-C standard and usually the use is specific to the device. One such use is to signal readiness to accept data from your computer (as opposed to sending XOFF/XON over a data line). Unfortunately, there is no standard of how many characters after the line goes OFF that the device will accept, nor a good way to determine where to start up again when the device becomes ready. You will have to familiarize yourself with your device's characteristics and then program you ATARI 800/400 computer and the 850 Interface Module accordingly.

TABLE II--RS-232-C ELECTRICAL SPECIFICATIONS

TYPE OF SIGNAL	FIRST STATE -24 volts to -3 volts	SECOND STATE +3 volts to +24 volts
Binary signal	1	0
Signal condition	MARK	SPACE
Control function	OFF	ON

It is common practice when using the 25-pin D connector most used with RS-232-C to connect XMT to pin 2, RCV to 3, RTS to 4, CTS to 5, DSR to 6, common signal ground to 7, CRX to 8, and DTR to 20. However, these conventions may not be followed, and you may also run into cases where the other pins in the connector have either entirely unrelated functions (such as other types of communication standards on the same connector) or possibly related functions (such as setting the Baud rate by connecting two pins). READ THE INSTRUCTIONS OF ANY DEVICE YOU INTEND TO CONNECT TO THE ATARI 850 INTERFACE MODULE CAREFULLY! You may have to make your own cable to connect the device to the Interface Module.

Believe it or not, the RS-232-C standard does not specify how data should be transmitted on XMT and RCV. In fact, RS-232-C explicitly avoids this issue. Fortunately, common convention and other standards have settled on a pretty universal serial data transmission convention. When data is not being sent, the data line sits idle in the MARK state. A data character (sometimes called a transmission WORD) is signalled by one START BIT, represented by the SPACE state. It is followed by the data bits (most commonly 8 of them), each bit being represented by SPACE for 0 and MARK for 1. The word is terminated by 1 (sometimes 2) STOP BIT, represented by the MARK state. The next word can immediately follow with its start bit, but if it does not, the line stays idle in the MARK state (effectively, the stop bit lasts indefinitely). The data bits are ordered least significant first, that is, the bit numbered 0 is sent first, 1 next, and so on. The receiver does not know when a character will be coming, so it has

to constantly monitor the stopped MARK state looking for the transition to a start bit. The receiver can then receive the rest of the bits in the word because it knows when each will arrive--each bit has the same duration as established by the BAUD RATE (bits-per-second rate) of the communication. Of course, both the transmitter and receiver must use the same Baud rate.

There are only a small number of common Baud rates, and the ATARI 850 Interface Module supports all of the most common ones. The most common transmission word size is 8 bits; when sending ASCII, which is a 7-bit code, the 8th bit usually represents the parity, is just set to 1 or 0, or is used as a marker bit of some sort. ASCII is very occasionally sent in 7-bit words. The ATARI 850 Interface module supports 7-bit words for these cases, and can also be used for communication with 7-bit or 6-bit codes such as BCD (with or without parity). Five-bit words are also allowed so you can communicate with old Baudot-code teletypes for radio-teletype and similar uses.