ATARI© 850 Interface Module Owner's Manual

COPYRIGHT 1980 ATARI, INC.

Every effort has been made to ensure that this manual accurately documents the Disk Operating System of the ATARI Personal Computer Systems. Howeven due to the ongoing improvement and update of the computer software ATARI, INC. cannot guarantee the accuracy of printed material after the date of publication nor can ATARI accept responsibility for errors or omissions.

C015953 rev. 1

TABLES and FIGURES

INTRODUCTION

Unpacking

Verify that the container for your ATARI 850CtmJ Interface Module contains the following items:
Interface Module
I/O Cable assembly (Data cord) No. CAO1412
Power Adapter No. CAO14748
Save all packing material in case you have to ship the Interface Module or store it for an extended period.

ATARI 850™ Interface Module

The ATARI 850 Interface Module allows communication between your ATARI Personal Computer and RS-232-C peripherals. The ATARI 410™ Program Recorder disk driver ATARI 820™ Printer and devices plugged in to a controller jack function without the interface module.
The interface module connects between the computer console and the peripherals. The interface module has four serial ports and one parallel port which is referred to as the Printer Port. The interface module contains a processor and memory and programmable ports. The programming of the interface module is controlled from the ATARI Personal Computer System.

ATARI 825™ Printer

The ATARI 825 Printer is an 80-column medium-speed Printer. It is much more versatile that the ATARI 820Ctm7 Printer. The 825 Printer requires an A1"ARI B50 Interface Module that is used in conjunction with ir
The 825 Printer connects to the Printer Port of the Interface Module.

ATARI 830™ Modem

The ATARI 830 Modem is a Bell 103-compatible modem that enables you to communicate over the switched telephone network with another terminal equipped with a similar modem. The ATARI 830 Modem requires an ATARI 850 Interface Module that is used in conjunction with ir
The modem usually connects to port 1 of the interface module. The modem under the control of the interface module produces tones that are sent out as electrical signals aver the telephone lines. Messages are received by the modem as tones that it decodes and sends to the interface module. The incoming/outgoing traffic is managed according to the programmed functions of the interface module. The technical specifications of the Modem are given in APPENDIX 13.

Contents of This Manual

The ATARI 850 Interface Module will be used with many different System configurations. Different configurations impose different demands on the user in what he has to know. The procedures are also different for different configurations. The general rules that must be observed in connecting and powering up are described and illustrated in Chapter 2.

Having connected your system and turned on the power. you then need to know how to program and use the ports to which you have connected your peripherals. The most complex technical material is in Chapter 3, which describes how to use the serial ports, and the supporting Appendices of Chapter 3.

The Appendices include user programs. code conversion tables. error codes. description of the RS-232-C communication standard and a technical description of the interface module. In addition. there are appendices that give the precise information necessary for configuring and using the serial ports.

How To Use This Manual

First determine the system configuration you are using. You may then use Table 1 of Chapter 2 to guide you in hooking up and powering your system.

If you are using the Interface Module with only the ATARI 825™ Printer all the necessary operating information is given in the manual supplied with the printer. You do not need the other information in this manual.

If you are using the Interface Module with a parallel printer that is not the ATARI 825 printer you will need to refer to APPENDIX 11 to understand the principles of operation of the Interface Module and timing constraints on the Printer Port and to APPENDIX 12 for pin connector information.

If you are using a serial port (with or without using the printer port as well) - you may need to configure the serial port. One case where you will not need to configure the serial port is where you are using a cartridge that requires a particular configuration which is imposed by the system when the power is turned on. For example, the ATARI TeleLink™ I uses serial port 1 but does not permit you to change the configuration or the operation of the port.

If your system configuration will support different configurations of the serial ports you should examine the default configuration (Chapter 3) to determine if the default settings are satisfactory and compatible with the specifications of your peripheral device. If the default configuration is satisfactory you can proceed to use the serial ports with proper I/O commands. The commands in BASIC are described briefly in Chapter 3 and described more fully in the Appendices. In this case you will not have to refer to the rather complex coding required for configuring a port.

If you do have to configure a port you should read the appropriate parts of Chapter 3 and then refer to the detailed information in Appendices 5 through 7. In case the material on configuring a port is confusing to you, you may find it helpful to read APPENDIX 1 (What is RS-232-C?) and/or APPENDIX 11 (Principles of Operation).

If you are familiar with the RS-232-C standard you will probably be able to proceed through Chapter 3, consulting the Appendices referred to there as necessary. However if you are not familiar with the RS-232-C standard it will probably be most efficient to read APPENDIX 1 on RS-32-C before you attempt to read Chapter 3.
The commands for configuring a port are covered very briefly in Chapter 3; little more than the form of the commands is give.

The detailed technical descriptions necessary for a full understanding of all aspects of configuring a port are placed in the Appendices that parallel the subsections in Chapter 3. Therefore you should read the port configuration parts of Chapter 3 to "see the big picture" and turn to the relevant Appendices for the detailed information necessary to write actual program segments.

The examples in APPENDIX 9 will show you how to control ports under the various conditions described in the examples. In addition, the examples should help you to understand how to configure and use a serial port if you run into problems with an application that is not very close to one of the examples itself.

Chapter 2

SYSTEM CONFIGURATIONS

This chapter c,ntains general rules governing the attachment and powering vf various configurations. The reasons for the procedures should be apparent frvm the explanations given in this chapter. but if the reasons are not apparent a more complete explanation of the principles of operation may be cvnsulted in APPENDIX li. Communicativn 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 abvut 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 memvry 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 guidelines and connector information on this subjecr

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 effett on its operation and is not recognized by the system. It is only when the peripheral is powered on that it has any effecr To have an effect, the peripheral must be powered on. Howeven 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. Howeven 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 printen 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 ompatible 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 environmenr The rubber muffa 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
ATARI 825 Printer	Language (BASIC, Assembly/Editon etc.)	No	No restriction on power-on sequence.
ATARI 830 Modem	TeleLink I	No	Turn on ATARI 850, then console.
ATARI 825 Printer ATARI 810 Disk Drive	Language	Yes	Turn on ATARI 810, then console, then ATARI 850. Position of ATARI 825 in sequence does not matter.
ATARI 825 Printer ATARI 830 Modem	TeleLink I	No	Turn on ATARI 850, then console. Position of ATARI 825 in sequence does not matter.
ATARI 830 Modem ATARI 810 Disk Drive	Language	Yes	Turn on ATARI 850 and Disk Drive, then console.
ATARI 825 Printer ATARI 830 Modem ATARI 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 procedure, exactly. We have tried to reduce the amount af technical detail in this chapten 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.

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 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 incorporate 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 DSN 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 DTN 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 DTN RTS are not ser

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 RTS XMT	DTR XMT	XMT
DSR CTS CRX	DSR	RTS

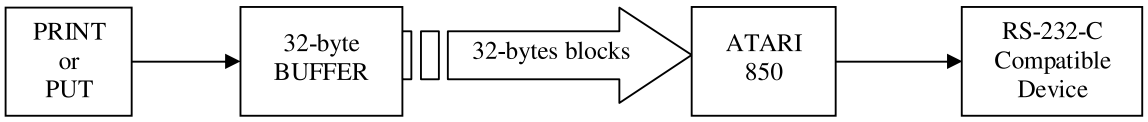
If you are using the modem set for Half Duplex, connected to a port of the Interface Module, then the port when outputing must be configured for Block Output and when inputing must be configured for oncurrent Mode I/O at not more than 300 Baud.
Other limitations on port configurations are imposed by 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 staternents, GET, INPUT, PUT and PRINR

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 PUR 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 inaccessable until you terminate the concurrent I/O mode. You terminate concurrent I/O by CLOSEing the port.

During Concurrent I/©, incoming data may overflow the conputer's buffer. In that case, data is losr 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 firsr Moreoven 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 tClear 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:

<u>Setting Translation Modes and Parity</u>

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

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 i, 2, 3 or 4. "R: " is interpreted as "R1:"

<u>Controlling the Outgoing Lines DTN RTS and XMT</u>

Format: **XIO 34, #Channel, Auxi, Aux,, "Rn: "**
Example: **XIO 34, #,, 160, 0, "R1: "**

This command determines the use of XMT and the outgoing control lines DTN 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 DTN RTS and XMR 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/C1 tXIO 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 ir

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 ir

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 ir

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

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 TNE 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-establi.shes 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, BASIG will close them when it interprets END or comes to the end of the program. Howeven 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", ,ust 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 convenienr APPENDIX 8 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 INPUR The BASIC output statements are PUT and PRINR 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 inpur 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 exponenr 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 outpur For this reason, full duplex operation is not allowed with block mode outpur

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 gou use block mode there are no restrictions on using 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 block being sent twice to the RS-232-C peripheral. To avoid this problem, use concurrent mode outpur

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 statemenr

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-32-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 INPUR 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.

<u>Other I/O commands from BASIC - LIST, SAVE, LOAD AND ENTER</u>

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. Z,his distinction is not important here, what we are lvoking at in thi5 5ection 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, befvre 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 therefare 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 RS-232-C 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 DSN 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 ir (See the overview section, the Baud section, and the STATUS REQUEST section for more information on the monitoring of DSN 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, Aux 1, 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 effecr STATUS also allows you to check the state of the RS-232-C control lines DSN CTS, CRX (and the state of RCV at the time 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 effecr 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 DTN CTS, CRX and RCV. (RCV can be directly checked, howeven 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 andlor 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 wenr 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 DSN CTS, CRX, and RCV on the RS-232-C port and locations 748 and 749 hold nothing.

Table 1 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 erron 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 ir When we add 0.5 in the next statement, we add a 1 to the result of the second INR 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 firsr

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". Howeven 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 ATAR1 Personal Computer System with the 850 Interface Module should be thought of as a unit comprising an RS--232-G Data Terminal.

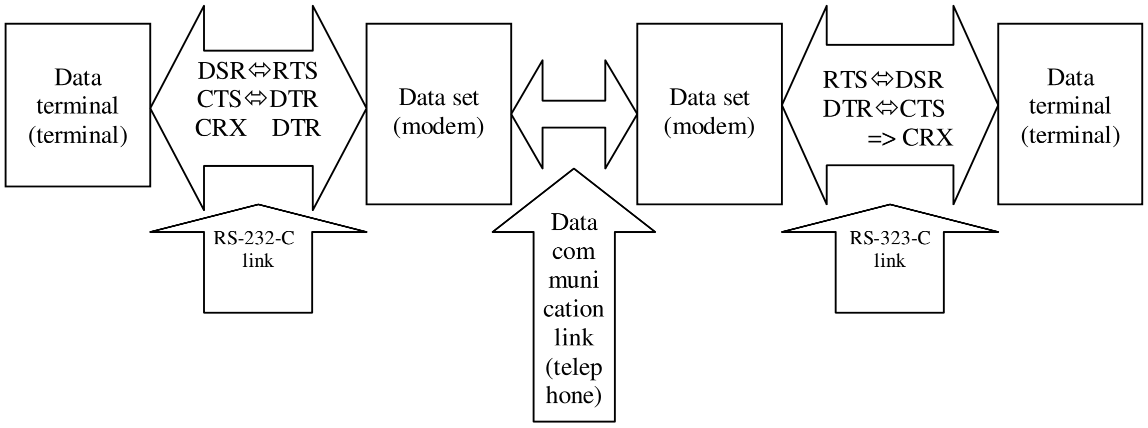


FIGURE 1: COMMUNICATIONS HOOK-UP SHOWING ROLE OF RS-232-C

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-23C- 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 stdandard 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" on "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)
	Terminal	Modem		
BA	$\longrightarrow$		Transmitted Data	XMT
BB	$\longleftarrow$		Received Data	RCV
CA	$\longrightarrow$		Request To Send	RTS
CB	$\longleftarrow$		Clear To Send	CTS
CC	$\longleftarrow$		Data Set Ready	DSR
AB	(none)		Signal Ground	-
CF	$\longleftarrow$		Signal (carrier) Detect	CRX
CD	$\longrightarrow$		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 DEVIGE. 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 DTN DSR to DSN 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.

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.

Signal (carrier) Detect 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 laten indicating that the other end has "hung up."

In normal operation, DTN DSR and CRX are all ON. For full duplex operation RTS and CTS are also both ON. Howeven 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 DSN 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 particulan this link seldom has more than the "equivalent" of XMT and RCV - that is, only data lines and no control. Howeven 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 communication, 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. Howeven 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 bark 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 DSN DTN RTS aud 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 400/800 computer and the 850 Interface Module accordingly.

TABLE II RS-232-C ELECTRICAL SPECIFICATIONS

Type of signal	First state -24 Volts t -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 bir 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 232 or is used as a marker bit of some sorr 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 communicatian 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.

APPENDIX 2

RS-232-C PORT ERROR CONDITIONS, CAUSES AND CORRECTIONS

This section contains descriptions of the errors you might encounter while using the ATARI 850 Interface Module. Many of these errors also occur with other ATARI peripherals, they are listed here so you can see what they mean when using the Interface Module.
There are a number of NEW errors which you can get from the Interface Module which no other peripheral produce. These new error codes are bolded.

ERROR 1 - Success. This is the status which successful completion of an I/O operation produces. BASIC does not report this to you except by continuing in normal fashion.

ERROR 128 - Break aborr This means you pressed the BREAK key while I/O was proceeding.

ERROR 129 - IOCB already OPEN. Your choice of channel number (#n) was that of a channel (IOCB) which was already OPEN. This can happen if you restart a program in a manner other than RUN (RUN closes files). Be careful not to put your OPEN statement inside a programmed loop. The second time OPEN is encountered it will produce ERROR 129.

ERROR 130 - Nonexistent device. You specified something other than R:, R1:, R2:, R3: or R4:. Perhaps you were trying to access a file on disk whose name starts with "R" and forgot the D: . THIS ERROR WILL OCCUR IF YOU ATTEMPT TO USE AN RS-232-C PORT AND THE RS232 HANDLER HAS NOT BEEN "BOOTED" WHEN THE SYSTEM WAS TURNED ON. In that case, you should save your program and start a new session. allowing the RS-232-C handler to boor See the section on automatic bootstrap.

ERROR 131 - Write only. You tried to read (GET, INPUT) from a port you OPENed as write only.

ERROR 132 - Invalid command. You specified something incorrectly in an XIO command to the Interface Module.

ERROR 133 - Channel not OPEN. You neglected to OPEN the channel (IOCB) to the I/O device you are trying to access.

ERROR 135 - Read only. You tried to write (PUT, PRINT) to a port you opened for read access only.

ERROR 138 - Device timeour The Interface Module did not respond to a command. Check the cables. Make sure the Interface Module is powered on.

ERROR 139 - NAK. The Interface Module refused to perform some command. You may issue a STATUS request to find out what was wrong. Mort common causes are: attempts to perform 5-, 6-, or 7-bit input at too high a Baud rate, automatic readiness checking was enabled and the connected device was not ready.

ERROR 150 - Port already OPEN. You attempted to OPEN an RS-232-C port but is was already OPEN through another channel (IOCB). You can access an RS-232-C port through only one channel at a time.

ERROR 151 - Concurrent mnde I/O not enabled. You attempted to start concurrent mode I/O (XIO 40) but the port was not opened with an odd number specified for Aux1 (Aux1 bit 0 not Set).

ERROR 152 - Illegal User-supplied Buffer. In the START CONCURRENT MODE I/O command with the user-supplied buffer, the buffer address and/or the buffer length were inconsistenr

ERROR 153 - Active Concurrent Mode I/O Error. You attempted to perform I/O to an RS-232-C port while Concurrent Mode I/O was active to some other RS-232-C port. Only input, output, CLOSE and STATUS commands to the active concurrent mode port are allowed while concurrent mode I/O is active, This error message is not always produced - attempts to do disallowed I/O while concurrent mode I/O is active may result in the computer "crashing".

ERROR 154 - Concurrent mode IIO not active. Concurrent mode I/O must be activated in order to perform input (GET, INPUT).

PRINTER PORT ERROR CONDITIONS, CAUSES AND CORRECTIONS

This section describes error conditions which could occur when using the printer port. There are no new error codes associated with the printer port however the meaning of some of the different between the Interface Module and other ATARI printers.

If an error occurs which is not listed here, consult the BASIC reference manual. Errors are listed here only if they have some new meaning when reported by the Interface Module.

ERROR 108 - Timeour The Interface Module did not respond to a request from the Computer. Check the cables. Make sure the Interface Module and attached printer are powered on. The Interface Module will NOT respond to printer control commands from the computer if the FAULT wire to the printer is low (caused by loose cable or printer off).

ERROR 139 - NAK. The interface Madule refused an illegal printer command. Make sure that Aux1 and Aux2 are specified as zero (0) in your OPEN command for the printer. This error also occurs when the printer appears active (FAULT line is high) but the printer fails to respond to characters sent to it within four seconds. Check the switches on the printer (online?). If the printer is not performing within 4 seconds, change your PRINT statements to break down transmissions into smaller chunks.

NOTE: Attempt5 to operate more than one printer at a time will result in unpredictable operation. While one printer may "win" most of the time, errors are always possible, and exactly which error occurs is a matter of chance. If you have more than one ATARI printer attached to your computer turn on only one of them at a time.

MEANING OF (ERROR) BITS IN LOCATION 746

TABLE I - Decimal Representation of the Error Bits in Location 746

Decimal Equivalent	Error
128	Received data framing error
64	Received data byte overrun error
32	Received data parity error
16	Received data buffer overflow error
8	Illegal option combination attempted
4	External device not fully ready flag
2	Error on block data transfer out
1	Error on command to Interface Module

Below are the descriptions of these error status bits in location 746 after STATUS command:

RECEIVED DATA FRAMING ERROR (bit 7, decimal value 128) This error bit indicates a framing error was encountered in the data coming from the external RS-232-C compatible device: the 10th bit of some character was not a STOP bit (9th, 8th or 7th in the cases of 7-, 6- or 5-bit received words). This error can be caused either by garbled data (e.g. noise on the phone lines) or by improper configuration to receive the data (e.g. wrong Baud rate). This condition is monitored in one of two places: in the 400/800 computer or in the Interface Module. The computer watches for this error in the case of 8-bit data. The Interface Module catches this error if you are receiving 7-, 6-, or 5-bit data. In both cases, the error status is set at the time the erroneous character is received (not the time you read it out of the holding buffer).

In the 8-bit data case, where the computer monitors the error, you may find out about the error any time after it occurs by issuing STATUS while the concurrent mode input is active. The error bit will be cleared when you issue the STATUS command. This error bit will be cleared also when you CLOSE the concurrent mode channel. In the 7-, 6-, and 5-bit cases, the error is monitored by the Interface Module and cannot be interrogated while the concurrent mode input operation is active. In this case, you must CLOSE and re-OPEN the concurrent mode channel and then issue STATUS to determine if the error occurred. The error bit in the Interface Module is cleared by STATUS when concurrent mode I/O is not active, it is also cleared by most of the configuring and control XIO's (but not all), and (it may be cleared) by CLOSE when concurrent mode I/O is not active.

In general, the error bits read from location 746 after a STATUS request apply only to the most recent I/O operation to the RS-232-C port that is, they are cleared as the I/O operation is started and then set if the error oceurred. Yau can see that the previous error is an exception to this rule. Other exceptions wll be noted.

RECEIVED DATA BYTE OVERRUN ERROR (bit 6, decimal value 64) This error bit is maintained by the computer and indicates that the computer got too busy to read all the data as it was arriving (due to overly heavy interrupt loading, or perhaps interrupts being masked off totally). This error is flagged when the first character of data following the error is read from the port and placed in the holding buffer. The error should not occur at all under normal conditions.

RECEIVED DATA PARITY ERROR (bit 5, decimal value 32) This error bit is maintained by the computer and indicates that a received character had the wrong parity. The bit will not be set if no parity checking has been enabled. This error occurs during the translation from the external (received) form of the character to the internal (INPUT, GET) form. The error flag bit is cleared by the STATUS command.

RECEIVED DATA BUFFER OVERFLOW ERROR (bit 4, decimal value 16) This error flag indicates that more data has arrived than can be held in the input buffer - data has not been read from the buffer (INPUT, GET) soon enough. This error is maintained by the computer and it occurs when the overflowing characteh arrives from the RS-232-C compatible device. The new character replaces the oldest one in the buffer. This error bit is cleared by the STATUS command.

ILLEGAL OPTION COMBINATION ATTEMPTED (bit 3, decimal value 8) This error flag is kept in the Interface Module and may be read by STATUS only if concurrent mode I/O is not active. It is set by an attempt to start concurrent mode input with short words (7-, 6-, or 5-bit) with the port open for both input and output (short words are allowed one direction at a time only) or too high a Baud rate (short words are allowed for input at a maximum rate of 300 Baud).

This error may be checked immediately after the Interface Module produces a NAK for the refused command. The bit is cleared by the

STATUS requesr Error bit (command erron decimal value 1> will always be set when this bit is ser

EXTERNAL DEVICE NOT FULLY READY (bit 2, decimal value 4) This bit is kept in the Interface Module and may be read by STATUS only when concurrent mode I/O is not active. It is set whenever a START CONCURRENT MODE I/O or block output command is refused by the Interface Module because one or more of the external status lines being monitored is not ON. Any of the external status lines not being monitored (as set by the SET BAUD RATE command) is ignored, and if none is being monitored this bit will not be set and the I/O operatian will proceed normally. Read this flag bit with a STATUS request immediately after the Interface Module refuses the operation with NAK. This flag is cleared by the STATUS command.

DATA BLOCK ERROR (bit 1, decimal value 2) This error bit is maintained in the Interface Module and may be read by STATUS immediately after a command is refused by NAK. In a block output, the data block was unsuccessfully received from the computer by the Interface Module. This error should not occur in normal operation, it indicates problems in communication between the computer and Interface Module.

COMMAND ERROR TO INTERFACE MDULE (bit 0, decimal value 1) This error bit is maintained in the Interface Module and may be read by STATUS immediately after a command is refused by a NAK from the Interface Module. This bit indicates that the Interface Module did not recognize a command sent to it from the computer or that the Interface Module is not configured properly to perform the command (see ILLEGAL OPTION COMBINATION ERROR).

During active concurrent mode I/O, the STATUS command will return the number of characters in the input buffer in locations 747 and 748, and the number of characters in the output buffer in location 749. To find the number of characters in the input buffer in BASIC:

```
LET BUFFERUSE = PEEK(747) + 256*PEEK(748)
```

If you want to find out only whether or not any characters are in the input buffer you do not need to multiply by 256:

```
IF PEEK(747)+PEEK(748)=0 THEN PRINT "input buffer empty..."
```

or:

```
IF PEEK(747)+PEEK(748)<>0 THEN PRINT "input buffer not empty..."
```

If you are using the built-in buffer or if your supplied buffer has fewer than 256 bytes, then location 748 will always be zero and you need to look only at location 747.

The output buffer holds only 32 characters, location 749 will never exceed 32.

When concurrent mode I/O is not active, location 747 will contain information about the monitored readiness lines (DSN, CTS and CRX) and the data receive line (RCV) of the specified port after a STATUS requesr Locations 748 and 749 will not contain anything useful after a STATUS request when there is no active concurrent I/O.

Location 747 will contain the sum of four numbers shown in table II. The current and past status of DSN, CTS, and CRX as well as the curnent status of RCV are included. The past status of DSN, CTS and CRX applies back to the time the Interface Module was booted, or to the most recent STATUS command to the specified port which was made while concurrent mode I/O was not active (i.e., the last time that DSN CTS and CRX were supplied to a STATUS request). No other operations affect the past status of these Iines.

Ports 2 and 3 will always show CTS and CRX as being ON. Port 4 will show CTS, CRX, and DSR as being ON.

A quick way to check whether or not a port is ready is this:

```
STATUS #n, XXX
IF PEEK(747)<128 THEN PRINT "not ready..."
```

or to check if it has stayed ready since the last check:

```
IF PEEK(747)>=192 THEN PRINT"always ready..."
```

In other words, the DSR status bits are the most significant bits in the sense byte, and you can check them this way without having to worry about the states of the other bits in the byte.

TABLE II - SENSE VALUES ADDED INTO LOCATION 747

DATA SET READY (DSR)	
192	Ready now (ON), on since previous STATUS
128	Ready now (ON), not always on since last STATUS
64	Not ready now (OFF), not always since last STATUS
0	Not ready now (OFF), always off since last STATUS
CLEAR TO SEND (CTS)	
48	Clear now (ON), on since previous STATUS
32	Clear now (ON), not always on since last STATUS
16	Not clear now (OFF), not always on since last STATUS
0	Not clear now (OFF), always off since last STATUS
CARRIER DETECT (CRX)	
12	Carrier now (ON), on since previous STATUS
8	Carrier now (ON), not always on since last STATUS
4	Not carrier now (OFF), not always off since last STATUS
0	Carrier now (ON), always off since last STATUS
DATA RECEIVE (RCV)*	
1	MARK (1) now
0	SPACE (0) now

* No information is supplied about the past status of RCV.

APPENDIX 5

SETTING THE BAUD, WORD SIZE, STOP BITS AND READY MONITORING

CONFIGURE BAUD RATE command allows you to set the Baud rate, "word" size, number of stop bits to transmit, and enable or disable checking of DSN, CTS and CRX. The command may be issued through an open channel to the RS-232-C port, or may be issued through a channel which isn't being used. If uou have opened a channel to the port you are corifiguring, you must use that channel. You cannot configure any port if a concurrent Mode I/O operation is active.

The CONFIGURE BAUD RATE command looks like this in BASIC:

```
XIO 36, #channel, Aux1, Aux2, "Rn: "
```

The **36** makes this a CNFIGURE BAUD RATE command.

Channel is the number of the channel that BASIC should use to execute the command. The channel should either be open to the port you are configuring. or should not be open at all. No concurrent mode I/O should be active when you issue this command.

Aux1 is a number or expression that specifies the Baud rate, "word" size, and number of stop bits to send with each "word." For each of these, pick a number from tables I, II, and III, and then add the numbers together to form Aux1. You may add them together yourself or you can let BASIC add them for you. For example:

```
XIO 36, #1, 128+0+10, 0, "R: "
```

and

```
XIO 36, 138, 0, "R: "
```

both specify the same thing.

Aux2 is a number or expression that specifies whether or not the Interface Module should check Data Set Ready (DSR), Clear to Send (CTS), and/or Carrier Detect (CRX) when a block mode output or START CONCURRENT MODE I/O operation is performed. If you ask to have the Interface Module check one or more of these, then the Interface Module will return error status if the line(s) checked is not ON. You may TRAP the error and program BASIC to take the action you desire. See table IV for values of Aux2.

The last XIO parameter: **"Rn: "** specifies which serial port of the Interface Module you are configuring. For n put 1, 2, 3, or 4, just as you would in the OPEN command.

TABLE I: BAUD RATE SPECIFIERS TO ADD TO AUX1

ADD	Baud rate	ADD	Baud rate
0	300 bps[i]	8	300 bps
1	45.5 bps*	9	600 bps
2	50 bps*	10	1200 bps
3	56.875 bps*	11	1800 bps
4	75 bps**	12	2400 bps
5	110 bps	13	4800 bps
6	134.5 bps***	14	9600 bps
7	150 bps	15	9600 bps

1) bps = bits per second = baud
*) These Baud rates are useful for communications with Baudot teletypes, for RTTY (radioteletype) applications. They are more commonly referred to as 60, 67, and 75 words per minute.
**) This Baud rate is sometimes used for ASCII communications, and may also be used for 5-bit Baudot RTTY. The latter is commonly referred to as 100 wpm.
***) This Baud rate is used by IBM systems.

TABLE II: WORD SIZE SPECIFIERS TO ADD TO AUX1

ADD	Word Size
0	8 bits
16	7 bits
32	6 bits
48	5 bits

TABLE III: SPECIFIER FOR 2 STOP BITS TO ADD TO AUX1

ADD	Stop bits send with each word
0	1
128	2

TABLE IV: AUX2 SPECIFICATION TO MONITOR DSR, CTS, CRX

ADD	TO MONITOR
0	None
1	CRX
2	CTS
3	CTS, CRX
4	DSR
5	DSR, CRX
6	DSR, CTS
7	DSR, CTS, CRX

Note that the default (pre-set) values of Aux1 and Aux2 for all four ports are zero, corresponding to 300 Baud, 8-bit words, one stop bit transmitted, and no checking of DSN CTS or CRX.

You should know the following things about this command:

The configured parameters will stay as you set them until you either reset them or until you reboot the system (turn the power off and back on). The SYSTEM RESET key will NOT reset any of these parameters.
You may configure each RS-232-C port independently.

If you specify 8-bit words, there are no restrictions on operationn of the port. However, the following restrictions apply to 7-, 6-, and 5-bit words: half-duplex operation only, some limitations on baud rates. Specifically, ail output baud rates are allowed in Block Output mode. In Concurrent Mode, either in or out, you are limited to operation at 300 Baud and below. If you specify 7-, 6-, or 5-bit words, there is no restriction on the number of stop bits you may specify.

Note that most applications of these word sizes will probably be to devices that require more than 1 stop bit - you should specify two.

If you specify 7-, 6-, or 5-bit words, each word sent or received will be converted from or to an 8-bit byte within the computer by ignoring the most significant bit(s). This will very likely interact with the translation operation, and in particular there may be no way you can receive an EOL. If this is the case, you cannot use the BASIC INPUT statement to read the port and you must retrieve characters one at a time using GER More details will be found in the section on translation modes. (APPENDIX 6)

If you specify that you want the Interface Module to check DSN, CTS and/or CRX, it will check them whenever you try to start concurrent mode I/O and whenever you try to send a block of data in block output mode. If any of the lines you asked to be checked is not ready (OFF), then the concurrent mode I/O will not be started or the block of data will not be senr The Interface Module will then return an error to BASIC, and you may TRAP the error and take connective action. Following the TRAP, you may perform a STATUS request from the Interface Module ta find out what the problem was.

Note that CTX and CRX are not supported on ports 2, 3, and 4, and that DSR is not on port 4. The Interface Module behaves as if they were really there, howeven and acts as if they were always ready (ON).

You may look at the states of DSN CTS and CRX at any time that concurrent mode I/O is not active (if you have a channel open to the port) by issuing a STATUS request for the port. Thus, enabling this automatic checking of these lines is not the only option available to you, and you may prefer checking them directly with STATUS.

APPENDIX 6

TRANSLATION AND PARITY HANDLING

The Interface Module handler can be configured to perform certain types of code conversions (translations) and do parity generating and checking for you. These two operations interact with each other. For this reason, they will be described together in this section. The various options you may select for each are even specified by executing the same command - CONFIGURE TRANSLATION AND PARITY.

There are three factors you need ta keep in mind when you are setting up your code translations. Translation, of course, is one of them, since it results in (possibly) changing one code into another. Parity generation and checking also may result in changing one code into another. The third factor you need to keep in mind is the word size you are transmitting/receiving. Inside the computer all words are the same as bytes: that is, all words are 8-bits. If you are sendinglreceiving 7-, 6- or 5-bit words, these shorter words have to come from 8-bit computer words by chopping out some bits, or expanded into 8-bit computer words by adding some bits. These operations obviously are the same as changing one code into another.

Each of these three possible code changes takes place separately from the others, one at a time. For output, translation comes first, followed by parity generation, and finally truncation (shortening by leaving bits off). Of course, at each stage a change may not occur, depending on what selection of options you have configured and depending on which character (code) the computer is sending. For example, if you have configured 8-bit words, the trunction operation does nothing. For input, the order of code changing is expansion (from short words to 8-bit words), followed by parity checking, and finally translation.

At each of the three stages, a code change may occur. If a change DOES occur then it is the CHANGED code which will be operated on in the next stage. For example, (in a particular configuration of translation and parity options) if you output an ATASCII EOL, it would first be translated to an ASCII CR and then parity would be generated for the CR. This is because the parity step operates on the result of the translation step, in this case the CR.

There is one other translation option which is very specific, namely, the option to have an ASCII LF (line feed) sent after each transmitted CR (Carriage Return). This code change occurs at the translation step. Consequently, the generated LF will go through the parity and truncation (small word) phases just like the CR.

TYPES OF CODE TRANSLATION

You have three options to choose from: no translation at all, "light" translation, or "heavy" translation. Whichever option you choose will apply both to incoming and outgoing characters. The "no translation" option is just what it says - no change is made to the characters, whether being received or senr This statement applies only to the translation step, of course - you can still get changes from parity and small words. The no-translate option is useful if you are going to do your own special processing on the characters you are sending and receiving. This can be particularly useful in the small-word situations, since many of the cases where small words are used do not (or cannot) involve ASCII. You may also want to use the no-translation option if the RS-232-C compatible device you are communicating with understands ATASCII.

No matter which translation option you choose, if you use a BASIC INPUT statement to read data the data must have an ATASCII EOL (End of Line) character at the end of each line. This requirement applies AFTER all translation. Thus, if you select the no-translation option, your incoming data must either contain EOL's or you should use GET instead of INPUR Remember also, that using short words and that checking parity also affect data coming in, so you may still need to use GER

Heavy and light translation are two ways to convert between ASCII and ATASCII. In either translation mode, the ATASCII EOL (9B in hexadecimal, 155 in decimal) is converted to and from the ASCII CR (0D in hex, 13 in decimal). In the case of output, EOL is changed to CR; if you also selected the Append LF option, EOL is changed to CR followed by LF, that is, the translation function produces two characters out for one in. On input, a CR will be translated to EOL. Both Heavy and Light translation modes assume ASCII in the outside world and they assume ATASCII in the computer. ASCII is treated as a 7-bit code, that is, the 8th (most significanta bit is always treated as if it is zero. On input, then, if you select Heavy or Light translation, the 8th bit of each word is cleared to zero. On output, the translation step will set this bit to zero.

Light translation performs the fewest changes between ASCII and ATASCII. The assumption is that you wish to work with ATASCII within the computer but treat it as if it were really ASCII. Note, for example, that the ATASCII graphics codes are the same (numerically, and for the most part the way you type them, too) as the ASCII control codes (1-26). So for input, the character has its high bit stripped (set to zero), and that's all-except if the code is found to be a CR it is changed to an EOL. For output, if the character being sent is EOL it is changed to CR; then, no matter what the character is, the high bit is set to zero. Light translation is the pre-set default mode.

Heavy translation is a more thorough translation mode. Here the assumption is that if there is no direct correspondence between the character in ASCII and ATASCII, then the code should not be translated. So for input, after the high bit is cleared to zero, if the character is CR it is changed to EOL, otherwise, the character is checked to see if it is the same in ATASCII as in ASCII. If it is not, it is translated to the WON'T TRANSLATE character. Specifically, if the code for the ASCII character is less than 32 decimal (i.e., the character is a control character) or greater than 124 decimal (7C hex) it will be translated to the "won't translate" character. Thus, heavily translated ASCII corresponds to the printable characters from blank through vertical bar. The "won't translate" character is specified by you in the CONFIGURE TRANSLATION command. If you do not specify it, the pre-set default value for it is zero (ATASCII graphic heart).

On output, heavy translation converts EOL to CR, and will output any character whose ASCII meaning is the same as it is in ATASCII. That is, characters whose values range from 0-31 decimal (ASCII control values) or whose values are above 124 decimal (7C hex) will not be senr Note that characters whose high bit is one will be translated to nothing, that is, characters which would show on the TV screen as INVERSE VIDEO WILL NOT BE SENT in heavy translation mode. Note also the difference between input and autput in the heavy translation mode: untranslatable characters in the input are converted to the "won't translate" value, where untranslatable output simply is not sent our

The (optionali sending of LF after CR is produced in the translation step. If you specify no translatian, the option of adding LF to CR is not available. If you specify light translation, LF will follow EOL (which of course becomes CR). Note that sending the 13 decimal code (CR) by itself EOL will be turned into a CR/LF pair (with the append LF feature turned on). Each character in the CR/LF pair is independently sent through the parity and word-shortening steps on its way out. The pre-set default setting of the append LF feature is OFF, that is, the default is to NOT append the LF.

PARITY

You may select input and output parity handling separately. Thus, you may choose to send, for example, even parity while you ignore the parity of what you are receiving. The parity is always the most significant bit of each 8-bit byte (bit number 7). Thus, this parity operation is not applicable if you are working with 7-, 6-, or 5-bit words.

In the default parity condition, the parity bit of neither input nor output is altered. Note, however, that the parity bit of out-going messages may have been changed during the translation step.

For output, you may select even parity, add parity, set parity bit or no parity.

For input, your choices are "don't touch", check even, check odd, and "don't check". Each of these last three options will clear the top bit to zero, whether or not a parity check is made. If an input parity error is found, the character will still be input as if it were all right; thF parity error flag will be turned on in the status bytes (see STATUS REQUEST).

SHORT-WORD CONVERSION

The third operation which affects your code translation is the short-word conversion (if you are using 8-bit words, this is a "no-effect" operation). Short words sent out are made from 8-bit computer characters by omitting the most significant bits. That is, a 7-bit word is bits 0-6 of the character a 6-bit word is bits 0-5, and a 5-bit word is bits 0-4. Thus the parity, if generated, is lost. ASCII is a 7-bit code; you can send ASCII in 7-bit form without parity (this is not common practice, though-usually 8 bits are sent even if the 8th bit is not used for parity). With 6-bit and 5-bit codes, you will not be using ASCII, so you will have to concern yourself with the codes you want to be sending. With these word sizes, you should turn translation off so the translation performed by the Interface Module handler will not affect the codes you are working with.

On input, small words are converted to 8-bit computer characters by adding high-order bits. These added bits are always set to 1. Thus, if you are receiving 7-bit ASCII, the parity and translation steps will be getting ASCII with the 8th bit set high. If you are receiving 6- or 5-bit codes, there is no way you can receive the 13 decimal (OD hex) code (ASCII CR) - after all. you cannot receive ASCII in 6 or 5 bits anyway. This means that in BASIC you will have to use the GET statemenr not INPUT. Of course, you will be doing your own code conversion, so you should turn off the conversions of the Interface Module handler.

The CONFIGURE TRANSLATION MODE command is specified in BASIC this way:

```
XIO 38, #channel, Auxi, Aux2, "Rn: "
```

38 specifies the CONFIGURE TRANSLATION MODE command.

#Channel specifies the channel number (IOCB number from 1 to 7) you wish to use to configure the translation mode. You may use an unopen channel if you have no channel open to the port you are configuring, otherwise you must use the channel you have opened to that port. You cannot issue the CONFIGURE TRANSLATION MODE command if any concurrent mode I/O is active.

Aux1 specifies the translation mode, the input parity mode, the output parity mode, and the Append LF option. You specify these options by adding numbers taken from tables I, II, III and IV.

You may add the numbers yourself and put the sum in your program for Aux1, or you may let BASIC add them for you (e.g., you can say either 2+8+32 or 42 to mean even parity in, even parity out and no translation). Do not add in more than one value from each table.

Aux2 is the numeric representation of the "won't translate" character for heavy translation. Remember that the BASIC function ASC will give you the numeric representation of a character. For example, 41 and ASC("A") mean the same number. The number you specify should be from 0 through 255.

"Rn: " specifies the port you are configuring. For n, you put 1, 2, 3, or 4. You may omit n, which will mean you are configuring port 1. The default configuration is Aux1=0 and Aux2=4. If you execute the CONFIGURE TRANSLATION MODE command for one of the RS-232-C ports, that configuration will remain in effect until you do another CONFIGURE TRANSLATION MODE for that port. SYSTEM RESET will not change the translation mode for any port. Of course, you can configure each port a different way.

TABLE I: TRANSLATION MODE OPTIONS ADDED TO AUX1

ADD	To Get
0	Light ATASCII/ASCII translation
16	Heavy ATASCII/ASCII translation
32	No translation

TABLE II: INPUT PARITY MODE OPTIONS ADDED TO AUX1

ADD	To Get
0	Ignore and do not change parity bit
4	Check for odd parity; clear parity bit
8	Check for even parity; clear parity bit
12	Do not check parity but clear parity bit

TABLE III: OUTPUT PARITY MODE OPTIONS ADDED TO AUX1

ADD	To Get
0	Do not change parity bit
1	Set output parity odd
2	Set output parity even
3	Set parity bit to 1

TABLE IV: APPEND LINE FEED OPTIONS ADDED TO AUX1

ADD	To Get
0	Do not append LF
64	Append LF after CR (translated from EOL)

APPENDIX 7

CONTROLLING THE OUTGOING LINES - DTR, RTS and XMT

There are up to three outgoing RS-232-C signals on each of the RS232 ports of the Interface Module: Data Terminal Ready (DTR), Request To Send (RTS), and Data Transmit (XMT). Each of these lines can be turned ON or OFF with the CONTROL command.

Port 1 supports all three outputs. Ports 2 and 3 have DTR and XMT, port 4 has only XMT. You may use this command the same way with any port - it is not an error to try to control a line that does not exist. Your attempt will simply have no effect.

You may control any or all of these lines on a single RS-232-C port with the CONTROL command (controlling lines on other ports requires one CONTROL command for each port). The CONTROL command may be issued to a port which is not OPEN through an I/O channel by specifying any unopen channel number in the CONTROL command. If the port has been opened through a channel, you must use that channel in the CONTROL command. You may not issue a CONTROL command if any concurrent mode I/O is active.

Controlling XMT line has very limited use and few users will be concerned with it. In its normal state XMT is passive. If you change XMT you are likely to interfere with the normal transmission of data. In the serial communication world the only practical use of control of the XMT line is to send a BREAK signal. The BREAK is simply a period of holding the XMT line out of its normal resting state. Specifically, the normal resting state is called MARK, which corresponds to the binary "1" state. A BREAK is a period of the state called SPACE, which corresponds to binary "0". (Actually, since MARK and SPACE are the only legal states of any RS-232-C signal, all data consists of alternating MARKS and SPACES. What distinguishes BREAK from other uses of SPACE is that a BREAK is a SPACE which is a lot longer in duration than the time that a transmitted word would be. This is so because any transmitted word ALWAYS has one or more MARK bits in it - in particular each word ends with one or more stop bits represented by MARK). Thus to send a BREAK, first issue a CONTROL command to set the XMT line to SAPCE tD), then a little while later issue a control to set it back to MARK (1).

The uses of the other lines will depend on your application. For some guidelines, see APPENDIX 1.

The pre-set default state of the DTR and RTS lines is OFF. The pre-set default state of the XMT line is MARK. Once you change any of them with the CONTROL command, the new setting will remain until you either turn the computer off or issue another CONTROL command to change things. The SYSTEM RESET key has no effect on these lines.

The form of the CONTROL command in BASIC is:

XIO 34, #channel, Aux1, Aux2, "Rn: "

34 specifies the CONTROL command.

#channel specifies the IOCB or channel number (1-7) you wish to use for the command. If no channel is open to the RS-232-C port. specify an unused channel. If the port is open through a channel, use that channel.

Aux1 is the sum of three numbers chosen from tables I, II and III to control DTR, RTS and XMT. Choose only one number from each table. You may add the numbers together yourself and put the resulting sum in your program for Aux1, or you may put an expression for the sum and let BASIC do the arithmetic for you.

Aux2 is not used by this command: the best value to specify is zero.

"Rn:" specifies the RS-232-C port you are acting on. For n you put 1, 2, 3 or 4. If you omit n, the Interface Module handler will assume you mean port 1.

TABLE I: CONTROL VALUES FOR DTR ADDED TO AUX1

ADD	To Get
0	No change from current DTR setting
128	Turn DTR OFF
192	Turn DTR ON

TABLE II: CONTROL VALUES FOR RTS ADDED TO AUX1

ADD	To Get
0	No change from current RTS setting
32	Turn RTS OFF
48	Turn RTS ON

TADLE III--CONTROL VALUES FOR XMT ADDED TO AUX1

ADD	To Get
0	No change from current XMT setting
2	Set XMT to SPACE (0)
3	Set XMT to MARK (1)

APPENDIX 8

STARTING CONCURRENT I/O MODE

Use the command START CONCURRENT I/O (XIO 40) to start concurrent I/O mode. This mode may be used for output and must be used for input or full duplex. The port must be open before you can start concurrent I/O. Once concurrent I/O is in effect no other I/O operations which use the computer I/O connector can be performed. I/O operation to another serial port, for example, can not be performed. I/O to the keyboard, the screen, the Editor and the controller jacks can still be performed.

The concurrent mode I/O operation may be terminated by SYSTEM RESET, BREAK, or by closing the port.

Operations which are allowed while concurrent mode I/O is active are input and output operations to the active port (GET, INPUT, PUT, PRINT), and STATUS commands to that port.

There are two different forms of the START CONCURRENT MODE I/O command. The main difference between them is that one specifies the use of a small input buffer built into the Interface Module handler (in the computer), and the other allows you to give your own buffer to the handler so it can be any size you wish. (NOTE: in Assembly Language these two options are realy just different forms of the same command).

The form of the START CONCURRENT MODE I/O command which allows you to specify your own I/O buffer has two disadvantages: the command is complicated to specify in this form, and the BASIC array you use as the buffer may be moved by the BASIC interpreter. Once created, BASIC arrays are NOT moved while a program is being run, but arrays are moved whenever you add or delete a BASIC statement, even in immediate mode. The handler for the Interface Module is told of the location of the buffer only when you start the comcurrent I/O; thus, if you allow BASIC to move the array, data will be inserted in unpredictable locations, possibly destroying even the BASIC program itself. Ongoing concurrent input could wind up in other arrays or variables, or even in your BASIC program! SO REMEMBER: IF A PROGRAM IS USING CONCURRENT MODE INPUT ALWAYS MAKE SURE THE CONCURRENT MODE OPERATION IS STOPPED WHEN YOUR PROGRAM STOPS. This will be done for you if you stop by using BREAK key, SYSTEM RESET key, or end your program with END or letting the program stop by "running off the end."

STOP does not terminate the concurrent input, and neither will it be stopped if an ERROR happens. IN THESE CASES, THE WAY TO STOP THE CONCURRENT I/O IS TO PRESS THE BREAK KEY.

None of these problems occur if you use the buffer which is built into the Interface Module handler since that buffer does not move!

On the other hand, that buffer is quite small (32 bytes) and this may not be adequate for all programs.

With a small input buffer you need to GET or INPUT the data from the buffer before the buffer fills up with data that you have not yet read. Of course, if in the long-range average you read the data out of the buffer more slowly than it is arriving, you will eventually lose data anyway. If this is the case, you will either have to put up with losing it (which is not all that bad in some cases), or you will have to figure out a way to slow down the device that is sending the data to you (such as setting a lower Baud rate). Even if your program processes the data fast enough in the long run, a small buffer puts demands on your program to get data quickly and often. Here are some things to consider.

The BASIC interpreter is quite slow relative to incoming data, if you want to do some processing on each and every character that comes in. In that case, 300 Baud would be fast. On the other hand, the system is more than fast enough to read in a line of data (terminated by CR) at 9600 Baud (960 cps) - as long as there is enough time between lines for your program to do its processing. It pays to read a whole line of input at a time (use INPUT wherever possible instead of GET), and it's really helpful if the inputting device will pause for you after each line. Even if the inputting device will not pause, reading a line at a time may buy you the processing time you need. The best thing to do is try it.

NOTE: In order to perform line-oriented input using the BASIC INPUT statement, the input must either have an ATASCII EOL at the end of each input line, or must have an ASCII CR terminate each line. In the latter case, you must configure the translation mode of the Interface Module port to convert the CR into EOL. This is discussed more fully in the section on configuring translation mode.

A large input buffer will be needed if you can read the data from the buffer only in large, occasional bursts. For example, if you do not know how long it will take to process a line of input because some lines require a lot of work, you will want to allow lines to "back up" in the input buffer. This will work fine as long as you do not get too many of these "slow" lines at once. You will probably have to determine the needed size of your input buffer by trial.

The number of characters that can come in every second depends on the Baud rate - the higher the Baud rate the faster characters can arrive. Thirty characters may arrive each second at 300 Baud, 480 may arrive in the same time at 4800 Baud. Of course, if the sending device does not run at the maximum possible speed - if there are "gaps" between characters anywhere - then the speed of characters will not be ???. Thus the Baud rate can control the MAXIMUM data transfer rate, but the actual or EFFECTIVE data transfer rate may be smaller.

What things boil down to is that your program in BASIC must INPUT data from the input buffer faster than the Interface Module puts them there from your RS-232-C compatible device; that is, your BASIC program must read the data faster than your device's effective data transmission rate (on average). You can control that rate by setting the Baud rate, and possibly there are other ways to control the transfer rate (that depends on the device itself). Be prepared to experiment to find the best mode of operation.

In BASIC, the START CONCURRENT MODE I/O operation which uses the built-in input buffer looks like this:

```
XIO 40, #channel, 0, 0, "Rn: "
```

Specify the appropriate open channel, and specify 1, 2, 3 or 4 for n in "Rn: ". If you leave n out (i. e. , "R: "), then port 1 is assumed. You MUST specify zero for both Aux1 and Aux2, since this is the way you tell the RS-232-C handler to use its own input buffer.

If you opened the port for output only, then only concurrent output is enabled. If the port is open for input only, then only concurrent input is started. If the port was opened for both, then concurrent mode input and output are started (full duplex). See the section about the input and output commands for details on how these various modes operate.

In BASIC the START CONCURRENT MODE I/O operation in which you supply the input buffer for the handler is specified by a series of POKEs followed by calling the Central I/O (CIO) through a USR function. The POKEs specify the type of operation, and specify the buffer address and length. You POKE these values into the I/O Control Block (IOCB) corresponding to the channel you have opened for the RS-232-C port.
Here is an example program:

```
10 DIM BUF$(500), RSTART$(7)
20 LET RSTART$ = "hhh*LVd"
25 REM NOTE: a underlined character in line above means inverse
       video.
30 LET FILE = 2
40 OPEN #FILE, 13, 0, "R4: "
50 LET IOCB = 16*FILE
60 LET BUF = ADR(BUF$)
65 LET BUFLEN = 500
70 LET RSTART = ADR(RSTART$)
80 POKE 832+IOCB+2, 40
90 POKE 832+IOCB+4, BUF-(INT(BUF/256)*256)
100 POKE 832+IOCB+5, INT(BUF/256)
110 POKE 832+IOCB+8, BUFLEN-(INT(BUFLEN/256)*256)
120 POKE 832+IOCB+9, INT(BUFLEN/256)
125 POKE 832+IOCB+10, 13
130 DUMMY = USR(RSTART,IOCB)
140 STARTSTATUS = PEEK(832+IOCB+3)
```

In this program, a full duplex file is opened through channel 2 to RS-232-C port number 4 (the 13 in line 40 specifies full duplex). Lines 50 through 70 set up some values which are used by the START CONCURRENT MODE I/O operation. The buffer is setup in lines 80 through 130. Line 140 gets the status value returned by the I/O call. Each POKE statement puts some needed value into the I/O Control Block (IOCB). The address to poke is specified as the sum of the following: the first address of the IOCB's (832), a value specifying which IOCB, and an "offset" into the IOCB for the particular value you are POKEing. The value specifying the IOCB is 16 times the channel number through which you have opened the RS-232-C port (in this case we set the variable IOCB to 32 in line 50, since the channel is 2).

The values poked into the IOCB are: 40 into offset 2, the buffer location (address) into locations 4 and 5: the buffer length into offsets 8 and 9, and 13 into offset 10. Pay special attention to the fact that the buffer address and the buffer length are both 2-byte values, requiring two POKEs to put them into the IOCB. Those complex-looking expressions in lines 90 through 120 are simply splitting the address and length into their low-part and high-part so each part can be POKEd individually.

Line 130 calls the I/O system through a USR function. This USR function has two arguments: the address of the function, and the IOCB specifier (the same as was used in specifying the POKE locations). The address of this USR function was found in line 70, so you see that the function is the character array called RSTART$. The function itself is the odd-looking sequence of characters in line 20. Be sure to type this character sequence carefully when before gou call this USR function - any mistakes and your program will probably produce an unrecoverable failure.

Assembler note: This USR function is the following in Assembly Language: PLA, PLA, PLA, TAX, JMP ,E456. The first four instructions get the IOCB number into the X register, and the return address is on the stack, so the I/O system is "called" by jumping to it!).

Line 140 gets the I/O status after the USR I/O call. You do not need to get the status if you do not want to. To get status PEEK at offset 3 in the IOCB. The status will be 1 if all went well. Otherwise, the status is the same as the error number that BASIC prints after an I/O call fails. (Note that the variable DUMMY in the program above does not get any meaningful value).

Once this START CONCURRENT MODE I/O operation has been performed, the concurrent I/O is active. The operation may be either in-only, or it may be full-duplex (as specified in the OPEN). If you are running full-duplex, the output buffer is built into the Interface Module handler. The input and output buffers are accessed through normal input and output statements in BASIC; see the section on input and output statements.

Once again, take note: BASIC MAY MOVE ARRAYS AROUND IF YOUR PROGRAM STOPS. IF THE CONCURRENT MODE INPUT CONTINUES AFTER YOUR PROGRAM STOPS, THIS MAY RESULT IN OVERWRITING SOMETHING OUTSIDE YOUR BUFFER ARRAY. IF YOU ARE NOT SURE WHETHER OR NOT THE CONCURRENT I/O HAS STOPPED, PRESS THE BREAK KEY TO STOP IT.

NOTE: there is a 256 byte area at address 1536 (decimal) which you may use as an input buffer or anything else. Be sure that area is only being used for the one thing you wish. No ATARI software uses this area except just after you turn the machine on, but you should be careful of non-ATARI software you use with BASIC. 1536 splits nicely into low- and high-parts (so does 256), so you could replace lines 90 through 120 of the above program:

```
90  POKE 832+IOCB+4, 0
100 P0KE 832+IOCB+5, 6
110 POKE 832+IOCB+8, 0
120 POKE 832+IOCB+9, 1
```

If you use this area, you do not need to worry about it when your program stops since BASIC will not move it.

APPENDIX 9

USER PROGRAMS

1. PROGRAMS TO TRANSFER BASIC SOURCE PROGRAMS FROM COMPUTER TO COMPUTER

Here is a pair of programs which you can use to transfer information from one ATARI 800 computer to another over the telephone. These two programs demonstrate an example of a technique called "handshaking". Handshaking is a rather over used term in the computer world, what we mean here is that the receiving program will respond to the sender with an "I've got it!" message of some sort when it has successfully received each line of information from the sender.

The trick here is that the sending program must not miss the "I've got it!" message; likewise, the receiving program must not only have got the line when it says "I've got it!", but the receiver must be ready to receive the next line immediately because, theoretically, the sender might send the next line immediately. These programs show how these things are done.

Both programs operate on one line (up to 255 characters) at a time. Each program starts by DIMensioning its line-array, and each asks its user for the filename to be sent/received. Each program then opens its modem port (R1:) and disk file (assuming the send/receive files are disk files).

The SEND program is started first. In line 540, the SEND program gets a line from the disk file. The program then prints the line on the TV screen (so you can watch the data being sent). Then (lines 570--590) the line is sent over the phone. Note that port R1: is opened full duplex: SEND assumes when it receives the line,that RECEIVE might reply IMMEDIATELY. In line 600, SEND waits for the reply (In this case, a line which is empty except for an EOL is used for the reply).

The RECEIVE routine, meanwhile, has set itself up to get a line from the modem (lines 280-290, 530). When line 530 completes (the line of data has been received), RECEIVE CLOSEs the modem port (R1:) in order to save the data on the disk (lines 540, 580) and echoes the data on the TV (line 590). Then RECEIVE OPENs the modem again and sends the reply (lines 610-630). Note that port R1: is opened full-duplex: RECEIVE assumes that it might start getting the next line IMMEDIATELY after it has sent its reply. Note also that it is not necessary for RECEIVE to INPUT the data immediately, but it is necessary that RECEIVE have started the concurrent-mode data receive (line 620).

When SEND gets the reply, it knows it can safely CLOSE the modem port (R1:) to get another line of data from its disk (lines 600-610). It then goes back to get another line of data (lines 530-540) and the whole cycle repeats. Note how the SEND program checks for the end of the disk file and how it sends a specially encoded line (EOF EOF EOF) to the RECEIVE program to signal this. Note that both programs explicitly CLOSE their files.

To use these programs, assume you and your friend are talking on the phone and you've prepared your computers (you have loaded your SEND progran, and your friend has the RECEIVE program). You each RUN your programs, and each program gets a filename from each of you-type the name but DON'T YET type RETURN. Now one of you sets his modem in ANSWER mode and the other sets his ORIGINATE. Looking at your watches, you decide that your friend will type his RETURN as soon as the READY light comes up on his modem and you will type your RETURN ten seconds later. In other words, the RECEIVE program must be ready to receive before the SEND program sends the first line! Now you each put your phone handsets in the modem cradles and you proceed to send a program to your friend.

Since these programs work on LINES of data, you cannot send tokenized BASIC. You should send BASIC source, that is, send a file you saved on the disk with the LIST (not SAVE) command. Your friend should ENTER the file he receives (not LOAD). You may modify these programs to send and receive the information one character at a time (using GET and PUT instead of PRINT and INPUT), doing the handshake every 40 characters or so. You'll have to pay particular attention to the question of sending the end-of-file information if you try this modification; however, such a modification should allow you to send any kind of data, not just lines of text.

Note that the RECEIVE program will probably need modification if you intend to put the received information on cassette. This is because the cassette handler requires that the first 128-byte record be written within about 30 seconds after you OPEN the cassette for output. A little experimentation should get you going.

SEND PROGRAM

```
110 DIM OUTLINE$(255)
200 REM
201 REM ==========
202 REM
210 LET TRANSLATE=32:REM [Full ATASCI]
220 XIO 38,#2,TRANSLATE,0,"R1:"
230 REM
240 PRINT "Send file's full name";
250 INPUT OUTLINE$
2b0 OPEN #1,4,0,OUTLINE$
500 REM
501 REM ==========
502 REM
510 FOR ETERNITY=1 TO 2 STEP 0
520 REM
530 TRAP 900:REM [Trap end file #1]
540 INPUT #1,OUTLINE$:REM [Get line]
550 PRINT OUTLINE$: REM [Echo onscreen]
560 REM
570 OPEN #2, 13, 0, "R1:"
580 XIO 40,#2,0,0,"R1:": REM [Start I/O]
590 PRINT #2; OUTLINE$:REM [Send line]
600 INPUT #2; OUTLINE$: REM [Get reply]
610 CLOSE #2: REM [Stop I/O]
620 REM
630 NEXT ETERNITY
900 REM
901 REM==========
902 REM
910 OPEN #2,8,0,"R1:": REM [Send EOF]
920 PRINT #2;"EOF EOF EOF"
930 CLOSE #2:CLOSE #1:REM [All done]
999 END
```

RECEIVE PROGRAM

```
110 DIM INLINE$(255)
200 REM
201 REM==========
202 REM
210 LET TRANSLATE=32:REM [Full ATASCII]
220 XIO 38,#1,TRANSLATE,0,"R1:"
230 REM
240 PRINT "Receive file's full name";
250 INPUT INLINE$
260 OPEN #2,8,0,INLINE$
270 REM
280 OPEN #1,13,0,"R1:"
290 XIO 40,#1,0,0,"R1:": REM [Start I/O]
500 REM
501 REM==========
502 REM
510 FOR ETERNITY=1 TO 2 STEP 0
520 REM
530 INPUT #1,INLINE$:REM [Get line]
540 CLOSE #1:REM [Stop I/O]
550 REM
560 IF INLINE$="EOF EOF EOF" THEN 900
570 REM
580 PRINT #2,INLINE$:REM [Save line]
590 PRINT INLINE$:REM [Echo onscreen]
600 REM
610 OPEN #1,13,0,"R1:"
620 XIO 40,#1,0,0,"R1:": REM [Start I/O]
630 PRINT #1:REM [Send reply]
640 REM
650 NEXT ETERNITY
900 REM
901 REM==========
902 REM
910 CLOSE #2:REM [EOF received]
999 END
```

2. BAUDOT TERMINAL EMULATOR

Here is a sample program showing the use of odd character transmission sizes and non-ATASCII (also non-ASCII) character codes. This program turns your ATARI computer into a BAUDOT teletype emulator.

WARNING: The ATARI 850 Interface Module was not designed for connection to old teletype equipment. Such equipment used 60 milliamp current loops rather than the more modern 20 milliamps, and high voltages (could be present) in such old equipment (which is dangerous) and could damage your 850 Interface Module. This program is intended to allow you to communicate, via a modem, over a telephone or radio link with someone owning a BAUDOT teletype.

The Baudot code is an old 5-bit serial code which is actually two codes in one. Half of the characters in Baudot are in the LETTERS SHIFT category and half are in the NUMBERS SHIFT category. The latter category includes digits 0-9 and some special characters. This program takes care of sending and receiving the shifting control characters.

This program is actually much simpler than it looks. In lines 110-210, the program's "constants" and starting values are set up. The "constants" are values which are not changed in the program, but for readability they are represented symbolically (as variables), Constants include: logical constants (YES and NO), PEEK and POKE addresses (SWITCH, KB), character constants (RETURN, FEED, UPSHIFT, DOWMSHIFT); BASIC line number constants for GOSUB's and GOTO's (RECEIVE, SEND, and TESTSWITCH); and useful numbers (NOPUSH, NOKEY). Setting INSHIFT to zero establishes LETTERS SHIFT for received data; setting ALPHA to YES establishes LETTERS SHIFT for sent data; and setting TALK to NO establishes LISTEN mode.

Lines 300-390 fill in the ASCII-BAUDOT translation tables from the data values in lines 2000-2460. REMarks are interspersed in the data to show what character is being translated. Notice that all the characters are represented within this program as numbers-the number is the "internal" character code for the corresponding letter (this is true for both ATASCII and BAUDOT, but, of course, the numbers representing a particular letter are different for each).

In order to make the code conversion easy this translation mode is set to 32 - no translation. The Baud rate is set to 45.5 Baud (60 wpm). This is the most common speed for old Baudot equipment. It is also the slowest speed configurable with the 850.

Lines 500-650 are the receive routine. The computer informs you that you are entering Listen mode, then OPENs the RS-232-C port R3: for input and starts the concurrent mode input (510-540). The receive loop (560-650, first does a GOSUB TESTSWITCH to check for switching to send mode (TESTSWITCH is discussed later). The STATUS and IF

PEEK... statements (580-585) see if there are any characters received. If there are, a character is input in line 590 and translated to ASCII in lines 600-630, and PRINTed to the TV in line 640. ATASCII table values less than zero mean untranslatable characters: 0 means the LETTERS SHIFT character is received: 1 mean NUMBERS SHIFT.

Lines 700-950 are the send routine. Talk mode is announced. and port R3: is OPENed for output. The first send loop (750-950) action is a GOSUB TESTSWITCH. Line 770 checks for the typing of a keyboard key. In lines 780-800 the key's value is retrieved and its high bit is stripped (it is forced to be less than 128 - this has the effect of disregarding inverse video and allows the conversion to table to require only 128 elements). The key is translated in line 810; if it translated to zero, that means it has no Baudot equivalent and line 820 restarts the loop. Otherwise. it is echoed to the TV (830); it then undergoes further translation in lines 840-890, where a LETTERS or NUMBERS shift character is added if needed. Line 900 sends the character itself, and if it was RETURN, lines 920-930 add a LINEFEED and LETTERS SHIFT.

The TESTSWITCH routine (lines 1000-1060) checks whether one of the function keys is pushed (START, SELECT or OPTION). If not pushed, TESTSWITCH just RETURNs. Otherwise, the subroutine waits for the button to be released, restores BASIC's GOSUB/FOR-NEXT stack, flips from SEND to RECEIVE mode (or vice-versa) and does a GOTO to the proper routine.

In operation, the 32-character internal buffer fills with characters to be sent. When the buffer is full the Interface Module sends the characters as a block. While the characters are being sent, the keyboard will accept one character (which you won't see on the screen), so you should type the next character you want to send and wait for it to appear on the TV. Note that this program, as written, sends the block immediately when you type RETURN. You may want to experiment with variations, such as sending each character as it is typed, or reading a line at a time rather than a character at a time from the keyboard (this allows you to use backspace to correct you typing, but the person at the other end of the connection won't see anything except when you type RETURN). Have fun!

```
110 DIM ATASCII(64),BAUDOT(128)
120 REM
121 REM(Set up constants...)
122 REM -------------------------------
130 LET YES=1:N0=0
140 LET SWITCH=53279:NOPUSH=7
150 LET KB=764:NOKEY=255
160 LET RETURN=8:FEED=2
170 LET UPSHIFT=27:DOWNSHIFT=31
180 LET RECEIVE=500: SEND=700
190 LET TESTSWITCH=1000
200 REM
201 REM (Starting values...)
202 REM --------------------------------
210 LET INSHIFT=0:ALPHA=YES:TALK=NO
300 REM
301REM(Fill Baudot --> ATASCII table...)
302 REM --------------------------------
310 FOR I=1 TO 64
320 READ IN
330 LET ATASCII(I)=IN
340 NEXT I
350 REM
351REM (Fill ATASCII --> Baudot table...)
352 REM --------------------------------
360 FOR I=1 TO 128
370 READ IN
380 LET BAUDOT(I)=IN
390 NEXT I
400 REM
401 REM (Set up I/O...)
402 REM --------------------------------
410 LET BAUD=128+48+1:TRANSLATE=32
420 XIO 36, #2,BAUD,0,"R3:"
430 XIO 38, #2,TRANSLATE,0,"R3:"
440 REM
450 OPEN #1,4,0,"K:"
500 REM
501 REM (Receive routine...)
502 REM --------------------------------
510 PRINT: PRINT "Listen..."
520 REM
530 OPEN #2,5,0,"R3:": REM [Input]
540 XIO 40,#2,0,0,"R3:": REM [Start]
550 REM
551 REM (Receive loop...)
552 REM --------------------------------
560 FOR INLOOP=0 TO 0 STEP 0
570 GOSUB TESTSWITCH
580 STATUS #2,PORT4
585 IF PEEK(747)=0 THEN NEXT INLOOP
590 GET #2,IN
```

```
600 LET IN=ATASCII(IN-224+INSHIFT+1)
610 IF IN<0 THEN NEXT INLOOP
620 IF IN=0 THEN INSHIFT=0:NEXT INLOOP
630 IF IN=1 THEN INSHIFT=32:NEXT INLOOP
640 PRINT CHR$(IN);
650 NEXT INLOOP
700 REM
701 REM (Send routine...)
702 REM --------------------------------
710 PRINT:PRINT "Talk..."
720 REM
730 OPEN #2,8,0,"R3:": REM [Output]
740 REM
741 REM (Send loop...)
742 REM --------------------------------
750 FOR OUTLOOP=0 TO 0 STEP 0
760 GOSUB TESTSWITCH
770 IF PEEK(KB)=NOKEY THEN NEXT OUTLOOP
780 GET #1,KEY
79G LET OUT=KEY
800 IF OUT>127 THEN LET OUT=OUT-128
810 LET OUT=BAUDOT(OUT+1)
820 IF OUT=0 THEN NEXT OUTLOOP
830 PRINT CHR$(KEY)
840 IF ALPHA THEN 880
850 IF OU<0 THEN 900
860 LET ALPHA=YES: PUT #2,DOWNSHIFT
870 GOTO 900
880 IF OUT>0 THEN 900
890 LET ALPHA=NO: PUT #2,UPSHIFT
900 PUT #2,ABS(OUT)
910 IF OUT<=>RETURN THEN NEXT OUTLOOP
920 PUT #2,FEED: PUT #2,DOWNSHIFT
930 XI0 32,#2,0,0,"R3:"
940 LET ALPHA=YES
950 NEXT OUTLOOP
1000 REM
1001 REM (Listen/Talk switch test...)
1002 REM --------------------------------
1010 IF PEEK(SWITCH)=NOPUSH THEN RETURN
1020 IF PEEK(SWITCH)<>NOPUSH THEN 1020
1030 POP: POP: REM [Pop GOSUB & FOR-loop]
1040 CLOSE #2
1050 IF TALK THEN TALK=NO:GOTO RECEIVE
1060 LET TALK=YES:GOTO SEND
2000 REM
2001 REM (Baudot to ATASCII table...)
2002 REM --------------------------------
2010 REM [NUL,E,LINEFEED,A,SPACE,S,I,U]
2020 DATA -1,69,-1,65,32,83,73,85
2030 REM [RETURN,D,R,J,N,F,C,K]
2040 DATA 155,68,82,74,78,70,67,75
```

```
2050 REM [T,Z,L,W,H,Y,P,Q]
2060 DATA 84,90,76,87,72,89,80,81
2070 REM [O,B,G,Numbers,M,X,V,Letters]
2080 DATA 79,66,71,1,77,88,86,0
2090 REM [NULL,3,LF,-,SPACE,BELL,8,7]
2100 DATA -1,51,-1,45,32,253,56,55
2110 REM [RETURN,$,4,',COMMA,!,:,(]
2120 DATA 155,36,52,39,44,33,58,40
2130 REM [5,",),2,#,6,0,1]
2140 DATA 53,34,41,50,35,54,48,49
2150 REM [9,?,+,Numbers,.,/,;,Letters]
2160 DATA 57,63,43,1,46,47,59,0
2200 REM
2201 REM (ATASCII to Baudot table...)
2202 REM ------------------------------------
2210 REM [Graphics characters incl. CR]
2220 DATA 0,0,0,0,0,0,0,0
2220 DATA 0,0,0,0,0,0,0,0
2220 DATA 0,0,0,0,0,0,0,0
2220 DATA 0,0,0,8,0,0,0,0
2260 REM [SPACE,!,",#,$,%,&,']
2270 DATA 4,-45,-17,-20,-9,0,-26,-11
2280 REM [( ,),*,+,COMMA,-,.,/]
2290 DATA -15,-18,0,-26,-12,-3,-28,-29
2300 REM [0,1,2,3,4,5,6,7]
2310 DATA -22,-23,-19,-1,-10,-16,-21,-7
2320 REM [8,9,:,;,<,=,>,?]
233G DATA -6,-24,-14,-30,0,0,0,-25
2340 REM [@,A,B,C,D,E,F,G]
2350 DATA 0,3,25,14,9,1,45,26
2360 REM [H,I,J,K,L,M,N,O]
237G DATA 20,6,11,15,18,28,12,24
2380 REM [P,Q,R,S,T,U,V,W]
2390 DATA 22,23,10,5,16,7,30,19
2404 REM [X,Y,Z,Graphics characters]
2410 DATA 29,21,17,0,0,5,0,0
2420 REM [A-Z again]
2430 DATA 0,3,25,14,9,1,45,26
2440 DATA 20,6,11,15,18,28,12,24
2450 DATA 22,23,10 5,16,7,30,19
2460 DATA 29,21,17,0,0,5,0,0
9999 END
```

3.EXAMPLE OF PROGRAMMING A PRINTER THROUGH AN RS-232-C PORT

Here are two examples of programming printers connected serially through RS-232-C ports. It is assumed that there are fundamental differences between the two - the charactersitics of each printer control how that printer must be programmed. These two sample programs (or program fragments) are not intended to show general techniques, but are examples of how certain specific needs can be met.

The printer being programmed here is able to buffer and hold characters ahead of its printing (or it is so fast that it is always ready to accept characters to print). When it does not want you to send more data, it sets a READY line OFF, that line is connected here to the DSR pin on the RS-232-C port. Howeven the printer sets its READY line OFF early - it is still able to collect up to 32 characters after it says it's full. In other words, since the RS-232-C ports block data out in blocks of up to 32 characters, it is only necessary to monitor the DSR line once per block.

The automatic moryitoring of DSR once per block is set up in line 150. In line 160, we tell the Interface Module to add LF to each CR (this printer wants the LF).

When a block is about to be sent, the Interface Module checks DSR (per our request). If it is OFF, the resulting NAK error is TRAPped (line 360), and in the TRAP routine (900 etc.) the program checks that the TRAP was really caused by the DSR being OFF. If this was the cause, the PRINT is simply retried - eventually it will succeed because the printer will become ready again.

```
      :
      :
      :
140 OPEN #2,8,0,"R2:"
150 XIO 36,#2,0,4,"R2:": REM [Monitor DSR]
160 XIO 38,#2,64,0,"R2:": REM [Add LF to CR]
      :
      :
360 TRAP 900
370 PRINT #2, ..... :REM [PRINT something to R2:]
      :
      :
900 STATUS #2,PORT2: REM [Get R2: status]
910 LET READY=PEEK(746)/8: REM [Check readiness error]
920 IF INT(READY)<>INT(READY+0.5) THEN 360: REM [If so, retry]
930 REM [If here then some error other than port not ready]
      :
      :
      :
```

4.ANOTHER EXAMPLE OF PRINTER CONTROL THROUGH AN RS-232-C PORT

The printer being programmed in this example also has a READY line to signal that it is not ready to accept data. However, when it is not ready, it cannot accept any data. Therefore, the data must be sent to the printer one character at a time, checking DSR before each character. Since the PRINT statement cannot be made to send data one character at a time, we assume that the file to be printed was first written to a disk or cassette. Here is a program to read that file off the disk or cassette and print it on this printer.

The operation of this program should be fairly obvious. Once again, we assume the printer wants both CR and LF at the end of a line (lines 230-240). The file is read from disk (or tape) one character at a time. Then if the printer on port R2: is ready (540-550), the character is PUT (560). The output is then forced (FORCE SHORT BLOCK) in line 570.

```
110 DIM FILE$(16)
200 REM
201 REM ---------------------------
202 REM
210 LET BAUD=13: REM [4800 Baud]
220 XIO 36,#2,BAUD,"R2:"
230 LET TRANSLATE=64: REM [Add LF to CR]
240 XIO 38,#2,TRANSLATE,0,"R2:"
250 REM
260 PRINT "List file's full name";
270 INPUT FILE$
280 OPEN #1,4,0,FILE$
290 REM
300 OPEN #2,8,0,"R2:"
500 REM
501 REM ---------------------------
502 REM
510 FOR ETERNITY=0 TO 0 STEP 0
520 TRAP 900: REM [Trap end of file]
530 GET #1,CHARACTER
540 STATUS #2,XXX: REM [Check ready]
550 IF PEEK(747)<128 THEN 540
560 PUT #2,CHARACTER
570 XIO 32,#2,0,0,"R2:"
580 NEXT ETERNITY
900 REM
901 REM ---------------------------
902 REM
910 CLOSE #2: CLOSE #1
920 END
```

5. READING A DIGITIZER

MORE INPUT THAN BASIC CAN HANDLE

his is an example of reading data from a digitizing pad. A digitizing pad is a device which is capable of sensing the position of a hand-held object (a special pen or whatever) and reporting its location to the computer.

The digitizing pad used here is capable of sending its information to the computer at speeds up to 4800 baud, so that is used here. Each sampled pen position is 14 characters long: a digit indicating whether or not the button on the pen is being pushed, the x-coordinate (6 characters), the y-coordinate (6 characters), a CR and a LF. Since the LF follow, the CR the Interface Module will read it as the first character on the following input line.

If we assume that the digitizer sends the pen coordinates as fast as it can, then BASIC will not be able to keep up at 4800 Baud. A lower Baud Rate might allow BASIC to get every sample, but at 300 Baud, for example, it would take about HALF A SECOND for each sample to come in (15 characters at 30 cps)! Thus we want the data to come in at the highest possible rate. It really doesn't matter if we miss samples, because the pen is usually in pretty much the same place sample after sample.

Therefore, it is OK. if the digitizer sends samples as fast as it can and the program just grabs them now and then when it can. However, we have to take into account the way the Interface Module behaves when data arrives too fast: when the computer's holding buffer fills up, the NEWEST data replaces the OLDEST. However, an INPUT statement reads the OLDEST data - which is messed up by being replaced by the newer data!

Th is is actually very trivial to solve. Look at line 100. A sample is INPUT twice! The first INPUT gets the messed-up sample which has been written oven by new data. Then the second INPUT gets a sample from the buffer which is unharmed. (This works because the sample contains enough characters to allow an INPUT to get significantly ahead of the arriving character stream, and because the sample contains fewer characters than the holding buffer).

Lines 110-130 extract the coordinates from the sample. It was not possible to use an INPUT statement with these number variables because the sample does not have commas between the sample numbers. The details of what the program does with the samples is not shown (in order to keep the example to the important points).

```
10 DIM IN$(16)
20 XIO 36,#1,13,0,"R2:"
30 OPEN #1,5,0,"R2:"
40 XIO 40,#1,0,0,"R2:"
     :
     :
100 INPUT #1, IN$: INPUT #1,IN$
110 LET BUTTON=VAL(IN$(2,2))
120 LET X=VAL(IN$(3,8))
130 LET Y=VAL(IN$(9,14))
     :
     :
590 GOTO 100: REM [Get next point]
```

APPENDIX 10

CODE TABLES DECIMAL HEX

Decimal code	Hex. Code	ASCII Character	ATASCII character Function/Display
0	0	NUL	
1	1	SOH	
2	2	STX	
3	3	ETX	
4	4	EOT	
5	5	ENQ	
6	6	ACK	
7	7	BEL	
8	8	BS	
9	9	HT	
10	A	LF	
11	B	VT	
12	C	FF	
13	D	CR	
14	E	SO	
15	F	SI	
16	10	DLE	
17	11	DC1	
18	12	DC2	
19	13	DC3	
20	14	DC4	
21	15	NAK	
22	16	SYN	
23	17	ETB	
24	18	CAN	
25	19	EM	
26	1A	SUB	
27	1B	ESC	(KEY ESC ESC)
28	1C	FS	MOVE UP ONE LINE =^
29	1D	QS	MOVE DOWN ONE LINE =V
30	1E	RS	MOVE LEFT ONE SPACE =<
31	1F	US	MOVE RIGHT ONE SPACE =>
32	20	SP	SPACE
33	21	!	!
34	22	"	"
35	23	#	#
36	24	$	$
37	25	%	%
38	26	&	&
39	27	'	'
40	28	(	(
41	29	)	)
42	2A	*	*
43	2B	+	+

44	2C	,	,
45	2D	-	-
46	2E	.	.
47	2F	/	/
48	30	0	0
49	31	1	1
50	32	2	2
51	33	3	3
52	34	4	4
53	35	5	5
54	36	6	6
55	37	7	7
56	38	8	8
57	39	9	9
58	3A	:	:
59	3B	;	;
60	3C	<	<
61	3D	=	=
62	3E	>	>
63	3F	?	?
64	40	@	@
65	41	A	A
66	42	B	B
67	43	C	C
68	44	D	D
69	45	E	E
70	46	F	F
71	47	G	G
72	48	H	H
73	49	I	I
74	4A	J	J
75	4B	K	K
76	4C	L	L
77	4D	M	M
78	4E	N	N
79	4F	O	O
80	50	P	P
81	51	Q	Q
82	52	R	R
83	53	S	S
84	54	T	T
85	55	U	U
86	56	V	V
87	57	W	W
88	58	X	X
89	59	Y	Y
90	5A	Z	Z
91	5B	[	[
92	5C	\	\
93	5D	]	]
94	5E	^	^
95	5F	_	_
96	60	`	

97	61	a	a
98	62	b	b
99	63	c	c
100	64	d	d
101	65	e	e
102	66	f	f
103	67	g	g
104	68	h	h
105	69	i	i
106	6A	j	j
107	6B	k	k
108	6C	l	l
109	6D	m	m
110	6E	n	n
111	6F	o	o
112	70	p	p
113	71	q	q
114	72	r	r
115	73	s	s
116	74	t	t
117	75	u	u
118	76	v	v
119	77	w	w
120	78	x	x
121	79	y	y
122	7A	z	z
123	7B	{	
124	7C		
125	7D	}	CLEAR SCREEN
126	7E	~	BACKSPACE
127	7F	DEL	TAB (10 SPACES)

Below is a table of the most common Baudot code. All Baudot codes are identical for letters, numbers, and control characters, but they differ somewhat in punctuation. The DECIMAL VALUE column gives the 5-bit Baudot serial binary code converted to decimal. When transmitted, a start bit (space) precedes the character, the character itself is sent low bit first, and 1.5 or 2 stop bits (mark) follow. Mark is sent for 1, space for 0.

The hex/dec columns show the value of the Baudot character when interpreted as an 8-ibt word with the three high-order bits set to 1. These are the codes which represent the Baudot characters with the Interface Module's no-translation mode (translation mode 32).

Letters	Figures	Decimal Value	HEX/DEC
A	- (dash)	3	E3/227
B	?	25	F9/249
C	:	14	EE/238
D	$	9	E9/233
E	3	1	E1/225
F	!	13	ED/237
G	+ or &	26	FA/250
H	# or STOP	20	F4/244
I	8	6	E6/230
J	' (apost.)	11	EB/235
K	(	15	EF/239
L	)	18	F2/242
M	. (period)	28	FC/252
N	, (comma)	12	EC/236
O	9	24	F8/248
P	0 (zero)	22	F6/246
Q	1 (one)	23	F7/247
R	4	10	EA/234
S	BELL	5	E5/229
T	5	16	F0/240
U	7	7	E7/231
V	;	30	FE/254
W	2	19	F3/243
X	/	29	FD/253
Y	6	21	F5/245
Z	"	17	F1/241
NULL	NULL	0	E0/224
RETURN	RETURN	8	E8/232
LINEFEED	LINEFEED	2	E2/226
SPACE	SPACE	4	E4/228
LETTERS	LETTERS	31	FF/255
FIGURES	FIGURES	27	FB/251

APPENDIX 11

PRINCIPLES OF OPERATION OF THE ATARI 850 INTERFACE MODULE

The ATARI 850 Interface Module is a computer: it contains a microprocessor , built-in program in ROM, and extensive I/O capability. The I/O forms the parallel (printer), and serial (RS-232-C ports, and is also used for communication between the Interface Module and the ATARI 400 or ATARI 800 computer.

This section presents the theory of operation of the Interface Module. Topics include the automatic bootstrap function, operation of the RS-232-C port handler which is loaded by the bootstrap function into the 400 or 800 computer operation of the Interface Module to execute the RS-232-C I/O commands, and operation of the printer port. The electrical interfaces of the RS-232-C and printer ports are shown, and signal handshake and timing on the printer port are discussed.

POWER-ON BOOTSTRAPPING OPERATION

NOTE THAT THE BOOTSTRAPPING OPERATION IS REQUIRED ONLY FOR OPERATION OF THE RS-232-C (SERIAL) PORTS AND NOT THE PRINTER PORT. The ATARI 400 and 800 computers already contain the necessary programming to operate the printer port on the 850 Interface Module. The automatic power-on bootstrapping operation, when enabled, loads the 1762-byte handler and tables required for operation of the serial ports.

The bootstrapping operation is enabled by turning ON the power to the Interface Module before the 400 or 800 computer.

Without Disk Drive

When the ATARI Personal Computer System's power is turned on, it issues a disk requesT. If there is no Disk Drive in the system (or if the Disk Drive is OFF), the Interface Module responds to the disk request. The computer then loads a special bootstrapping program from the Interface Module, as if it were reading from a disk. The bootstrapping program is then run, and it gets the RS-232-C handler from the Interface Module and relocates it into the computer's RAM. The memory occupied by the bootstrapping program is then freed (but the handler remains}.

With Disk Drive

If there is a Disk Drive attached to the system (Drive 1 only), it responds to the disk request issued by the 400 or 800 computer at power-on. The computer then reads a start-up program from that disk. Most commonly, this program is an ATARI Disk Operating System (DOS). The Interface Module does not respond to the disk request if a Disk Drive responds first, therefore, the program loaded from the disk must load the handler from the Interface Module. In the DOS II, this job is handled by a special AUTORUN.SYS file supplied with your DOS II diskette.

The AUTORUN.SYS program is loaded and executed by the DOS, it finds the Interface Module and loads the bootstrapping program from it. The bootstrapping program then loads and relocates the RS-232-C handler from the Interface Module. Read the instructions supplied with your DOS II for details on AUTORUN.SYS.

PRINTER SOFTWARE OPERATION

The Interface Module responds to commands to an ATARI printer whenever it senses a printer attached to the parallel port (see the electrical section on the printer port for signal requirements between the Interface Module and a printer).

The ATARI 400/800 Operating System contains a printer handler program which will address one printer, called P: . Four commands are allowed by the P: handler: OPEN, CLOSE, output (represented by PUT, PRINT, and LIST in BASIC), and STATUS.

To use the printer, one must OPEN a channel (IOCB) to the printer. CLOSE releases the channel when it is no longer needed.

ATARI printers (including the Interface Module) operate in Block Output Mode (as described elsewhere in this manual for the RS-232-C port operation). The printer handler builds a 40-byte buffer, and when the buffer fills, the 40 bytes are sent to the printer. When a printer is attached to the Interface Module, the Interface Module accepts the 40 characters and sends them, one at a time, over the parallel port to the attached printer. The printer must acknowledge all 40 characters within 4 seconds (see the electrical section for a discussion of the handshake between the Interface Module and a printer).

There is one exception to the above description: When the printer handler is asked to print an ATASCII EOL (End-Of-Line) character, it fills any unused part of the 40-character buffer with blanks (following the EOL) and sends it immediately. For this reason, the Interface Module ignores any characters in the buffer which follow an EOL.

The Interface Module translates EOL into ASCII CR (Carriage Return, 13 decimal). No other translations are made. In particular, bit 7 (high bit) of each byte is not changed, and LF is not added following CR.
However. multiple EOL's in a row, without intervening characters, are sent to the printer as alternating CR's and blanks.

A special note about LPRINT in BASIC: LPRINT is equivalent to OPEN, PRINT and CLOSE all in one. Execution of an LPRINT statement with a comma or semicolon at the end will send to the Interface Module a 40-character buffer which is padded with blanks but does NOT have an EOL character. The Interface Module will send all 40 characters to the printer (including the blanks), but the printer will probably not respond because most printers wait for CR before activating a print cycle.

The STATUS request for device P: is answered by the Interface Module if there is a printer attached and powered ON. The status returned in location 746 (decimal) contains 128 if the previous operation to the printer was successful, 129 if the previous command to the Interface Module printer port was bad, 130 if the previous 40-byte data frame had an error (this should not happen); and 132 if the previous command timed out - that is, the printer stayed BUSY past 4 seconds.

RS-232-C PORTS SOFTWARE OPERATION

Once booted, the RS-232-C port handler is linked in as the R: device. This handler contains code to re-establish itself whenever a warm start (RESET) occurs.

The RS-232-C handler is called by CIO to execute each type of I/O operation for the R: device (except output calls from BASIC which bypass CIO by calling the RS-232-C handler directly). Some of the commands are executed entirely by the handler (set-up), but most are passed on to the Interface Module. Some commands cause set-up in both the handler and in the Interface Module.

The CONFIGURE BAUD RATE command is a set-up command which is executed by both the handler and the Interface Module. Both the handler and the Interface module keep separate tables for each of the four RS-232-C ports.

The SET TRANSLATION MODE command is executed by the handler. This command sets values which control the translation and parity handling during I/O.

The CONTROL command is executed by the Interface Module. Outgoing control lines for the indicated port are set ON (or MARK), set OFF (or SPACE), or left alone, as specified by the control parameter. Each line is left alone until another CONTROL command is executed. Note that, if the XMT line is set to SPACE, it will return to SPACE following any subsequent data transmission, until another CONTROL command sets it to MARK.

The OPEN command is executed entirely by the handler. It establishes control information for the port being OPENed. The CLOSE command is executed mostly by the handler: OPEN flags are cleared, any data in output buffers is sent, concurrent mode I/O is shut down. Any data in an input buffer is lost at CLOSE time.

Block mode output takes data from BASIC PRINT or PUT statements, puts each character through translation, and puts each character into the 32-byte output buffer. The buffer is transmitted when it fills, or when 13 (hex) is stored into the buffer (automatic short block on CR). Data from the buffer is sent to the Interface Module as 8-bit bytes. If 7-, 6-, or 5-bit words are configured, the

Interface Module strips the necessary number of high-order bits from each byte before transmitting it to the port. If monitoring of any external status line has been configured for the port, the readiness is checked by the Interface Module whenever a block is sent to it. If not ready, the Interface Module returns a NAK. The 400/800 computer waits while the Interface Module transmits a block.

The FORCE SHORT BLOCK command causes the handler to transmit the block of data before 32 bytes have been collected. If there is no data in the buffer, the FORCE SHORT BLOCK command has no effect.

When START CONCURRENT MODE I/O is performed, a number of things occur. The handler marks the concurrent mode I/O as active (if there are no errors while starting concurrent mode I/O). The handler sets up its own serial input/serial output interrupt handlers as necessary (depending on I/O direction) to field data going in and out. The handler sets itself up to monitor the BREAK key so BREAK will stop the concurrent mode I/O. The handler establishes the initial (empty) state of the input and output buffers. Then the handler informs the Interface Module that concurrent mode I/O has started.

During concurrent mode I/O, each character being received from the Interface Module is taken in by the handler's interrupt driver, put through translation, and placed in the input buffer. Characters to be sent to the Interface Module are translated and put in the output buffer. As the serial hardware in the computer finishes sending each character, the output interrupt driver immediately sends another character from the buffer (unless it is empty). If the input buffer overflows, an error is flagged, output buffer overflow stops putting data into the buffer until data is sent to free buffer space.

Input and output statements (GET, PUT, PRINT, INPUT) executed to a channel through which concurrent I/O is active do not directly cause any I/O to the RS-232-C port. Rather, input statements simply retrieve data that is in the input buffer and output statements put data into the output buffer. If an input statement wants more data but the input buffer is empty, BASIC will wait until the data arrives. If an output statement attempts to put data into a full output buffer BASIC will wait until space becomes available (as a result of the interrupt-driven sending of data from the output buffer). The interrupt-driven sending of data from the output buffer starts as soon as data is put into the buffer. The data is moved into and out of each buffer circularly - that is, the buffer is automatically re-used. The maximum amount of data a circular buffer can hold at once is one less byte than its size.

The Interface Module handles concurrent I/O in one of two ways. The most common mode is used when 8-bit words are being transmitted, no matter what the rate of I/O direction. In this mode, the interface module "connects" (through the interface module's microprocessor) the XMT and RCV lines of the selected port to the I/O connector going to the computer. The data is not interpreted by the Interface Module in this mode, all serialization of the data is performed by the serial I/O hardware in the 400/800 computer. Note that the "connection" between the RS-232-C port and the computer's peripheral I/O port is handled by software. Each line coming in to the Interface Module (one from the computer, one on the RS-232-C port) is sampled (checked) over and over, and its value is then passed on to "connected" outgoing line. The sampling rate is 34.6 kHz; the lines are sampled every 28.9 microseconds.

The other concurrent I/O mode is established in the Interface Module for low speed (300 Baud or less) 7-, 6- or 5-bit input (half-duplex). In this mode, the Interface Module receives a 7-, 6- or 5-bit character from the port and then transmits a corresponding 8-bit character to the computer. This is done because the computer's hardware is not capable of receiving anything but 8-bit serial words. The Interface Module receives the data by sampling it at a rate of 16 samples per bit (similar to a typical UART). As each character is sent from the Interface Module to the computer extra high-order 1-bits are added to get 8-bit words. The Interface Module sets an internal error flag if a framing error occurs in the incoming data. This flag may be queried with STATUS after the concurrent I/O is stopped.

The Interface Module leaves concurrent mode when told to by the handler when the concurrent I/O channel is CLOSED, or when BREAK or RESET is pushed.

The Interface Module is constantly keeping track of all incoming RS-232-C readiness lines, for the purpose of being able to report their history to the STATUS command. This does not apply to the RCV lines or any lines on the printer port. The readiness lines are checked periodically through sampling. The sampling rate depends on the activities the Interface Module is asked to perform. In order not to be missed, a pulse on a readiness line should be at least a few dozen milliseconds in duration.

The STATUS command is performed either by the RS-232-C handler alone (when concurrent I/O is active) or by both the handler and the Interface Module. In the former caser the handler supplies the user with information about its current operation. In the latter case, the handler combines some of its own information with status and sense information supplied by the Interface Module.

ELECTRICAL SPECIFICATIONS OF RS-232-C SERIAL PORTS

(You may refer to the schematic diagram of the 850 Interface Module - APPENDIX 12 Figure 6 - while reading this section).

There are basically two types of circuit for the serial port lines: a receiving circuit, and a transmitting circuit. One of these circuits connects each RS-232-C signal line to a pin of one of the two computer I/O chips in the Interface Module.

The sending circuit consists of an operational amplifier (op-amp) followed by a 10 Ohm protective resistor. The op-amp is driven "to the rail", and produces approximately +9 volts for SPACE, and -5.5 volts for MARK guaranteed at least + or -5V, when driving a 3000 Ohm load (3 kOhm is the worst-case load allowed by the RS-232-C standard, any lower resistance may result in improper operation). The driver circuit will withstand short circuits to ground, and will withstand connection to voltages within their driving range.

Shorting a driver to a voltage outside the range -5.5 to +9 volts may result in damage to the Interface Module.

The receiving circuit consists of a diode and transistor whose function is to convert the minus/plus RS-232-C voltages to the voltages used by the I/O chips. A 4700 Ohm input resistor protects the outside device from having to deliver too much current. Notice that the DSR inputs have 1800 Ohm resistors attached to ground which insure that DSR will seem OFF if nothing is attached to DSR (however, this is no protection against the "antenna effect" of a long unterminated wire attached to DSN which will cause DSR to go ON and OFF if there is activity in other leads in the same cable). Port 4 may be set up for 20 mA current loop operation. In current-loop operation, pins 4 and 7 (RTS +10V, and RCV) are tied together (Pin numbers are of the 9-pin connector of the Interface Module). When the attached Teletype keyboard-sending contacts are closed, pin 9 pulls RCV negative (MARK). This is the idle state of the Teletype. Whenever the switch opens during transmission of a character from the Teletype, RCV is pulled positive (SPACE). Notice that if the Teletype if turned off this switch may be open and the 850 Interface Module will receive a BREAK signal.

For current loop output, the Teletype's printer solenoid is tied between pins 1 and 3 (+10v DTR and XMT). XMT is normally negative (MARK), thus the solenoid is activated in the MARK state. XMT goes to nearly +10V for SPACE so very little current passes through the solenoid and it disengages. Be careful when connecting a current loop device that it does not apply excessive voltages to the Interface Module. Also note that if the send and receive loops are connected together within the Teletype the send or receive loop may not work correctly (the signal may be shorted out). If this happens, try swapping the send or receive wire pairs.

PARALLEL PRINTER PORT SPECIFICATIONS

While reading this section you may refer to the Interface Module schematic - APPENDIX 12 Figure 6 - and printer port timing diagrams at the end of this section.

All signals on the printer port are TTL level (0 to +5V). The output lines are buffered by transistors to supply the necessary drive for the printer electronics. Input lines are buffered to protect the I/O chips.

The output circuit can sink 5 milliAmps. That is, the circuit is capable of pulling 1000 Ohm pull-up resistors in the printer to TTL zero. The output circuit expects some such pull-up in the printer; if they are not present, the output lines will be pulled to +5V only by the internal 10000 Ohm pull-up resistors and the lines may slew too slowly to TTL one.

The Interface Module detects the presence of the printer via the FAULT line. If this line is low, the Interface Module will not respond to printer requests from the 400 or 800 computer. This line is low if the 825 printer is turned off (or disconnectedl. This feature allows you to connect more than one ATARI printer to the computer I/O port, and switch between them by powering only one of them on at a time. If you attach your own printer to the printer port, FAULT must be high (TTL one) for the Interface Module to operate the printer. If there is no appropriate signal from your printer to which FAULT may be attached, you may connect FAULT (pin 12) to the +5V pull-up at pin 9. Be sure you do not connect FAULT to a busy-type line which will alternate on and off, FAULT should stay on.

The eight data lines are positive-logic. The data lines normally rest at zero (ASCII NULL). A data byte is sent to the printer (when it is not BUSY) by placing the data on the 8 data lines and pulsing the DATA STROBE. The STROBE is normally high, and goes low during the strobe pulse.

After sending each data byte to the printer, the Interface Module waits for a BUSY signal. The ATARI 825 printer sends a positive-logic BUSY signal as it processes each byte of data. The BUSY is quite short for most data bytes since the printer merely saves each character in its own memory, but BUSY is quite long when the printer prints. The Interface Module does not care how long the printer is BUSY - the only requirement is that the printer respond to all 40 characters (that is, go not BUSY after the last character) within 4 seconds. Immediately after BUSY goes low again, the Interface Module sends the next character to the printer. When all the characters have been accepted by the printer, the Interface Module signals the 400/840 computer that the print operation is finished.

Some printers which use the Centronix-type interface do not signal BUSY for each character received, but only go BUSY during printing.

For this reason, the Interface Module only waits 200 microseconds for BUSY after sending a data byte. If BUSY does not go on within this time, the Interface Module sends the next character assuming the printer has completed its processing of the preceding character.

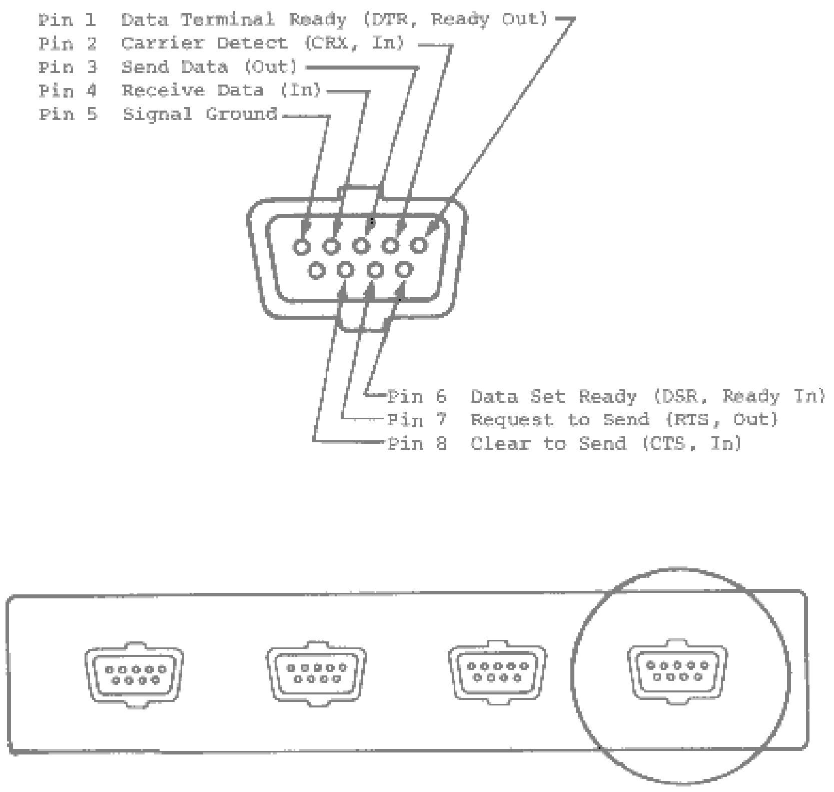

Pin 1 Data Terminal Ready (DTR, Ready Out)
Pin 2 Carrier Detect (CRX, In)
Pin 3 Send Data (Out)
Pin 4 Receive Data (In)
Pin 5 Signal Ground
Pin 6 Data Set Ready (DSR, Ready In)
Pin 7 Request to Send (RTS, Out)
Pin 8 Clear to Send (CTS, In)

Figure 1. Pin functions of Serial Port No. 1 in 850™ Interface Module (9-pin female connector)

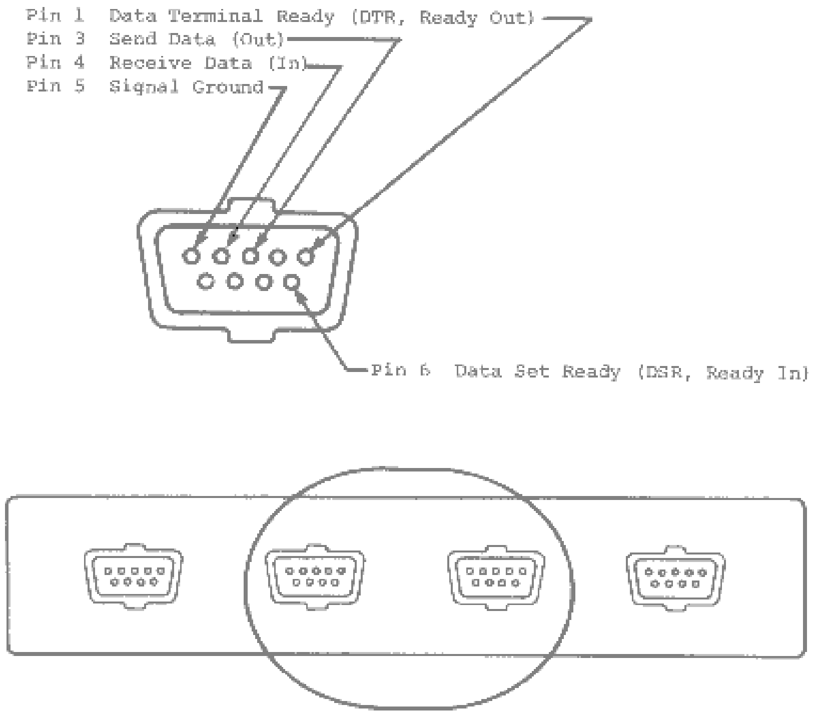

Pin 1 Data Terminal Ready (DTR, Ready Out)
Pin 3 Send Data (Out)
Pin 4 Receive Data (In)
Pin 5 Signal Ground
Pin 6 Data Set Ready (DSR, Ready In)

Figure 2. Pin functions of Serial Port Nos. 2 and 3 in 850^{TM} Interface Module (9-pin female connector)

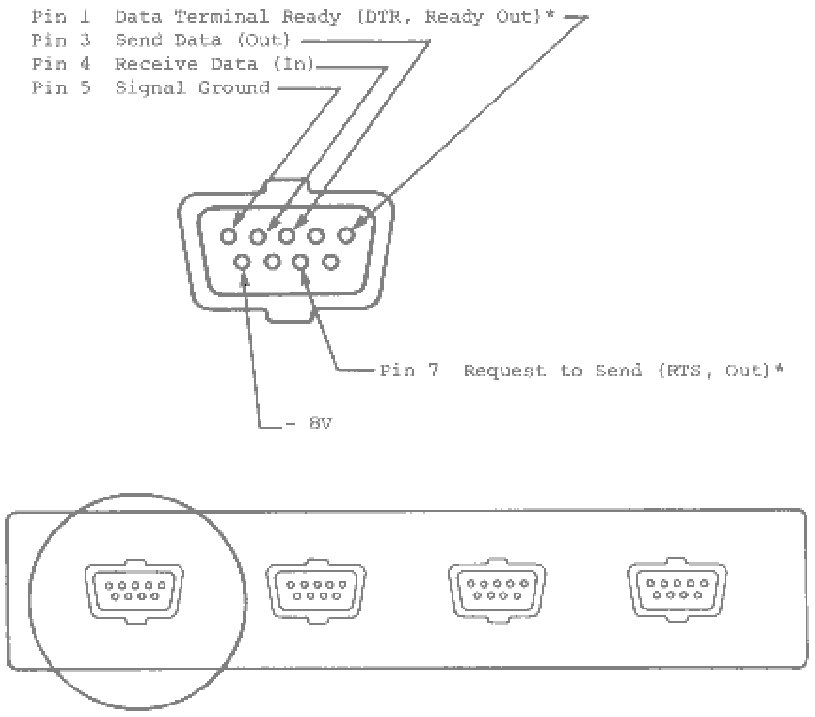

Pin 1 Data Terminal Ready (DTR, Ready Out)*
Pin 3 Send Data (Out)
Pin 4 Receive Data (In)
Pin 5 Signal Ground
Pin 7 Request to Send (RTS, Out)*
– 8V

Figure 3. Pin functions of Serial Port No. 4 in 850™ Interface Module (9-pin female connector)

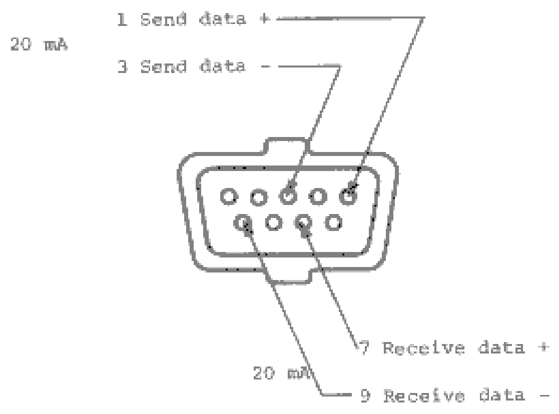


Figure 4. Hook-up of Serial Port No. 4 used with a 20 mA loop device.

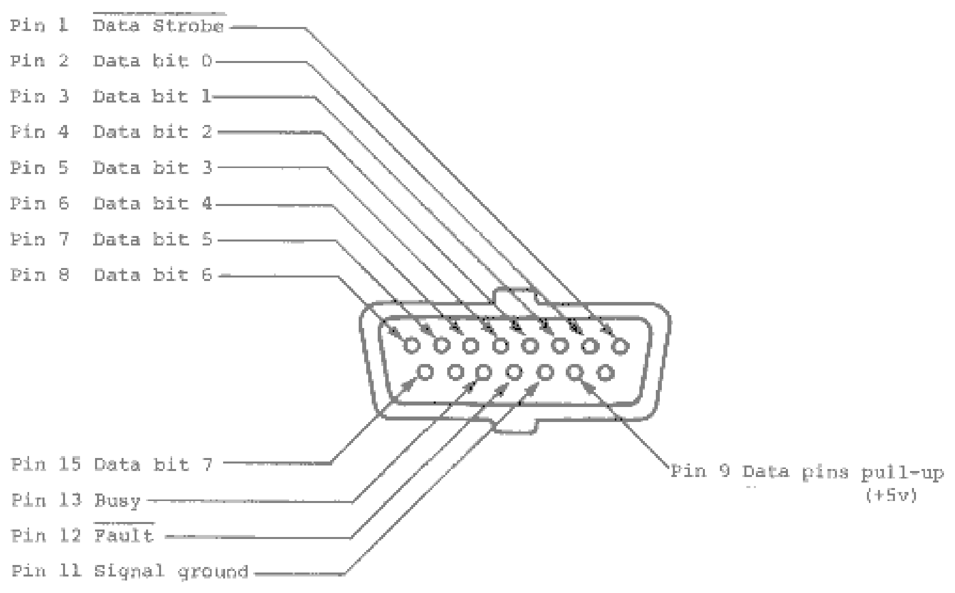
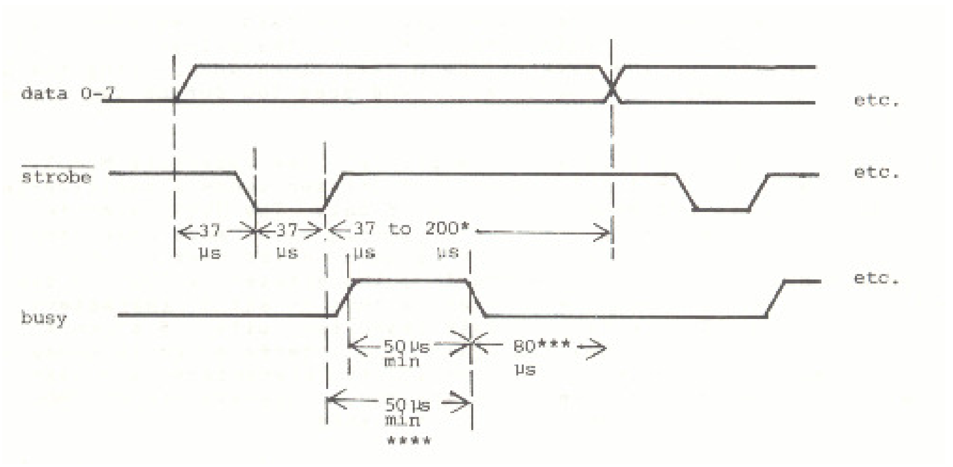


Figure 5. Pin functions of the Printer Port of the 850™ Interface Module (15-pin female connector

Note 1: This port is designed for connecting the ATARI 825™ Printer. If you are going to connect a different printen consult the specifications of its input connector. Make sure that the cable you use to connect between the Printer Port of the 850 and the input to the printer is wired correctly to connect Bit 0 to Bit 0, and so on. You may have to construct your own cable.

Note 2: $\overline{\text{FAULT}}$ must be +5V for printer port to operate. If your printer has nothing appropriate to connect to FAULT (such as +5V power or some ONLINE-type signal), connect pin 9 to in 12.



*   one byte sent every 280 us without Busy (see text)

**   pulse must be 950us, no maximum. Howeven 40 characters must be accepted by printer in 4 sec. (see text)

***   approximate

**** Busy may follow either leading or trailing edge of strobe, however, it must remain at least 50us after trailing edge of strobe.

Figure 6. Printer Port timing diagram.

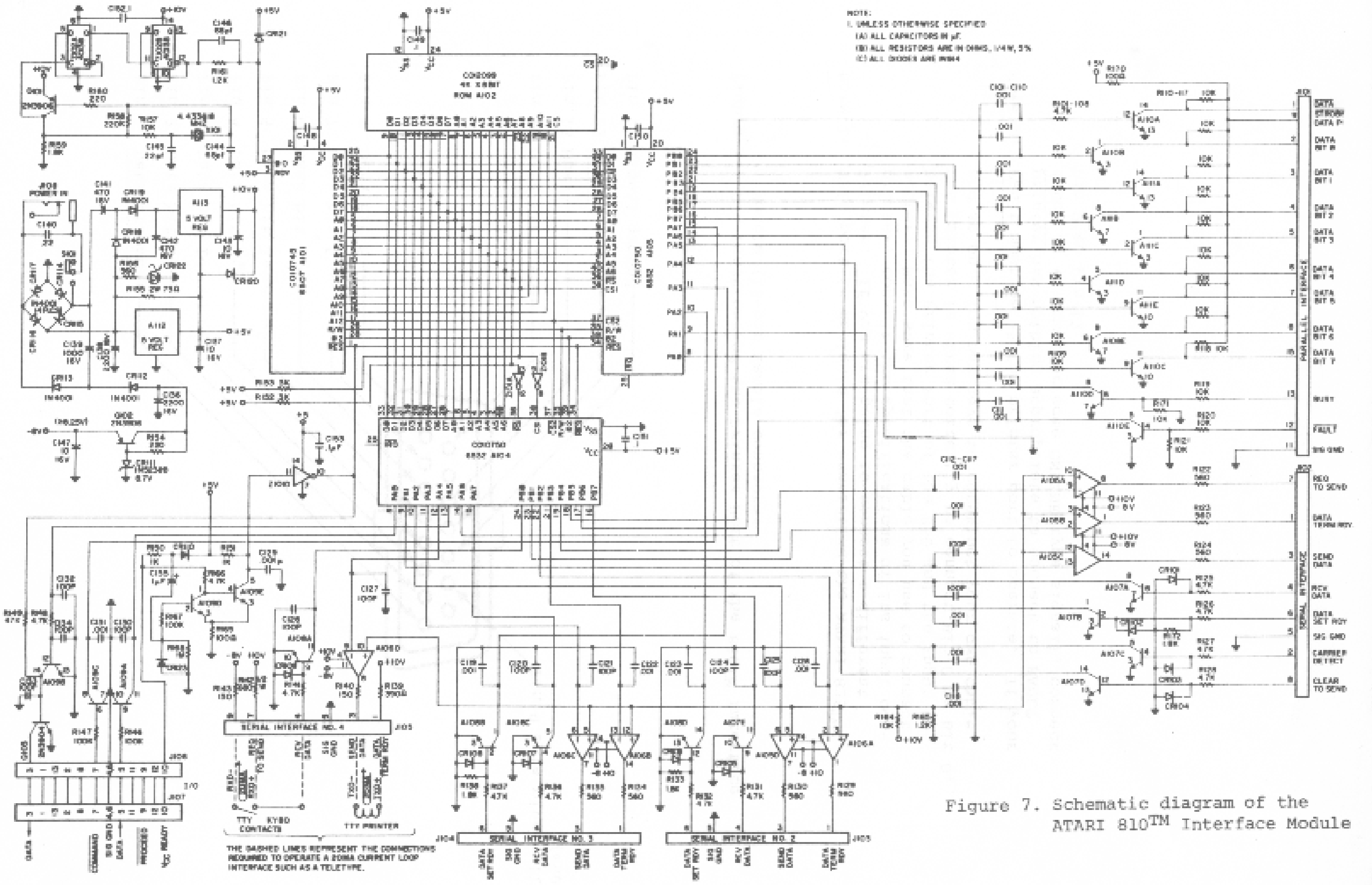


Figure 7. Schematic diagram of the ATARI 810TM Interface Module

APPENDIX 13

ATARI 830™ Modem

The term "modem" is a contraction of "modulator/demodulator". In this context, modulate means to convert from serial binary data to signals of particular frequencies and demodulate means to convert from signals of particular frequencies to serial binary data. The ATARI 830 is a modem. It allows you to send and receive data over telephone lines at rates up to 300 Baud (equivalent to 30 characters/second).

To use the modem you need an ATARI 400 or 800, the ATARI 850 Interface Module and a telephone. Hook-up is shown on the packing box of the modem.

The modem uses frequencies in the audio range. The communication paths include small volumes of air between the modem and the ear-piece and mouthpiece of the telephone. Hence, a modem of this type is said to be "acoustically coupled".

The common standard for communications between computer-related equipment is the RS-232-C standard of the Electronic Industries Association (see APPENDIX 1). The ATARI 830 modem conforms with that standard. The most common type of Modem in use is the Bell 103/113 series. The ATARI 830 modem can communicate with Bell 103/113 equipment.

SIZE

10.2 x 4.7 x 2.3 inches

WEIGHT

1.5 lbs.

TEMPERATURE

Operating environment: 32 to 122 degrees Fahrenheit. (0 to 50 degrees Centigrade)
Storage: -40 to 140 degrees Fahrenheit. (-40 to 60 degrees Centigrade)

ELECTRICAL REQUIREMENTS

24 VAC/150mA supplied by UL-listed wall-mount transformer with 6 foot cord.

HUMIDITY

Operating Environment: 10% to 90% relative humidity (no condensation).
Storage: 5% to 95% (no condensation).

TRANSMITTER FREQUENCIES

Originate:
  Mark: 1270Hz
  Space: 1070Hz
Answer:
  Mark: 2225Hz
  Space: 2025Hz

RECEIVE FREQUENCIES

Originate:
  Mark: 2225Hz
  Space: 2025Hz
Answer:
  Mark: 1270Hz
  Space: 1070Hz

MAXIMUM TRANSMISSION RATE

300 Baud

RECEIVE SENSITIVITY

-45dBm

CONTROLS

FULL/TEST,/HALF

FULL: Sets full duplex operation.
TEST: Sets up audio self-test mode.
HALF: Sets half duplex operatian.
Receive data wiil copy transmit data.

ANS/OFF/ORIG
  ANS: Sets answer mode.
  OFF: Turns unit power off.
  ORIG: Sets originate mode.

INDICATORS

POWER: Power on.
READY: Ready to communicate.

DATA INTERFACE

The modem provides an RS-232-C interface via a standard 25 pin female D-connector. The table below lists the signals used by the modem.

```
OUTPUTS: Mark(OFF): -8V
         Space(ON): +10V

INPUTS: Mark(OFF): -3 to -25V
        Space(ON): +3 to +25V
```

PIN CONNECTIONS

PIN #	SIGNAL MNEMONICS	FUNCTION	SIGNAL DIRECTION
2	XMT	Transmit Data	Input to Modem
3	RCV	Receive Data	Output to Terminal
5	CTS	Clear To Send (On with Carrier Detect)	Output to Terminal
6	DSR	Data Set Ready (On with Carrier Detect)	Output to Terminal
7	-	Signal Ground	Common
8	CRX	Carrier Detect	Output to Terminal

INSTALLATION

Set modem power off, connect wall-mount transformer to unit and 115 VAC outlet.

Connect terminal RS-232-CC cable to interface connector on modem.

Turn modem to ORIG or ANS. POWER indicator should be ON.

OPERATION - ORIGINATE MODE

Set ANS/OFF/ORIG control to ORIG. and FULL/TEST/HALF control as required.

Dial remote number. After hearing answering tone, place handset into acoustic muffs with cord as indicated on label.

When READY indicator turns on, the modem is ready to communicate. Proceed with communication.

OPERATION - ANSWER MODE

Manually answer telephone, switch ANS/OFF/ORIG control to ANS.

NOTE: In answer mode a tone should be heard at all times with or without the READY indicator being activated.

Place handset into muffs with cord as indicated by label.

When READY indicator turns on, proceed with normal data exchange.

To terminate call, switch back to OFF and hang up telephone.

TEST

This test mode is desgined to verify that the modem is functioning properly. It does so by switching the transmitter channel frequencies to match those of the receiver. All data into the modem will be looped back to the terminal for verification. It requires a telephone set to provide an isolated acoustic path between speaker (transmitter) and microphone (receiver).

ORIGINATE MODE

Set terminal for full duplex operation and Modem to ORIG. Set FULL/TEST/HALF switch ta TEST. A continuous tone should be heard from the speaker muff. If no tone is present, unit is defective.

Dial a single digit on the telephone to obtain a quiet line situation. Immediately place handset into acoustic muffs with cord as indicated on label.

NOTE: A quiet line is required for this test to prevent dial tone interference. The line # may remain quiet for only 30 seconds. Repeat as necessary. A longer quiet time can be obtained by dialing a telephone extension or another number under control. The mouthpiece of the second telephone set must be covered to prevent room noise interference.

Wait for READY indication, type message an keyboard. The TEST function will display message. Check terminal for message accuracy.

ANSWER-MODE

Once Modem passes test in originate mode, quickly switch to answer mode with telephone handset still in the rubber muffs.

Wait for READY indication, type message on keyboard. The TEST function will display message. Check terminal for message accuracy.

NOTE: If Modem gives READY indication in both answer and originate modes yet no message or incorrect message (such as double characters) is displayed on the terminal, the RS-232-C cable or terminal interface may be at fault. If terminal can be looped back with RS-232-C cable (pin 2 tied to pin 3 at modem end) and correct message is displayed, then the modem is defective.

TROUBLE SHOOTING

If you have problems, the most likely reason is the phone line. Noise on the line or a weak phone line signal can often result in lost or invalid data. Try to re-dial the call to insure connection is noise-free and there is no interference.

If communication still cannot be established and modem checks out in the TEST mode, see tables below for other possible causes for failure.

Ready Light Off: Is modem power ON?
Is handset in proper position?
Label indicates direction of cord.
Are mode switches set properly?

1. When communicating with a time share computer the modem should be set to ORIG mode. Modem at remote computer end will be in answer mode.
2. When commmunicating with another terminal, mode selection is determined by prior agreement between users. Remember one modem must be in answer mode, the other in originate mode.
3. For proper communication, modem must be either FULL or HALF, not in TEST.
4. Is modem at the other end compatible with the modem. Remote modem must be another modem or a Bell 103-compatible modem. Communication cannot be established with a Bell 202 type.

Double Character Display: Is Modem in half duplex mode?

1. If remote computer echoes all characters both modem and your terminal should be in FULL duplex mode.
2. If communication system is half duplex (no echo>, either your terminal or the modem (not both) should be in HALF duplex.

Garbled Display: Is telephone handset fully seated in the rubber muffs?
Is baud rate correct?
Both local and remote terminals must send data at the same baud rate (300 baud or less).
Is received signal too weak or noisy?
Pick-up handset and listen for a clean tone (if remote modem is in answer mode). If additional tones, dialing pulses, static noise or voices are present, data may be garbled. Re-dial call.

NOTICE

Use of multiple printer control codes that involve carriage motion (with the exception of end-of-line), can cause an ERROR 139 (Device NAK). Carriage motion includes backspace, forward and reverse linefeeds, and partial linefeeds.

The ATARI 850™ Interface Module sends data to the printer in 40-character blocks. If there is more than one carriage motion in each block, the printer cannot recover in time to receive the next 40-character block.

If you should have this problem, check your program. Try to arrange your printer control codes in such a way that these is no more than one carriage motion in each 40-character block. This can be done by preceding each carriage motion with forty "null" characters. Null characters can be generated with control comma ([CTRL] [,]) or with CHR$(0).