The SuperPET

Serial Port

WATCOM

Quality and Service

The SuperPET

Serial Port

Written by: J. B. Schueler

Distributed by: WATCOM Products Incorporated
415 Phillip Street
Waterloo, Ontario
Canada N2L 3X2

Revised: August 1984

Printed in Canada

WATCOM

Disclaimer

WATCOM Systems Inc. makes no representation or warranty with respect to the adequacy of this documentation or the programs which it describes for any particular purpose or with respect to its adequacy to produce any particular result. In no event shall WATCOM Systems Inc., its employees, its contractors or the authors of this documentation be liable for special, direct, indirect or consequential damages, losses, costs, charges, claims, demands, or claim for lost profits, fees or expenses of any nature or kind.

Table of Contents

The SuperPET Serial Port

1 RS-232C Communications

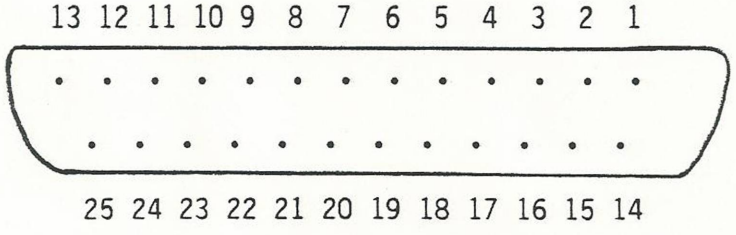
The RS-232C serial communications standard was introduced in 1962 and was adopted by the Electronics Industries Association. You often hear the terms EIA and RS-232C used interchangably. "RS" stands for "recommended standard" and "C" is the most recent revision level.

The 25-pin connector found inside your SuperPET is the most widely used in the industry although you may find examples where manufacturers have employed different types of connectors. Originally, the 232 standard defined the connection between a modem and a terminal but it is now used in a more general way. The standard describes the function of all 25 lines or conductors in the connector. In practice, much less than the 25 are used; sometimes as few as 3 lines. On the SuperPET, 9 are used and we will describe which these are and what they are used for.

We will refer to the 25-pin connector as the serial "port" from here on in.

2 Control and Data Lines of the Serial Port

A cable must be provided that connects the serial port of the SuperPET to a similar port on another device, for example a printer, a modem, or another computer. The line joining the two devices requires a "male" connector on one end which is plugged into the serial port of the SuperPET. A male connector has pins; a female connector has sockets. The male connector plugs into the "female" connector of the serial port in the SuperPET.



Serial Port Female Connector
Front View

The SuperPET uses the following pin connections of the RS-232C port.

Pin	Definition	Symbol
1	Protective Ground	(PG)
2	Transmitted Data	(TxD)
3	Received Data	(RxD)
4	Request To Send	(RTS)
5	Clear To Send	(CTS)
6	Data Set Ready	(DSR)
7	Signal Ground	(SG)
8	Data Carrier Detect	(DCD)
20	Data Terminal Ready	(DTR)

In the following description, we will use two terms to describe the devices on each end of the line. The SuperPET is an example of Data Terminal Equipment (DTE). Terminals and printers are also examples of DTE's. On the other end of the line is the Data Communications Equipment (DCE). A modem is an example of such a device. In many cases, a modem is not required and a connection is made directly to another DTE. For example, the SuperPET may be connected to a serial printer. For consistency throughout the following discussion, we will call the SuperPET, the DTE, and anything that we connect to it, the DCE (even another computer).

PG Protective or "chassis" Ground is normally connected to "earth" ground. Although most equipment manufacturers provide chassis ground, they may recommend that you provide this for their device when you connect it to the SuperPET. In this case pin 1 of the DCE must be connected to pin 1 of the DTE.

TxD Transmitted Data is the bit stream of logical 0's (+3 to +30 volts) and 1's (-3 to -30 volts) being sent from the SuperPET (the DTE) to the other device (the DCE).

RxD Received Data is the stream of logical 0's and 1's coming from the DCE to the DTE (the SuperPET).

RTS Request To Send is a control signal (+5 volts) instructing the DCE that the DTE wishes to send data. The presence of -10 volts indicates that the DTE does not wish to send data. The state of this line can be controlled from the "command register" of the serial device (this will be discussed later).

CTS Clear To Send is a control signal sent by the DCE to the DTE indicating that the DCE is ready to accept data.

DSR Data Set Ready is a control signal sent by the DCE, instructing the DTE that it is in a "powered-up" state. No other meaning is implied.

SG Signal Ground is used to establish a common ground reference between the two devices, enabling them to detect when 0's and 1's are being received.

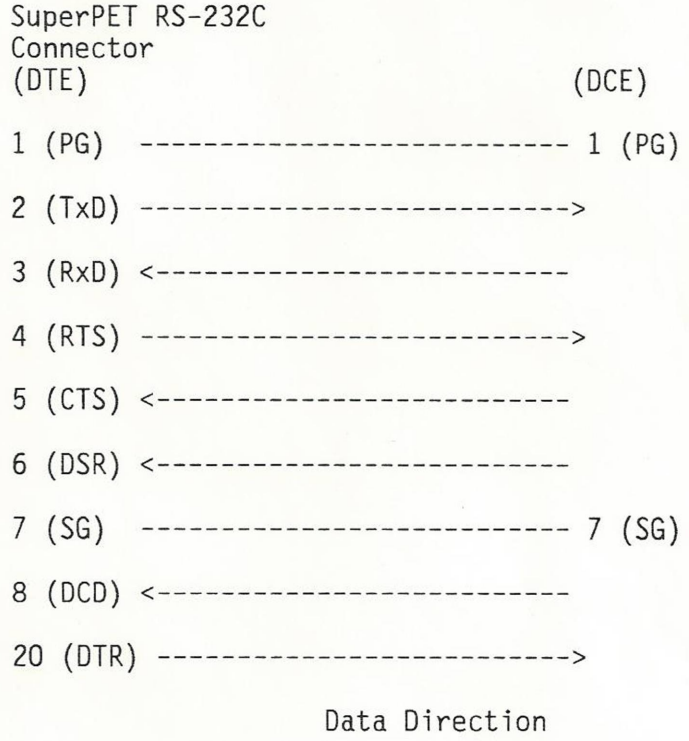
DCD Data Carrier Detected is a control signal from the DCE which tells the DTE that a "carrier" signal has been established by the modem with another modem (usually over a telephone line).

DTR Data Terminal Ready is a control signal sent by the DTE to the DCE indicating that it is ready to operate. +5 volts indicates "ready" and -10 volts indicates "not

ready". The state of this line can be controlled from the "command register" of the serial device.

Another commonly used connection is pin 22, called the Ring Indicator (RI). This is a signal from a modem that is equipped with an auto-answer feature that indicates that a call is coming in over the telephone line. The serial device of the SuperPET does not support a status flag or bit to indicate this condition. Hence this feature is not supported.

The following diagram illustrates the direction of flow of information.



3 Mapping Between RS-232C and TTL Voltages

The previous section described voltage levels at the pins of the connector. These are RS-232C voltages and are converted to TTL (Transistor - Transistor Logic) voltages in the circuitry of the SuperPET.

A pair of chips, 1488 (designated U39 on the SuperPET Model 2 circuit board) and 1499 (U38) perform this conversion. The following chart illustrates the correspondence between RS-232C voltages and TTL voltages.

RS-232C Voltages	TTL Voltages	TTL Level
+5 (+3,+30)	+0 (+0,+2)	low
-10 (-3,-30)	+5 (+2.5,+5)	high

The figures inside parenthesis indicate acceptable ranges. For example, a TTL voltage in the range (+0,+2) is nominally +0 or "low", whereas a TTL voltage in the range (+2.5,+5) is nominally +5 or "high". The RS-232C voltages (+5 and -10), quoted above, are for illustrative purposes only. As the diagram shows, the range for acceptable positive and negative RS-232C voltages is quite wide.

Later on in this document, you will see references to "low" and "high". It will help to keep in mind that these are references to TTL voltage levels. This chart will enable you to determine what RS-232C voltage you should expect to see on a particular pin for a "low" or a "high" condition.

4 Connecting the Serial Port to A Modem

4.1 Full Connection

Some DTEs require only connections to TxD, RxD, and the ground pins. The SuperPET serial port, however, requires signals at all nine pin connections. You should try to obtain a cable with at least nine lines. When the available cable has less than 9 lines, all is not lost. The next section describes what to do when you cannot connect all 9 lines.

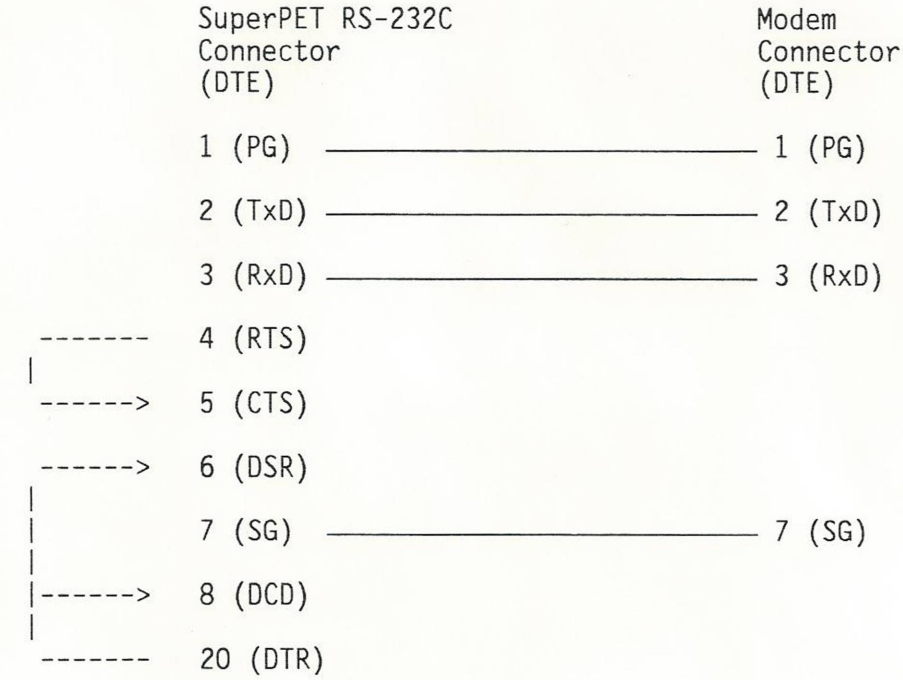
The wiring for all 9 lines is shown as follows:

SuperPET RS-232C Connector (DTE)		Modem Connector (DCE)
1 (PG)	———	1 (PG)
2 (TxD)	———	2 (TxD)
3 (RxD)	———	3 (RxD)
4 (RTS)	———	4 (RTS)
5 (CTS)	———	5 (CTS)
6 (DSR)	———	6 (DSR)
7 (SG)	———	7 (SG)
8 (DCD)	———	8 (DCD)
20 (DTR)	———	20 (DTR)

The documentation for your modem should describe the pin arrangement of the modem. Generally the connection is "straight-through", that is, pin 1 of the modem is connected to pin 1 of the serial port, pin 2 is connected to pin 2, pin 3 is connected to pin 3, etc.

4.2 Partial Connection

As we said above, the SuperPET serial port requires signals at all nine pin connections. To satisfy this requirement, when only lines 1, 2, 3, and 7 are delivered to the serial port, the pins can be connected as follows:



In the RS-232C port of the SuperPET, the Clear To Send (CTS) line must be at the right voltage level for the transmitter (serial port) to work. The CTS input line is used to control the transmitter operation. The enable state is with CTS low (TTL +0 volts). The transmitter is automatically disabled if CTS is high (TTL +5 volts). If the CTS line is not provided from the DCE, the correct level is achieved by connecting the Request To Send (RTS) directly to CTS; thereby, the signal going out to pin 4 goes directly back into pin 5 indicating that it is "clear to send".

Similarly, most software that runs on the SuperPET would expect to receive the signals Data Set Ready (DSR) and Data Carrier Detect (DCD) (this is illustrated later on in a programming example). Again, if there are no lines coming from the DCE cable for these two signals, the correct voltage is achieved by connecting DSR and DCD (pins 6 and 8) directly to Data Terminal Ready (DTR, pin 20). When the SuperPET indicates DTR on pin 20, this signal feeds pins 6 and 8 of the serial port, thus returning DSR and DCD respectively.

You should note that the above arrangement results in the loss of a certain amount of "protocol" information. This step should only be taken when you are sure that the DCE does not need to know the status of Request To Send (RTS) and Data Terminal Ready (DTR). If you are uncertain then you should fully connect all the lines as shown in the previous section.

5 Connecting the Serial Port to A Non-Modem Device

The serial port of the SuperPET may be connected to a device other than a modem, for example, a printer or another computer. The connection in such case may be less than straight-forward. Generally, the cabling arrangement must involve the crossing of pin 2 of the DCE to pin 3 of the DTE and pin 3 of the DCE to pin 2 of the DTE. In other words, the SuperPET's receive line is connected the DCE's transmit line and the SuperPET's transmit line is connected to the DCE's receive line. This kind of cabling is often called a "null modem". A "real" modem provides the cross-connection of send and receive between itself and another modem.

5.1 Connecting to a Serial Printer

A printer may not have a transmission capability and so this line need not be connected. Those printers that support the XON/XOFF synchronization protocol transmit as well as receive data and the line must be connected in order for this protocol to work.

Note:

The SuperPET's "serial" input/output routines do not support the XON/XOFF protocol. You must provide your own support in your application program.

The best way to determine how the cabling should be done is to consult the manufacturer's documentation of the "pinout" of the device's "serial" port. One device with which we have had experience had all the appropriate connections made internally. The connection in this case was identical to that of connecting to a modem. This was not what we expected and, as a result, things did not work. A thorough reading of the documentation solved the mystery.

5.2 Connecting to a Host Computer

Connection to a host computer system can be even more complex, usually because of the length of wire involved and the fact that it terminates somewhere inside the host computer. In general, the best policy is to seek help from the host computer's system-support personnel.

When connecting to a host computer system consider the following. If you can connect the cable to a regular ASCII terminal and it works then you should have few problems connecting to the SuperPET. If the cable ends in the wrong type of connector (i.e., a female connector) then you must either replace the connector or add an adapter. If you replace the connector, jot down on a piece of paper the configuration of the wires. If you fail to keep track of which wire went to which pin, you may spend several frustrating hours trying to make the right connections. If you add an adapter, don't cross any of the wires (they are already crossed somewhere along the line).

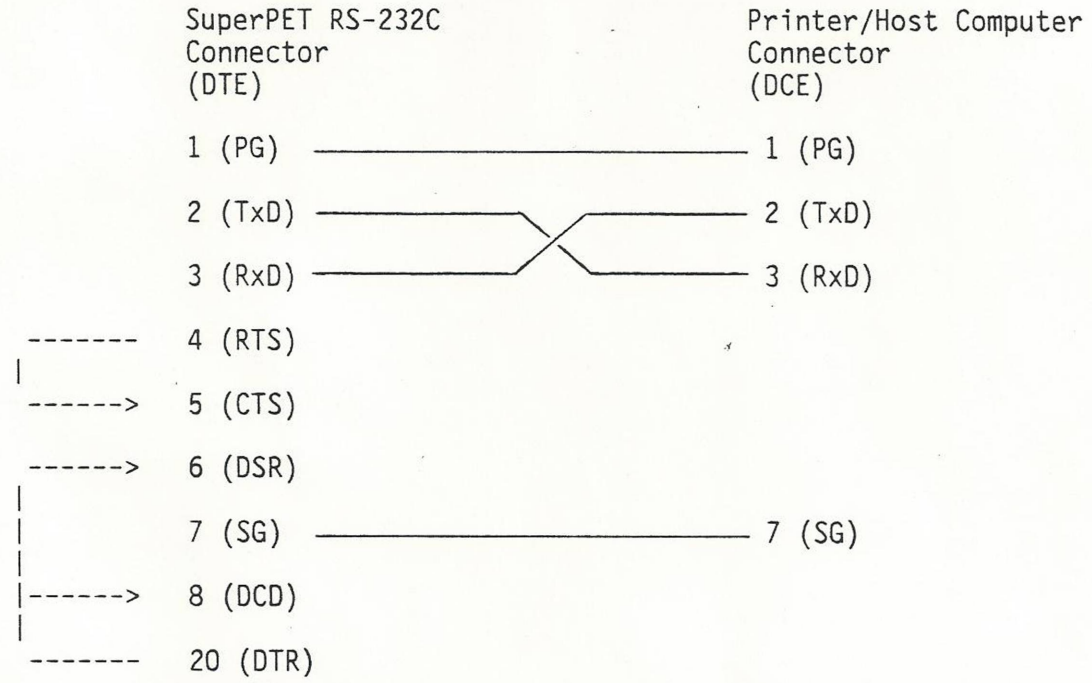
If the cable works with an ASCII terminal but not the SuperPET consider that the SuperPET requires more connections than the 3 or 4 that you can usually "get away with" on an ASCII terminal.

To sum up, the most common misunderstanding that people seem to have is that the SuperPET is somehow "different" from a terminal. Many terminal manufacturers have already "hot-wired" pins 4, 5, 6, 8 and 20 inside the terminal. Quite likely the pins in

these terminals can be reenabled by removing or adding certain "jumper" wires. The SuperPET arrives without such hot wiring because the serial port was intended to be connected to devices other than host computers.

5.3 Partial Connection

The following illustrates a common way to wire the connector and cable.



In the RS-232C port of the SuperPET, the Clear To Send (CTS) line must be at the right voltage level for the transmitter (serial port) to work. The CTS input line is used to control the transmitter operation. The enable state is with CTS low (TTL +0 volts). The transmitter is automatically disabled if CTS is high (TTL +5 volts). If the CTS line is not provided from the DCE, the correct level is achieved by connecting the Request To Send (RTS) directly to CTS; thereby, the signal going out to pin 4 goes directly back into pin 5 indicating that it is "clear to send".

Similarly, most software that runs on the SuperPET would expect to receive the signals Data Set Ready (DSR) and Data Carrier Detect (DCD) (this is illustrated later on in a programming example). Again, if there are no lines coming from the DCE cable for these two signals, the correct voltage is achieved by connecting DSR and DCD (pins 6 and 8) directly to Data Terminal Ready (DTR, pin 20). When the SuperPET indicates DTR on pin 20, this signal feeds pins 6 and 8 of the serial port, thus returning DSR and DCD respectively.

You should note that the above arrangement results in the loss of a certain amount of "protocol" information. This step should only be taken when you are sure that the DCE does not need to know the status of Request To Send (RTS) and Data Terminal Ready (DTR). If you are uncertain then you should fully connect all the lines as described in the manufacturer's documentation.

Quite often serial printers use a "line" synchronization protocol, involving one or all of Clear to Send (CTS), Data Set Ready (DSR), or Data Terminal Ready (DTR). The "hand-shaking" protocol used by a printer should be described in the manufacturer's documentation. The primary purpose of the protocol is to avoid "overrunning" the printer (i.e., sending it data faster than it can print it). Without the appropriate connections, this protocol cannot be implemented.

Note:

The SuperPET's "serial" input/output routines do not support any line protocol. You must provide your own support in your application program.

6 The SuperPET Serial Device

The SuperPET is equipped with a chip that makes possible serial communications with another device that possesses a serial port. The serial communications device on the SuperPET is called an "Asynchronous Communication Interface Adapter" (abbreviated ACIA). The ACIA in use in the SuperPET is the 6551 which is produced by several chip manufacturers (e.g., Synertek). The 6551 is designated U40 on the SuperPET Model 2 circuit board. The chip manufacturers produce documentation for their devices in the form of "data sheets".

Much of the information presented here was distilled from Synertek's data sheet for the 6551. Here are some of the characteristics of the device.

- On-chip baud rate generator: 15 programmable baud rates derived from a standard 1.8432 MHz external crystal (50 to 19,200 baud).
- Programmable interrupt and status register to simplify software design.
- Single +5 volt power supply.
- Serial echo mode.
- False start bit detection.
- 8-bit bi-directional data bus for direct communication with the microprocessor.
- External 16x clock input for non-standard baud rates (up to 125K baud).
- Programmable: word lengths; number of stop bits; and parity bit generation and detection.
- Data set and modem control signals provided.
- Parity: (odd, even, none, mark, space).
- Full-duplex or half-duplex operation.
- 5, 6, 7, 8 and 9 bit transmission.

A good deal of information is provided on the electrical characteristics, operating temperatures, and timings of the chip. The data sheet also meticulously describes all

the input/output lines to the chip, its registers, and various states. Very little information is provided on the proper way to program the chip. Much of that is discovered through trial and error.

The ACIA has 4 8-bit registers and, on the SuperPET, these are located in the address range EFF0 (decimal 61424) through EFF3 (decimal 61427). The registers are described as follows:

6.1 Data Register (EFF0)

Incoming data from the DCE is stored in the Data Register. The bits are ordered from most significant (bit number 7) to least significant (bit number 0). For example, the ASCII letter "A" is 01000001 in binary or 65 in decimal. The left-most bit is the most significant bit (MSB) and the right-most bit is the least significant bit (LSB).

```
;
; ACIA Data Register
;
;    7      6      5      4      3      2      1      0
; +------+------+------+------+------+------+------+------+
; | MSB  |      |      |      |      |      |      | LSB  |
; |  0   |  1   |  0   |  0   |  0   |  0   |  0   |  1   |
; +------+------+------+------+------+------+------+------+
;
; The decimal value 65 (ASCII "A") is illustrated.
;
```

Up to 8 bits of data may be received and it is stored "right justfied" in the data register. Received data may simply be read ("peeked") from this register. Note that any parity information is stripped from the data before placing it in the data register.

Similarly, up to 8 bits of data (a byte) may be transmitted to the DCE by storing ("poking") the data into this register. Note that the device takes care of adding any parity information to the data that is transmitted to the DCE.

The number of data bits sent or received is governed by the choice of "word length" (see description of the Control Register).

Data is sent and received 1 bit at a time, least significant bit first. The rate of transmission is governed by the BAUD rate. The rate of reception may be governed by the BAUD rate or by an externally generated receiver clock (see description of the Control Register).

6.2 Status Register (EFF1)

The Status Register is used to indicate to the SuperPET the status of various 6551 functions. It is a read-only register (i.e., you cannot write new data into it). The status bits of this register are described below. The INT (interrupt) flag is cleared whenever the status register is read. The status of DSR (pin 6) and DCD (pin 8) on the serial port can be determined by examining the appropriate bits of this register.

```
;
; Flags for ACIA Status Register
;
;    7      6      5      4      3      2      1      0
; +------+------+------+------+------+------+------+------+
; |      |  ___ |  ___ |      |      |      |      |      |
; | IRQ  | DSR  | DCD  | TDRE | RDRF | OVRN | FE   | PE   |
; +------+------+------+------+------+------+------+------+
;
PARITY  equ $01; Parity error (1=Error/0=No Error)
FRAMING equ $02; Framing error (1=Error/0=No Error)
OVERRUN equ $04; Overrun error (1=Error/0=No Error)
RDRF    equ $08; Receiver data reg. full (1=Full/0=Not Full)
TDRE    equ $10; Transmit data reg. empty (1=Empty/0=Not Empty)
DCND    equ $20; Data carrier detected (1=Not Detect/0=Detect)
DSNR    equ $40; Data set ready (1=Not Ready/0=Ready)
INT     equ $80; Interrupt occurred (1=Interrupt/0=No Interrupt)
;
```

Note that when the "Data Set Ready" status is "Ready", the DSR status bit is 0. When it is "Not Ready", it is 1. What is happening at the serial port connector? A positive RS-232C voltage on the DSR pin is converted to a TTL +0 volts or "low" and is reflected in the status register as a 0. A negative RS-232C voltage is converted to a TTL +5 volts or "high" and is reflected in the status register as a 1.

The same is true of "Data Carrier Detected". When a carrier signal is "Detected", the DCD status bit is 0. When it is "Not Detected", it is 1.

You can use the SuperPET monitor to display the contents of the ACIA registers ("d eff0"). Without a connector present in the serial port, the status of bits 5 and 6 (DCD and DSR respectively) will be 1. When a connector is present and both DCD and DSR are asserted (i.e., a positive RS-232C voltage is present), the status should be 0.

Earlier, we described one way of ensuring that DCD and DSR will always be asserted. That was with the "hot-wired" connector.

Let us consider what should happen when a modem is fully connected to the serial port. When the modem is turned on, the DSR status bit should go from 1 to 0. When a telephone call is made and a carrier signal (a high-pitched tone) is obtained, the DCD status bit should go from 1 to 0.

6.3 Command Register (EFF2)

The Command Register is used to control specific transmit/receive functions. It is a read/write register. Parity control is included in this register. The bits of this register are described below. Note that bits 2 and 3 of this register are used to govern interrupts resulting from transmission of data as well as the state of Request to Send (RTS). RTS should be set "low" in order to produce +5 volts on the RTS pin (the desired state).

```
;
; Flags for ACIA Command Register
;
;    7      6      5      4      3      2      1      0
; +------+------+------+------+------+------+------+------+
; |    PARITY CHECK    | NRML/| TRANSMITTER |RX INT| DTR  |
; |      CONTROLS      | ECHO |  CONTROLS   | E/D  | E/D  |
; +------+------+------+------+------+------+------+------+
;
DTRE    equ $01; Enable receiver / transmitter
;                0 = Disable Receiver/Transmitter /  ___  \ high
;                1 = Enable Receiver/Transmitter  \ DTR  / low
RID     equ $02; Receiver interrupt disable
RIE     equ $FD; Receiver interrupt enable mask (complement of RID)
;                0 = IRQ Interrupt Enabled from bit 0 of Status Reg
;                1 = IRQ Interrupt Disabled
TIE     equ $04; Transmitter interrupt enable
TID     equ $08; Transmitter interrupt disable
TBREAK  equ $0C; Transmit Break
;              +-----+----------+----------+----------------+
;              | BIT | TRANSMIT |          | OTHER          |
;              | 3 2 | INTERRUPT| RTS LEVEL|                |
;              +-----+----------+----------+----------------+
;              | 0 0 | Disabled | High     | -              |
;              | 0 1 | Enabled  | Low      | -              |
;              | 1 0 | Disabled | Low      | -              |
;              | 1 1 | Disabled | Low      | Transmit Break |
;              +-----+----------+----------+----------------+
ECHO    equ $10; Normal/Echo mode for receiver (0=NORMAL/1=ECHO)
P_NONE  equ $00; No parity     (--0x xxxx)
P_ODD   equ $20; Odd parity    (001x xxxx)
P_EVEN  equ $60; Even parity   (011x xxxx)
P_MARK  equ $A0; Mark parity   (101x xxxx)
P_SPACE equ $E0; Space parity  (111x xxxx)
;      +-------+-----------------------------------------------+
;      | BIT   | Operation                                     |
;      | 7 6 5 |                                               |
;      +-------+-----------------------------------------------+
;      | - - 0 | Parity disabled - No parity bit generated;    |
;      |       | No parity bit received                        |
;      | 0 0 1 | Odd parity Receiver and Transmitter           |
;      | 0 1 1 | Even parity Receiver and Transmitter          |
;      | 1 0 1 | Mark parity bit x'mitted; no rx parity check  |
;      | 1 1 1 | Space parity bit x'mitted; no rx parity check |
;      +-------+-----------------------------------------------+
;
```

6.4 Control Register (EFF3)

The Control Register is used to select the desired mode for the 6551. The word length, number of stop bits, and clock controls are all determined by the Control Register. The bits of this register are described below.

```
;
; Flags for ACIA Control Register
;
;    7      6      5      4      3      2      1      0
; +------+------+------+------+------+------+------+------+
; | STOP |      WORD    |RX-CLK|         BAUD RATE         |
; | BITS |    LENGTH    | SRC  |         GENERATOR         |
; +------+------+------+------+------+------+------+------+
;
SP_0     equ     $00     ; 16x External Clock    (xxxx 0000)
SP_50    equ     $01     ; 50 Baud               (xxxx 0001)
SP_75    equ     $02     ; 75 Baud               (xxxx 0010)
SP_110   equ     $03     ; 109.92 Baud           (xxxx 0011)
SP_135   equ     $04     ; 134.58 Baud           (xxxx 0100)
SP_150   equ     $05     ; 150 Baud              (xxxx 0101)
SP_300   equ     $06     ; 300 Baud              (xxxx 0110)
SP_600   equ     $07     ; 600 Baud              (xxxx 0111)
SP_1200  equ     $08     ; 1200 Baud             (xxxx 1000)
SP_1800  equ     $09     ; 1800 Baud             (xxxx 1001)
SP_2400  equ     $0A     ; 2400 Baud             (xxxx 1010)
SP_3600  equ     $0B     ; 3600 Baud             (xxxx 1011)
SP_4800  equ     $0C     ; 4800 Baud             (xxxx 1100)
SP_7200  equ     $0D     ; 7200 Baud             (xxxx 1101)
SP_9600  equ     $0E     ; 9600 Baud             (xxxx 1110)
SP_19200 equ     $0F     ; 19200 Baud            (xxxx 1111)
EXT      equ     $00     ; External Receiver Clk (xxx0 xxxx)
BRD      equ     $10     ; Baud Rate Generator   (xxx1 xxxx)
WORD_8   equ     $00     ; Data Word Length 8    (x00x xxxx)
WORD_7   equ     $20     ; Data Word Length 7    (x01x xxxx)
WORD_6   equ     $40     ; Data Word Length 6    (x10x xxxx)
WORD_5   equ     $60     ; Data Word Length 5    (x11x xxxx)
STOP_1   equ     $00     ; 1 Stop Bit            (0xxx xxxx)
STOP_2   equ     $80     ; 2 Stop Bits           (1xxx xxxx)
;                          1 Stop Bit if Word Length = 8 Bits
;                            and Parity
;                          1 & 1/2 Stop Bits if Word Length =
;                            5 Bits and No Parity
;
```

6.5 Register Offsets

From an assembly language programming point of view, it is useful to describe the registers of the ACIA symbolically. Below, we describe them in terms of offsets from the starting address of the 4 ACIA registers.

```
;
; Offsets to ACIA Registers
;
IOR     equ     0       ; ACIA Input/Output Data Register
STATR   equ     1       ; ACIA Status Register
CMDR    equ     2       ; ACIA Command Register
CNTLR   equ     3       ; ACIA Control Register
```

6.6 Selecting Parity, Word Length, and Stop Bits

You must select the parity, word length and number of stop bits to match what the DCE expects. When a packet of data is sent to or received from the DCE it is enclosed by a "start" bit, an optional parity bit and 1 or 2 "stop" bits.

A start bit tells the receiver that a character is about to begin. The stop bits tell the receiver that the entire character has been transmitted. The start and stop bits serve the same function as blank spaces between printed words.

Actually, the stop bits perform two functions. First, it returns the line to a state wherein the next start bit can be recognized and, second, it allows the receiver time to prepare for the next character. It is the transition from stop to start bit that causes the receiver to start processing the incoming stream of bits.

The term "asynchronous" is used to describe this method for data transmission. For example, if you were communicating with a host computer using the SuperPET as a terminal, the time between characters sent to the host varies with the rate at which you type. Characters arrive at the host computer "asynchronously".

```
;
; Data Packet
;                                          Optn'l        Optn'l
;  Start  <--- Up to 8 Data Bits --->  Parity Stop   Stop
;   Bit                                  Bit   Bit 1  Bit 2
; +------+------+------+------+......+------+------+------+
; |  0   | 0/1  | 0/1  | 0/1  |      | 0/1  |  1   |  1   |
; +------+------+------+------+......+------+------+------+
;
```

The choice of word length governs how many data bits will be sent or received. The minimum is 5 and the maximum is 8. The least significant bits of the Data Register are sent whenever the word length is less than 8. As the description of the ACIA Control Register says, only 1 stop bit is added if the selected word length is 8.

Let's consider what data is sent to the DCE when even parity, a 7-bit word length, and 1 stop bit is selected. The start bit is sent first. Parity is calculated for the 7 least significant bits in the data register such that the sum of the data bits plus the parity bit is even. For example, if the data bits are 1000001 then the parity bit will be 0 since 1 + 0 + 0 + 0 + 0 + 0 + 1 + 0 = 2. Thus, starting with the least significant bit, 7 data bits are sent and then a 0 parity bit. The single stop bit is sent last. This is a total of 10 bits.

A very common selection for serial printers is even parity, a word length of 7, and 1 stop bit. Another is no parity, a word length of 8, and 1 stop bit. In both cases, 10 bits of

information are transmitted. If the selected BAUD rate (or number of bits/second) was 300 then the character transmission rate would be 300 divided by 10 or 30 CPS (characters per second).

You may now understand what "NO", "EVEN" and "ODD" parity means but what about "SPACE" and "MARK" parity? The following table should summarize what all of these mean.

Parity	Meaning
No	No parity bit is generated
Odd	Sum of parity + data bits is odd
Even	Sum of parity + data bits is even
Mark	Parity bit is always 1
Space	Parity bit is always 0

Question:

What is the difference between "no parity" with a word length of 8 and "space parity" with a word length of 7 when the most significant data bit (of 8 bits of data) is always 0?

Answer:

None!

It is very important that you establish the correct settings for parity, word length and number of stop bits. Of course, the correct BAUD rate is important too. The manufacturer's documentation for your modem, printer, etc. should be consulted to determine what these parameters should be.

7 Summary

Having now examined the registers of the serial port in some detail, the following points should be noted.

(1) The CTS signal must be high in order for the ACIA to operate.

(2) DSR (pin 6) and DCD (pin 8) are inputs to the serial device.

(3) Both DSR and DCD can be used to provide status information on the device with which we are communicating. For example, we could choose not to transmit data to the DCE if either one or both of the status flags "DSR and DCD are not 0. Typically, a serial device handler or "driver" will do this. This is why we recommended "hot wiring" the DSR and DCD pins to the DTR pin whenever these two signals were not provided by the DCE.

(4) By convention, DSR and DCD were defined to mean certain things. The programmer, however, could choose to assign different meanings to these pins. For example, the DCE's Data Terminal Ready (DTR) status may alternately change between high and low to signal a "Ready" or "Not Ready" status. We could hook up the DCE's pin 20 (DTR) to the DTE's pin 6 (DSR) and then check the status of DSR to determine if we should transmit data to the DCE.

(5) You will probably be confused by the terminology "high" and "low" when applied to the various signals. For example, the receiver / transmitter is enabled setting bit 0 in the "Command Register" to 1. This is noted above as setting DTR low. If you enable DTR and use a volt meter to check the DTR pin, you'll see that it is at +5 volts (or some appropriate positive voltage). Remember that this RS-232C voltage corresponds to a TTL voltage of +0 volts which is "low".

We conclude this description of the serial port with two programming examples. The second example was arrived at after many months of experimentation with the ACIA.

Note:

An appendix at the end of this document lists some additional references which you may find interesting and useful.

If you have any suggestions for improving this document, please write to WATCOM at the address shown on the title page.

Appendix A

Simple Programming Example Using BASIC

We will demonstrate how to program the serial device using BASIC as an example. The following excerpts are taking from a BASIC program which is used to log records from a telephone switchboard system which is equipped with a serial port. The switchboard or DCE obeys the Data Terminal Ready (DTR) protocol. It will transmit a record of information only when DTR is enabled. The serial port settings for communicating with this device are a BAUD rate of 2400, a word length of 7, 1 stop bit, and EVEN parity.

```
250 acia%=hex("eff0")
260 !            stop=1,   wrdlen=7, brd,      spd=2400
270 poke acia%+3,hex("00")+hex("20")+hex("10")+hex("0A")
280 ! parity=even, tx int disable, rx int disable, DTR enable
290 ENABLE%=hex("60")+hex("08")+hex("02")+hex("01")
300 ! parity=even, tx int disable, rx int disable, DTR disable
310 DISABLE%=hex("60")+hex("08")+hex("02")+hex("00")
320 open #3,"serial",input
```

At this point the number of stop bits, word length, and BAUD rate have been established. Two variables, ENABLE% and DISABLE% will be used to control the Data Terminal Ready status as well as the parity setting. The "serial" device has been opened for reading of records.

The next program fragment shows how the arrival of records from the serial port are controlled.

```
480 loop
490    poke acia%+2,ENABLE%
500    linput #3,msg$
510    poke acia%+2,DISABLE%
520    print msg$
530 endloop
```

For each record, DTR is first enabled, a record is read (the record is terminated upon reception of ASCII CR or carriage return), and then DTR is disabled. This allows the program to continue processing the record just received without having to worry about the untimely arrival of a new record. Note that once the DCE has started to transmit a stream of data, the BASIC "linput" routine must be ready to accept and store each data byte elsewhere in memory. Failure to do so will result in lost data. The DCE does check to see whether anyone on the other end of the line is picking up the incoming data. If the next data byte is stored overtop of the data byte which is already in the Data Register without it having been read then we call this an "overrun" condition.

In this example, the processing of the record involves printing it to the screen. The program from which this example was adapated actually performed quite a bit more processing.

Appendix B

Transfering Data Between Two Computers

In this section, we will describe how to send and receive files of textual data between two computers. Again, we will use BASIC to illustrate how this may be done.

The various implementations of our software treat the serial port as a file with the special name "serial". Some knowledge of the interaction of BASIC's input/output routines with the serial port may aid in understanding how a program should deal with this device. Two example programs show a simple means of text file transfer. The first program is used to "send" files, over the serial port, to another computer. The second program is used to "receive" a file that is being "sent" by the first program.

```
 1 ! File Transmission
 2 on ioerr ignore
 3 on eof ignore
 4 open #3,"serial",inout
 5 linput "File to send? ",file$
 6 open #4,file$,input
 7 if iostatus <> 0
 8   print "No such file"
 9 else
10   linput #3,rply$
11   loop
12     linput #4,rec$
13     if iostatus <> 0 then quit
14     print #3,rec$
15     linput #3,rply$
16     if rply$(1:1)=chr$(10) then rply$(1:1)=""
17     if rply$ <> "ok" then quit
18   endloop
19   close #4
20 endif
21 print #3,"<eof>"
22 close #3
23 end
```

```
 1 ! File Reception
 2 on ioerr ignore
 3 open #3,"serial",inout
 4 linput "File to get? ",file$
 5 open #4,file$,output
 6 if iostatus <> 0
 7   print "Cannot open file"
 8 else
 9   print #3,"go"
10   loop
11     linput #3,rec$
12     if rec$(1:1)=chr$(10) then rec$(1:1)=""
13     if rec$ = "<eof>" then quit
14     print #4,rec$
15     if iostatus <> 0 then quit
16     print #3,"ok"
17   endloop
18   close #4
19 endif
20 close #3
21 end
```

The serial port is capable of both transmitting and receiving data. If we wish to both send and receive records via this port then, in BASIC, we open the "serial" device for input/output (e.g., OPEN #3,"serial",INOUT).

When you "print" a string to the device (e.g., PRINT #3,"go"), the characters 'g', 'o', a carriage return (chr$(13)), and a line feed (chr$(10)) are transmitted over the serial port. If the port is connected to the serial port of another computer then each of these characters will appear as input data on the port of this computer. A program must be executing on the receiving computer which "grabs" each character as it appears. If there is no such program then the transmitted data will be lost. The rates of transmission and reception must be identical, otherwise the transmitted data will not be "assembled" correctly by the receiver. Therefore the BAUD rates of the two serial ports must be the same. You must also ensure that the parity and word length settings for the two ports are compatible.

If characters arrive at a rate faster than the receiving program can deal with them, then some of the transmitted data will be lost. This is a problem that will likely show up at high BAUD rates. It is a problem that cannot be easily dealt with from a BASIC program since it typically executes much slower than is necessary to ensure no loss of data.

In addition, even if data arrives at a slow enough rate, there may be points at which we wish to do other processing. For example, we may wish to write a number of characters, that have been received, to a file. It would be convenient if we could somehow inform the program executing on the "sending" computer that it should wait a while until we are ready for more data. This problem is fairly easy to solve in the program. We can invent a simple protocol which will enable the two programs executing on different computers to "talk" to each other. One program will "drive" the conversation. In other words, one program will instruct the other program when to "speak" (i.e., send a record) and when to "be silent" (i.e., wait until it is alright to send the next record).

This is very simple to do using BASIC input/output statements. If you examine the "receive" program above, it may be apparent to you that it is the program that drives the conversation. The "send" program will not "speak" until it is "spoken" to. Since the two programs must synchronize their dialogue it is very important to know which of the above programs must be "run" first. In this case it is the "send" program which must be run first. If it is not executing at the time the "receive" program says "go" then it will not "hear" the command to begin transmission.

If the programs are started in the wrong order, both will be "listening" and neither will "say" anything. Clearly, it is desirable to avoid such a "standoff" or "deadlock" situation. If this event occurs, both computers may no longer respond. If you type in the above programs, save them first before you attempt to run them.

The two programs, shown above, were written to use the serial port of a microcomputer. On large computers, this port may also be the one to which your terminal/microcomputer is connected. If this is the case, the programs may require some slight modification. The device name and dialogue synchronization may need to be changed.

Appendix C

Complete Programming Example Using Assembly Language

The following assembly language routines provide the building blocks for writing a device driver for the serial port. Each segment, shown below, contains the assembler directive, "include <srdefn>". The contents of the file, "srdefn", are shown in the section entitled "Serial Definitions".

These routines could, for example, be incorporated into an ASCII terminal emulator. The last section illustrates how this may be done by providing an example of a simple terminal emulator.

C.1 ACIA Handler Initialization

The routine "SRTie" expects to be passed the address of a buffer which is to be used to store received characters. The length of this buffer is defined by "BFLEN" and, in this example, is set to hexadecimal 78 (see Serial Definitions).

```
;include <srdefn>
;
; ACIA Handler Initialization
;
xref    ACIAHOST_
;
xdef    SRTie_
SRTie_  equ     *
  STD SRBuffer          ; set buffer address
  STD BfSPtr            ; set buffer start
  STD BfEPtr            ; set buffer end
  CLR Key               ; init peek/poke window
  CLR ReceiveLock       ; our XOFF/XON status is "ON"
  CLR SendLock          ; host XOFF/XON status is "ON"
  CLR SendInProg        ; indicate o.k. to send next byte
```

```
  CLR Break              ; no "BREAK" condition
  CLR BreakUnderway      ; break sequence not underway
  CLR OverrunError       ; no overrun errors
  CLR FramingError       ; no framing errors
  CLR ParityError        ; no parity errors
  CLR Count              ; zero buffer fill count
  LDB ACIAHOST_+CMDR     ; get current command register setting
  ANDB #$FO              ; preserve some bits
  ORB #(TID+DTRE)        ; set command register bits
  STB ACIAHOST_+CMDR     ; set parity, tx interrupt disable,
                         ;   rx interrupt enable
  RTS                    ; end of handler initialization

  end
```

C.2 ACIA Interrupt Connection / Disconnection

The routines "SRConnect" and "SRDisconnect" are used to connect and disconnect from the SuperPET interrupt handling system.

```
;include <srdefn>
;
IntVctr equ     $0100
IRQ     equ     8
;
; Connect to IRQ Interrupts
;
xref    ConBInt_
;
xdef    SRConnect_
SRConnect_ equ  *
  LDD IntVctr+IRQ       ; get address of current IRQ handler
  STD IRQHndlr          ; save old IRQ handler address
  LDD #IRQ              ; install new IRQ handler
  PSHS D                ;
  LDD #SRIRQHndlr       ;
  JSR ConBInt_          ;
  LEAS 2,S              ; pop parameter
  RTS                   ; return to caller
;
; Disconnect from IRQ Interrupts
;
;
xdef    SRDisconnect_
SRDisconnect_ equ       *
  LDD IRQHndlr          ; get old IRQ handler address
  STD IntVctr+IRQ       ; restore IRQ handler address
  CLRA                  ; indicate that we
  CLRB                  ; have disconnected
  STD IRQHndlr          ;
  RTS                   ; return to caller
;
```

```
xref    ACIAHOST_
xref    SRIntH
;
xdef    SRIRQHndlr
SRIRQHndlr      equ     *
  LDA ACIAHOST_+STATR   ; get host ACIA status
  if lt                 ; if INTERRUPT bit on then
    JSR SRIntH          ; - call our ACIA handler
  else                  ; else
    JMP [IRQHndlr]      ; - call regular IRQ handler
  endif                 ; endif
  RTS                   ; end of IRQ handler routine

  end
```

C.3 ACIA Interrupt Handling

The routine "SRIntH" performs the handling of interrupts caused by the transmission and reception of data from the serial port.

```
;include <srdefn>
;
; ACIA Interrupt Handler
; ACIA Status in A Accumulator
;
xref    ACIAHOST_
xref    SRStuff
;
xdef    SRIntH
SRIntH  equ     *
  BITA #TDRE            ; if tx data register empty
  if ne                 ; then
    TST SendInProg      ; - if character was transmitted
    if ne               ; - then
      LDB ACIAHOST_+CMDR; - - get command register
      ANDB #$F2         ; - - select bits to preserve
      ORB #(TID+DTRE)   ; - - select tx interrupt disable and
                        ; - - enable receiver/transmitter
      STB ACIAHOST_+CMDR; - - update command register
      CLR SendInProg    ; - - indicate o.k. to send next byte
    endif               ; - endif
  endif                 ; endif
  BITA #RDRF            ; if rx data register full
  if ne                 ; then
    JSR SRStuff         ; - stuff character into ring buffer
    CLR BreakUnderway   ; - break sequence must be complete
  endif                 ; endif
  BITA #OVERRUN         ; if overrun error
  if ne                 ; then
    INC OverrunError    ; - tally another overrun error
  endif                 ; endif
```

```
  TST BreakUnderway     ; if no break sequence in progress
  if eq                 ; then
    BITA #FRAMING       ; - if framing error
    if ne               ; - then
      INC FramingError  ; - - tally another framing error
      INC Break         ; - - indicate "BREAK" condition
      DEC BreakUnderway ; - - indicate break sequence underway
    endif               ; - endif
  endif                 ; endif
  BITA #PARITY          ; if parity error then
  if ne                 ; then
    INC ParityError     ; - tally another parity error
  endif                 ; endif
  RTS                   ; end of interrupt handler

  end
```

C.4 Getting Received Data

The routine "SRGetChar" returns a byte from the reception buffer to the caller. If no data are present in the buffer then a NULL character is returned (decimal value 0). The data are returned in the "B" register, one at a time, in the same order as they were received.

```
;include <srdefn>
;
; Get Next Byte From Ring Buffer
;
xref    SRPutX
;
xdef    SRGetChar_
SRGetChar_ equ  *
  CLRB                  ; return NULL if buffer empty
  LDX BfSPtr            ; get ring buffer start pointer
  CMPX BfEPtr           ; if ring buffer not empty
  if ne                 ; then
    LDD SRBuffer        ; - calculate ring buffer
    ADDD #BFLEN         ; - end address
    PSHS D              ; - save ring buffer end address
    DEC Count           ; - reduce buffer use count
    guess               ; - guess that it's time to restart host
      TST TTSync        ; - - quit if we cannot synch. data reception
      quif eq           ; - -
      TST ReceiveLock   ; - - quit if we have not sent XOFF
      quif eq           ; - -
      LDA Count         ; - - get buffer use count
      CMPA #BFLEN-BFLOW ; - - quit if buffer is not near empty
      quif gt           ; - -
      LDB #XON          ; - - transmit XON character
      JSR SRPutX        ; - - to restart the host
      CLR ReceiveLock   ; - - indicate that we have sent XON
    endguess            ; - endguess
```

```
    LDB 0,X             ; - get next byte
    LEAX 1,X            ; - increment start pointer
    CMPX ,S             ; - if at end of ring buffer
    if eq               ; - then
      LDX SRBuffer      ; - - point at beginning of buffer
    endif               ; - endif
    STX BfSPtr          ; - update ring buffer start pointer
    LEAS 2,S            ; - pop ring buffer end address
  endif                 ; endif
  STB Key               ; pass current byte through memory also
  CLRA                  ; high byte of D gets 0
  RTS                   ; end of character get routine

  end
```

C.5 Transmitting Data

The routine "SRPutChar" transmits a data byte out the serial port. The data byte is passed in the "B" register.

The routine "SRBreak" generates a "break" condition on the serial line.

```
;include <srdefn>
;
; ACIA Output Routine
;
xref    ACIAHOST_
;
xdef    SRPutKey_
SRPutKey_ equ   *
  LDB Key               ; load byte to transmit
;
xdef    SRPutChar_
SRPutChar_ equ  *
  loop                  ; loop (host synchronization)
    TST SendLock        ; - check XON/XOFF status
  until eq              ; until not XOFF
;
```

```
xdef    SRPutX          ; special entry for XOFF/XON transmission
SRPutX  equ     *
  loop                  ; loop
    TST SendInProg      ; - check status of last byte transmitted
  until eq              ; until last byte was transmitted

  PSHS A                ; save accumulator A
  loop                  ; loop
    LDA ACIAHOST_+STATR ; - check for data carrier not detected
    ANDA #DCND+DSNR     ; - and data set not ready status
  until eq              ; until both status bits are zero
                        ;   (i.e., DSR + DCD)
  LDA ACIAHOST_+CMDR    ; get current command register
  ANDA #$F2             ; select bits to preserve
  ORA #(TIE+DTRE)       ; select tx interrupt enable and
                        ; enable receiver/transmitter
  DEC SendInProg        ; indicate transmission started
  STB ACIAHOST_+IOR     ; start data transmission out ACIA port
  STA ACIAHOST_+CMDR    ; update command register
  PULS A                ; restore accumulator A
  RTS                   ; end of output routine

xdef SRBreak_
SRBreak_ equ    *
  LDA ACIAHOST_+CMDR    ; get current command register
  ANDA #$F2             ; select bits to preserve
  ORA #(TBREAK+DTRE)    ; select transmit BREAK
                        ;   and enable receiver/transmitter
  STA ACIAHOST_+CMDR    ; update command register
                        ; delay long enough for host computer
                        ; to recognize a "break" condition
  LDX #0                ; zero counter
  loop                  ; loop
    LEAX 1,X            ; - increment counter
  until eq              ; until zero again
                        ; and then turn "break" off
  ANDA #$F2             ; select bits to preserve
  ORA #(TID+DTRE)       ; set transmitter interrupt
                        ;   disable and DTR enable
  STA ACIAHOST_+CMDR    ; update command register
  RTS                   ; end of serial break routine

  end
```

C.6 Ring Buffer Handler

The routine "SRStuff" places received data into a "ring" buffer if there is room to do so. This routine also handles host XON/XOFF synchronization, both for data reception and transmission.

```
;include <srdefn>
;
; Stuff Received Byte into Ring Buffer If Room is Available
;
; Note: This routine executes in "interrupts disabled" state
;
xref    ACIAHOST_
xref    SRPutX
;
xdef    SRStuff
SRStuff equ     *
  LDB ACIAHOST_+IOR     ; get data byte
  guess                 ; guess : special char
    CMPB #XON           ; - quit if CTRL/Q not received
    quif ne             ; -
    CLR SendLock        ; - o.k. to transmit more characters
  admit                 ; admit
    CMPB #XOFF          ; - quit if CTRL/S not received
    quif ne             ; -
    DEC SendLock        ; - stop transmission of characters
  admit                 ; admit
    TSTB                ; - quit if NULL character received
    quif eq             ; -
    CMPB #$7F           ; - quit if DEL character received
    quif eq             ; -
    LDX SRBuffer        ; - calculate ring buffer
    LEAX BFLEN,X        ; -   end address
    PSHS X              ; - save ring buffer end address
    LDY BfEPtr          ; - get ring buffer end pointer
    LEAX 1,Y            ; - increment it
    CMPX ,S             ; - if at end of ring buffer
    if eq               ; - then
      LDX SRBuffer      ; - - point at beginning of buffer
    endif               ; - endif
```

```
    LEAS 2,S              ; - pop ring buffer end address
    CMPX BfSPtr           ; - if ring buffer not full
    if ne                 ; - then
      STB 0,Y             ; - - save received character
      LDB ACIAHOST_+CMDR; - - get command register
      BITB #RID           ; - - if receiver interrupt not disabled
      if eq               ; - - (i.e. we really wanted this character)
        STX BfEPtr        ; - - - update ring buffer end pointer
        INC Count         ; - - - increment buffer use count
      endif               ; - - endif
      guess               ; - - guess that it's time to halt host
        TST TTSync        ; - - - quit if we cannot synchronize
        quif eq           ; - - -   data reception
        TST ReceiveLock ; - - - quit if we already sent XOFF
        quif ne           ; - - -
        LDB Count         ; - - - get buffer use count
        CMPB #BFLEN-BFHI; - - - quit if buffer is not near full
        quif lt           ; - - -
        DEC ReceiveLock ; - - - indicate that we have sent XOFF
        CLI               ; - - - enable interrupts since
                          ; - - -   we want to send
        LDB #XOFF         ; - - - transmit XOFF character
        JSR SRPutX        ; - - - to halt the host
      endguess            ; - - endguess
    endif                 ; - endif
  endguess                ; endguess
  RTS                     ; end of ring buffer stuffer

  end
```

C.7 Serial Definitions

The following set of "equates" is used by all of the above routines.

```
opt nolist
;
; Offsets to ACIA Registers
;
IOR     equ     0       ; ACIA Input/Output Data Register
STATR   equ     1       ; ACIA Status Register
CMDR    equ     2       ; ACIA Command Register
CNTLR   equ     3       ; ACIA Control Register
```

```
;
; Flags for ACIA Status Register
;
;    7      6      5      4      3      2      1      0
; +------+------+------+------+------+------+------+------+
; |      |  ___ |  ___ |      |      |      |      |      |
; | IRQ  | DSR  | DCD  | TDRE | RDRF | OVRN | FE   | PE   |
; +------+------+------+------+------+------+------+------+
;
PARITY  equ $01; Parity error (1=Error/0=No Error)
FRAMING equ $02; Framing error (1=Error/0=No Error)
OVERRUN equ $04; Overrun error (1=Error/0=No Error)
RDRF    equ $08; Receiver data reg. full (1=Full/0=Not Full)
TDRE    equ $10; Transmit data reg. empty (1=Empty/0=Not Empty)
DCND    equ $20; Data carrier detected (1=Not Detect/0=Detect)
DSNR    equ $40; Data set ready (1=Not Ready/0=Ready)
INT     equ $80; Interrupt occurred (1=Interrupt/0=No Interrupt)
;
;
; Flags for ACIA Command Register
;
;    7      6      5      4      3      2      1      0
; +------+------+------+------+------+------+------+------+
; |    PARITY CHECK    | NRML/| TRANSMITTER |RX INT| DTR  |
; |      CONTROLS      | ECHO |  CONTROLS   | E/D  | E/D  |
; +------+------+------+------+------+------+------+------+
;
DTRE    equ $01; Enable receiver / transmitter
;                0 = Disable Receiver/Transmitter / ___ \ high
;                1 = Enable Receiver/Transmitter  \ DTR / low
RID     equ $02; Receiver interrupt disable
RIE     equ $FD; Receiver interrupt enable mask (complement of RID)
;                0 = IRQ Interrupt Enabled from bit 0 of Status Reg
;                1 = IRQ Interrupt Disabled
TIE     equ $04; Transmitter interrupt enable
TID     equ $08; Transmitter interrupt disable
TBREAK  equ $0C; Transmit Break
;             +-----+----------+----------+----------------+
;             | BIT | TRANSMIT |          | OTHER          |
;             | 3 2 | INTERRUPT | RTS LEVEL |               |
;             +-----+----------+----------+----------------+
;             | 0 0 | Disabled | High     | -              |
;             | 0 1 | Enabled  | Low      | -              |
;             | 1 0 | Disabled | Low      | -              |
;             | 1 1 | Disabled | Low      | Transmit Break |
;             +-----+----------+----------+----------------+
```

```
ECHO    equ $10; Normal/Echo mode for receiver (0=NORMAL/1=ECHO)
P_NONE  equ $00; No parity     (--0x xxxx)
P_ODD   equ $20; Odd parity    (001x xxxx)
P_EVEN  equ $60; Even parity   (011x xxxx)
P_MARK  equ $A0; Mark parity   (101x xxxx)
P_SPACE equ $E0; Space parity  (111x xxxx)
;       +-------+-----------------------------------------------+
;       | BIT   | Operation                                     |
;       | 7 6 5 |                                               |
;       +-------+-----------------------------------------------+
;       | - - 0 | Parity disabled - No parity bit generated;    |
;       |       | No parity bit received                        |
;       | 0 0 1 | Odd parity Receiver and Transmitter           |
;       | 0 1 1 | Even parity Receiver and Transmitter          |
;       | 1 0 1 | Mark parity bit x'mitted; no rx parity check  |
;       | 1 1 1 | Space parity bit x'mitted; no rx parity check |
;       +-------+-----------------------------------------------+
;
```

```
;
; Flags for ACIA Control Register
;
;    7      6      5      4      3      2      1      0
; +------+------+------+------+------+------+------+------+
; | STOP |      WORD    |RX-CLK|         BAUD RATE         |
; | BITS |    LENGTH    | SRC  |         GENERATOR         |
; +------+------+------+------+------+------+------+------+
;
SP_0     equ     $00     ; 16x External Clock    (xxxx 0000)
SP_50    equ     $01     ; 50 Baud               (xxxx 0001)
SP_75    equ     $02     ; 75 Baud               (xxxx 0010)
SP_110   equ     $03     ; 109.92 Baud           (xxxx 0011)
SP_135   equ     $04     ; 134.58 Baud           (xxxx 0100)
SP_150   equ     $05     ; 150 Baud              (xxxx 0101)
SP_300   equ     $06     ; 300 Baud              (xxxx 0110)
SP_600   equ     $07     ; 600 Baud              (xxxx 0111)
SP_1200  equ     $08     ; 1200 Baud             (xxxx 1000)
SP_1800  equ     $09     ; 1800 Baud             (xxxx 1001)
SP_2400  equ     $0A     ; 2400 Baud             (xxxx 1010)
SP_3600  equ     $0B     ; 3600 Baud             (xxxx 1011)
SP_4800  equ     $0C     ; 4800 Baud             (xxxx 1100)
SP_7200  equ     $0D     ; 7200 Baud             (xxxx 1101)
SP_9600  equ     $0E     ; 9600 Baud             (xxxx 1110)
SP_19200 equ     $0F     ; 19200 Baud            (xxxx 1111)
EXT      equ     $00     ; External Receiver Clk (xxx0 xxxx)
BRD      equ     $10     ; Baud Rate Generator   (xxx1 xxxx)
WORD_8   equ     $00     ; Data Word Length 8    (x00x xxxx)
WORD_7   equ     $20     ; Data Word Length 7    (x01x xxxx)
WORD_6   equ     $40     ; Data Word Length 6    (x10x xxxx)
WORD_5   equ     $60     ; Data Word Length 5    (x11x xxxx)
STOP_1   equ     $00     ; 1 Stop Bit            (0xxx xxxx)
STOP_2   equ     $80     ; 2 Stop Bits           (1xxx xxxx)
;                          1 Stop Bit if Word Length = 8 Bits
;                            and Parity
;                          1 & 1/2 Stop Bits if Word Length =
;                            5 Bits and No Parity
```

```
;
; Zero Page Location For Serial I/O Variables
;
SRBuffer        equ     $00     ; - $01
; $02-$03 reserved
OverrunError    equ     $04
FramingError    equ     $05
ParityError     equ     $06
Key             equ     $07
ReceiveLock     equ     $08
SendLock        equ     $09
; $0a-$0b reserved
BfSPtr          equ     $0c     ; - $0d
BfEPtr          equ     $0e     ; - $0f
Count           equ     $10
; $11-$12 reserved
; $13-$14 reserved
HostSync        equ     $15
TTSync          equ     $16
SendInProg      equ     $17
Break           equ     $18
BreakUnderway   equ     $19
IRQHndlr        equ     $1A     ; - $1B
; $1C-$1D reserved
; $1E-$1F reserved
;
;
XON     equ     $11     ; ctrl q
XOFF    equ     $13     ; ctrl s
;
BFLEN   equ     $78     ; buffer length
BFLOW   equ     64      ; restart reception factor
BFHI    equ     16      ; halt reception factor
;
opt list
```

C.8 A Simple Terminal Emulator

The following is a listing of a simple terminal emulation routine (see Note below). It illustrates how the routines, shown above, may be incorporated into a terminal emulator.

```
;include <srdefn>
;
; Simple ASCII/APL Terminal Emulator
;
; Special Features:
; 1. XON/XOFF support for transmission to host (HostSync)
; 2. XON/XOFF support for reception from host (TTSync)
; 3. Break generation using STOP key
; 4. Scroll control using Shift/Keypad 0 key
; 5. Emulation exit using Shift/Keypad 8 key
; 6. Character set (ASCII/APL) support
;    using Shift In (SI) / Shift Out (SO) characters
; 7. CTRL key support using RVS key and other key (@,A,B,...,_)
;    (keys must be pressed one after the other, not simultaneously)
;
; Known Problems:
; 1. Wrap at column 80 may require host setup.
; 2. Holding down the SHIFT key generates continual interrupts
;    such that incoming characters may be lost.
; 3. The duration of the BREAK interval in SRBreak_ may need
;    to be increased for certain systems.
;
; To link this emulator, create and use the following command file:
;       term
;       org $A00
;       include "disk/1.watlib.exp"
;       export ACIAHOST_ = $EFF0
;       "emuterm.b09"
;       "srget.b09"
;       "srinth.b09"
;       "srirq.b09"
;       "srput.b09"
;       "srstuff.b09"
;       "srtie.b09"
;
        xref    Openf_
        xref    FGetChar_
        xref    Closef_
        xref    PutChar_
        xref    TBreak_
        xref    TSetChar_

        xref    SRTie_
        xref    SRConnect_
        xref    SRDisconnect_
        xref    SRGetChar_
        xref    SRPutChar_
        xref    SRBreak_

Echo    equ     $7f     ; Use location $7f for Echo control
Service equ     $32     ; System service code
Buffer_ equ     $400    ; Buffer for storing up received characters
Screen_ equ     $8000   ; Location of screen memory
SKD_    equ     $10e    ; Location of screen / keyboard descriptor
```

```
                                ; offsets from SKD_
CURSOR  equ     20              ; cursor : int
APLSET  equ     26              ; aplset : bool

KBDEL   equ     $04             ; Keyboard DEL
KBESC   equ     $06             ; Keyboard ESC
CTRL_G  equ     $07             ; ASCII Bell
CURUP   equ     $0b             ; SuperPET Cursor Up
CR      equ     $0d             ; Carriage Return
CTRL_N  equ     $0e             ; ASCII Shift Out
CTRL_O  equ     $0f             ; ASCII Shift In
CTRL_Q  equ     $11             ; ASCII DC1 (XOn)
CTRL_S  equ     $13             ; ASCII DC3 (XOff)
ESCAPE  equ     $1b             ; ASCII Escape
RUBOUT  equ     $7f             ; ASCII Del (Rubout)
KBCTRL  equ     $ff             ; Keyboard OFF/RVS

PFKEY   equ     $80                     ; Function keypad
KBEXIT  equ     PFKEY + 8               ; Keypad 8 (shifted)
KBHOLD  equ     PFKEY + 10              ; Keypad 0 (shifted)

ASCII   equ     1                       ; ASCII character set selection
APL     equ     2                       ; APL character set selection

xdef    EmuTerm_
EmuTerm_ equ    *
                                        ; * Start of parameter section
                                        ;   A more sophisticated terminal emulator
                                        ;   would allow User setting of the following:
        LDB     #$00                    ;   FALSE value (patchable)
        STB     Echo                    ;   Disable local echoing of typed characters
        LDB     #$FF                    ;   TRUE value (patchable)
        STB     HostSync                ;   Recognize host XON/XOFF synchronization
        LDB     #$FF                    ;   TRUE value (patchable)
        STB     TTSync                  ;   Perform terminal XON/XOFF synchronization
                                        ; * end of parameter section
        CLRB                            ; FALSE value
        STB     Hold                    ; Scroll control is off
        STB     PrevKey                 ; Last key typed is initialized NULL
                                        ; Openf( "KEYBOARD", "R" )
        LDD     #ReadMode               ; pointer to open mode string
        PSHS    D                       ; push parameter 2
        LDD     #KeyBoard               ; pointer to file name
        JSR     Openf_                  ; open keyboard
        LEAS    2,S                     ; pop parameter
        STD     KBIn                    ; save keyboard file pointer
        LDD     #Buffer_                ; get address of buffer
        JSR     SRTie_                  ; SRTie( Buffer )
        JSR     SRConnect_              ; SRConnect()
        LDB     #CTRL_Q                 ; SRPutChar( CTRL_Q)
        JSR     SRPutChar_              ; transmit XON in case host is in XOFF state
        loop                            ; loop (talk to host computer)
        JSR     TBreak_                 ; - check for STOP key
        TSTB                            ; - if TBreak()
```

```
if      ne              ; - then
JSR     SRBreak_        ; - - SRBreak()
endif                   ; - endif
LDD     KBIn            ; - Key := FGetChar( KBIn )
JSR     FGetChar_       ; -
STB     Key             ; -
CMPB    #KBEXIT         ; - quit if Key = KBEXIT
quif    eq              ; -
guess                   ; - guess
CMPB    #KBCTRL         ; - - quit if Key <> KBCTRL
quif    ne              ; - -
STB     PrevKey         ; - - remember CTRL toggle
admit                   ; - admit
CMPB    #KBHOLD         ; - - quit if Key <> KBHOLD
quif    ne              ; - -
LDA     #$FF            ; - -
STA     TTSync          ; - - TTSync := TRUE
COM     Hold            ; - - Hold := ~ Hold
admit                   ; - admit
TSTB                    ; - - quit if Key = NULLCHAR
quif    eq              ; - -
CLRA                    ; - -
STA     Hold            ; - - Hold := FALSE
BSR     TranOut         ; - - translate key before sending to host
TST     Echo            ; - - if local echoing
if      ne              ; - - then
BSR     LocalEcho       ; - - - LocalEcho ( Key )
endif                   ; - - endif
STB     PrevKey         ; - - PrevKey := Key
JSR     SRPutChar_      ; - - SRPutChar( Key )
endguess                ; - endguess
TST     Hold            ; - if ~ Hold
if      eq              ; - then
loop                    ; - - loop
JSR     SRGetChar_      ; - - - SRGetChar()
TSTB                    ; - - - quit if Key = NULLCHAR
quif    eq              ; - - -
BSR     LocalEcho       ; - - - echo received char on screen
TST     ReceiveLock     ; - - -
until   eq              ; - - until ~ ReceiveLock
endif                   ; - endif
endloop                 ; endloop
LDB     #CTRL_S         ; transmit XOFF in case host has more for us
JSR     SRPutChar_      ; SRPutChar( CTRL_S )
                        ; get out those last few characters
loop                    ; loop
JSR     SRGetChar_      ; - SRGetChar()
TSTB                    ; - quit if character = NULLCHAR
quif    eq              ; -
BSR     LocalEcho       ; - echo received char on screen
endloop                 ; endloop
JSR     SRDisconnect_   ; SRDisconnect()
LDD     KBIn            ; get keyboard file pointer
JSR     Closef_         ; Closef( KBIn )
```

```
            CLR     Service         ; set exit code
            RTS                     ; end of terminal emulation

TranOut     equ     *
            guess                   ; guess
            CMPB    #KBESC          ; - quit if Key <> KBESC
            quif    ne              ; -
            LDB     #ESCAPE         ; - Key := ESCAPE
            admit                   ; admit
            CMPB    #KBDEL          ; - quit if Key <> KBDEL
            quif    ne              ; -
            LDB     #RUBOUT         ; - Key := RUBOUT
            admit                   ; admit
            LDA     PrevKey         ; - quit if PrevKey <> KBCTRL
            CMPA    #KBCTRL         ; -
            quif    ne              ; -
            ANDB    #$1f            ; - Key := Key & $1f
            endguess                ; endguess
            RTS                     ; end of keyboard translation for output

LocalEcho   equ     *
            PSHS    B               ; save character
            CLRA                    ; zero hi part of D
            guess                   ; guess
            CMPB    #CR             ; - quit if Key <> CR
            quif    ne              ; -
            JSR     PutChar_        ; - PutChar( CR )
            LDB     #CURUP          ; - PutChar (CURUP )
            JSR     PutChar_        ; - counteract Linefeed
            admit                   ; admit
            CMPB    #CTRL_G         ; - quit if Key <> CTRL_G
            quif    ne              ; -
            BSR     Beep_           ; - ring the bell
            admit                   ; admit
            CMPB    #CTRL_N         ; - quit if Key <> CTRL_N
            quif    ne              ; -
            LDB     #APL            ; -
            JSR     TSetChar_       ; - APL character set selected
            admit                   ; admit
            CMPB    #CTRL_O         ; - quit if Key <> CTRL_O
            quif    ne              ; -
            LDB     #ASCII          ; -
            JSR     TSetChar_       ; - ASCII character set selected
            admit                   ; admit
            guess                   ; - guess APL character set & '_'
            TST     SKD_+APLSET     ; - - quit if not APL character set
            quif    eq              ; - -
            CMPB    #'F'            ; - - quit if not APL '_'
            quif    ne              ; - -
            LDX     SKD_+CURSOR     ; - - get cursor offset
            LEAX    Screen_,X       ; - - point at current character
            LDA     ,X              ; - - get character already there
            CMPA    #'a'+$80        ; - - quit if character < APL A
            quif    lo              ; - -
```

```
        CMPA    #'z'+$80        ; - - quit if character > APL Z
        quif    hi              ; - -
        TFR     A,B             ; - - use reverse video character
        endguess                ; - endguess
        CLRA                    ; - zero hi part of D
        JSR     PutChar_        ; - echo character on screen
        endguess                ; endguess
        PULS    B               ; restore Key
        RTS                     ; end of local echo routine


;
; BEEP the SuperPET Buzzer
;
; Registers:    A,B (D) destroyed
;               X       preserved
;
Freq    equ     $E848   ; frequency
Wave    equ     $E84A   ; waveform
Music   equ     $E84B   ; music on/off
;
xdef    Beep_
Beep_   equ     *
        pshs    X       ; save X reg
        LDB     #16     ; on=16
        STB     Music   ; turn on the music
        LDB     #$0f    ; get waveform 00001111 off/off/off/off/on/on/on/on
        STB     Wave    ; and set it
        LDX     #3      ; set counter
        loop            ; loop
        LDB     #55     ; - get frequency
        BSR     Play    ; - and play it
        LDB     #60     ; - get frequency
        BSR     Play    ; - and play it
        LEAX    -1,X    ; - bump counter
        until   eq      ; until count exhausted
        LDB     #00     ; off=00
        STB     Music   ; turn off the music
        puls    X       ; restore X reg
        RTS             ; return to caller

Play    equ     *
        STB     Freq    ; set frequency
        LDD     #0      ; set counter
        loop            ; loop
          ADDD  #1      ; - increment
          CMPD  #$0BFF  ; - check for duration
        until eq        ; until duration over
        RTS             ; end of play routine

KeyBoard fcc    "KEYBOARD"
        fcb     0

ReadMode fcc    "R"
```

```
        fcb     0

KBIn    rmb     2       ; keyboard file pointer
Hold    rmb     1       ; hold screen (scroll/no scroll toggle)
PrevKey rmb     1       ; previously typed key

        end
```

Note:

A more sophisticated terminal emulator is available from WATCOM Products Inc. For more information, write to:

WATCOM Products Inc.,
415 Phillip St.,
Waterloo, Ontario,
CANADA N2L 3X2

Appendix D

EIA RS-232C Pin Assignments

The following is a list of some of the 25 pins defined by the RS-232C standard.

Pin No.	EIA Desig.	CCITT Desig.	Description	Abbrev.
1	AA	101	Protective Ground	PG
2	BA	103	Transmit Data	TxD
3	BB	104	Receive Data	RxD
4	CA	105	Request to Send	RTS
5	CB	106	Clear to Send	CTS
6	CC	107	Data Set Ready	DSR
7	AB	102	Signal Ground	SG
8	CF	109	Data Carrier Detect	DCD
12	SCF	122	Secondary Data Carrier Detect	SDCD
13	SCB	121	Secondary Clear to Send	SCTS
14	SBA	118	Secondary Transmit Data	STxD
15	DB	114	Serial Clock Transmit	SCT
16	SBB	119	Secondary Receive Data	SRxD
17	DD	115	Serial Clock Receive	SCR
19	SCA	120	Secondary Request to Send	SRTS
20	CD	108.2	Data Terminal Ready	DTR
21	CG	110	Signal Quality Detect	SQD
22	CE	125	Ring Indicator	RI
23	CH	111	Data Rate Select (DTE source)	DRS
	CI	112	Data Rate Select (DCE source)	
24	DA	113	External Transmit Clock	EXT

Note:

CCITT is another standards organization- the International Telegraph and Telephone Consultative Committee (a unit of the International Telecommunications Union). CCITT's standards are V.24/V.28 and are the international counterpart of the U.S. Electronics Industries Association's RS-232C standard.

Appendix E

Glossary

The following are a few of the glossary items published in the "Terminal and Communications Handbook" published by Digital Equipment Corporation (see References). The handbook contains several very informative chapters. Among them are Chapter 10 - "Basic Concepts of Data Communication" and Chapter 19 - "Communications Glossary".

asynchronous transmission Transmission in which time intervals between transmitted characters may be of unequal length. Transmission is controlled by start and stop elements at the beginning and end of each character. Also called start/stop transmission.

BAUD A unit of signaling speed equal to the number of discrete condition or signal events per second. In asynchronous transmission, the unit of signaling speed corresponding to one unit interval per second. Baud is the same as "bits per second" only if each signal event represents exactly one bit.

carrier A continuous frequency capable of being modulated or impressed with a signal.

CCITT Comite Consultatif Internationale de Telegraphie et Telephonie.

data communications equipment (DCE) The equipment that provides the functions required to establish, maintain, and terminate a connection, the signal conversion and coding required for communication between data terminal equipment and data circuits. The data communication equipment may or may not be an integral part of a computer (e.g., a modem). DCE has lately been enhanced to mean "data circuit-terminating equipment", a more general term.

control procedure The means used to control the orderly communication of information between stations on a data line. Also called: line discipline.

data terminal equipment (DTE) 1) The equipment comprising the data source, the data sink, or both. 2) Equipment usually comprising the following functional units: control logic, buffer store, and one or more input or output devices or computers. It may also contain error control, synchronization, and station identification capability.

full duplex Simultaneous two-way independent transmission in both directions.

half duplex A circuit designed for transmission in each direction but not in both directions simultaneously.

host computer A computer attached to a network providing primarily services such as computation, data base access, or special programs, or programming languages.

mark Presence of a signal. In telegraphy, mark represents the closed condition or current flowing. Equivalent to a binary one condition.

modem (modulator-demodulator) A device that modulates signals transmitted over communications circuits. Also called "data set".

modulation The process by which some characteristic of a high frequency carrier signal is varied in accordance with another, a lower frequency "information" signal. This technique is used in modems to make business machine signals compatible with communications facilities.

null modem A device which interfaces between a local peripheral that normally requires a modem, and the computer near it that expects to drive a modem to interface to that device.

protocol A formal set of conventions governing the format and relative timing of message exchange between two communicating processes. See also: control procedure.

serial transmission A method of transmission in which each bit of information is sent sequentially on a single channel rather than simultaneously as in parallel transmission.

start element In start/stop transmission, the first element in each character, which serves to prepare the receiving equipment for the reception and registration of the character.

start/stop transmission Asynchronous transmission in which a group of code elements corresponding to a character signal is preceded by a start element and is followed by a stop element.

stop element In start/stop transmission, the last element in each character, to which is assigned a minimum duration, during which the receiving equipment is returned to its rest condition in preparation for the reception of the next character.

Appendix F

References

The following references provide additional insight into serial communications.

[1] Asynchronous Communication Interface Adapter - SY6551, Synertek 1981-1982 Data Catalog, pp. 3-169 to 3-176, Synertek Incorporated, P.O. Box 552-MS/34, Santa Clara, California, U.S.A. 95052.

[2] Terminals and Communications Handbook 1981-82, Digital Equipment Corporation, New Products Marketing, PK3-1/M92, Maynard, Massachusetts, U.S.A. 10754.

[3] "The Lowly Modem", George M. Dick, Datamation, Vol. 23, No. 3 (March 1977), pp. 69-73.

[4] "Line Control Procedures", James P. Gray, Proceedings of the IEEE, Vol. 60, No. 11 (Nov. 1972), pp. 1301-1312.

Articles [3] and [4] are reprinted in "Computer Networks: A Tutorial", Abrams, Blanc and Cotton, Third Edition - 1980, IEEE Catalog No. EHO162-8 or IEEE Computer Society Catalog No. 297. Copies may be obtained from

Computer Society Publications Office,
5855 Naples Plaza, Suite 301,
Long Beach, California, U.S.A. 90803

or

IEEE Service Centre,
445 Hoes Lane,
Piscataway, New Jersey, U.S.A. 08865

The WATCOM Group Inc.
WATCOM Products Inc.
WATCOM Systems Inc.
WATCOM Seminars
WATCOM Publications Ltd.

415 Phillip Street
Waterloo, Ontario, Canada
N2L 3X2
(519) 886-3700
Telex 06-955 458