Owner's Manual

Model 7710

Asynchronous Serial Interface

California Computer Systems

CALIFORNIA COMPUTER SYSTEMS

APPLE II™ ASYNCHRONOUS SERIAL INTERFACE

MODEL 7710A

OWNER'S MANUAL

TABLE OF CONTENTS

89000-07710B

California Computer Systems
250 Caribbean Drive
Sunnyvale, CA 94086
USA

INTRODUCTION

Congratulations! In purchasing this small, 2.75" X 5" Asynchronous Serial Interface card, you acquired a very powerful accessory to your Apple II~ computer. Now your computer can "talk" to many types of computer peripherals, such as video terminals, line printers and paper tape units, or even to another computer. The only requirement is that the peripheral possess an electrical interface point which complies with the Electronic Industries Associations (EIA) RS-232C interface definition in any configuration up to "E".

Take a few minutes to study this manual ANL the manual for the device you wish to connect to your computer before you install the interface card, power up the computer, and expect it to work. While this card is easy to use, it must be configured to your peripheral properly if they are to talk to each other.

I. THEORY OF OPERATION

Before we go into any detail about the interface card's theory of operation, let's discuss definitions and why we need this kind of an interface.

A computer is a very expensive do-nothing unless you can give it data, instruct it what to do with the data and then have it present the results. To help do this, peripheral devices were designed to read and/or store the data (and instructions),then transfer it to the computer. Unless the peripheral was designed specifically to interface to your computer, you still had a problem. Your computer, for instance, expects 8 bits of data (one byte) to be transferred at one time through its peripheral connecters. Further, certain "handshake" signals are exchanged which tell the receiving unit when it can have the data. The receiver can then answer back, "I got it" or "Wait a minute, I'm busy". This type of interface works, but has its limitations. First, it requires the computer and the peripheral to be perfectly matched to each other; and second, all eight data bits must be available at the same time. In the first case, peripheral manufacturers cannot afford to engineer a special interface to every computer which might use their product. The second problem limits the length of cables connecting the computer and the peripheral. Nothing's perfect! In this case, the longer the cable, the more distorted the signals are until they become unusable. If so, what happens when the two devices are in different cities or states? Who can afford the ten or so telephone lines needed for this eight bit parallel data transfer?

Many years ago, the telegraph solved the last problem by devising a method of encoding messages in bits of information and then transmitting them long distances, one bit at a time through one wire. The receiving end then had to reassemble the bits of information into the message. This same idea of serial by bit data transfer we use with computers.

To make this scheme work, the sender must have a method to convert the parallel data into serial data, while the receiver must be able to reconvert the serial data to parallel. This presents a problem for the receiver because the serial data is nothing more than a continuous stream of ones and zeroes. How can the receiver tell which 8 bit group is the correct group of data? Two schemes have been devised to solve this problem: Synchronous and Asynchronous modes of transmission. During synchronous transmissions, a predefined pattern of synchronization bits are sent out first. When the receiver finds this pattern, it "locks" onto it. Then it automatically divides up all subsequent bits into 8 bit groups for the rest of the transmission. Asynchronous interfaces handle the

problem differently. They add a START BIT to the front of the bit groups, plus one to two STOP BITS at the tail. This total group of bits is then sent, one bit at a time, over the interfaces. The computer has no trouble identifying the data bits, since they are a fixed number of mixed bits appearing between the the logical zeros of the start bits and the logical ones of the stop bits.

To transmit the serial data over long distances, you need a Modulator/Demodulator, or Modem. Modems need the serialized data and also some "handshaking" control signals from the computer. Since this is the interface point between computer and communications equipment, the EIA created the RS-232C interface specification to allow manufacturers of either type of equipment to know what to expect from the "other side". For short distances (less than several hundred feet), the modems and telephone wires are not needed. The catch here is that one side of the interface must be made to think it's a modem, or Data Communications Equipment (DCE), as RS-232C calls it. Because modems usually connect to some type of computer terminal,the RS-232C calls the other side of the interface the Data Terminal Equipment (DTE).

RS-232C defines the necessary protocol between the DCE and the DTE for many possible configurations. Of these, the first five (Configurations A, B, C, D, and E) are most commonly used. Configuration A defines a one way, transmit only interface. A simple serial keyboard might use this type. Configuration B is also a one way, transmit only; but it has more "handshaking." A paper tape reader would effectively use this type. Configuration C is also one way, but this time it is receive only. Here, a serial printer is a logical candidate. In Configuration D, we start getting into two way traffic. This one is HALF DUPLEX link. Although it can carry traffic both ways, it can only go one way at a time.

When a modem is used in this configuration, it is often called a "two wire modem" because the telephone line is connected with only two wires. This type of interface is not often used. Configuration E is a FULL DUPLEX link. This means that the link can support traffic in both directions at the same time. CRT terminals most often use this type of link. If a modem is used, it is called -- guess what? -- a "four wire modem." In any case, our interface card supports all five of these configurations. One additional "handshake" line, the Data Terminal Ready (DTR) line, has also been provided on this card. Normally, this line is used only on synchronous interfaces, but some types of serial printers make use of this line.

One more problem needs solving before our interface is functionally complete: How fast should each bit be sent? Common usage has defined many standard signalling rates. Usually, they are an even multiple of 75 baud (bits per second, including all overhead bits, such as start and stop bits). A few other rates are used by the industry giants. These rates are 50, 110, and 134.5 baud. RS-232C does not specify any standard signalling rates, but suggests a practical upper limit of 20 kilobaud and indirectly establishes a theoretical upper limit of 50 kilobaud.

Now that we've discussed the general principles of serial data transmission, let's discuss the specific workings of the CCS Asynchronous Serial Interface. The card can be divided into four sections: a) the transmitter/receiver section; b) the baud rate generator clock; c) the control section; and d) the interface program memory.

a) Transmitter/Receiver Section

Here is the heart of the interface, for this section does the parallel to serial and serial to parallel data conversion. It adds the start/stop bits when transmitting, removes them when receiving, talks to the computer in 8 bit bytes, and provides the necessary line drivers/receivers for the serial interface to comply with the RS-232C electrical specifications.

The major component of this section is a 6850 Asynchronous Communications Interface Adapter, or (ACIA) for short. This integrated circuit provides the data formatting and the control needed to interface the serial asynchronous data information to your computer. The computer parallel data is serially transmitted and received, formatted properly, and checked for errors by the ACIA. The functional configuration of the ACIA is programmed via the parallel data bus during initialization of the chip. A programmable control register allows specification of variable word lengths, clock division ratios, transmit control, receive control, and interrupt control. On the serial side, the ACIA provides three control lines for peripherals/modems. One of these lines, in particular, allows the peripheral to tell the computer to slow down, to not send the data so fast. This line is the Data Terminal Ready (DTR) signal, which appears on pin 20 of the RS-232 connector. The ACIA will stop transmitting when DTR goes low (more negative than -3 volts) at the RS-232 connector. When DTR goes high (more positive than +3 volts), the ACIA will resume transmitting.

The ICs 75154 and 75150 perform the line receiver and line driver functions, respectively. They merely translate the standard Transistor-Transistor Logic (TTL) signals of the ACIA to the +5 volt to -5 volt (nominal) signal levels required by RS-232C. They also provide for fail-safe operation should your interface cable short or disconnect accidently.

Again, note that your CCS Asynchronous Serial Interface implements the DCE side of the RS-232C interface. This allows direct connection to most peripherals, i.e. video terminals, serial printers, combination keyboard/printers, and other similar serial devices. If you examine the ACIA's specification sheet, you may think it conflicts with our module's use certain ACIA control lines. There is no conflict: The ACIA's control lines were named on the assumption it would be used in a DTE device. Since we are using it in a DCE device, the roles of these pins change. For example, the DTR line mentioned above is connected to the ACIA's Clear to Send (CTS) pin.

Before the ACIA can send or receive data bits, it needs a clock to tell it how often to send or receive each bit. So let's discuss the clock next.

b) Baud Rate Generator Clock Section

Different serial devices require the serial data to be clocked at many different rates. To allow the greatest flexibility, we have provided a 4702 baud rate generator. This chip contains both an oscillator for the on-board quartz crystal and a controllable frequency divider circuit.

The oscillator generates a highly stable 2.4576 megahertz square wave signal. This signal is then counted down by a divisor determined by the settings of the four switches on the card. The output of the 4702 is 16 times the actual selected baud rate. It is connected to the ACIA at the transmit and receive clock pins. The ACIA can and must be programmed to expect a x16 clock. Note that the 4702, by itself, can only generate 13 different baud rates, from 50 (x16) to 9600 (x16) baud, although the switches allow us 16 different choices. One of these choices is taken away because the 4702 will generate a 2400 (x16) clock on two different switch settings. The remaining two settings specify an external clock source. One is used to generate a 19,200 (x16) clock. Normally, the 4702 prescales the 2.4576 Mhz signal by dividing it first by 16 to yield a 9600 (x16) signal. The prescaler, however, allows us a connection to the signal after it has been divided by 8. Thus, we now have a 19,200 (x16) baud signal available. The last setting allows a signal from pin 24 of the RS-232C connector to be used as the clock. This allows the

terminal (or any other source you may desire) to generate the clock.

Clock rates of up to 50,000 baud may be used. Although the ACIA and 75154 will respond to higher clock rates (up to about 500,000 hz), clocks above 50 kbs are non-compliant with RS-232C.

c) Control Section

The control section of the interface consists primarily of a buffer between the ACIA and the computer data lines, and some interrupt arbitration logic. The ACIA has only limited ability to drive the data lines of the computer. To insure that the data successfully passes to and from the computer and the ACIA, we have placed a bidirectional line buffer (the 8304) between them. This IC can drive the bus with enough power to insure good data transfer. To provide this power, though, the chip must consume a lot of power. Since the driver is used for only short periods of time, a power down feature has been added to conserve power. This transistor monitors the DEVICE SELECT line of the slot from the computer. If the DEVICE SELECT goes true, the transistor turns on power to the 8304 allowing the computer to talk with the ACIA. When it is finished talking, the computer removes the DEVICE SELECT signal. The transistor, seeing this, then turns off the power to the 8304. As a result, more power becomes available to the other peripheral controls. The 8304 monitors the R/W (Read/Write) line to determine which way to pass the data.

The interrupt arbitration logic is but one link in the interrupt daisy chain. The entire daisy chain prioritizes peripheral generated interrupts insuring that only one device interrupts the computer at a time. If this board wants service from the processor, it can present an interrupt request at arbitration logic. If no higher priority interrupt is in progress (connecter pin 28 active high), then an interrupt request is issued to the computer (connecter pin 30). Concurrently, pin 23 is driven low, maintaining the interrupt daisy chain and signalling lower priority devices that an interrupt is pending, forcing them to wait their turn. After being serviced, the ACIA removes its interrupt request, and the arbitration logic sets the daisy chain signal high again. Now the lower priority devices can request interrupts as they need. Note that the highest priority device should be installed in the leftmost peripheral connecter in the group 1 through 7 in your computer. Slot 0 does not support the daisy chain. Also, empty slots are NOT allowed between the cards because they break the daisy chain.

d) Control Program Memory

Your computer dedicates 256 bytes of memory space to each peripheral connector. This space allows sufficient room for most interface unique software or firmware. The CCS card has provisions to place two 256 x 4 bit Read Only Memories (ROMs) on the card to contain your interface firmware. Since these ROMs consume quite a bit of power, they are also equipped with a power down feature like the one described in the control section. To save you the trouble of developing your own firmware, CCS offers several standard firmware packages to control the interface. Should you desire to develop your own software, this board allows installation of Random Access Memories (RAMs). CCS offers a "RAM-Pak" to support this feature. Of course, the memory power down feature must be disabled and the R/W control line enabled to the RAMs. This is done by installing one jumper wire on the interface card. By using the RAMs, you can fully develop and test your interface program thoroughly before committing it to ROMs.

Either type of memory is read by the computer which provides an address on address lines A0 - A7. Then the computer enables the memories by making the specific peripheral connector I/O SELECT line active. The memories then fetch the content of the specified address and offer it to the computer on the data lines D0 - D7. The computer then reads it and does with it as it would with the content of any other memory location. If you use RAMS, the computer can write into these dedicated locations just as if they were regular memory.

Tying all of the parts together, we find that the program memory provides necessary instructions to the computer on how to use the interface. This program must first initialize the ACIA for operation, then cause the computer to give data to and get data from the ACIA. Once the ACIA gets some data from the computer, it adds the start/stop bits and sends the whole group out, one bit at a time. If so commanded, the ACIA will interrupt the computer whenever it can take another byte of data. The receiver watches the incoming serial data line for a start bit. When it finds one,it assembles the bits that follow until the stop bit arrives. It then checks the data for errors and notifies the computer of the data's readiness to be read and its error status. If so commanded, the receive section will also generate interrupts. The clock tells the ACIA how fast to send out or receive each bit of data on the serial side of the interface. Finally, the control logic arbitrates the interrupt requests and makes sure the computer can read the ACIA's data or status.

II. INSTALLATION AND CHECKOUT

Now that we have an understanding of how the card works, let's try it out. The ultimate test is using it in its actual operating environment. But before we can plug in the card and expect it to work, we need to match the interface card to the peripheral.

Selecting the proper baud rate

Your peripheral manual should tell you at which baud rate or rates the peripheral will operate. If it can handle several baud rates, select the highest one available which is common to the interface card. Now set your terminal for that baud rate, following its instructions. Next, set the CCS card switches to match. With the card in front of you, position it so the switches are in the upper right hand corner. Find your baud rate in the following table, and set the switches as shown for that baud rate. In this table, o means to push that side of the rocker switch down.

TABLE 1 - Baud Rate Selector Switch

		1234			1234
50 Baud	ON	:o oo:	75 Baud	ON	: oo:
	OFF	: o :		OFF	:oo :
110 Baud		: :	134.5 Baud		:oo o:
		:oooo:			: o :
150 Baud		:o :	200 Baud		: o o:
		: ooo:			:o o :
300 baud		: o :	600 Baud		:o o:
		:o oo:			: oo :
1200 Baud		: o :	1800 Baud		:o o :
		:oo o:			: o o:
2400 Baud		:oo :	2400 Baud		: o:
		: oo:			:ooo :
4800 Baud		: oo :	9600 Baud		:ooo :
		:o o:			: o:
19200 Baud		: ooo:	External Clock		:oooo:
		:o :			: :

Other Set up for the Peripheral

To make your terminal and computer function as a system, you must set up several other items on the terminal. For instance, if it allows you to select between half or full duplex operation, select the FULL DUPLEX option. Your computer's firmware expects a full duplex keyboard/display. What this means is that the computer will, after receiving a character from the keyboard, send that character back out to the display. This "echo" feature allows a quick check that the computer really received what you wanted it to get. When a terminal is in a half duplex mode, however, it will display the character as it is typed in. Then, after the computer echoes the character, it gets displayed a second time. As an example, if you typed in "RUN" with the terminal in a half duplex mode, you would see "RRUUNN" on the display. Cure this ill by setting the terminal for full duplex operation.

Next, your computer does not generate any Line Feed control characters. It expects your terminal to do this each time a Carriage Return (Control D) is sent. If your terminal has an Auto Line Feed option, use it. Otherwise, you will have to program the interface software to do it.

Your computer expects all alphabetic characters to be in upper case. If the terminal has an Upper Case Only option, use it. Of course, the interface software can also be programmed to convert all lower case characters to upper case. An example of how to do so is in the interface software listing.

The ACIA must be commanded to match the parity option selected in the terminal. Or, conversely, the terminal's parity option needs to be set to match what the ACIA has been commanded to generate. Data transfer usually is so reliable on asynchronous links that no parity generation or checking is needed.

Card Installation

Now we're ready to install the interface card in the computer. First of all, plug the I/O cable (W1) onto the card. Align pin 1 of the cable connector with pin 1 of the mating connector on the PC board. Pin 1 can be identified by the outside stripe on the cable; it can be identified also by a triangular mark or other type tick on the dual 13-pin connector. When all pins are properly aligned, push down firmly on the connector until you can no longer see the metal connector pins. Gently fold the ribbon cable on the diagonal toward the ROMs/RAMs. Crease the fold only slightly; too much crease might fatigue and break the

the wires in the ribbon. Now gently fold the ribbon back under itself, so that the back panel connector points to the right of the board. This slack in the cable is needed for strain relief. The card is now ready to be inserted into the computer.

```
**********************************************************
*                                                        *
* WARNING: Do not remove the computer top cover if       *
* the line power cord is plugged in. Damage may          *
* result to you and your computer.                       *
*                                                        *
**********************************************************
```

Place the computer directly in front of you. Remove the top cover by placing the palms of your hands on the back edge of the computer, with your fingers hanging over the rear. Curl your fingers around the rear edge until you feel a ridge at your finger tips. Gently but firmly pry up until you hear two distinct "pops." Don't lift the cover any further. Slide it to the rear to remove it from the computer. Toward the inside rear of the computer, you will see eight 50 pin connecters. From left to right, they are numbered #0, #1, ... ,#7. Place the CCS Asynchronous Serial Interface card into any of these connectors except #0, the leftmost; it is reserved for other use. We suggest you use slot #2, if it is not already occupied. Insert the card by holding it and the cable so that the component side of the card is to the right and the cable assembly is to the rear. Align the card edge into the chosen connector; then gently push the card down until it is firmly seated. Now slide the cable assembly through the nearest back panel slot, and replace the cover on the computer. Plug one end of a signal cable (you supply this) to the external connector and the other end to the appropriate place on your peripheral. Finally, plug in the line power cord, and we're ready to test the module.

CHECKOUT

The best test for any computer hardware is its actual use. There are some tests we should perform first, though, to be confident that the interface will really work. For the remainder of this section, we will assume the interface card is in slot #2. If you put it in a different slot, you will have to modify the tests accordingly.

Before you test the board's operation, you need to know what type memory you have on your board. The CCS Asynchronous Serial Interface card allows you the option of installing either ROMs or RAMs. We do not supply either assembled boards or kits with RAMs in them; if your kit or board came supplied with memory ICs, they are ROMs. The only way you could have RAMs on your board is if you purchased them separately and installed them yourself. Use only the test below that is designed for your type memory.

Test 1 - ROM Test (if installed)

This test displays the contents of the ROMs on the TV screen. From there, you can compare the contents with the ROM program listing. This verifies that the ROMs are properly installed and can be read by the computer. Note: Your computer's disassembler cannot recreate any assembler pseudo operation codes, such as ORG or EQU. Occasionally, use of the ORG instruction could hide an instruction from the disassembler. For instance, the code:

```
BCS     *
ORG     *-1
SEC
```

will disassemble as

```
BCS     *+$38
```

Watch for this kind of programming trick when comparing the listings. It is valid code, but may make you think you have bad ROMs. Programming tricks such as this are used to conserve memory in tight situations.

a. Turn on and Reset your computer.
b. Type in C200L (CR).
c. Compare program listing to the TV display.
d. When you run out of screen display, type in L (CR).
e. Repeat c. and d. until all 256 bytes of ROM are read.
f. If problems result, compare the hexadecimal values of the memory locations. The ROMs may be reversed on your board. If not, see your CCS dealer.

Test 2 - RAM Test (if installed).

This test verifies that we can read and write to all locations of the program RAMs. It copies a 256 byte segment of your computer's firmware into the RAMs, then compares this copy to the original. Any errors will displayed on the TV screen.

a. Turn on and Reset the computer.
b. Type in C200<F000.F0FFM (CR).
c. Type in C200<F000.F0FFV (CR).
d. A * should appear almost immediately on the screen if all is OK. If it doesn't, check that you've installed the RAM jumper under U10. Otherwise, see your CCS dealer.

Test 3 - Serial Data Loop Test.

This test checks out the ACIA, the clock, and the line drivers. It does this by sending out a known byte of data, looping it back to the receive section, reading the data and comparing the result.

For this test, we will need a "loop-back" test fixture. This may be made by taking a standard DB-25S socket which mates to the end of the signal cable and wiring pins 2 and 3 together, pins 4 and 5 together, and pins 8 and 20 together. This fixture allows the transmitted data to be looped back into the ACIA receiver.

a. Disconnect the signal cable from the peripheral device.
b. Install the loop-back test fixture onto the end of the signal cable.
c. Insure that the External Clock is NOT selected on the card's switches.
d. Turn on and Reset your computer.
e. Type in C0A0:03 (CR) to reset the ACIA.
f. Type in C0A0:11 (CR) to initialize the ACIA operating mode.
g. Type in C0A1:55 (CR) to write to the ACIA an alternating bit pattern.
h. Type in C0A1 (CR) to read the receive data.
i. Compare to see if the display from h matches what was sent in step g.
j. Repeat Steps g, h, and i using AA in place of the 55 in g.
k. Repeat Steps f thru i, using different baud rates, ACIA commands, and data patterns until you are satisfied that the interface works properly.
l. If you have any problems, contact your CCS dealer.

After completing Test 3, turn off the power, disconnect the test fixture, and reconnect the peripheral. If you changed the baud rate, set it back to the correct rate. If you have ROMs on your board, you are now ready to use your CCS Asynchronous Interface. If you installed RAMs instead, you are now ready to develop your custom interface software.

Notes:

III. INTERFACE SOFTWARE/FIRMWARE

The CCS Asynchronous Serial Interface is a very flexible device. It obtains this flexibility by achieving a balance between the hardware and its interface software. The hardware takes care of most of the tasks which are the same, regardless of use. The software is left to do what it does best, the application unique tasks. With this "personality" contained in the software, it is impossible to create one program which is everything to everybody. CCS offers some standard "ROM-Paks" which should meet most of your needs. Review their specifications carefully. If one fits your needs, then use it. If not, then read on. In this section, we will provide you with the information needed to control the ACIA, including many of your computer's firmware "hooks" for peripheral operation and a sample program which allows the transfer of your computer's keyboard function to a terminal's keyboard, or the display of information on the terminal instead of the TV set, or total control/display of your computer from the terminal.

First, let's take a closer look at the ACIA as seen by the computer. From our earlier discussion, we saw that the computer must be able to write or read to/from the ACIA, to tell the ACIA how to work with the data, and to read the ACIA status to find out what is happening in the ACIA.

Your computer dedicates 16 memory addresses to each peripheral connector slot (except slot #0) for the memory mapped input/output. These 16 addresses are above and beyond the 256 dedicated program memory addresses. The I/O addresses are located at $C0xy where:

x = 8 + n
n = the peripheral slot number (= 1, 2,...,7)
y = the specific address (= 0, 1,...,$E, $F)

In our specific case,

$C0x0 (write) = the ACIA command register
$C0x0 (read) = the ACIA status register
$C0x1 (write) = the ACIA transmit data register
$C0x1 (read) = the ACIA receive data register

NOTE: The last address digit (y) is not decoded beyond even or odd. This means that data can be passed on any odd address within the range, while the ACIA's command/status registers can be accessed on any even address.

What commands are valid for the ACIA? Here, we must look at each bit of the command byte:

```
Bit 7654 3210
    xxxx xx00: The clock is 1x the baud rate
    xxxx xx01: The clock is 16x the baud rate
    xxxx xx10: The clock is 64x the baud rate
    xxxx xx11: ACIA Master Reset
    xxx0 00xx: 7 data + Even parity + 2 stop bits
    xxx0 01xx: 7 data + Odd parity + 2 stop bits
    xxx0 10xx: 7 data + Even parity + 1 stop bit
    xxx0 11xx: 7 data + Odd parity + 1 stop bit
    xxx1 00xx: 8 data + No parity + 2 stop bits
    xxx1 01xx: 8 data + No parity + 1 stop bit
    xxx1 10xx: 8 data + Even parity + 1 stop bit
    xxx1 11xx: 8 data + Odd parity + 1 stop bit
    x00x xxxx: Set CTS active
               Disable transmit interrupts
    x01x xxxx: Set CTS active
               Enable transmit interrupts
    x10x xxxx: Set CTS inactive
               Disable transmit interrupts
    x11x xxxx: Set CTS active
               Transmit break on transmit data
               Disable transmit interrupts
    0xxx xxxx: Disable Receive Interrupts
    1xxx xxxx: Enable Receive Interrupts on:
               -Receiver data register full
               -Receive data overrun
               -RTS signal went inactive
```

NOTE: In this card, the Data Carrier Detect line from the ACIA has been connected to the Request To Send (RTS) line at the RS-232C interface. As a result, whenever your terminal makes the RTS line inactive and if you have the receive interrupts enabled, the computer will be interrupted. Also, in the coding for bits 5 and 6 of an ACIA command, the ACIA's RTS pin is connected to the Clear To Send (CTS) line at the RS-232C connector. As was noted earlier, the last ACIA handshake line, CTS, is connected to the RS-232C DTR line. Keep these changes in mind when using the ACIA's data sheets to obtain more information on the ACIA.

Typical commands which are used in the sample program are:

```
0000 0011 = $03: ACIA Master Reset
0001 0001 = $11: Make CTS active
                 Disable all interrupts
                 8 data + no parity + 2 stop bits
                 x16 clock.
```

The Status Register Bits, if = 1, mean:

Bit 0: Receive data is ready for the computer.
1: The transmit data register can take another byte of data.
2: The RTS signal went inactive, (see the above note on RTS usage).
3: The Data Terminal Ready signal is inactive, don't send data.
4: A Framing Error occurred on receive data.
5: Receiver Overrun - another byte of data was received before the previous byte was read; the previous byte is lost.
6: Parity Error occurred on the receive data.
7: The ACIA generated either a transmit or a receive interrupt.

For a more complete discussion of ACIA command and status structures, see a 6850 ACIA data sheet.

Now, let's look at some of your computer's firmware "hooks". For instance, the computer looks at two page 0 locations to find out where the current keyboard handler and the console output driver programs are located. These locations are:

$36 - $37: Address of console output handler
$38 - $39: Address of keyboard input handler

When you type in the BASIC command, IN#n, the firmware writes a 00 in location $38 and a $Cn in location $39. The equivalent monitor command, n[Cntl]K, also does the same thing. This creates an effective address of $Cn00 for the keyboard handler initialization program. Then, the next time any keyboard input is wanted, the initialization routine gets called. It must set everything up, then pass control to the input routine to actually do the input. Part of the initializer's tasks is to change location $38 to identify the input handler's correct entry point. Then, the next time input is wanted, we can go straight to the input routine. We don't need to reset up everything again. Likewise, when OUT#n (or n[Cntl]P) is typed in, location $36 is set to 0 and $37 set to $Cn. On the first output, control will be passed to Cn00 for output initialization. Location $36 must then be set to match the output handler's entry point for all subsequent console output.

Input and output with your computer is handled on a byte by byte basis. As a result, the input or output data can passed between the handlers and the computer in the accumulator (A register). So, for input, the data should be left in the accumulator when control is returned to the caller. On output, the handler can find the data in the accumulator.

III. INTERFACE SOFTWARE/FIRMWARE

The input and output routines will be called as subroutines. Control can be returned to the caller by issuing a simple, "RTS" (Return from Subroutine) instruction. Good programming practice says to save all register contents on entry to subroutines, then restore the registers to their original contents just before leaving a subroutine (except for parameter passing registers). This practice saves many head aches and program debugging time later on.

What about scratch pad memory, if we need it? The video display refresh memory locations (addresses $400 - $7FF) use only the first 120 of every 128 locations for the display data. The remaining 24 addresses can be used for other purposes. Use them cautiously, though. Some other programmer may have beaten you to them. Two sets of locations which are available include $6F(n+8) and $77(n+8) where n is the slot number. For most programs, these should be enough to save, for instance, the last issued ACIA command, etc.

We now have enough information to program a simple remote console interface program. Our program will consist of three parts: initialization, input, and output.

For initialization, we must:

a. Save the registers;
b. Reset the ACIA;
c. Initialize the ACIA with the correct command word;
d. Establish the proper input and/or output entry point;
e. Allocate and/or initialize any special pointers, counters, etc;
f. Go to Step b of the appropriate I/O routine, depending if input or output is desired.

For input, we must:

a. Save the registers;
b. Check the ACIA status and wait until the input data is ready;
c. Read the input data;
d. Do any special data conversion as needed: i.e. set bit 7 = 1, lower to upper case conversion, etc;
e. Restore the registers;
f. Return to the caller.

For output, we must:

a. Save the registers;
b. Do any special preprint control (tabs, form feeds, etc.);
c. Wait until the ACIA can take more data;
d. Write the data to the ACIA;
e. Do any postprint control (line or page control, insert a line feed after a carriage return, etc.);
f. Restore the registers;
g. Return to the caller;

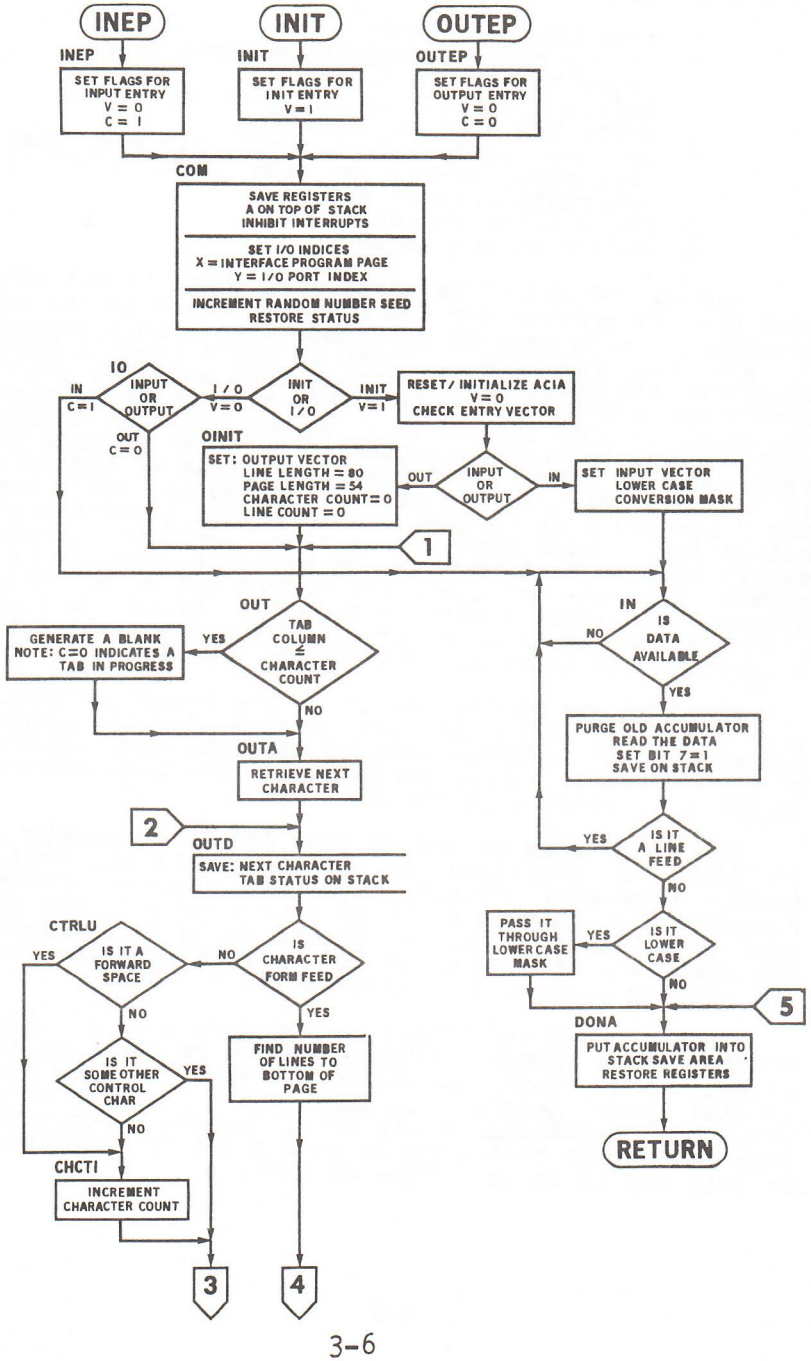
Several of the above tasks are common to all the routines. To stretch our 256 bytes of space as far as possible, let's make as much code as possible common to all the routines. Since we can't predict what absolute addresses will contain this code, we can't create any subroutine calls. This also means we must use relative code throughout. We can use some unused status flags to indicate which entry point we came in from. So, let's use the V (overflow) flag to indicate if we are initializing or not. Likewise, let's use the C (carry) flag to indicate if we are inputting or outputting. With care, these flags won't be altered except when we want to. We now have the means to use common segments of code, yet branch out to the task unique code streams when we need to. After the flag bits have served their purpose, they can be reused to indicate a tab in progress or other function.

Although we don't need it in the sample program, one other scratch location merits identification. Address $7F8 is often used to hold the page address of the currently active peripheral. The page address is $C0 + n, where n is the slot number.

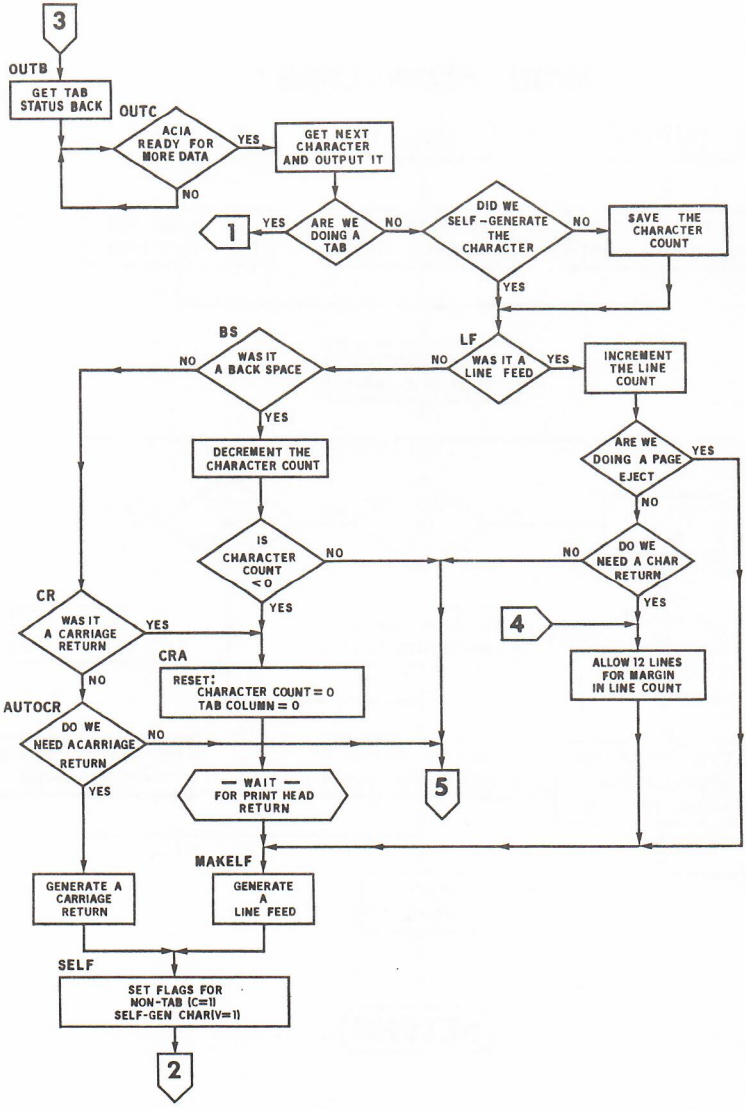
One other item we can do that is really unrelated to the I/O functions is to randomize the random number generator "seed". Console I/O occurs rather randomly, so this is a golden opportunity to mix the seed up a little. All we need to do is to increment the seed each time we do an I/O function. Notice that the seed is located at locations $4E and $4F.

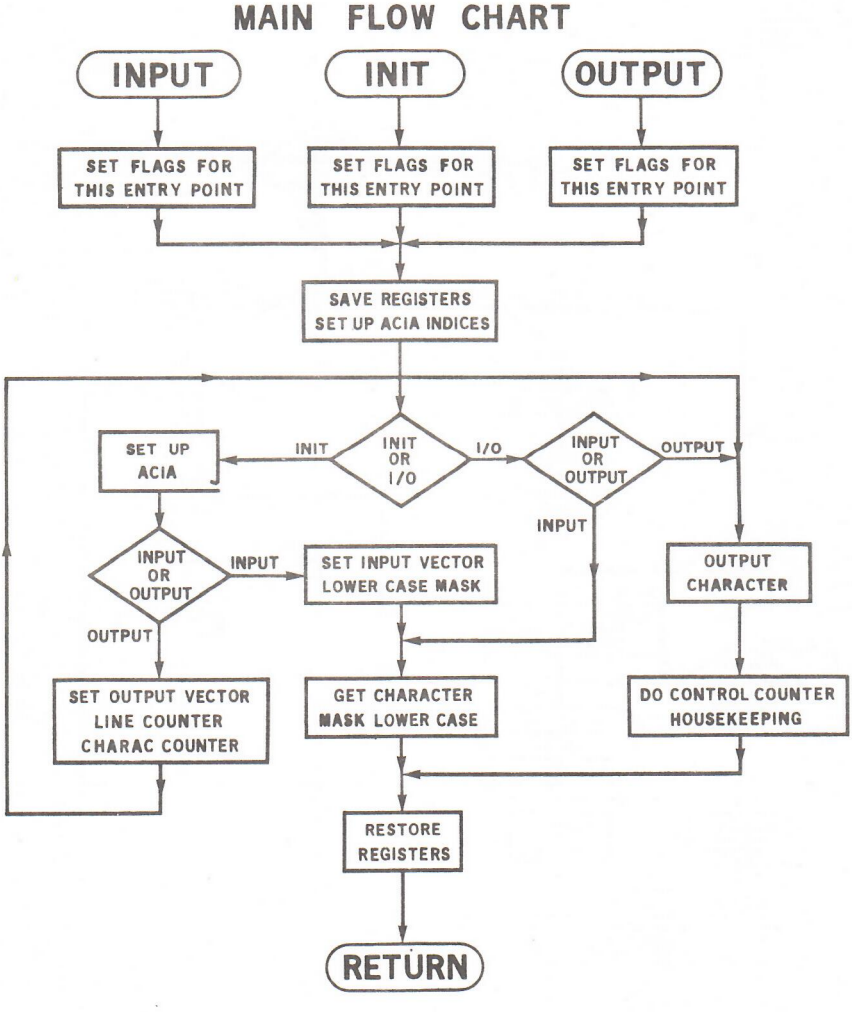
That's what our interface software must do. Flow charts and a program listing follow on the next few pages. Study them carefully... They contain some special line length and page length features which were not explained in any detail above. Then write your own using this program as a point of departure, or just use it as is. They do work!

DETAIL FLOW CHART



DETAIL FLOW CHART




MAIN FLOW CHART
INPUT
INIT
OUTPUT
SET FLAGS FOR THIS ENTRY POINT
SET FLAGS FOR THIS ENTRY POINT
SET FLAGS FOR THIS ENTRY POINT
SAVE REGISTERS
SET UP ACIA INDICES
SET UP ACIA
INIT
INIT OR I/O
I/O
INPUT OR OUTPUT
OUTPUT
INPUT
INPUT OR OUTPUT
INPUT
SET INPUT VECTOR LOWER CASE MASK
OUTPUT CHARACTER
OUTPUT
SET OUTPUT VECTOR LINE COUNTER CHARAC COUNTER
GET CHARACTER MASK LOWER CASE
DO CONTROL COUNTER HOUSEKEEPING
RESTORE REGISTERS
RETURN

```
0000                    ;
0000                    ; SAMPLE DRIVER PROGRAM FOR THE
0000                    ; CCS 7710A ASYNCHRONOUS SERIAL INTERFACE CARD
0000                    ;
0000                    ; COPYRIGHT 1979 CALIFORNIA COMPUTER SYSTEMS
0000                    ;
0000                    ;This program is an example of an input and output driver for
0000                    ;the Asynchronous Serial Interface card.  It contains all the
0000                    ;necessary code to allow direct logical replacement of your
0000                    ;computer's keyboard and TV output.  The output defaults to 127
0000                    ;characters per line and no paging.  Three entry points
0000                    ;are defined: initialization, input, and output.  When the
0000                    ;commands IN#n or PR#n are issued, the computer sets up a
0000                    ;jump vector address to the initialization entry point.  The
0000                    ;initialization routine will adjust the input or output entry
0000                    ;point vector to the correct address when the first input or
0000                    ;output occurs.
0000                    ;
0000                    ; System Equates
0000                            ORG     0
0000                    MAXLN:  EQU     54      ;54 lines per page default
0000                    MAXCHR: EQU     127     ;127 characters per line default
0000                    BKSP:   EQU     $87     ;ASCII Back Space - 1 (for carry)
0000                    LNFD:   EQU     $8A     ;ASCII Line Feed
0000                    FF:     EQU     $8C     ;ASCII Form Feed
0000                    CARRET: EQU     $8D     ;ASICII Carriage Return
0000                    FS:     EQU     $95     ;Forward Space
0000                    SPACE:  EQU     $A0
0000                    CH      EQU     $24     ;Tab column pointer
0000                    CSWL    EQU     $36     ;Location of output driver vector
0000                    KSWL    EQU     $38     ;Location of input driver vector
0000                    KSWH    EQU     $39
0000                    CPL     EQU     $5F8-$C0 ;MAXCHR SAVE LOCATION
0000                    LPP     EQU     $678-$C0 ;MAXLN SAVE LOCATION
0000                    LOCASE  EQU     $6F8-$C0 ;LC conversion mask hold
0000                    LNCNT   EQU     $778    ;Line counter
```

```
0000                    CHCNT   EQU     $778-$C0 ;Character counter
0000                    CMD     EQU     $C080    ;+ $n0 for the ACIA Command port
0000                    STATUS  EQU     CMD      ;This is the ACIA Status port
0000                    DATA    EQU     CMD + 1  ;This is the ACIA Data port
0000                    WAIT    EQU     #FCA8    ;Address of wait routine
0000                    RETURN  EQU     $FFCB    ;Used to find the slot address
0000                    ;
0000                    ; The common code
0000                    ;
0000 2C CB FF   INIT:   BIT     RETURN   ;Set V = 1
0003 70 04              BVS     COM
0005 18         OUTEP:  CLC              ;Clear the carry for output
0006 B0                 DFB     $B0      ;Make a BCS to skip next instruction
0007 38         INEP:   SEC              ;Set the carry for input
0008 B8                 CLV              ;Clear the V Flag for I/O
0009 48         COM:    PHA              ;Save the registers and status
000A 8A                 TXA
000B 48                 PHA
000C 98                 TYA
000D 48                 PHA
000E 08                 PHP
000F 78                 SEI              ;Disable interrupts
0010 20 CB FF           JSR     RETURN   ;Put slot address on the stack
0013 BA                 TSX              ;Put the Slot Page
0014 BC 00 01           LDY     $100,X   ;Number into Y
0017 68                 PLA              ;Recover the output data (if any)
0018 68                 PLA              ;
0019 68                 PLA
001A 68                 PLA
001B 9A                 TXS              ;Restore  the Stack Pointer
001C 48                 PHA              ;Save the data in the stack top
001D 98                 TYA              ;Get the Slot Page Number
001E AA                 TAX              ;Establish X Index
001F 0A                 ASL     A        ;Multiply by 16 to get the
0020 0A                 ASL     A        ;  $n0 index to access the ACIA
0021 0A                 ASL     A
```

```
0022 0A             ASL
0023 A8             TAY             ;Establish Y Index
0024 68       COMA: PLA             ;Get the saved status codes
0025 28             PLP
0026 48             PHA
0027 50 1B          BVC     IO      ;Routine IO branch
0029                ;
0029                ; Initialization Routine
0029                ;
0029                ;This code handles the first input or the first output request
0029                ;after invoking the IN#n or the PR#n commands.  It initializes
0029                ;the ACIA, sets up the lower case conversion mask, the desired
0029                ;entry point vector, and finally goes to the appropriate input
0029                ;or output routine.
0029                ;
0029 A9 23          LDA     #$23    ;Reset the ACIA
002B 99 80 C0       STA     CMD,Y
002E A9 11          LDA     #$11    ;Set for 8 + 2, No parity, No int
0030 99 80 C0       STA     CMD,Y
0033 B8             CLV             ;Clear the initialization flag
0034 A5 38          LDA     KSWL    ;See if input is wanted
0036 D0 29          BNE     OINIT   ;No, branch for output initialization
0038 E4 39          CPX     KSWH    ;Maybe, make sure
003A D0 25          BNE     OINIT   ;No, branch for output
003C 9D 38 06       STA     LOCASE,X ;Set no lower case conversion mask
003F A9 07          LDA     #7      ;Set normal input entry point vector
0041 85 38          STA     KSWL
0043 38             SEC             ;Fall through next branch
0044 90 31    IO:   BCC     OUT     ;Go to normal output
0046                ;
0046                ; Input Routine
0046                ;
0046                ;This routine expects no parameters from calling routines.
0046                ;It waits until data has been typed in from the keyboard, and
0046                ;then returns that data in the accumulator to the caller.  It
0046                ;ignores all line feeds and converts all lower case letters to
0046                ;upper case (if the locase mask = #$20).
```

```
0046                    ;
0046 B9 80 C0   INPUT:  LDA     STATUS,Y ;Get ACIA status
0049 4A                 LSR     A        ;Isolate Rx Rdy bit
004A 90 FA              BCC     INPUT    ;Loop until data is ready
004C 68                 PLA              ;Get rid of data on stack top
004D B9 81 C0           LDA     DATA,Y   ;Read the new data
0050 09 80              ORA     #$80     ;Set Bit 7 for normal video
0052 48                 PHA              ;Save data on stack
0053 C9 8A              CMP     $LNFD    ;Ignore Line Feeds
0055 F0 EF              BEQ     INPUT
0057 68                 PLA
0058 C9 E0              CMP     #$E0     ;See if it is lower case
005A 90 74              BCC     DONA     ;Go finish up if not
005C 5D 38 06           EOR     LOCASE,X ;Convert to lower if mask = $20
005F B0 6F              BCS     DONA     ;Now go finish up
0061                    ;
0061                    ;Output Initialization
0061                    ;
0061 A9 05      OINIT:  LDA     #5       ;Set normal output entry point vector
0063 85 36              STA     CSWL
0065 A9 7F              LDA     #MAXCHR  ;Set defaults
0067 9D 38 05           STA     CPL,X
006A A9 36              LDA     #MAXLN
006C 9D B8 05           STA     LPP,X
006F A9 00              LDA     #0
0071 8D 78 07           STA     LNCNT
0074 9D B8 06           STA     CHCNT,X ;Zero counters
0077                    ;
0077                    ; Output Routine
0077                    ;
0077                    ;This routine does the actual output of the data.  It expects
0077                    ;to find the data for output on the top of the stack (where
0077                    ;the common code put it).
0077                    ;
0077 BD B8 06   OUT:    LDA     CHCNT,X ;See if tab wanted
007A C5 24              CMP     CH
```

```
007C B0 03              BCS     OUTA    ;Branch if no tab
007E A9 A0              LDA     #SPACE  ;Space out to column
0080 48                 PHA             ;Save on stack
0081 68         OUTA:   PLA             ;Retrieve character
0082 48         OUTD:   PHA             ;Resave it
0083 08                 PHP             ;Save Tab Status
0084 C9 8C              CMP     #FF     ;Check for form feed
0086 D0 0A              BNE     CRTLU   ;Branch if not
0088 28                 PLP             ;Restore tab flag
0089 AD 78 07           LDA     LNCT    ;Develop number of lines
008C FD B8 05           SBC     LPP,X   ;  to end of page
008F 38                 SEC             ;Insure no tab on form feed
0090 D0 2D              BNE     LFA     ;Go finish the form feed
0092 C9 95      CRTLU:  CMP     #F5     ;See if Control U
0094 F0 04              BEQ     CHCTI   ;Branch if so
0096 29 60              AND     #$60    ;Test for other control character
0098 F0 03              BEQ     OUTB    ;Skip counter increment
009A FE B8 06   CHCTI:  INC     CHCNT,X ;Bump counter
009D 28         OUTB:   PLP             ;Reget tab flag
009E B9 80 C0   OUTC:   LDA     STATUS,Y ;Get ACIA status
00A1 29 02              AND     #2      ;Isolate Tx buffer bit
00A3 F0 F9              BEQ     OUTC    ;Wait if not ready
00A5 68                 PLA             ;Get data for output
00A6 99 81 C0           STA     DATA,Y  ;Output it
00A9 90 CC              BCC     OUT     ;Branch if tab
00AB 70 01              BVS     LF      ;Branch if self generating character
00AD 48                 PHA             ;Resave character
00AE C9 8A      LF:     CMP     #LNFD   ;See if line feed
00B0 D0 14              BNE     BS      ;No, branch
00B2 EA                 NOP             ;Disable line counters
00B3 EA                 NOP
00B4 EA                 NOP
00B5 30 34              BMI     MAKELF  ;Branch if eject in the progress
00B7 AD 78 07           LDA     LNCNT   ;Get the line count
00BA FD B8 05           SBC     LPP,X   ;Ready for eject?
00BD 30 10              BMI     DONE    ;No, done
```

```
00BF E9 0C      LFA:    SBC     #$C     ;Allow for margin
00C1 8D 78 07           STA     LNCNT   ;Reset line count
00C4 D0 25              BNE     MAKELF  ;Do the eject
00C6 E9 87      BS:     SBC     #BKSP   ;See if back space
00C8 D0 13              BNE     CR      ;No, branch
00CA DE B8 06           DEC     CHCNT,X ;Yes, adjust counter
00CD 30 12              BMI     CRA     ;Disallow negative count
00CF                    ;
00CF                    ;  Final Common Code
00CF                    ;
00CF                    ;This part of the code restores the registers and returns to
00CF                    ;the caller.  It is used by all of these routines
00CF                    ;
00CF 68         DONE:   PLA             ;Set the stack straight
00D0 BA         DONA:   TSX             ;Modify A register value in
00D1 E8                 INX             ;  Stack to insure it is
00D2 E8                 INX             ;  Restored to the right value
00D3 E8                 INX
00D4 9D 00 01           STA     $100,X
00D7 68                 PLA             ;Restore registers
00D8 A8                 TAY
00D9 68                 PLA
00DA AA                 TAX
00DB 68                 PLA
00DC 60                 RTS             ;Done!
00DD                    ;
00DD                    ;  Rest of output code
00DD                    ;
00DD E9 06      CR:     SBC     #6      ;See if a Carriage Return
00DF D0 12              BNE     AUTOCR  ;No, branch
00E1 9D B8 06   CRA:    STA     CHCNT,X ;Zero out counter
00E4 85 24              STA     CH      ;  and tab pointer
00E6 A9 C0              LDA     #$C0    ;Wait for print head to return
00E8 20 A8 FC           JSR     WAIT    ;
00EB A9 8A      MAKELF: LDA     #LNFD   ;Make a line feed
00ED 2C CB FF   SELF:   BIT     RETURN  ;V=1 for self generating character
```

```
00F0 38                  SEC          ;C=1 for no tab
00F1 B0 8F               BCS  OUTD    ;Always branch
00F3 BD B8 06 AUTOCR:    LDA  CHCNT,X ;End of line yet?
00F6 DD 38 05            CMP  CPL,X
00F9 30 D4               BCC  DONE    ;No, done
00FB A9 8D               LDA  #CARRET ;Yes, get a Carriage Return
00FD D0 EE               BNE  SELF    ;Process it
0100                     END  $100
```

IV. OPERATION

In this section, we will give you a description of how to generate the interface driver routine and how to route the console input - output through this card. Little more can be said since the rest of the operating procedures are determined by the software/firmware you install into the RAMs/ROMs on the card. We suggest that after you install the programs, attach a copy of the unique operating instructions to this chapter of the manual. Should you opt to use the sample driver program from above, some instructions are provided to guide you in selecting some nondefault parameters in realtime.

Driver Generation

The interface software must be loaded into the computer before you can use it. Of course, if you installed ROMs on the card, the soft(firm)ware is already there. But if you selected RAMs, they must be loaded every time you turn your computer on and want to use the interface. The following procedure was devised for floppy disk systems; if you use some other storage media, you'll need to devise your own scheme.

The first chore is to get the interface software initially into memory. The firmware miniassembler works nicely for this; see your Red Book for details on how to use it. Assemble the driver directly into the interface's memory. For instance, if the interface is in slot #1, use address $C100 as the base address. After you have assembled your driver into memory, save a copy of it on disk. To do this, first move a copy down into the "lower 32". Your disk can only load and save programs from the lower 32K of memory. Location $A00 is a good spot for the copy. It doesn't interfere with the Integer BASIC or the Disk Operating System (DOS). This Monitor command performs the move nicely:

```
*A00<C100.C1FFM(cr)
```

Now, transfer control over to >BASIC under the DOS. If the DOS is already in memory, just type in:

```
*3D0G
```

Otherwise, do a disk boot:

```
*6(ctrl P)(cr)
```

Finally, we are ready to actually save the driver on disk:

```
>BSAVE ASYN1.0,A$A00,L$100(cr)
```

Your driver software is now saved on your disk, with a file name of ASYN1.0 if you followed the above example. We are now ready to check out the driver and modify it as necessary until you are happy with its performance. Don't forget to save a copy after each modification. There's nothing more frustrating than to try to check out a routine only to have it bomb out and erase itself in the process. After you're happy with it, save it for one last time. You're now ready to routinely use the driver.

Power-On Loading of the Driver

Two methods of loading the software from disk are outlined here: a) direct commands, b) under program control.

a. Direct Commands:
To load the driver with direct commands, perform the following sequence after first turning the computer on:

1. Boot in the DOS:

   *6(ctrl P)(cr)

2. Now, read in the driver file:

   >BLOAD ASYN1.0(cr)

3. Return control to the monitor:

   >CALL-155(cr)

4. Finally, upload the driver into the interface's RAM. Assuming the interface is in slot #2:

   *C200<A00.AFFM(cr)

5. Go do your thing!

b. Loading the program under program control:
This alternate method combines steps 2 and 4 above into one automated step. That's what a computer is all about, isn't it? A simple >BASIC program to perform this is:

```
10 INPUT "ASYNC INTERFACE SLOT IS: ",S
20 IF S<1 OR S>7 THEN GOTO 10
30 DEST = -16384 + 256 * S
40 PRINT "(ctrl D)BLOAD ASYN1.0,A$A00"
50 FOR I = 0 TO 255
60 POKE DEST + I, PEEK (2560 + I)
70 NEXT I
80 END
```

Assuming this program has been saved on disk under the file name of ASYN, all we have to do now is:

1. Boot in the DOS:

   *6(ctrl P)(cr)

2. Execute the ASYN program:

   >RUN ASYN(cr)

3. Answer the question when it appears:

   AYNC INTERFACE SLOT IS: ?2(cr)

4. It's loaded and ready to use!

Input

The programs you install in the RAM/ROMs won't do any good unless control can be passed to them for input or output. To do this, type in:

IN#n where n is the slot number (>or] BASIC)
n(ctrl K) (Monitor)

Either of these commands will cause your computer to go to the installed input program on the card for all subsequent input to the computer. On the very first input, the ACIA will be initialized if the interface program works like the sample above. If the output function has already been invoked, the reinitializing of the ACIA will not cause any problem unless the ACIA has been commanded with a non-default command. If this is the case, the ACIA will return to its default command. Be aware that if any peripheral control options are allowed in the program, invoking the IN#n command will cause the default options to be selected for both input and output, unless the options are initialized after the input versus output init decision is made. The sample program, for instance, waits until after this decision is made before it initializes the character counter and line counter. This way, the IN command has no effect on the counters.

Output

To cause all output to be controlled by the programs on the card, type in one of the following commands:

PR#n where n ic the slot number (> or] BASIC)
n(ctrl P) (Monitor)

All subsequent output from the computer will now be routed to the card's interface program. Depending on the installed initialization routine, the same "be aware" applies here as well as for the IN#n command.

Changing default parameters

If the sample program has been chosen, there are three default parameters which can be changed after the IN#n or OUT#n (as appropriate) commands have been executed.

a. ACIA operating mode command:

The ACIA operating mode can be changed by selecting the appropriate command as defined in Section III above, and POKEing it into location -16256 + 16*n ($C080 + n) where n is the slot number. Remember, though, that any subsequent IN#n or OUT#n command will recommand the ACIA to the default value.

b. Lines Per Page:

The sample driver will automatically issue 12 line feeds after every 54 lines have been printed. To change the number of printed lines which trigger the automatic 12 line feeds, POKE the new maximum lines per page into location $6F8 (1528d) + the slot number. This number should be in the range of 1 < LPP < 127 or funny results will occur!

c. Characters Per Line:

The sample driver will automatically initiate a carriage return, line feed sequence after every 80 characters have been printed. If you want some other number of characters per line, simply POKE your value into location $5F8 (1784d) + the slot number. Make sure it is in the range 0 < CPL <= 255.

HAVING TROUBLE?

If your computer and peripheral are not talking to each other, you may be encountering one of the following common problems:

Problem: Your computer is not sending data. For the interface card to send data, the request to send line, pin 4, must be at a positive voltage greater than +3 volts.

Remedy: A handshake line from the receiving device or a jumper from pin 4 to pin 6 or 8 of the RS-232 connector. If you use a jumper, the interface card will send data at the proper baud rate but might overrun the receiving buffer, causing some data loss. If this happens, use a slower baud rate.

Problem: Your computer is not receiving data. For the interface card to receive data, the data terminal ready line must be at a positive voltage greater than +3 volts.

Remedy: A handshake line from the sending device or a jumper from pin 20 to pin 6 or 8 of the connector.

Problem: When you type in PR#n or IN#n, nothing happens. You might not have connected the receiving device to your computer. Your computer is doing its best, but is outputting to thin air.

Remedy: Connect your receiving device.

IV. OPERATION (continued)

Control Program Defined Operating Instructions

(Attach your unique instructions here.)

V. TECHNICAL INFORMATION

Section V Contents:

V. TECHNICAL INFORMATION (continued)

SPECIFICATIONS:

SIZE:	5"l x 2.75"h x .75"w(max)
WEIGHT:	< 5 oz. (w/cable)

SYSTEM INTERFACE

Internal:	APPLE II Peripheral slots 1 through 7
External:	EIA RS-232C (1969) Configuration A thru E Primary Circuits Only Interface Type: DCE Failsafe Input Circuits (Shorts and Opens)

DATA TRANSFER MODE:	7 or 8 Data Bits Odd, Even, or No Parity Bits 1 or 2 Stop Bits Asynchronous Serial by word Serial by bit

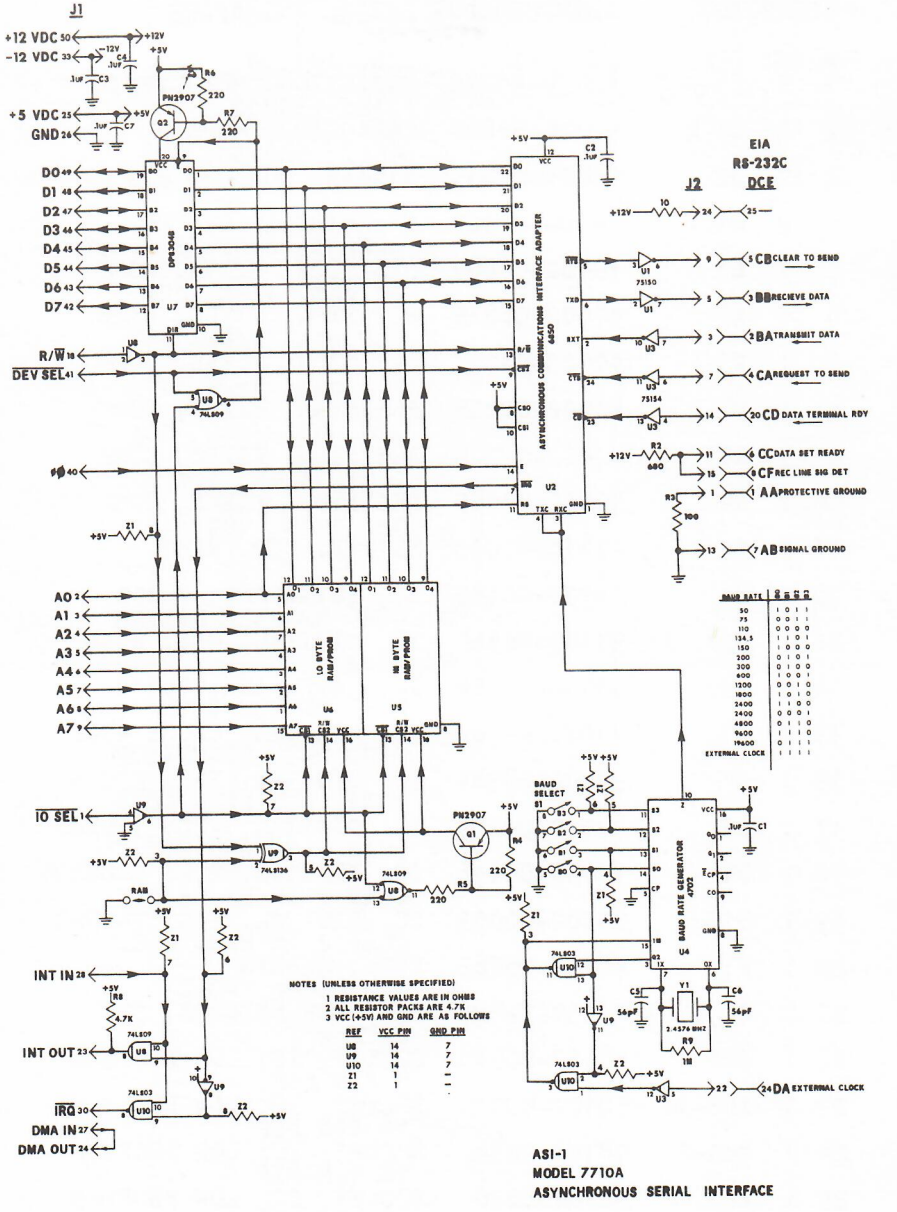
DATA TRANSFER RATES: (16X Clock)

50 baud	75 baud	110 baud
134.5 baud	150 baud	200 baud
300 baud	600 baud	1200 baud
1800 baud	2400 baud	4800 baud
9600 baud	19200 baud	External

PROGRAM MEMORY:	ROM (Mask)
or	PROM (Fuse link)
or	RAM (Static: 2112's)
Size:	256 bytes
Note:	ROM/PROM Auto Powered Down

REQUIRED POWER:	+5vdc +12vdc -12vdc
SALIENT FEATURES:	For other features, see a 6850 Data Sheet for details Supports Daisy Chain Interrupts with on board arbitration logic Allows DMA Daisy Chain (Pass thru) Crystal Controlled Baud Rates Baud Rates Switch Selectable Glass Epoxy (FR-4) PC Board Gold Plated Connector Fingers Solder Mask both sides of board Component Silkscreen

Schematic/Logic Diagram



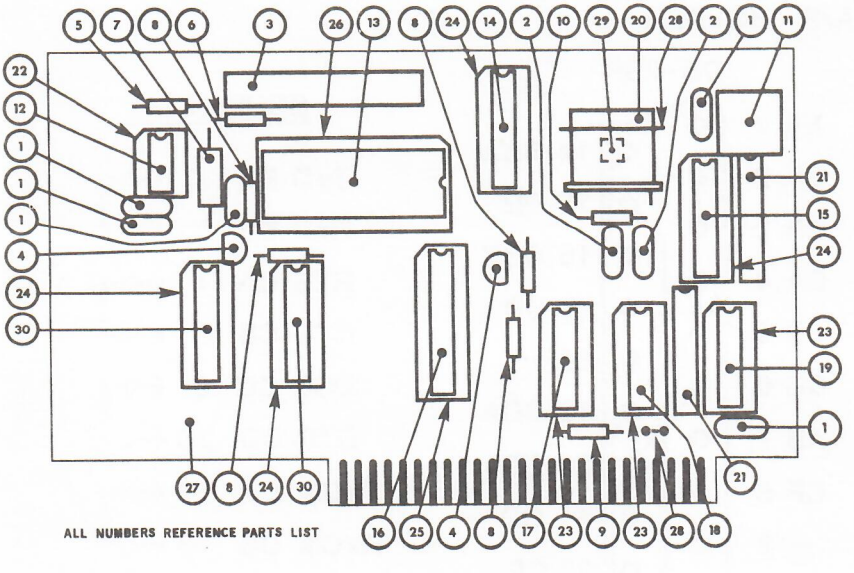
ASI-1
MODEL 7710A
ASYNCHRONOUS SERIAL INTERFACE

PARTS LIST - ASI-1 (Assy 00000-7710A)
CCS 7710A ASYNCHRONOUS SERIAL INTERFACE

#	QTY	REF	CCS P/N	DESCRIPTION
1	5	C1-4,7	42034-21046	CAPACITOR, MONOLYTHIC .1uf, 50vdc
2	2	C5,6	42215-55605	CAPACITOR, MICA 56pf, 500vdc, 10%
3	1	J2	56004-02013	HEADER, DUAL 13 PIN
4	2	Q1,2	36100-02907	TRANSISTOR, SI; PNP GENERAL PURPOSE, PN2907
5	1	R1	40002-01005	RESISTOR, FIXED, COMP 10 ohm, 1/4W, 10%
6	1	R2	40002-06815	RESISTOR, FIXED, COMP 680 ohm, 1/4W, 10%
7	1	R3	40003-01015	RESISTOR, FIXED, COMP 100 ohm, 1/2W, 10%
8	4	R4-7	40002-02215	RESISTOR, FIXED, COMP 220 ohm, 1/4W, 10%
9	1	R8	40002-02725	RESISTOR, FIXED, COMP 2.7K, 1/4W, 10%
10	1	R9	40002-01055	RESISTOR, FIXED, COMP 1M, 1/4W, 10%
11	1	S1	27111-41010	SWITCH, DIP, 4PST
12	1	U1	30300-00150	IC, INTERFACE; 75150 DUAL RS-232C DRIVER
13	1	U2	31100-06850	IC, DIGITAL, MOS; 6850 ACIA
14	1	U3	30300-00154	IC, INTERFACE; 75154 QUAD RS-232 RCVR
15	1	U4	31000-04702	IC, DIGITAL, CMOS; 4702 BAUD RATE GENERATOR
16	1	U7	30900-08304	IC, DIGITAL, TTL; 8304B OCTAL BUS DRVR/RCVR
17	1	U8	30000-00009	IC, DIGITAL, TTL; 74LS09 QUAD 2 IN AND O.C.
18	1	U9	30000-00136	IC, DIGITAL, TTL; 74LS136 QUAD 2 IN EX-OR O.C.
19	1	U10	30000-00003	IC, DIGITAL, TTL; 74LS03 QUAD 2 IN NAND O.C.
20	1	Y1	48132-45762	XTAL, QUARTZ 2.4576MHz, HC-6
21	2	Z1,2	40930-72726	RESISTOR NETWORK, SIP 2.7K x 7
22	1	XU1	58102-00080	SOCKET, IC; LOW PROFILE 8 PIN DIP
23	3	XU8-10	58102-00140	SOCKET, IC; LOW PROFILE 14 PIN DIP
24	4	XU3-6	58102-00160	SOCKET, IC; LOW PROFILE 16 PIN DIP
25	1	XU7	58102-00200	SOCKET, IC; LOW PROFILE 20 PIN DIP
26	1	XU2	58102-00240	SOCKET, IC; LOW PROFILE 24 PIN DIP

PARTS LIST - ASI-1 (Assy 00000-7710A)
CCS 7710A ASYNCHRONOUS SERIAL INTERFACE
(continued)

#	QTY	REF	CCS P/N	DESCRIPTION
27	1	-	07710-00002	BOARD, PC ASI-1, REV A
28	-	(Y1)	51000-01220	WIRE, BARE; TINNED #22 AWG
29	-	(Y1)	60003-00001	TAPE, FOAM; 2 SIDED
30	1	U5,U6	00000-7610C	IC, DIGITAL, TTL, ROM-PAK



PARTS BREAKDOWN ASYNCHRONOUS SERIAL INTERFACE

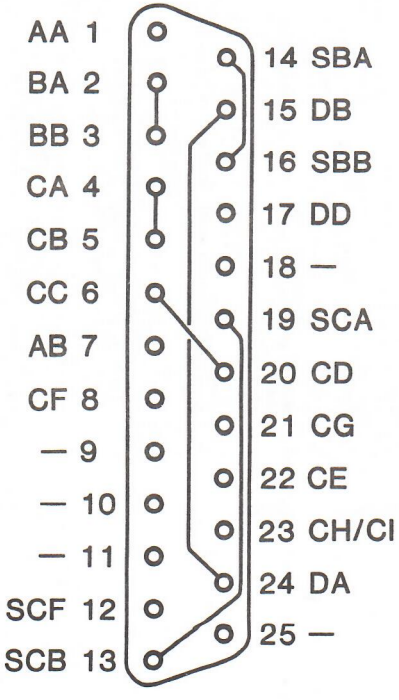
RS-232C ECHOPLEX WIRING DIAGRAM

The Echoplex Connector is a Loop-Back testing module in which a peripheral can "talk" with itself to determine proper functioning of the peripheral. An Echoplex Connector can be constructed using the information below. Secondary functions may also be tested with this configuration.

PARTS LIST

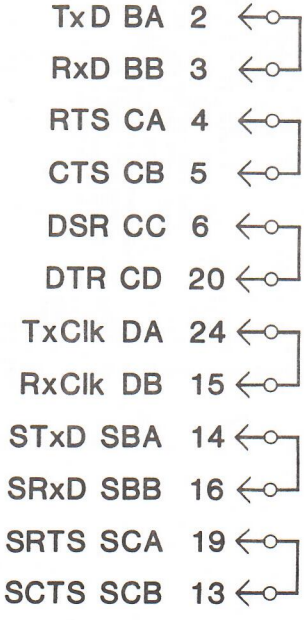
QTY	
1	DB-25 CONNECTOR
A/R	INSULATED 24g WIRE
A/R	SOLDER

DB-25P

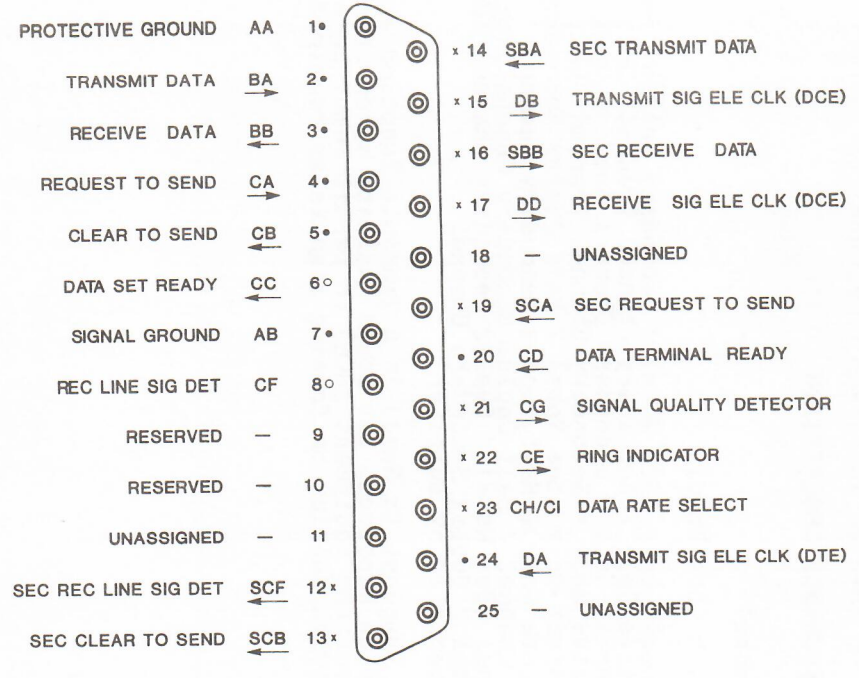


BACK SIDE

SCHEMATIC



FRONT VIEW



DB-25S (FEMALE)

7710A
•FULL/ACTIVE SUPPORT
○PASSIVE SUPPORT
×NOT SUPPORTED

EIA RS-232C
DCE TYPE CONNECTOR PIN ASSIGNMENT

DEFINATION OF RS-232C INTERFACE CONFIGURATION TYPES

CONFIGURATION	DATA INTERCHANGE DESCRIPTION
A	Transmit Only
B	Transmit Only (See Note)
C	Receive Only
D	Half Duplex; or Duplex (See Note)
E	Full Duplex
F	Primary Channel Transmit Only (See Note) / Secondary Channel Receive Only
G	Primary Channel Receive Only / Secondary Channel Transmit Only (See Note)
H	Primary Channel Transmit Only / Secondary Channel Receive Only
I	Primary Channel Receive Only / Secondary Channel Transmit Only
J	Primary Channel Transmit Only (See Note) / Half Duplex Secondary Channel
K	Primary Channel Receive Only / Half Duplex Secondary Channel
L	Half Duplex Primary Channel / Half Duplex Secondary Channel; or Duplex Primary Channel (See Note) / Duplex Secondary Channel (See Note)
M	Duplex Primary Channel / Duplex Secondary Channel
Z	Special (Circuits specified by supplier)

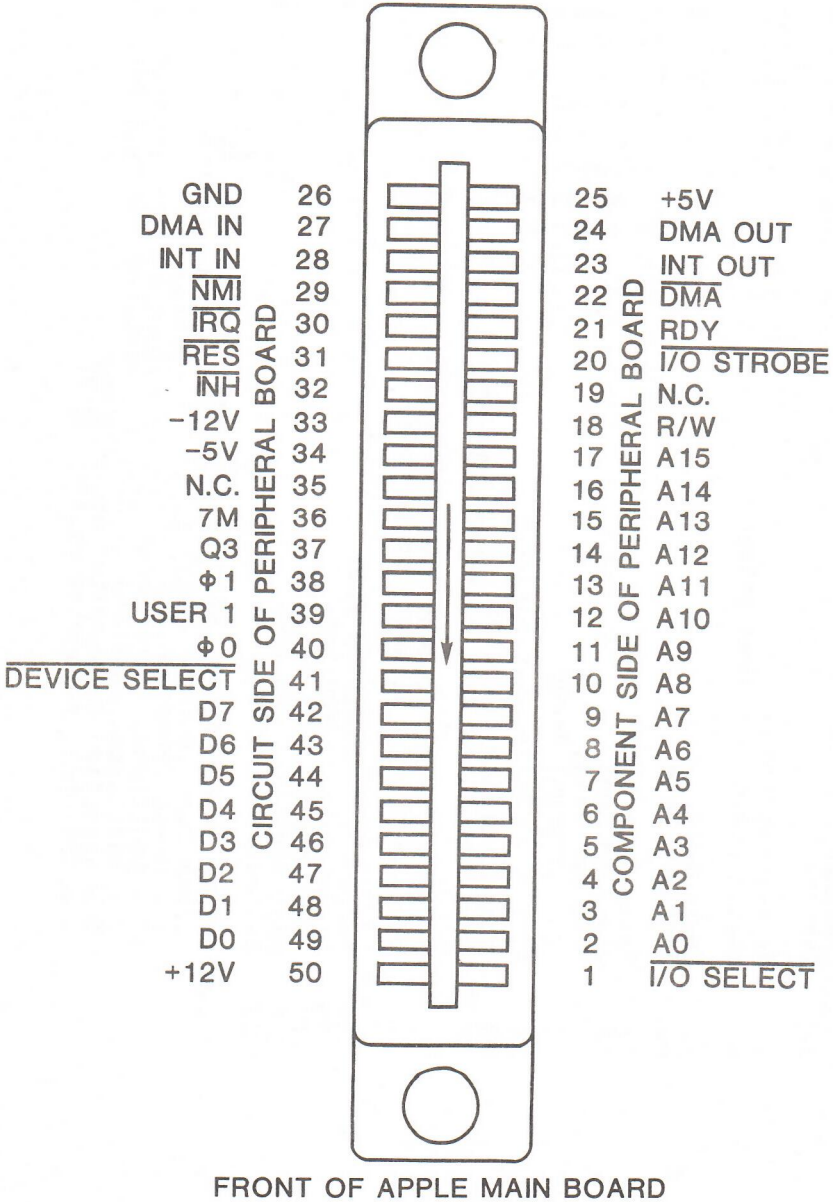
Note: The inclusion of CA (Request to Send) in a Transmit Function where it would not ordinarily be expected, but could indicate a non-transmit mode to the data communications equipment (DCE) to permit removing a line signal, or sending synchronizing or framing signals as required.

STANDARD INTERFACE CONFIGURATIONS FOR RS-232C

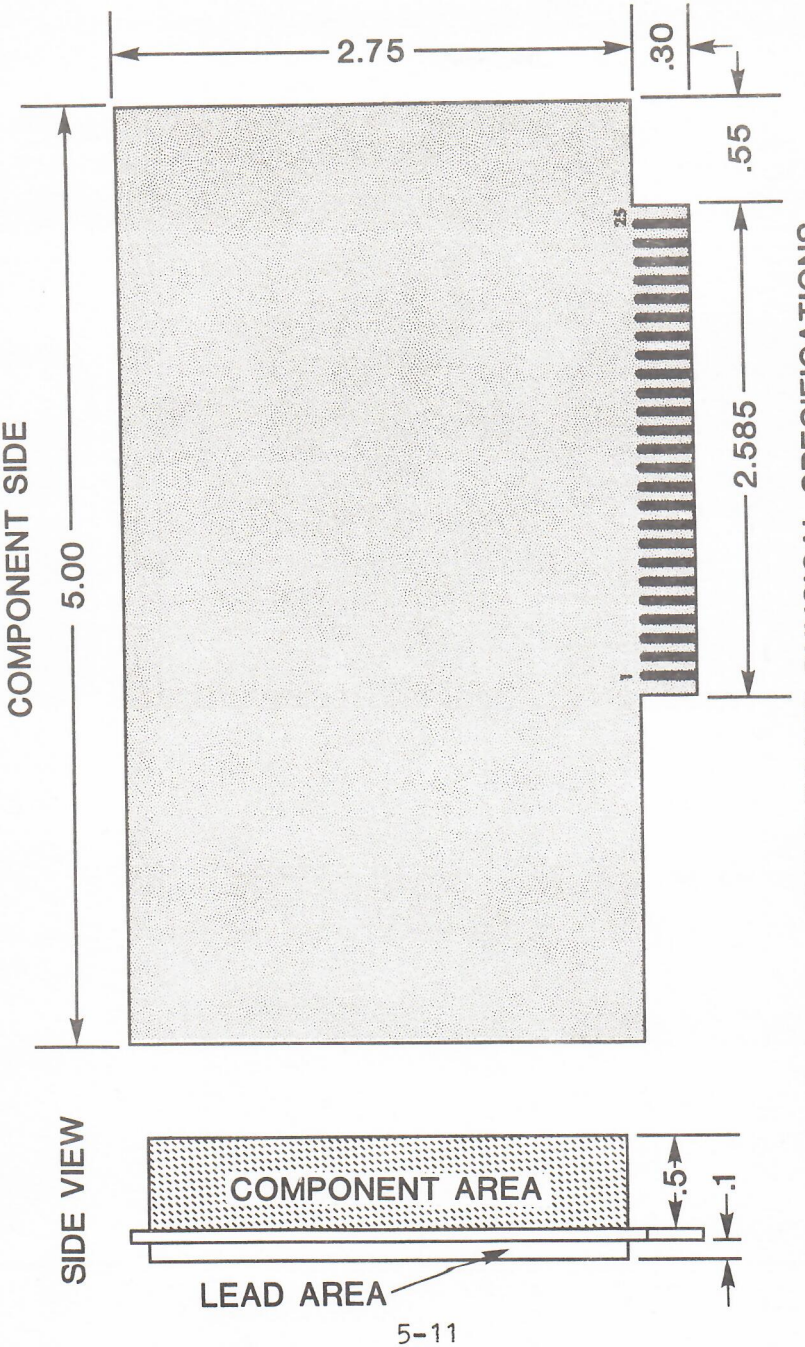
Interchange Circuit Ref	Circuit Name	Interface Configuration A	B	C	D	E	F	G	H	I	J	K	L	M	Z
AA	Protective Ground	X	X	X	X	X	X	X	X	X	X	X	X	X	X
AB	Signal Ground / Common Return	X	X		X	X	X		X		X		X	X	O
BA	Transmitted Data			X	X	X		X		X		X	X	X	O
BB	Received Data		X		X		X				X		X		O
CA	Request to Send	x	x		x	x	x		x		x		x	x	o
CB	Clear to Send	X	X	X	X	X	X	X	X	X	X	X	X	X	O
CC	Data Set Ready	S	S	S	S	S	S	S	S	S	S	S	S	S	O
CD	Data Terminal Ready	s	s	s	s	s	s	s	s	s	s	s	s	s	o
CE	Ring Indicator			X	X	X		X		X		X	X	X	O
CF	Received Line Signal Detector														o
CG	Signal Quality Detector														o
CH/CI	Data Signalling Rate Selector (DTE/DCE)														
DA/DB	Transmit Signal Element Timing (DTE/DCE)	T	T		T	T	T		T		T	T	T	T	O
DD	Receiver Signal Element Timing (DCE)			t	t	t		t		t		t	t	t	o
SBA	Secondary Transmitted Data							x		x	x	x	x	x	o
SBB	Secondary Received Data						x		x		x	x	x	x	o
SCA	Secondary Request to Send							x			x	x	x		o
SCB	Secondary Clear to Send							x		x	x	x	x	x	o
SCF	Secondary Received Line Signal Detector						x		x		x	x	x	x	o

Notes: Upper Case Letters indicate the line is supported by the CCS 7710A.
Lower Case letters indicate the line is not supported by the CCS 7710A.
X = Basic Interchange Circuits, All Systems
T = Additional Interchange Circuits required for Synchronous Channel
S = Additional Interchange Circuits required for Switched Service
O = Specified by supplier as required
- = Optional, supported by the CCS 7710A

PERIPHERAL CONNECTOR PINOUT
TOP VIEW
BACK OF APPLE MAIN BOARD



FRONT OF APPLE MAIN BOARD



TYPE A APPLE BOARD PHYSICAL SPECIFICATIONS

VI. LIMITED WARRANTY

LIMITED WARRANTY

California Computer Systems (CCS) warrants to the original purchaser of its products that

1.) its CCS assembled and tested products will be free from materials defects for a period of one (1) year, and be free from defects of workmanship for a period of ninety (90) days; and

2.) its kit products will be free from materials defects for a period of ninety (90) days.

The responsibility of CCS hereunder, and the sole and exclusive remedy of the original purchaser for a breach of any warranty hereunder, is limited to the correction or replacement by CCS at CCS's option, at CCS's service facility, of any product or part which has been returned to CCS and in which there is a defect covered by this warranty; provided, however, that in the case of CCS assembled and tested products, CCS will correct any defect in materials and workmanship free of charge if the product is returned within ninety (90) days of original purchase from CCS; and CCS will correct defects in materials in its products and restore the product to an operational status for a labor charge of $25.00, provided that the product is returned to CCS within ninety (90) days in the case of kit products, or one (1) year in the case of CCS assembled and tested products. All such returned products shall be shipped prepaid and insured by original purchaser to:

Warranty Service Department
California Computer Systems
250 Caribbean Drive
Sunnyvale, California
94086

CCS shall have the right of final determination as to the existence and cause of a defect, and CCS shall have the sole right to decide whether the product should be repaired or replaced.

This warranty shall not apply to any product or any part thereof which has been subject to

(1) accident, neglect, negligence, abuse or misuse;

(2) any maintenance, overhaul, installation, storage, operation, or use, which is improper; or

(3) any alteration, modification, or repair by anyone other than CCS or its authorized representative.

THIS WARRANTY IS EXPRESSLY IN LIEU OF ALL OTHER WARRANTIES EXPRESSED OR IMPLIED OR STATUTORY INCLUDING THE WARRANTIES OF DESIGN, MERCHANTABILITY, OR FITNESS OR SUITABILITY FOR USE OR INTENDED PURPOSE AND OF ALL

VI. LIMITED WARRANTY (continued)

OTHER OBLIGATIONS OR LIABILITIES OF CCS. To any extent that this warranty cannot exclude or disclaim implied warranties, such warranties are limited to the duration of this express warranty or to any shorter time permitted by law.

CCS expressly disclaims any and all liability arising from the use and/or operation of its products sold in any and all applications not specifically recommended, tested, or certified by CCS, in writing. With respect to applications not specifically recommended, tested, or certified by CCS, the original purchaser acknowledges that he has examined the products to which this warranty attaches, and their specifications and descriptions, and is familiar with the operational characteristics thereof. The original purchaser has not relied upon the judgement or any representations of CCS as to the suitability of any CCS product and acknowledges that CCS has no knowledge of the intended use of its products. CCS EXPRESSLY DISCLAIMS ANY LIABILITY ARISING FROM THE USE AND/OR OPERATION OF ITS PRODUCTS, AND SHALL NOT BE LIABLE FOR ANY CONSEQUENTIAL OR INCIDENTAL OR COLLATERAL DAMAGES OR INJURY TO PERSONS OR PROPERTY.

CCS's obligations under this warranty are conditioned on the original purchaser's maintence of explicit records which will accurately reflect operating conditions and maintenance preformed on CCS's products and establish the nature of any unsatisfactory condition of CCS's products. CCS, at its request, shall be given access to such records for substantiating warranty claims. No action may be brought for breach of any express or implied warranty after one (1) year from the expiration of this express warranty's applicable warranty period. CCS assumes no liability for any events which may arise from the use of technical information on the application of its products supplied by CCS. CCS makes no warranty whatsoever in respect to accessories or parts not supplied by CCS, or to the extent that any defect is attributable to any part not supplied by CCS.

CCS niether assumes nor authorizes any person other than a duly authorized officer or representative to assume for CCS any other liability or extension or alteration of this warranty in connection with the sale or any shipment of CCS's products. Any such assumption of liability or modification of warranty must be in writing and signed by such duly authorized officer or representative to be enforceable. These warranties apply to the original purchaser only, and do not run to successors, assigns, or subsequent purchasers or owners; AS TO ALL PERSONS OR ENTITIES OTHER THAN THE ORIGINAL PURCHASER, CCS MAKES NO WARRANTIES WHATSOEVER, EXPRESS OR IMPLIED OR

VI. LIMITED WARRANTY (continued)

STATUTORY. The term "original purchaser", as used in this warranty shall be deemed to mean only that person to whom its product is originally sold by CCS.

Unless otherwise agreed, in writing, and except as may be necessary to comply with this warranty, CCS reserves the right to make changes in its products without any obligation to incorporate such changes in any product manufactured theretofore.

This warranty is limited to the terms stated herein. CCS disclaims all liability for incidental or consequential damages. Some states do not allow limitations on how long an implied warranty lasts and some do not allow the exclusion or limitation of incidental or consequential damages so the above limitations and exclusions may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.