Serial-to-Parallel Converter Using the MC68705P3

INTRODUCTION

This application note describes a software use of the MC68705P3 EPROM-equipped microcomputer unit (MCU) device of the M6805 family of microcomputers. In this application the MC68705P3 converts asynchronous serial data on an input port to parallel data on an output port. Since no serial transmit or receive hardware is provided on the MC68705P3, the serial-to-parallel conversion function is implemented in software.

This document emphasizes the following features:

- Software serial receiver techniques
- Character queueing/dequeueing techniques
- Development of a general software delay routine.

OVERVIEW OF THE MC6805

The Motorola M6805 is a family of 8-bit microcomputers that allows programmer-friendly characteristics of the M6800 microprocessor family to be used in embedded control applications. A wide variety of M6805s are available with differing amounts and kinds of on-chip RAM, ROM, and peripherals. Low-power applications are served with CMOS versions, and prototyping and short-run production needs are met with EPROM versions in both HMOS and CMOS. To facilitate programming this wide variety of devices, the designers have kept the instruction set consistent across family members; the only instruction extensions are on the CMOS family members to enable low-power shut down, and a multiply instruction on the HCMOS family members.

OVERVIEW OF THE SERIAL-PARALLEL CONVERTER

The serial-to-parallel converter described in this application note gives a simple means of interfacing asynchronous serial data output from a device to a printer that requires parallel input (for example, a Centronics-style printer). The converter implements an internal character queue, allowing some low-cost printers to benefit in performance from this feature. A variety of speeds (baud rates) are recognized, as well as both ASCII and Baudot codes. The converter provides data flow control ("data restraint") via a stop-sending signal that it sends back to the source of the serial data. Data restraint is based on the status of the internal character queue.

HARDWARE CONSTRUCTION AND HOOKUP

Figure 1 is a logic diagram that implements the MC68705P3 converter in a serial-to-parallel conversion scheme.

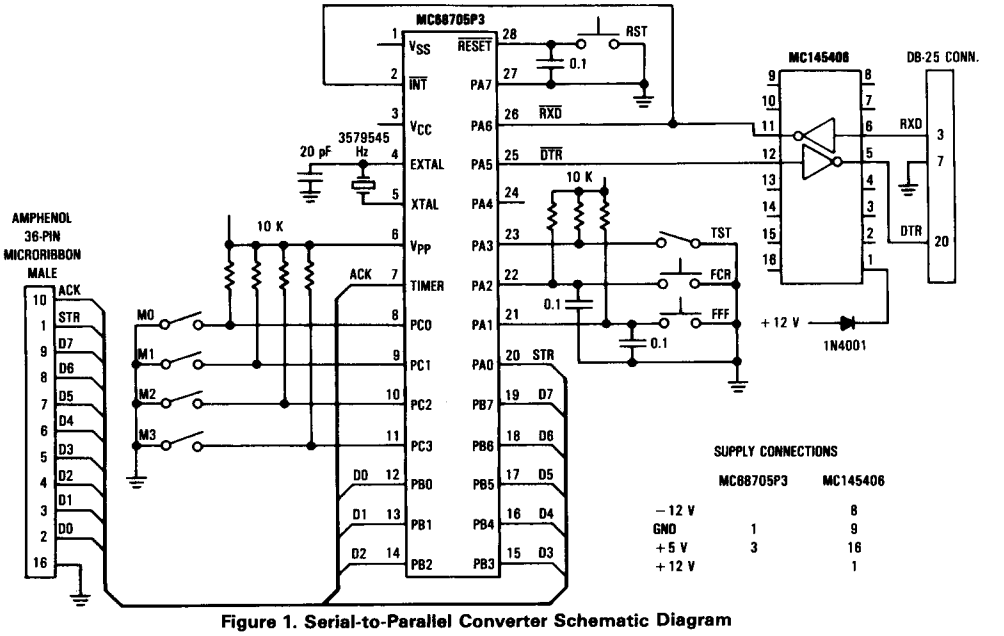

Figure 1. Serial-to-Parallel Converter Schematic Diagram

Table 1. Parts List

Quantity	Description
1	MC68705P3 microcomputer
1	3.579545 MHz crystal (color TV subcarrier type)
1	20 pF capacitor
3	0.1 μF capacitor
7	10 kΩ resistor
5	Single-pole, single-throw (SPST) toggle switch
3	SPST normally-open (NO) momentary switch
1	MC145406 EIA RS-232C line driver/receiver*
1	1N4001 diode (or equivalent)*
1	36-pin microribbon style male connector (for parallel printer connection)
1	Connector as appropriate for serial data source
1	+5 Vdc @ 150 mA supply
1	± 12 Vdc 20 mA supply*

*Optional, in order to meet full EIA serial specification.

EIA RS-232C SERIAL INTERFACE

RS-232C is an Electronic Industry Association (EIA) recommended standard for the interconnection of data equipment for the purpose of serial data communications. It specifies voltage and impedance levels, signal names and recommended functions, and a suggested class of connectors with their pin assignments.

The major connections in an EIA interface are as follows:

- Signal and chassis grounds
- Data in both directions: Receive Data (RXD) and Transmit Data (TXD)
- Various control signals indicating readiness to send or receive data in each direction, as follows:
  - Data Set Ready (DSR)
  - Data Terminal Ready (DTR)
  - Request to Send (RTS)
  - Clear to Send (CTS)
  - Data Carrier Detect (DCD)
  - Ring Indicator (RI)

An additional factor is the characterization of each device as either "Data Terminal Equipment" (DTE), or "Data Communications Equipment" (DCE), which is a means of agreement as to which end of an interconnection will drive each interface line, and which one will read that line. Most peripherals for small computer systems are configured as DTE, with the exception of modems which are consistently DCE. The serial interface on this converter is normally wired as DTE, although it is up to the builder of the converter to choose which lines will be driven and which ones monitored.

Data restraint works as follows: when the printer stops accepting characters for a possibly indefinite period of time and the data source continues to send characters, the queue internal to the converter fills up and causes the data restraint output line (DTR, Data Terminal Ready) to go low. This line should be connected to whatever input connection on the data source will cause sending to cease. If there is no such connection possible (such as when the data source is at the far end of a modem or radio link), then the printer must be fast enough to keep up without the loss of characters. The presence of character buffering alleviates this problem by averaging bursts of characters. The converter in this application note contains a small example of such a queue, and most printers to which it will be attached contain a more substantial internal queue.

The voltage levels recommended are both positive and negative with respect to ground; the amplitude of a valid voltage is between 3 and 15 volts. These levels are incompatible with modern digital logic such as transistor-to-transistor (TTL) and the metal oxide semiconductor (MOS) technology from which microcomputers are made. Thus, level conversion is necessary for both transmitting and receiving data. Accordingly, Motorola offers the MC145406 EIA line driver/receiver to accomplish data conversion. However, if the serial data source operates at TTL rather than EIA levels, the MC145406, its protective power supply diode, and the ± 12 volt supplies may be eliminated.

THE PARALLEL INTERFACE

The parallel interface is characterized by the parallel transfer of 8 bits of information (7 bits on some printers). The source of the data (the converter) presents the parallel data bits to the interface and then produces a negative-going pulse on a data strobe line. The printer acknowledges this transfer in a variety of ways; a negative-going pulse on an acknowledge (ACK) line is the most widely-implemented method. When the ACK pulse occurs, the printer is indicating its readiness to take the next character. On some printers this occasionally takes a considerable time after the previous character was strobed in, due to the time taken for the print mechanism to operate.

Electrically, the printer interface lines look like TTL receivers and drivers. The MC68705P3 can sense and drive these lines directly.

USAGE

CODE/BAUD RATE SELECTOR

Upon reset, the configuration of port C, bits 0 through 3, determine the serial format and rate according to Table 2.

Table 2. Code/Baud Rate Switch Settings

Value	Baud Rate	Code	WPM
0	45.45	Baudot	61.33-65
1	50	Baudot	66.67-71.43
2	56.88	Baudot	76.67
3	74.2	Baudot	100-106
4	110	ASCII	
5	300	ASCII	
6	600	ASCII	
7	1200	ASCII	
8	2400	ASCII	
9	4800	ASCII	

FFF AND FCR MOMENTARY SWITCHES

When the momentary switch marked FFF (force form feed) is closed, a form feed character is put into the internal queue for transmission to the printer. When the printer is idle, the queue is empty; hence, the effect of the switch is essentially immediate. The FCR (force carriage return) switch functions in a like manner, except that it transmits a carriage return character.

TEST MODE SWITCH

When the test mode switch (TST) is closed, the converter ignores the serial input line and sends a stream of characters to the printer, consisting of the alphabet continuously repeated. When the test switch is opened, the test mode stops and the converter reverts to its normal function. The test switch comes in handy for troubleshooting the printer side of the system.

SOFTWARE

OVERALL ORGANIZATION

The software is implemented in the following sequence:

- An interrupt process receives characters from the serial line and queues them up;
- A background process dequeues characters and places them on the parallel port;
- A separate test mode verifies the printer interface.

THE SERIAL RECEIVER

The serial receiver runs in response to an interrupt generated by a negative-going transition on the interrupt request (IRQ) input which indicates the beginning of a character. The receiver then reads the character completely (in a manner described in detail in the following paragraphs), places the received character into a queue (FIFO buffer), and then dismisses the interrupt.

The number of bits to be received, and the timing constants to be used, are variables set up at the time of program initialization according to the code/speed switches or jumpers.

DESCRIPTION OF SERIAL DATA FORMATS

First, some terms are defined as follows: the terms "Mark" and "Space" refer to the two valid conditions that a serial data line can be in. When the data line is idle, it is in Mark condition; this corresponds to the negative valid voltage range described earlier.

The MC145406 EIA receiver section inverts these voltage levels so that the negative EIA voltage becomes the high TTL level, and the positive EIA voltage becomes the low TTL level. Therefore, the Mark condition means a logic-high level and the Space condition equates to a logic-low level.

All asynchronous data is sent in packages of a certain number of bit times; the frequency with which these bits come across the serial line is known as the "baud rate" (after Charles Baudot, a Belgian inventor of early teleprinters in the late 19th century). Occasionally, the unit of time itself is called a "baud".

It is evident that some kind of synchronization is necessary between the transmitter and the receiver. In the asynchronous serial formats, synchronization is accomplished by preceding every character with a single bit-time of Space signal; this is called the "start" bit. The character itself follows, beginning with its least significant bit. Finally, one or more bits of Mark condition are inserted; these are the "stop" bit(s). Any arbitrary amount of time is allowed between the end of the stop bits of one character and the beginning of the start bit of the next character. Hence, synchronization is accomplished on a character-by-character basis. The term "asynchronous" refers to the lack of restriction on the time interval between characters, given that stop and start bits are properly formed.

DETAILED DESCRIPTION OF THE SERIAL RECEIVER ALGORITHM

As previously mentioned, the beginning of the character is denoted by a Mark-to-Space transition (logic high to logic low at the MCU). This is connected to the MC68705P3 interrupt line and thus produces an interrupt. The interrupt routine is entered at the location labeled "SerHandler". It waits one bit-time for the start bit to pass, and then another half bit-time to get to the center of the first data bit. These delays are generated by software in timing subroutines which are described in detail in Appendix I. The routine then samples each bit at the center of its respective bit time, meanwhile shifting it into the high-order bit of the accumulator. It repeats this five or eight times, depending on whether Baudot or ASCII characters are being received.

When the routine finishes the last data bit, it calls "BuffIn" which puts the received character into the internal queue. This routine is described in detail in Appendix II. Note that in the interest of minimizing time spent in the interrupt routine, Baudot characters are not translated upon receipt, but rather when taken off the queue. This occurs in the background process.

A side effect of the character queueing routine is that it raises the data restraint condition by sending a signal to the data source if the queue is almost full. This data restraint signal prevents a slow-responding data source from sending additional characters after the queue is already full.

THE PARALLEL TRANSMITTER

The parallel transmitter runs in the background; that is, it is interrupted as necessary by the serial receiver interrupt process. The transmitter routine consists of a continuous loop whose first location is at label "Main".

The reason that the parallel transmitter is the background process is because the parallel interface operates in a completely synchronous manner, meaning that any part of the character transfer process can be subjected to unplanned delays. Quite the opposite is true of the receive process, where the constancy of the timing intervals used in the receiving of the successive bits of the character are critical. What this means is that the parallel transmit process can be interrupted, whereas the serial receive process must not be interrupted. Thus, the serial receive process runs as an interrupt, not subject to further interrupts, while the parallel process runs in the background, subject to interruption.

The parallel transmit program repeatedly calls "DumpBuff", a routine that attempts to empty the character queue to the parallel port. If DumpBuff finds the queue empty or the printer not ready, it returns without doing anything.

As long as the queue contains any characters, however, DumpBuff calls "ParOut" to handle the details of putting out a single character to the parallel interface. This resets the ACK pulse detector, presents the data bits to the parallel printer interface, and forms the data strobe. Waiting for the ACK pulse to occur takes place outside the ParOut routine.

An interesting aspect of the parallel transmit process is the use of the M6805 timer hardware to detect the ACK pulse from the printer. This is done by initializing the timer count register to 1, with the result that the request flag in the timer control register goes true upon the negative-going edge of the

ACK pulse. The reason that the parallel transmit process does not watch for the ACK pulse directly is that on many printers this pulse is too short to catch reliably in a program loop.

BAUDOT-TO-ASCII CONVERSION

If the converter is working in Baudot mode, the parallel transmit process converts each character obtained from the queue into ASCII before transmitting it. This conversion is described in detail in Appendix III.

DATA RESTRAINT

As mentioned earlier, data restraint is based on the number of characters in the queue. Since the parallel transmit process tends to empty the queue, it is the responsibility of this process to re-enable transmission from the serial source under the appropriate condition. That condition exists whenever the queue empties below a certain number of characters; this is handled in the queue output routine. Note that some hysteresis is built into the system, by making the number of queued characters which begins data restraint slightly more than the number of characters which ends data restraint. This tends to reduce the frequency of transitions of the data restraint line.

APPENDIX I
SOFTWARE DELAY ROUTINE

This appendix describes a general algorithm for producing a wide range of delays, with specific application to the bit timing of the serial receiver process.

ALGORITHM

This algorithm uses two 8-bit input parameters (the A and X registers) to control the amount of delay obtained. The delay routine is in the form of two nested loops, the inner one controlled by A and the outer one controlled by X. On the MC67805P3, the delay is given by the formula:

$$8AX + 12X + 19 + E \text{ cycles},$$

where $1 <= (A, X) <= 256$; the register contents of 0 map onto the value 256. E is a parameter representing the cycles encountered outside of the routine; in our serial input example, E computes to 30 cycles.

To get the best approximation to a given time delay, first calculate the desired delay in terms of processor cycles. Then perform substitutions of A and X into the above formula until the best fit is found. Sometimes the parameter combinations are non-obvious, so an exhaustive test of parameter pair values is useful; a Pascal program to do this is given in the following section.

Note that in the serial-to-parallel converter, the initialization logic reads the data rate/code switches and transforms these into an index, which the timing routines "Baud" and "HalfBaud" use to lookup their A and X parameters when each routine is called.

PARAMETER CALCULATOR PROGRAM

The following program is written in standard Pascal; it reads a file which is composed of a number of lines of the following structure:

```
clock frequency in MHz          (integer plus decimal fraction)
external cycles of delay        (integer)
target delay in microseconds    (integer)
target delay in microseconds    (integer)
. . .                           . . .
target delay in microseconds    (integer)
0
```

The program incorporates two parameters that should be customized for the application, and which are read in as the first two records of the file. The first parameter represents the MCU oscillator frequency in megahertz; the second parameter accounts for the cycles external to the timing routine which must be included in the overall delay calculation. In the serial-to-parallel converter these values are 3.579545 and 30, respectively.

The Parameter Calculator Program written in standard Pascal is as follows:

```
program CalcXA (input, output);

  var     XtalFreq:       real;     {in units of MHz}
          ExternCycles:   integer;
          PeriodUSec:     integer;
          PeriodCyc:      integer;
          DelayCyc:       integer;
          X, A:           integer:

  procedure find (  IdealCyc: integer;
                    var Cyc: integer;
                    var X, A: integer);
    label 49,99;
    var   TrialCyc, TrialX, TrialA: integer;
          epsilon: integer;
    function Delay ( x, a: integer ): integer;
      begin Delay := 8 * a * x + 12 * x + 49 + ExternCycles end;
    begin {procedure find}
      epsilon := IdealCyc;
      for TrialX := 1 to 256 do begin
        for TrialA := 1 to 256 do begin
          TrialCyc := Delay (TrialX, TrialA);
          if abs (Trial Cyc-Ideal Cyc) < epsilon
            then begin
              Cyc := TrialCyc;
              X := TrialX;
              A := TrialA;
              epsilon := abs(Cyc - IdealCyc);
              if epsilon = 0 then go to 99
              end;
          if TrialCyc > IdealCyc then go to 99
          end; {for TrialA do begin}
49      end; {for TrialX do begin}
99:   end; {find}

  begin {program CalcXA}
    writeln (output, '     Period,   Ideal     Actual     X  A');
    writeln (output, '     uSec      Cycles    Cycles          ');
    writeln (output);
                        readln (input, XtalFreq);
                        readln (input, ExternCycles);
    repeat
      readln (input, PeriodUSec);
      if PeriodUSec > 0 then begin
        PeriodCyc := round((XtalFreq / 4.0) * PeriodUSec);
        find (PeriodCyc, DelayCyc, X, A);
        writeln(output, PeriodUSec:10, PeriodCyc:10
                        ,DelayCyc:10, X:6, A:5)
      end
    until PeriodUSec <= 0
  end.
```

APPENDIX II
CHARACTER QUEUING SYSTEM

The system used to queue up characters in the converter is of a type often referred to as a "character buffer". A fixed area of RAM is set aside for the queue, and three auxiliary variables keep track of the queue's state. The character buffer system consists of the following:

- An input pointer (BuffInPtr);
- An output pointer (BuffOutPtr);
- The number of items in the queue (BuffCount).

The routines in the converter which handle input and output are called "BuffIn" and "BuffOut", respectively. These routines are structured very similarly. Each one uses indexed addressing to locate the character of interest, and updates that pointer by simple incrementation. This is followed by a reset to the beginning of the buffer area if the resulting pointer was out of range.

In addition, both routines manipulate BuffCount. Notice that when BuffInPtr equals BuffOutPtr, the queue is either full or empty. The value of BuffCount indicates whether the full or the empty condition actually exists. The queue system is initialized by setting BuffInPtr and BuffOutPtr to the beginning of the buffer area, and setting BuffCount to zero.

In the converter application, the job of setting the data restraint line true and false has been moved into the BuffIn and BuffOut routines, as this was much simpler than trying to pass back result codes to the calling routines.

BuffIn and BuffOut could easily be generalized to work on any queue, and the address of the variables could be supplied in the X register. In this case, such side effects would have to be removed from the routines, and information about buffer empty/full, or near empty/full, would need to be returned as result codes in some manner. Also, the generalized routines would protect the integrity of the queue by doing nothing when overflow or underflow would result.

APPENDIX III
BAUDOT TO ASCII CONVERSION

The Baudot code is an encoding of printable characters that considerably predates ASCII, and is still common in older teleprinters and on the shortwave radio bands. Transmission of Baudot characters is normally asynchronous, with start bits as described at the beginning of the serial receiver software section.

In contrast to ASCII, a Baudot character consists of only 5 bits. Although the resulting 32 codes can represent the uppercase alphabet, they cannot represent the decimal digits at the same time, nor any reasonably full set of punctuation symbols and control functions.

This is alleviated by giving most of the binary codes two interpretations, depending on the state of the transmitter and receiver. Specifically, there is a "Lets" mode which encompasses the alphabet and a small number of punctation marks, and a "Figs" mode which encodes the decimal digits and more punctuation. Two shift codes (called "Lets" and "Figs") put the equipment into one or the other of these modes. The shift codes, along with the space character and other printer control characters, are given codes which have the same meaning regardless of the shift mode.

IMPLEMENTATION

If the converter is working in Baudot mode, the serial receiver process shifts in only 5 bits per character. Shift is to the right (from MSB to LSB); the character is in the upper 5 bits of the byte and is enqueued in that form. When the parallel transmitter dequeues the byte, it finishes shifting the character to the LSB and then looks up its ASCII equivalent in one of two tables, depending on the shift mode in effect. Two characters, however, are not run through the lookup tables; these are the "Figs" and "Lets" shift characters, which instead determine which table will subsequently be used. One other character is treated specially. The Space character is converted via the lookup table and transmitted. It also causes the shift mode to be forced to "Lets". This implements the "unshift-on-space" convention prevalent on Baudot equipment.

APPENDIX IV
SERIAL-TO-PARALLEL CONVERTER
SOURCE CODE

```
Aug 11 12:58 1987  705p3.lst Page 1

0001                    *-----------------------------------------------------------------------*
0002                    *
0003                    *       Serial-to-Centronics converter    Version 4.0 4/29/87
0004                    *
0005                    *       Donald G. Weiss  1/83
0006                    *
0007                    *   Based on the MC68705P3. Converts asynchronous serial data stream
0008                    *   in either 5-unit Baudot or 8-unit ASCII code to the Centronics
0009                    *   interface.
0010                    *
0011                    *   Serial connections:
0012                    *
0013                    *       RXD     Receive data input to converter.
0014                    *               Hi=mark;  lo=space.
0015                    *       DTR     Data terminal ready from converter.
0016                    *               Hi=restrain; lo=proceed.
0017                    *
0018                    *               Note: DTR may not need to be observed by the data source
0019                    *               if the baud rate is slow enough and the printer is fast
0020                    *               enough.  The exact maximum speed depends also on the
0021                    *               number of characters per line -- e.g., CR-LF's (blank
0022                    *               lines) can be accepted by some units at a continuous rate
0023                    *               of only about one per second.  This is partly alleviated
0024                    *               by an 80-plus-character buffer in this converter.
0025                    *
0026                    *   Connections to centronics parallel port:
0027                    *
0028                    *       D0-7    Data lines from converter.
0029                    *       Strobe  Data strobe from converter. Normally high; pulses
0030                    *               low for about 5 microseconds, then high.
0031                    *       Ack     Ack line to converter.  Low-going pulse indicates
0032                    *               printer ready to accept next character.
0033                    *
0034                    *   Control connections:
0035                    *
0036                    *       Reset   Reset/reinitialize the interface.  Accomplished
0037                    *               by grounding momentarily.
0038                    *       Test    Test mode switch.  When low, enters a mode which
0039                    *               exercises the printer with a continuous string
0040                    *               of printable ASCII characters.
0041                    *       FCR     Force Carriage Return to the printer.
0042                    *       FFF     Force Form Feed to the printer. (ASCII modes only)
0043                    *       Rate    Baud rate / Code select.  4 lines select the following:
0044                    *
0045                    *             value       baud rate     code    wpm
0046                    *
0047                    *               0             45.45     Baudot  61.33-65
0048                    *               1             50        Baudot  66.67-71.43
0049                    *               2             56.88     Baudot  76.67
0050                    *               3             74.2      Baudot  100-106
0051                    *               4             110       ASCII
0052                    *               5             300       ASCII
0053                    *               6             600       ASCII
0054                    *               7             1200      ASCII
0055                    *               8             2400      ASCII
0056                    *               9             4800      ASCII
0057                    *
0058                    *-----------------------------------------------------------------------*
```

```
temp                                    page   2
0060                        *---------------------------------------------------------------------*
0061                        *           I/O definitions                                           *
0062                        *---------------------------------------------------------------------*
0063 0004                   DDR     equ     4               offset to DDR's
0064                        *---------------------------------------------------------------------*
0065                        *       Port A:  Serial data; Serial/Parallel/General Control         *
0066                        *---------------------------------------------------------------------*
0067 0000                   Ctrl    equ     0               General control
0068 0000                   SerData equ     Ctrl            Serial data
0069 0000                   SerCtrl equ     Ctrl            Serial control
0070 0000                   ParCtrl equ     Ctrl            Parallel control
0071                        *
0072                        *                               Bit position definitions:
0073 0006                   RXD     equ     6           i   Incoming Serial data
0074 0005                   DTR     equ     5           o   Data terminal ready
0075 0004                   TXD     equ     4           o   Outgoing Serial data (for monitor pgm)
0076 0003                   Test    equ     3           i   low --> Test Mode
0077 0002                   FCR     equ     2           i   low --> force CR to printer
0078 0001                   FFF     equ     1           i   low --> force FF to printer
0079 0000                   Strobe  equ     0           o   Parallel port strobe
0080                        *                           Direction mask:
0081 0031                   CtrlDDMask   equ %00110001
0082                        *                           Initial bit values:
0083 0011                   CtrlInitMask equ %00010001
0084                        *---------------------------------------------------------------------*
0085                        *       Port B: Parallel output register                              *
0086                        *---------------------------------------------------------------------*
0087 0001                   ParData     equ 1
0088                        *                           Direction mask:
0089 00FF                   ParDDMask   equ %11111111
0090                        *                           Initial bit values:
0091 0000                   ParInitMask equ 0
0092                        *---------------------------------------------------------------------*
0093                        *       Port C: Serial rate/mode select register                      *
0094                        *---------------------------------------------------------------------*
0095 0002                   SerRate     equ 2
0096                        *                           Direction mask:
0097 000F                   SerRateMask equ %00001111
0098                        *---------------------------------------------------------------------*
0099                        *       Timer port --  Ack pulse detector.                            *
0100                        *---------------------------------------------------------------------*
0101 0008                   AckCount    equ 8           actually, timer count
0102 0009                   AckCtrl     equ 9           actually, timer control
0103                        *
0104                        *                       initial bit values:
0105                        *                           TIR   <- don't care
0106                        *                           TIM   <- interrupt masked
0107                        *                       The following are mask options -- see MOR.
0108                        *                           TIN   <- external clock source
0109                        *                           TIE   <- external source enabled
0110                        *                           PSC   <- prescaler cleared
0111                        *                           PS2-0 <- prescaler bypassed
0112                        *
0113 0078                   AckInitMask equ %01111000
0114                        *
0115                        *                       bit position definitions:
0116 0007                   Ack         equ 7           actually, timer interrupt request
```

```
temp                                    page   3
0118                    *---------------------------------------------------------------------*
0119                    *
0120                    *       Mask Option Register -- an actual register in the EPROM parts such
0121                    *       as the 705P3; a mask option directive on parts such as the 05P2.
0122                    *
0123                    *---------------------------------------------------------------------*
0124 0784                       org     $784
0125 0784 60                    fcb     %01100000
0126                    *
0127                    *       CLK     7       Crystal
0128                    *       TOPT    6       P2/P4 type timer
0129                    *       CLS     5       clock source = event counter
0130                    *       TIE     4       not used in P2/P4 mode
0131                    *       PSC     3       not used  "   "    "
0132                    *       PS2     2       Prescaler option 2 = 0
0133                    *       PS1     1           "       "    1 = 0
0134                    *       PS0     0           "       "    0 = 0
0135                    *
0136                    *---------------------------------------------------------------------*
0137                    *           RAM definitions
0138                    *---------------------------------------------------------------------*
0139 0010                         org  $10
0140                    *---------------------------------------------------------------------*
0141                    *       Baudot conversion variables and definitions
0142                    *---------------------------------------------------------------------*
0143 0010               BaudotCtrl rmb  1               Baudot Shift indicator  flag word
0144 0000               FigsCase   equ  0                                       flag bit #
0145                    *---------------------------------------------------------------------*
0146                    *       Serial variables
0147                    *---------------------------------------------------------------------*
0148 0011               NbrBits    rmb  1               # of data bits (5 = Baudot; 8 = ASCII)
0149 0012               BTemp      rmb  3               Baud timer temp variables
0150 0015               BIndex     rmb  1               Pointer into baud rate table
0151                    *---------------------------------------------------------------------*
0152                    *       Buffer variables and definitions
0153                    *---------------------------------------------------------------------*
0154 0016               BuffInPtr  rmb  1               Buffer pointer
0155 0017               BuffOutPtr rmb  1                   "
0156 0018               BuffCount  rmb  1               # of chars in buffer
0157 0019               Buff       equ  *               Buffer area definition
0158 0070               BuffEnd    equ  $70                 "
0159 0057               BuffSize   equ  BuffEnd-Buff        "
0160                    *---------------------------------------------------------------------*
0161                    *       Remainder of RAM (to $7f) for stack.
0162                    *---------------------------------------------------------------------*
```

```
temp                                    page  4
0164                            *---------------------------------------------------------------------*
0165                            *       Data ROM area
0166                            *---------------------------------------------------------------------*
0167 0080                               org     $80
0168                            *---------------------------------------------------------------------*
0169                            *       Table of 2-byte rate constants to establish baud timing.
0170                            *---------------------------------------------------------------------*
0171                            *   Notes
0172                            *       The following are calculated on the basis of the formula
0173                            *
0174                            *           D = 8ax + 12x + 79,
0175                            *
0176                            *       where x is the first parameter to the delay routine, a is the
0177                            *       second parameter, and the constant term includes both the
0178                            *       delay routine overhead (49 cycles), plus 30 cycles of outer
0179                            *       loop overhead.
0180                            *
0181                            *       Due to the structure of the timing loops, X and A must be >= 1.
0182                            *
0183                            *       Cycle counts are based on the use of a 3.579545 MHz MCU clock.
0184                            *
0185                            *                   1/2 Baud time                   1 Baud time
0186                            *   Baud    ----------------------          ----------------------
0187                            *   rate        ideal   actual   X    A      ideal   actual   X    A
0188                            *               cycles  cycles                cycles  cycles
0189                            *
0190                            *   45.45        9845    9847    6   202       19689   19687   38   63
0191                            *   50.          8949    8943    8   137       17898   17899    9  246
0192                            *   56.88        7866    7867   11    87       15733   15731   13  149
0193                            *   74.2         6030    6031   16    45       12060   12063   28   52
0194                            *  110           4068    4071    2   248        8135    8135   38   25
0195                            *  300           1491    1491    1   175        2982    2983    2  180
0196                            *  600            746     747    1    82        1491    1491    1  175
0197                            * 1200            373     371    1    35         746     747    1   82
0198                            * 2400            186     187    1    12         373     371    1   35
0199                            * 4800             93      99    1     1         186     187    1   12
0200                            *
0201                            *---------------------------------------------------------------------*
0202                            RateTable
0203 0080 06 CA 26 3F                   fcb     6,  202,  38,   63      45b   Baudot
0204 0084 08 89 09 F6                   fcb     8,  137,   9,  246      50b     "
0205 0088 0B 57 0D 95                   fcb    11,   87,  13,  149      57b     "
0206 008C 10 2D 1C 34                   fcb    16,   45,  28,   52      74b     "
0207 0090 02 F8 26 19                   fcb     2,  248,  38,   25     110b   ASCII
0208 0094 01 AF 02 B4                   fcb     1,  175,   2,  180     300b     "
0209 0098 01 52 01 AF                   fcb     1,   82,   1,  175     600b     "
0210 009C 01 23 01 52                   fcb     1,   35,   1,   82    1200b     "
0211 00A0 01 0C 01 23                   fcb     1,   12,   1,   35    2400b     "
0212 00A4 01 01 01 0C                   fcb     1,    1,   1,   12    4800b     "
```

```
temp                                    page   5
0214                    *---------------------------------------------------------------------*
0215                    *       Baudot to ASCII conversion table.
0216                    *       2 bytes per entry.  First byte for lets mode, second for figs.
0217                    *---------------------------------------------------------------------*
0218 001B               Figs    equ     $1B
0219 001F               Lets    equ     $1F
0220 0004               Space   equ     $04
0221 0008               BCR     equ     $08
0222 0000               ANul    equ     $00
0223 0005               AEnq    equ     $05
0224 0007               ABel    equ     $07
0225 000C               AFF     equ     $0C
0226 000A               ALF     equ     $0A
0227 000D               ACR     equ     $0D
0228 0020               ASpc    equ     $20
0229                    *
0230                    *               Character   Baudot
0231                    *               lets,figs    code
0232                    BtoATable
0233 00A8 20 20                 fcb     ASpc,   ASpc    00      Blank
0234 00AA 45 33                 fcb     'E,     '3      01
0235 00AC 0A 0A                 fcb     ALF,    ALF     02      Line feed
0236 00AE 41 2D                 fcb     'A,     '-      03
0237 00B0 20 20                 fcb     ASpc,   ASpc    04      Space
0238 00B2 53 27                 fcb     'S,     ''      05
0239 00B4 49 38                 fcb     'I,     '8      06
0240 00B6 55 37                 fcb     'U,     '7      07
0241 00B8 0D 0D                 fcb     ACR,    ACR     08      Car. ret.
0242 00BA 44 05                 fcb     'D,     AEnq    09      D, WRU
0243 00BC 52 34                 fcb     'R,     '4      0a
0244 00BE 4A 07                 fcb     'J,     ABel    0b      Bell
0245 00C0 4E 2C                 fcb     'N,     ',      0c
0246 00C2 46 24                 fcb     'F,     '$      0d
0247 00C4 43 3A                 fcb     'C,     ':      0e
0248 00C6 4B 28                 fcb     'K,     '(      0f
0249 00C8 54 35                 fcb     'T,     '5      10
0250 00CA 5A 22                 fcb     'Z,     '"      11
0251 00CC 4C 29                 fcb     'L,     ')      12
0252 00CE 57 32                 fcb     'W,     '2      13
0253 00D0 48 23                 fcb     'H,     '#      14
0254 00D2 59 36                 fcb     'Y,     '6      15
0255 00D4 50 30                 fcb     'P,     '0      16
0256 00D6 51 31                 fcb     'Q,     '1      17
0257 00D8 4F 39                 fcb     'O,     '9      18
0258 00DA 42 3F                 fcb     'B,     '?      19
0259 00DC 47 26                 fcb     'G,     '&      1a
0260 00DE 00 00                 fcb     ANul,   ANul    1b      Figs shift
0261 00E0 4D 2E                 fcb     'M,     '.      1c
0262 00E2 58 2F                 fcb     'X,     '/      1d
0263 00E4 56 3B                 fcb     'V,     ';      1e
0264 00E6 00 00                 fcb     ANul,   ANul    1f      Lets shift
```

```
temp                                    page   6
0266                        *---------------------------------------------------------------*
0267                        *            ROM Program
0268                        *---------------------------------------------------------------*
0269 03C0                           org     $3C0
0270                        SCInit
0271 03C0 CD 04 8A                  jsr     Debounce        wait out reset line bounce
0272 03C3 A6 11                     lda     #CtrlInitMask   set up Ctrl init pattern
0273 03C5 B7 00                     sta     Ctrl
0274 03C7 A6 00                     lda     #ParInitMask    set up Parallel init pattern
0275 03C9 B7 01                     sta     ParData
0276 03CB A6 31                     lda     #CtrlDDMask     set up Ctrl DDR
0277 03CD B7 04                     sta     Ctrl+DDR
0278 03CF A6 FF                     lda     #ParDDMask      set up ParData DDR
0279 03D1 B7 05                     sta     ParData+DDR
0280 03D3 A6 78                     lda     #AckInitMask    init the timer as an ack pulse catcher
0281 03D5 B7 09                     sta     AckCtrl
0282 03D7 A6 01                     lda     #1
0283 03D9 B7 08                     sta     AckCount
0284 03DB B6 02                     lda     SerRate         determine the baud rate and code type
0285 03DD A4 0F                     and     #SerRateMask        get the number
0286 03DF A1 09                     cmp     #9
0287 03E1 23 02                     bls     In5                 if rate > 9   (max rate is 9)
0288 03E3 A0 0A                     sub     #10                     then reduce it by 10
0289 03E5 AE 05                 In5 ldx     #5
0290 03E7 A1 04                     cmp     #4                  if rate >= 4 then install 8 bits
0291 03E9 25 02                     blo     In6                                 else install 5 bits
0292 03EB AE 08                     ldx     #8
0293 03ED BF 11                 In6 stx     NbrBits
0294 03EF 48                        lsla
0295 03F0 48                        lsla                    mult by table entry size (=4)
0296 03F1 B7 15                     sta     BIndex          set up rate pointer
0297 03F3 11 10                     bclr    FigsCase,BaudotCtrl init Baudot to unshifted mode
0298 03F5 AE 19                     ldx     #Buff           init the buffer pointers
0299 03F7 BF 16                     stx     BuffInPtr
0300 03F9 BF 17                     stx     BuffOutPtr
0301 03FB 3F 18                     clr     BuffCount
0302 03FD 20 00                     bra     Main
```

```
temp                                    page   7
0304                            *---------------------------------------------------------------------*
0305                            *       Main Loop
0306                            *---------------------------------------------------------------------*
0307 03FF                       Main    equ     *               Loop
0308 03FF CD 04 8A                      jsr     Debounce            Wait for a debouncing period
0309 0402 9A                            cli                         Allow interrupts
0310                            MLoop
0311 0403 06 00 04                      brset   Test,Ctrl,ML2       If Test low Then
0312 0406 9B                            sei                             Inhibit interrupts
0313 0407 AD 2A                         bsr     TMode                   Invoke Test Mode
0314 0409 9A                            cli                             Allow interrupts
0315 040A 04 00 10              ML2     brset   FCR,Ctrl,ML4        If FCR low Then
0316 040D A6 0D                         lda     #ACR                    load up an Ascii CR
0317 040F 06 11 02                      brset   3,NbrBits,ML3           If Baudot then get (raw form of)
0318 0412 A6 40                         lda     #BCR*8                      Baudot CR instead
0319 0414 AD 57                 ML3     bsr     BuffIn                  buffer up the CR
0320 0416 AD 72                         bsr     Debounce                let FCR debounce
0321 0418 05 00 FD                      brclr   FCR,Ctrl,*              wait for FCR to rise
0322 041B AD 6D                         bsr     Debounce                let FCR debounce
0323 041D 02 00 0E              ML4     brset   FFF,Ctrl,ML6        If FFF low Then
0324 0420 07 11 0B                      brclr   3,NbrBits,ML6           If Not Baudot Then
0325 0423 A6 0C                         lda     #AFF                        load up an Ascii FF
0326 0425 AD 46                         bsr     BuffIn                      buffer up the FF
0327 0427 AD 61                         bsr     Debounce                    let FFF debounce
0328 0429 03 00 FD                      brclr   FFF,Ctrl,*                  wait for FFF to rise
0329 042C AD 5C                         bsr     Debounce                    let FFF debounce
0330 042E CD 04 B7              ML6     jsr     DumpBuff            dump buffer (if nonempty) to printer
0331 0431 20 D0                         bra     MLoop
0332
0333                            *---------------------------------------------------------------------*
0334                            *       TMode subroutine -- put out a continuous series of printable
0335                            *       characters to the printer.
0336                            *---------------------------------------------------------------------*
0337
0338 0433                       TMode   equ     *               While true do
0339 0433 AD 55                         bsr     Debounce            Wait for a debouncing period
0340 0435 A6 41                         lda     #'A'                For char := 'A' to 'Z' do
0341 0437 07 00 01              TLoop   brclr   Test,Ctrl,TM2           If Test high
0342 043A 81                            rts                             Then return
0343 043B AD 0A                 TM2     bsr     ParOut                  Else Output the character
0344 043D 0F 09 FD                      brclr   Ack,AckCtrl,*                   await ack from printer
0345 0440 4C                            inca
0346 0441 A1 5A                         cmp     #'Z'
0347 0443 23 F2                         bls     TLoop
0348 0445 20 EC                         bra     TMode
0349
0350                            *---------------------------------------------------------------------*
0351                            *       Transmit character in A out the parallel port.
0352                            *       A register preserved; X reg, status bits klobbered.
0353                            *---------------------------------------------------------------------*
0354                            ParOut
0355 0447 AE 01                         ldx     #1              prime the ack detector for the next pulse
0356 0449 BF 08                         stx     AckCount
0357 044B 1F 09                         bclr    Ack,AckCtrl
0358 044D B7 01                         sta     ParData         stuff char to parallel port
0359 044F 11 00                         bclr    Strobe,ParCtrl  make strobe (about 5 uSec.)
0360 0451 10 00                         bset    Strobe,ParCtrl
0361 0453 81                            rts                     return
```

```
temp                                    page   8
0363                            *-----------------------------------------------------------------------*
0364                            *       Null Interrupt Handler -- for unimplemented interrupts
0365                            *-----------------------------------------------------------------------*
0366                            NullHandler
0367 0454 80                            rti
0368                            *-----------------------------------------------------------------------*
0369                            *       SerHandler -- Input a character to the char buffer from the serial
0370                            *       line.
0371                            *       Entered by low-going transition on interrupt line, denoting the
0372                            *       leading edge of the start bit.
0373                            *
0374                            *       Assumes serial format consisting of:
0375                            *       ~   indefinite mark interval  (hi signal)
0376                            *       1   spacing start bit         (lo signal)
0377                            *       N   data bits                 (lsb first; hi=1; N = 5 or 8)
0378                            *       1+  marking stop bit(s)       (hi signal)
0379                            *       ~   indefinite mark interval  (hi signal)
0380                            *-----------------------------------------------------------------------*
0381                            SerHandler
0382 0455 AD 39                         bsr     HalfBaud        wait for middle of start bit            8
0383                            *   Delay = 1/2 average instr. time + interrupt response time
0384                            *           + (HalfBaud routine) + the following instrs.
0385                            *         = 3 + 11 + (8ax + 12x + 49) + 16  =  8ax + 12x + 79
0386 0457 0C 00 12                      brset   RXD,SerData,SI9 if not still there, dismiss interrupt   10
0387 045A BE 11                         ldx     NbrBits         init the data bit counter               4
0388 045C A6 00                         lda     #0              init the data accumulator               2
0389                            SILoop
0390 045E AD 3C                         bsr     Baud            wait till middle of next bit time       8
0391                            *   loop length = (Baud routine) + (these instrs)
0392                            *               = (8ax + 12x + 49) + (30) = 8ax + 12x + 79
0393 0460 0D 00 00                      brclr   RXD,SerData,*+3 get RXD voltage state into C bit        10
0394 0463 46                            rora                    rotate C into MSB of A reg.             4
0395 0464 5A                            decx                    count down the number of bits           4
0396 0465 26 F7                         bne     SILoop          repeat until count exhausted            4
0397 0467 AD 04                         bsr     BuffIn          Now have character; buffer it up
0398 0469 0D 00 FD                      brclr   RXD,SerData,*   Spin until RXD in stop (hi) state
0399 046C 80                    SI9     rti                     return to interruptee
0400                            *-----------------------------------------------------------------------*
0401                            *       BuffIn -- Put char in A reg. into the buffer.
0402                            *       If Buffer full, discard character.
0403                            *       Sets DTR low whenever buffer is close to full.
0404                            *       Klobbers X.
0405                            *-----------------------------------------------------------------------*
0406 046D BE 18                 BuffIn  ldx     BuffCount       If Buffer not full
0407 046F A3 57                         cpx     #BuffSize
0408 0471 27 16                         beq     BI9
0409 0473 3C 18                 BI2     inc     BuffCount           Then update char count
0410 0475 BE 16                         ldx     BuffInPtr               store char
0411 0477 F7                            sta     0,x
0412 0478 5C                            incx                            update input pointer
0413 0479 A3 70                         cpx     #BuffEnd
0414 047B 25 02                         blo     BI4
0415 047D AE 19                         ldx     #Buff
0416 047F BF 16                 BI4     stx     BuffInPtr
0417 0481 BE 18                         ldx     BuffCount
0418 0483 A3 53                         cpx     #BuffSize-4             If buffer getting full
0419 0485 25 02                         blo     BI9
0420 0487 1A 00                         bset    DTR,SerCtrl                 Then restrain input data
0421 0489 81                    BI9     rts                     return
```

```
temp                                    page   9
0423                    *---------------------------------------------------------------------*
0424                    *       Delay for debounce period.
0425                    *       Klobbers A, X.
0426                    *---------------------------------------------------------------------*
0427                    Debounce
0428 048A AE 02                 ldx     #2
0429 048C A6 00                 lda     #0
0430 048E 20 18                 bra     Delay
0431                    *---------------------------------------------------------------------*
0432                    *       HalfBaud -- Wait a half-baud period.
0433                    *       Preserves A, X.
0434                    *
0435                    *       Delay = 8AX + 12X + 49 mpu cycles
0436                    *               where A and X are bytes 1 and 0 of the rate table entry.
0437                    *---------------------------------------------------------------------*
0438                    HalfBaud
0439 0490 BF 12                 stx     BTemp                                           5
0440 0492 B7 13                 sta     BTemp+1                                         5
0441 0494 BE 15                 ldx     BIndex                                          4
0442 0496 E6 81                 lda     RateTable+1,x                                   6
0443 0498 EE 80                 ldx     RateTable+0,x                                   6
0444 049A 20 0C                 bra     Delay                                           4
0445                    *---------------------------------------------------------------------*
0446                    *       Baud -- Wait One Baud period.
0447                    *       Preserves A, X.
0448                    *
0449                    *       Delay = 8AX + 12X + 49 mpu cycles
0450                    *               where A and X are bytes 3 and 2 of the rate table entry.
0451                    *---------------------------------------------------------------------*
0452                    Baud
0453 049C BF 12                 stx     BTemp                                           5
0454 049E B7 13                 sta     BTemp+1                                         5
0455 04A0 BE 15                 ldx     BIndex                                          4
0456 04A2 E6 83                 lda     RateTable+3,x                                   6
0457 04A4 EE 82                 ldx     RateTable+2,x                                   6
0458 04A6 20 00                 bra     Delay                                           4
0459                    *---------------------------------------------------------------------*
0460                    *       Delay -- delay for a value denoted by A, X registers.
0461                    *       Delay = 8AX + 12X + 19 mpu cycles
0462                    *               where A, X values of zero are taken as 256.
0463                    *       X restored from BTemp, A restored from BTemp+1.
0464                    *---------------------------------------------------------------------*
0465                    Delay
0466 04A8 B7 14                 sta     BTemp+2                         5
0467 04AA B6 14         Dl1     lda     BTemp+2                         4       x
0468 04AC 4A            Dl2     deca                                    4   a   x
0469 04AD 26 FD                 bne     Dl2                             4   a   x
0470 04AF 5A                    decx                                    4       x
0471 04B0 26 F8                 bne     Dl1                             4       x
0472 04B2 BE 12                 ldx     BTemp                           4
0473 04B4 B6 13                 lda     BTemp+1                         4
0474 04B6 81                    rts                                     6
```

```
temp                                    page 10
0476                    *---------------------------------------------------------------------*
0477                    *       DumpBuff -- Dump as much as possible of the contents of the
0478                    *       buffer to the printer port; translate Baudot to Ascii if needed.
0479                    *       Klobbers A, X.
0480                    *---------------------------------------------------------------------*
0481                    DumpBuff
0482 04B7 3D 18        DBLoop  tst     BuffCount       While buffer not empty
0483 04B9 27 12                beq     DBEnd
0484 04BB AD 30                bsr     BuffOut             get char
0485 04BD 06 11 05             brset   3,NbrBits,DB12      If Baudot Mode
0486 04C0 44                   lsra                            Then finish shifting char
0487 04C1 44                   lsra
0488 04C2 44                   lsra
0489 04C3 AD 09                bsr     BtoA                    Translate to Ascii
0490 04C5 CD 04 47      DB12    jsr     ParOut              output the char
0491 04C8 0F 09 FD             brclr   Ack,AckCtrl,*       await ack from printer
0492 04CB 20 EA                bra     DBLoop
0493 04CD 81            DBEnd   rts
0494                    *---------------------------------------------------------------------*
0495                    *       BtoA -- Convert the char in A from Baudot to Ascii.
0496                    *       Klobbers X.
0497                    *---------------------------------------------------------------------*
0498 04CE A1 1B         BtoA    cmp     #Figs                   If Figs shift then
0499 04D0 26 04                 bne     BA4
0500 04D2 10 10                 bset    FigsCase,BaudotCtrl         record it
0501 04D4 20 0E                 bra     BA8                         translate & return
0502 04D6 A1 1F         BA4     cmp     #Lets                   If Lets shift then
0503 04D8 26 04                 bne     BA6
0504 04DA 11 10                 bclr    FigsCase,BaudotCtrl         record it
0505 04DC 20 06                 bra     BA8                         translate & return
0506 04DE A1 04         BA6     cmp     #Space                  If space code then
0507 04E0 26 02                 bne     BA8                         {auto unshift on space}
0508 04E2 11 10                 bclr    FigsCase,BaudotCtrl         record it and output it
0509 04E4 48            BA8     asla                            Turn char into lookup index
0510 04E5 97                    tax
0511 04E6 01 10 01              brclr   FigsCase,BaudotCtrl,BA10 If in Figs case then
0512 04E9 5C                    incx                                add figs offset
0513 04EA E6 A8         BA10    lda     BtoATable,x             Look up Baudot from Ascii
0514 04EC 81                    rts
0515                    *---------------------------------------------------------------------*
0516                    *       BuffOut -- return a character from the buffer in Accumulator.
0517                    *       Set DTR high whenever buffer is not too full.
0518                    *       ** Assumes Buffer is nonempty **
0519                    *       Klobbers X.
0520                    *---------------------------------------------------------------------*
0521 04ED 3A 18         BuffOut dec     BuffCount       update char count
0522 04EF BE 17                 ldx     BuffOutPtr      fetch char
0523 04F1 F6                    lda     0,x
0524 04F2 5C                    incx                    update output pointer
0525 04F3 A3 70                 cpx     #BuffEnd
0526 04F5 25 02                 blo     B04
0527 04F7 AE 19                 ldx     #Buff
0528 04F9 BF 17         B04     stx     BuffOutPtr
0529 04FB BE 18                 ldx     BuffCount
0530 04FD A3 4F                 cpx     #BuffSize-8     if buffer not too full
0531 04FF 22 02                 bhi     B09
0532 0501 1B 00                 bclr    DTR,SerCtrl         then unrestrain input data
0533 0503 81            B09     rts                     return.
0534                    ************************************************************************
0535                    *
0536                    *       interrupt vectors
0537                    *
0538 07F8                       org     $7F8            start of vectors
0539                    *
0540 07F8 04 54                 fdb     NullHandler     timer interrupt
0541 07FA 04 55                 fdb     SerHandler      serial-centronics inerrupt
0542 07FC 04 54                 fdb     NullHandler     sw interrupt
0543 07FE 03 C0                 fdb     SCInit          reset
```