Stamp Applications no. 20 (October '96):

Use the BS1's Debug Output For Stamp-PC Communication

Plus a big-digit clock demo
for the BASIC Stamp II

by Scott Edwards

GIVE ME ONE MORE PIN! That seems to be the battle cry of Stamp users everywhere. This month, we'll show you a sneaky way to use the Stamp's debug capability to send data to a PC through the serial port.

For you BS2 users, I'll also present a big-digit clock application. A sneaky technique uses graphics symbols and lookup tables to create 1"-high digits on an ordinary a 4x20 serial LCD.

The Debug protocol.

I can almost guarantee you that your mental model of how BS1 Debug works is entirely wrong. Don't feel bad; mine was too.

Debug is the instruction that lets you examine the contents of any variable through a window in the STAMP.EXE host program. For example, debug b2 shows you the contents of variable b2. Before I started researching this article, I'd have guessed that the debug instruction caused the Stamp to send the contents of b2 up the programming cable via some custom serial protocol. I would have been wrong on both counts!

Any debug instruction in your program causes the BS1 to send the contents of *all* variables up the programming cable in plain, old 4800-baud asynchronous serial, true polarity. That means that if you invert this data and convert it to RS-232 levels, you can receive it with a PC running terminal software.

You can program a Stamp with listing 1 and a PC with listing 2, connect them together as shown in figure 1, and bingo, you can examine every byte of the Stamp's memory.

This approach has a couple limitations. First of all, at 4800 baud, the Stamp takes almost two-tenths of a second to send a debug string.

Table 1. Debug Data Format

Byte No.	Use/Meaning
0—63	Synchronization
64—70	Used by STAMP.EXE
41	Pins (input)
72—80	Used by STAMP.EXE
81	Pins (output)
82	Dirs (direction register)
83—96	B0—B13 user variable3

Since only 17 bytes of the debug format are of any real use, that works out to only 80 bytes per second (equivalent to 800 baud).

And while I was working on the program in listing 2, I got occasional device errors from the PC com port. I never did figure out what caused them, since the error message provides no additional clues as to the origin of the problem.

Despite those drawbacks, I can't help but think that this information will be mighty handy in some applications.

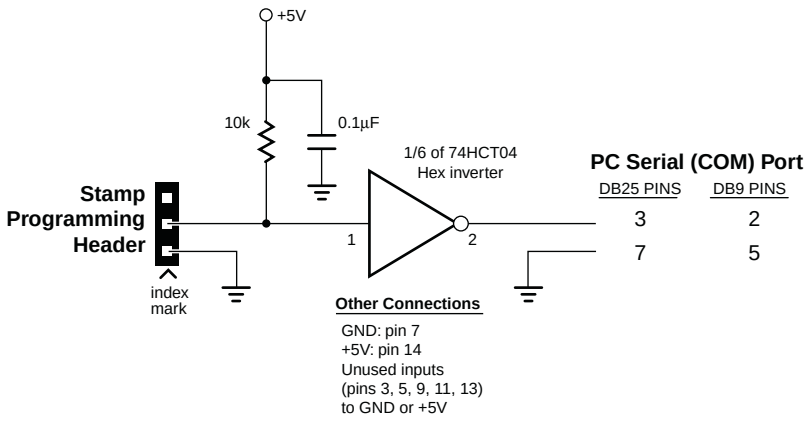

Figure 1. Hookup diagram for viewing debug output.

Giant-Character Clock Demo. My company makes those serial LCD modules (LCD plus our custom Backpack daughterboard) that so many Stamp users incorporate into their applications. My customers often call looking for really BIG displays that can be read across the room. The largest standard modules on the market have characters less than a half inch tall, and most have serious price tags.

To make matters worse, many of these applications have the conflicting requirement that the display also be able to show lots of data at a glance. So the goal is a display that has a few big characters, and lots of little characters.

Our next application is my solution to this dilemma, based on a 4x20 serial LCD module. Those of you who are interfacing LCDs directly can undoubtedly adapt the approach to your own application.

The idea is to use the LCD's eight user-defined characters as building blocks to construct 4-line-tall symbols. Figure 2 shows the custom characters I used; figure 3 is the hookup diagram that also illustrates how the custom characters can be arranged to create giant numerals.

For the clock portion of the demo, I used the real-time clock chip discussed in my Data Collection Proto Board article (*N&V*, March '96).

Listing 3 shows how the demo works. The part of the program that generates the big characters depends heavily on a series of lookup tables. Since the custom symbols fall in the range of 0 to 7, which can be expressed as a 3-bit number, they are represented by 4-bit nibbles in the lookup table. Each 16-bit word of the lookup tables actually represents four custom symbols. The BS2's nibble-addressing capability makes it a snap to unpack the symbols to send them to the display.

I liked this technique so much that I altered the firmware of my product, the LCD Serial Backpack, to load the custom symbol set of figure 2 upon initialization. All 4x20 LCD modules sold after July '96 have these symbols built in. If you want to define your own characters, you still can. Just download the new ones as usual.

But if you're using this giant-character display procedure, you can save more than 64 bytes of program memory by skipping the character downloading step at the beginning of the program.

An interesting side effect of the way the program works—reserving place for the colon between the hours and minutes digits with space characters—has the useful side effect of blinking the colon without any code overhead!

#	Graphic	Data	#	Graphic	Data
0		0,0,0,1,3,7,15,31	4		0,0,0,0,31,31,31,31
1		0,0,0,16,24,28,30,31	5		31,31,31,31,0,0,0,0
2		31,15,7,3,1,0,0,0	6		31,31,31,31,31,31,31,31
3		31,30,28,24,16,0,0,0	7		0,0,0,0,0,0,0,0

Figure 2. Custom symbols that make up the giant numerals.

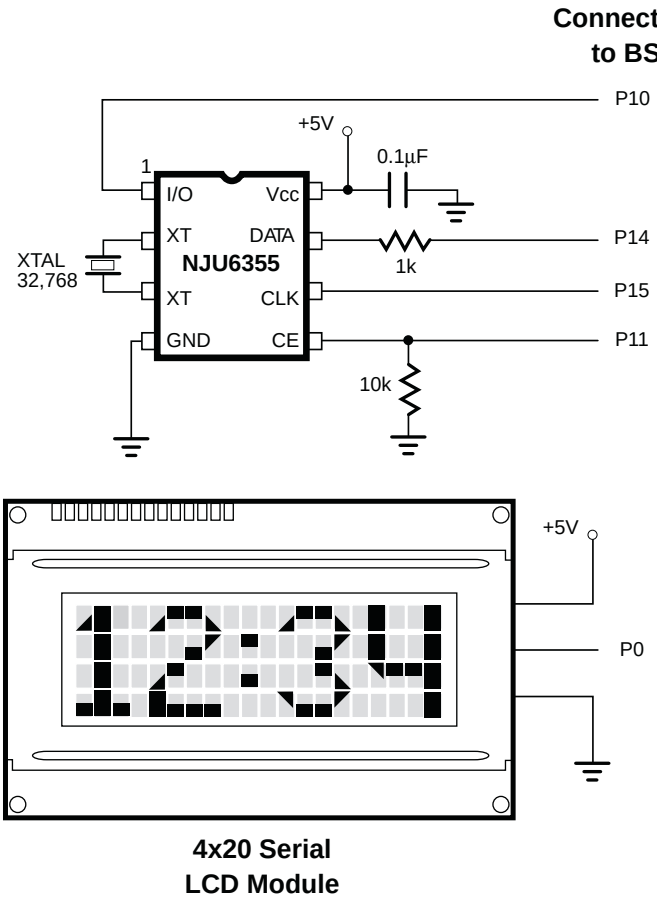

Figure 3. Hookup diagram for the BIG_TIME demo program.

NOTE:

The big-character procedure described here is pretty much obsolete. All 4x20 and 4x40 displays available from www.seetron.com have automatic big-character generation built in (and have had this capability for several years).

This article was originally published in 1996. The Stamp Applications column continues with a changing roster of writers. See www.nutsvolts.com or www.parallaxinc.com for current Stamp-oriented information.

Listing 1. BS1 Program to Demonstrate Debug Output

```
'Program: DBUG.BAS
' This program sets the various memory variables to known
' values for viewing with the PC program in listing 2.

b0 = "A": b1 = "B": b2 = "C": b3 = "D"
b4 = "E": b5 = "F": b6 = "G": b7 = "H"
b8 = "I": b9 = "J": b10 = "K": b11 = "L"
b12 = "M":b13 = "N"
dirs = %00001111: pins = 0

loop:
  pause 2000           ' Wait 2 seconds.
  debug b0             ' Debug output.
  b0 = b0 + 1          ' Increment B0.
  random w6            ' Random number in B12 and B13 (W6).
goto loop              ' Repeat forever.
```

Listing 2. QBASIC Debug Viewer (for PCs running DOS)

```
DECLARE SUB showDebug ()
' Program: DBUG_IN.BAS (QBASIC program to work with BS1 debug output)
' This program demonstrates how to capture and interpret the serial
' data output by the BS1's debug instruction _without_ use of the
' STAMP.EXE software. This capability can be very handy when you
' need one more output from the Stamp, and only require one-way
' communication with the PC (Stamp -> PC), as for data acquisition.
' The center pin of the Stamp's 3-pin programming header must be
' connected to the serial data in of COM1 through a CMOS inverter
' or RS-232 line driver, as shown in the accompanying article.

' See the text and table 1 for the debug protocol.

DIM SHARED i AS INTEGER
DIM SHARED item AS INTEGER
DIM SHARED row AS INTEGER
DIM SHARED debugData$

' Open communications through com1 serial port. Set up the following
' parameters: 4800 baud, no parity, 8 data bits, 1 stop bit.
' Disable handshaking by setting the timeout values for all of the
' handshake inputs to zero; carrier detect (CD), clear to send (CS),
' data set ready (DS). In addition, disable the timeout for OPENing
' the port itself with OP0. Finally, set the port for INPUT access
' and assign it a 1024-byte receive buffer.

CLOSE                   ' In case port is left open from previous run.
' Open the com port for input at 4800 baud with 4096-byte buffer.
OPEN "com1:4800,N,8,1,CD0,CS0,DS0,OP0" FOR INPUT AS #1 LEN = 4096

' Now print the labels to the screen.
CLS : PRINT "                      =====BS1 DEBUG VIEWER====="
LOCATE 4, 25: PRINT "ASC"
LOCATE 4, 35: PRINT "DEC"
LOCATE 4, 45: PRINT "HEX"
LOCATE 5, 10: PRINT "INs:"
LOCATE 6, 10: PRINT "OUTs:"
LOCATE 7, 10: PRINT "Dirs:"
FOR i = 0 TO 13
  LOCATE (i + 8), 10: PRINT "B"; LTRIM$(RTRIM$(STR$(i))); ": "
NEXT

' Collect the debug data in a string variable. The loop below
' synchronizes on the 64 $F0 characters (240 decimal) sent by
' the Stamp at the beginning of a debug.
start:
i = 0
DO WHILE i < 64
again:
```

```
 IF LOF(1) = 0 THEN GOTO again
 IF NOT EOF(1) THEN debugData$ = INPUT$(1, #1) ELSE GOTO again
 IF debugData$ = CHR$(240) THEN i = i + 1 ELSE i = 0
LOOP

' After the 64 sync characters, this instruction grabs the
' next 33 bytes that make up the debug output. Of these,
' only 17 hold useful information, but this is the
' easiest way to collect the data.
hold:
IF LOF(1) > 33 THEN debugData$ = INPUT$(33, #1) ELSE GOTO hold

' Show the INs register, which is located 10 items below the
' other registers in the string.
i = 0: row = 5: item = 8: showDebug

' Now show the other registers, OUTs, Dirs, and B0 through B13.
row = 6: item = 18
FOR i = 0 TO 15
  showDebug
NEXT

GOTO start              ' Repeat until CTL-Break

SUB showDebug
  LOCATE (row + i), 26: PRINT MID$(debugData$, (item + i), 1); "  ";
  LOCATE (row + i), 34: PRINT ASC(MID$(debugData$, (item + i), 1)); "   ";
  LOCATE (row + i), 46: PRINT HEX$(ASC(MID$(debugData$, (item + i), 1))); "
";
END SUB
```

Listing 3. Program Demonstrating Big Numerals on Serial LCD

```
' Program: BIG_TIME.BS2
' This program demonstrates a method for using a 4-line by
' 20 character serial LCD module to display 1-inch high
' numerals. In this demo, the BS2 displays the current time
' (HH:MM) in 1" digits, thanks to the assistance of an NJU6355
' clock chip, connected as shown in the accompanying article.
' (Owners of the Data Collection Proto Board can run this
' program without modification as the pin assignments for the
' clock are the same. Use the unswitched +5V supply for the
' LCD.)

'=====================================================================
'        PIN ASSIGNMENTS, SYSTEM CONSTANTS, TEMPORARY VARIABLES
'=====================================================================

CLK     con     15      ' Clock line for all serial peripherals.
DATA_   con     14      ' Data line for all serial peripherals.
NJU_CE  con    11       ' Chip-enable for NJU6355 clock/calendar.
NJU_IO  con    10       ' IO (read/write) for NJU6355; 1=write.
temp    var    byte     ' Temporary variable used in several routines.
nbl     var    nib      ' Temporary nibble.

'=====================================================================
'            NJU6355 CLOCK/CALENDAR CONSTANTS AND VARIABLES
'=====================================================================
' The NJU6355ED clock/calendar chip maintains a 13-digit BCD account
' of the current year, month, day, day of week, hour, minute, and
' second. The clock subroutines transfer this data to/from a 13-nibble
' array in the BS2's RAM called "DTG" for "date-time group." The
' constants below allow you to refer to the digits by name; e.g.,
' "Y10s" is the tens digit of the year. Note that there's no "am/pm"
' indicator--the NJU6355 uses the 24-hour clock. For instance, 2:00 pm
' is written or read as 14:00 (without the colon, of course).
Y10s     con    1        ' Array position of year 10s digit.
Y1s      con    0        '  "       "     "  year 1s    "
Mo10s    con    3        '  "       "     "  month 10s  "
Mo1s     con    2        '  "       "     "  month 1s   "
D10s     con    5        '  "       "     "  day 10s    "
D1s      con    4        '  "       "     "  day 1s     "
H10s     con    8        '  "       "     "  hour 10s   "
H1s      con    7        '  "       "     "  hour 1s    "
M10s     con    10       '  "       "     "  minute 10s "
M1s      con    9        '  "       "     "  minute 1s  "
S10s     con    12       '  "       "     "  second 10s "
S1s      con    11       '  "       "     "  second 1s  "
day      con    6        '  "       "     "  day-of-week (1-7) digit.
digit    var    nib      ' Number of 4-bit BCD digits read/written.
DTG      var    nib(13)  ' Array to hold "date/time group" BCD digits.
```

```
'=====================================================================
'            LCD SERIAL BACKPACK CONSTANTS/VARIABLES
'=====================================================================
' The display for this application is a 4x20 alphanumeric LCD
' equipped with a Backpack daughterboard to convert it to a serial
' device. Newer Backpacks (sold July 96 and after) have the big-
' character building-block symbols preprogrammed; older units require
' that they be downloaded. If you're unsure, program the BS2 with
' just the line "serout 0,$4054,[0,1,2,3,4,5,6,7]" You should see
' an orderly row of ramp- and block-shaped symbols if the characters
' are built in. Otherwise, you'll see random patterns of dots. If
' that's the case, remove the comment marks from the sections of
' code indicated below.

I       con    254      ' Instruction prefix.
ClrLCD  con    1        ' Clear-LCD instruction.
N96N    con    $4054    ' 9600 baud, inverted, no parity.
cgRAM   con    64       ' Address 0 of CG RAM.

EEptr   var    word     ' Pointer into EEPROM.
pat     var    EEptr    ' Alias for EEptr.
line    var    nib      ' LCD line

' If the 4x20 serial LCD module you're using was purchased after July
' 1996, you may omit this code. Otherwise, remove the comment marks
' (') from the beginning of the lines below to activate this code.
'bitPat0       DATA    0,0,0,1,3,7,15,31         ' Left-right up-ramp
'bitPat1       DATA    0,0,0,16,24,28,30,31      ' Right-left "   "
'bitPat2       DATA    31,15,7,3,1,0,0,0         ' Left-right down ramp.
'bitPat3       DATA    31,30,28,24,16,0,0,0      ' Right-left "    "
'bitPat4       DATA    0,0,0,0,31,31,31,31       ' Lower block.
'bitPat5       DATA    31,31,31,31,0,0,0,0       ' Upper block.
'bitPat6       DATA    31,31,31,31,31,31,31,31   ' Full block.
'bitPat7       DATA    0,0,0,0,0,0,0,0           ' Full blank

'=====================================================================
'                         DEMONSTRATION PROGRAM
'=====================================================================
  DIRS = $FFFF                  ' Write 1s to all direction bits.

' setup =============
' Set the clock.

  DTG(Y10s)=9: DTG(Y1s)=6       ' Year = 96.
  DTG(Mo10s)=0: DTG(Mo1s)=7     ' Month = 07.
  DTG(D10s)=0: DTG(D1s)=5       ' Day = 05.
  DTG(day) = 2                  ' Day of week (1-7) = 2 (Tuesday).
  DTG(H10s)=1: DTG(H1s)=2       ' Hour = 12.
  DTG(M10s)=5: DTG(M1s)=1       ' Minute = 50.
  gosub write_clock             ' Write data to clock.
```

```
low 0                   ' Make the serial output low
pause 1000              ' Let the LCD wake up.

' ============================================================================
'                 Define Symbols in CG RAM
' ============================================================================
' If the 4x20 serial LCD module you're using was purchased after July
' 1996, you may omit this code. Otherwise, remove the comment marks
' (') from the beginning of the lines below to activate this code.
'serout 0,N96N,[I,cgRAM]        ' Enter CG RAM.
'for EEptr = 0 to 63            ' Write the bit patterns..
'  Read EEptr,temp              ' ..to the LCD.
'  serout 0,N96N,[temp]
'next

serout 0,N96N,[I,ClrLCD]        ' Clear the LCD.
pause 1

' demo  =============
' Continuously display the current date and time to the debug screen.
demo:
  gosub read_clock              ' Update DTG data.
  gosub bigClock                ' Display on LCD.
  serout 0,N96N,[I,201,5,I,157,4]
  pause 1000                    ' Wait a second.

goto demo                       ' Do it again.

' ============================================================================
'                   NJU6355 CLOCK/CALENDAR SUBROUTINES
' ============================================================================

' read_clock =============
' Get the current date/time group from the NJU6355 clock and store
' it in the array DTG(n).
read_clock:
  low NJU_IO                                    ' Set for read.
  high NJU_CE                                   ' Select the chip.
  for digit = 0 to 12                           ' Get 13 digits.
    shiftin DATA_,CLK,lsbpre,[DTG(digit)\4]     ' Shift in a digit.
  next                                          ' Next digit.
  low NJU_CE                                    ' Deselect the chip.
return                                          ' Return to program.

' write_clock =============
' Get the time stored in DTG(n) and write it to the NJU6355 clock.
' Note that the NJU6355 does not allow you to write the seconds digits.
' If clears the seconds digits when written, so if you set it for
' 08:30 (hh:mm), when the write is complete, the NJU6355 starts at
' 08:30:00 (hh:mm:ss).
```

```
write_clock:
  high NJU_IO                                    ' Set for write.
  high NJU_CE                                    ' Select the chip.
  for digit = 0 to 10                            ' Write 11 digits.
    shiftout DATA_,CLK,lsbfirst,[DTG(digit)\4]   ' Shift out a digit.
  next                                           ' Next digit.
  low NJU_CE                                     ' Deselect the chip.
return                                           ' Return to program.

' ===========================================================================
'                 Subroutine Displaying Large Numbers
' ===========================================================================
bigClock:
for line = 0 to 3                          ' Four lines to display.
  lookup line,[128,192,148,212],temp       ' Get start address of line.
  serout 0,N96N,[I,temp]                   ' Position the cursor on line.
  for digit = 3 to 0                       ' For each digit:
    lookup digit,[9,10,7,8],nbl            '  Get clock data.
    nbl = DTG(nbl)
    gosub getPattern                       '  Get symbols for line/digit.
    serout 0,N96N,[pat.nib3,pat.nib2,pat.nib1,pat.nib0]  ' Send to LCD.
    if digit <> 2 then skip1               ' Make space for colon..
    serout 0,N96N,[32,32,32]               ' ..after 2nd digit.
skip1:
  next                                     ' next digit.
next                                       ' next line.
return

' ===========================================================================
'              Subroutines Defining Big-Character Patterns
' ===========================================================================
' Each digit is represented by four lines of four symbols. The branch
' instruction below picks the appropriate lookup table for the current
' line. The lookup table then returns the corresponding four symbols
' packed into a single word (16-bit) variable.

getPattern:
branch line,[first,second,third,fourth]

'             0     1     2     3     4     5     6     7     8     9
'            ---   ---   ---   ---   ---   ---   ---   ---   ---   ---
first:
lookup nbl,[$0551,$7067,$0551,$0551,$6776,$6555,$0557,$2556,$0551,$0551],pat
return

second:
lookup nbl,[$6776,$7767,$7743,$7743,$6776,$2441,$6041,$7703,$2443,$6776],pat
return
```

```
third:
lookup nbl,[$6776,$7767,$0577,$7751,$2556,$7776,$6776,$7767,$0551,$2536],pat
return

fourth:
lookup nbl,[$2443,$7464,$6444,$2443,$7776,$2443,$2443,$7767,$2443,$7443],pat
return
```