LCD Serial Backpack App Note No. 2

Getting Graphic: Defining Custom LCD Characters

Block graphics add new symbols and enable animated effects

ALPHANUMERIC LCDS are inexpensive and easy to use. But there are times when you'd like to add some graphical pizzazz to your display *without* the expense and code overhead of a graphics display.

With a little ingenuity, you may find that the block-graphics capabilities of standard LCDs can fill the bill without busting your budget. This application note will show you how to:

- Define and use custom characters.
- Create simple animations.
- Redefine visible symbols for complex animations.

Background. This application note is intended primarily for displays equipped with the LCD Serial Backpack™, a daughterboard that gives 14-pin alphanumeric LCDs an easy-to-use serial interface. However, the principles described here apply to standard LCD modules as well.

Character-Generator ROM and RAM. When you send the ASCII code for a character like "A" to an LCD module, the module's controller looks up the appropriate 5x8-pixel pattern in ROM (read-only memory) and displays that pattern on the LCD. That *character-generator* ROM contains 192 bit maps corresponding to the alphabet, numbers, punctuation, Japanese Kanji characters, and Greek symbols.

The ROM is part of the main LCD controller (e.g., HD44780, KS0066, etc.), is mask-programmed, and cannot be changed by the user. The manufacturers do offer alternative symbols sets in ROM for European and Asian languages, but most U.S. distributors stock only the standard character set shown in the LCD Serial Backpack manual.

Alphanumeric LCD controllers do not allow you to turn individual pixels on or off—they just let you pick a particular pattern (corresponding to an ASCII code) and display it on the screen.

If you can't change the ROM and you can't control pixels, how do you create graphics on these LCDs? Easy. There's a 64-byte hunk of RAM (random-access memory) that the LCD controller uses in the same way as character-generator (CG) ROM. When the controller receives an ASCII code in the range that's mapped to the CG RAM, it uses the bit patterns stored there to display a pattern on the LCD. The main difference is that you can write to CG RAM, thereby defining your own graphic symbols.

Custom-Symbol Worksheet

Use this worksheet to design custom-character bitmaps for alphanumeric LCDs. Color in the squares representing black pixels. Then, for each row, add the numbers from the tops of each column in which there's a colored-in square. Write the result in the Data (value) blank. These are the byte values you will download to CG RAM to create your symbols.

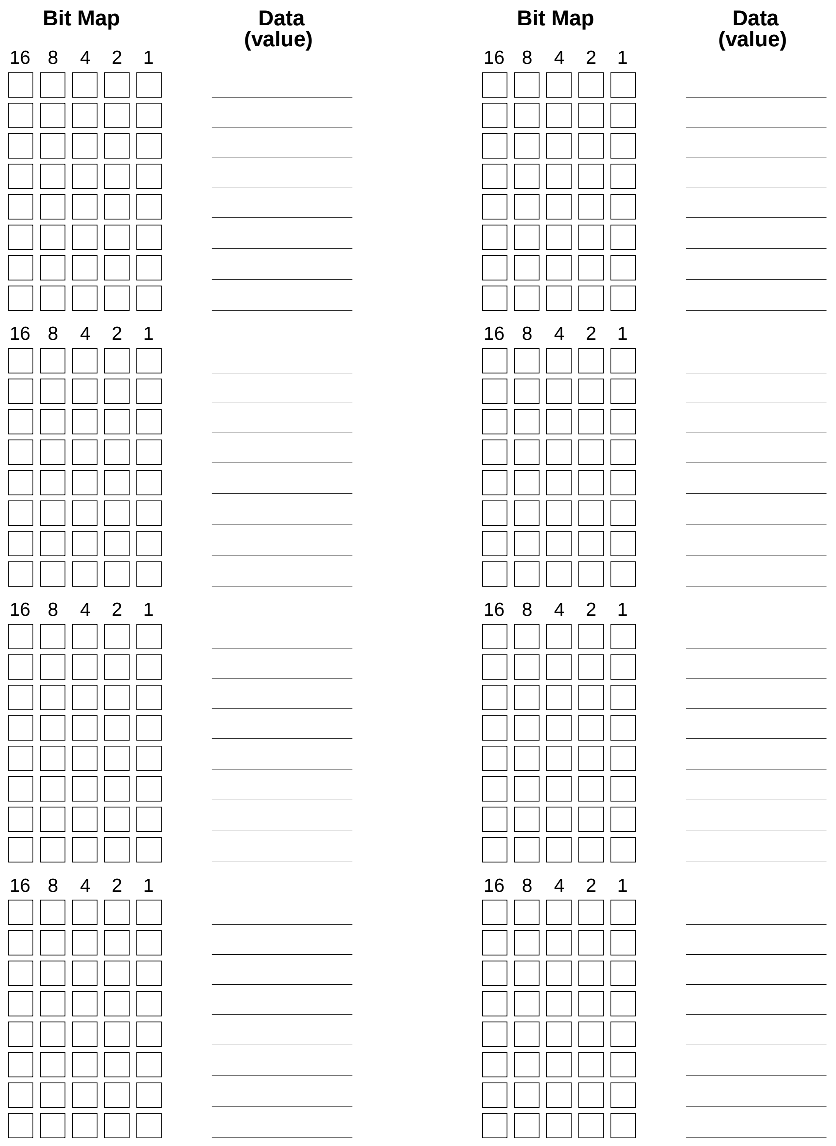

Figure 2.1 Worksheet for calculating custom-character bitmaps.

Each byte of CG RAM is mapped to a five-bit horizontal row of pixels, and LCD characters are typically eight rows high, so 64 bytes of CG RAM is enough to define eight custom characters. These characters correspond to ASCII codes 0 through 7, which normally serve as control codes for marking the beginning of a serial transmission or ringing the bell on a terminal. Since these have no meaning to an LCD module, the designers appropriated them for CG RAM.

When an LCD is first powered up, CG RAM contains random bits—garbage. LCDs with newer LCD Serial Backpacks are initialized with a set of wedge- and block-shaped symbols that can be used to draw large characters on 4-

line displays; see app note 1 in this series. Either way, you may overwrite the CG RAM with your own symbols.

Writing to CG RAM. As the Backpack manual shows, writing to CG RAM is a lot like moving the cursor to a particular position on the display and displaying characters at that new location. The steps are:

- Send the instruction-prefix byte (254). For non-Backpack users, this is the same as setting the RS bit for the next byte. With the Backpack, RS is automatically reset after one byte.
- Set the CG RAM address by sending a byte from 64 to 127 (locations 0–63 in CG RAM).
- Send bytes with the bit patterns for your symbol(s). The LCD controller automatically increments CG RAM addresses, just as it does cursor positions on the display.
- To leave CG RAM, send the instruction prefix (byte 254) followed by a valid display address (e.g. 128, 1st character of 1st line); the clear-screen instruction (byte 1); or the home instruction (byte 2). Now bytes are once again being written to the visible portion of the display.

The Backpack manual shows an example that defines character 0 as a diamond shape and displays a row of diamonds across the screen. Figure 2.1 is a worksheet to help you calculate bit patterns for your own symbols. Figure 2.2 shows an example symbol, while listings 2.1 through 2.3 are PBASIC (Stamps I and II) and QBASIC examples that define and use these symbols.

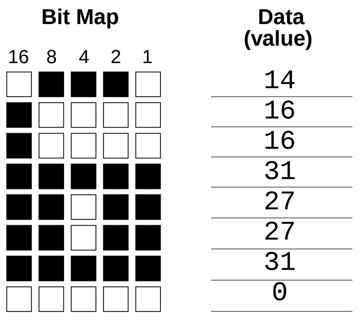

Figure 2.2 Example custom symbol.

Simple Animation. You can create a sort of animation by rapidly printing a series of custom characters at the same screen position. Program listings 2.4 through 2.6 (BS1, BS2 and QBASIC) define a rotating stick symbol using four graphic characters and “spin” this symbol in the middle of the second line of the display.

Animation in Multiple Locations. There’s a useful side effect to the way the LCD controller uses CG RAM. Normally, we define a pattern in CG RAM, then print the character. But you can also change the CG RAM for characters that are already on the screen, and their appearance will change. This opens up two neat possibilities:

- You can create animations with more frames than the eight CG RAM locations. Since new bit patterns are loaded from an external computer, the only limit on the number of frames is that computer’s storage capacity.
- You can animate multiple screen locations simultaneously. Just print the graphics character you want to animate in multiple screen locations, then cycle different bit patterns through its CG RAM. All instances of that character will change.

Listings 2.7 through 2.9 show an example of this kind of animation in BS1, BS2 and QBASIC programs. The second line of the display is filled with character 0, then that character is redefined in an expanding-box pattern. The result is 16 simultaneous animations.

Sources

Scott Edwards Electronics (SEE) manufactures and sells the LCD Serial Backpack™, a daughterboard that gives 14-pin alphanumeric LCDs an easy-to-use serial interface. The Backpack automatically initializes the LCD, receives serial data at a user-selectable rate of 2400 or 9600 baud and converts received data into LCD-compatible parallel output. The Backpack supports all standard alphanumeric LCDs up to 80 characters total (4x20 or 2x40).

The Backpack is available by itself for installation on a user-supplied LCD, or preinstalled to one of several high-quality supertwist LCDs. See the catalog for current offerings.

SEE also offers 4x40 serial LCD modules with additional terminal-emulation and special-features including automatic large-number display.

You may download a current SEE catalog from the customer-support FTP archive: ftp.nutsvolts.com, in pub/nutsvolts/scott. Or contact—

Scott Edwards Electronics
PO Box 160
Sierra Vista, AZ 85635
ph: 520-459-4802 fax: 520-459-0623

```
' Listing 2.1: GRPHIX1.BAS
' (Defining custom graphics symbols with LCD Backpack & BS1)
' This program demonstrates how to define and use custom
' symbols through the LCD Serial Backpack. The symbol patterns
' are loaded into the LCD from the Stamp's EEPROM. Thereafter,
' whenever you send an ASCII value of 0 - 7 to the LCD, it
' displays the custom symbol defined (until power is turned off).

SYMBOL  lcd    = 0      ' Serial output to LCD.
SYMBOL  I      = 254    ' Instruction prefix for LCD.
SYMBOL  lcdCls = 1      ' Clear-screen instruction for LCD.
SYMBOL  cgRAM  = 64     ' Starting point of CG RAM.
SYMBOL  line2  = 192    ' Position: start of 2nd line.
SYMBOL  count  = b11    ' Counter for EEPROM addressing.
SYMBOL  char   = b10    ' EEPROM byte.

' Custom-character definitions stored as bit patterns in the
' Stamp's EEPROM. Each character is made up of 8 bytes. Each byte
' represents a row of 5 pixels (only the lowest 5 bits are
' actually used). Comments at the end of each EEPROM entry show
' the appearance of the corresponding graphic.
EEPROM (0,4,2,31,2,4,0,0)        '   right arrow
EEPROM (0,4,8,31,8,4,0,0)        '   left arrow
EEPROM (4,14,21,4,4,0,0,0)       '   up arrow
EEPROM (0,0,0,4,4,21,14,4)       '   down arrow
EEPROM (8,20,20,8,5,6,5,5)       '   single-character "OK"
EEPROM (14,17,17,31,27,27,31,0)  '   locked
EEPROM (14,16,16,31,27,27,31,0)  '   unlocked
EEPROM (17,18,23,9,19,4,7,0)     '   single-character "1/2"

' =====Program begins by downloading custom character patterns
' from EEPROM after a 1-second delay for LCD initialization.
pause 1000                       ' Wait a second
serout lcd,n2400,(I,cgRAM)       ' Point to character-generator RAM.
for count = 0 to 63              ' For each of 8 chars x 8 bit rows.
  read count,char                ' Get the bit pattern from EEPROM.
  serout lcd,n2400,(char)        ' Write the data to LCD CG RAM.
next
```

```
serout lcd,n2400,(I,lcdCls)              ' Clear the LCD.
serout lcd,n2400,("CUSTOM GRAPHICS")     ' Label top line.

' ============= DEMO PROGRAM ============
' Move to line 2, show custom-character symbols.
serout lcd,n2400,(I,line2)
for count = 0 to 7
  serout lcd,n2400,(count," ")  ' Print character, then a space.
next
end
```

```
' Listing 2.2: GRPHIX1.BS2
' (Defining custom graphics symbols with LCD Backpack & BS2)
' This program demonstrates how to define and use custom
' symbols through the LCD Serial Backpack. The symbol patterns
' are loaded into the LCD from the Stamp's EEPROM. Thereafter,
' whenever you send an ASCII value of 0 - 7 to the LCD, it
' displays the custom symbol defined (until power is turned off).
' Install a jumper at BPS on Backpack for 9600 baud operation

lcd     con    0       ' Serial output to LCD.
I       con    254     ' Instruction prefix for LCD.
lcdCls  con    1       ' Clear-screen instruction for LCD.
cgRAM   con    64      ' Starting point of CG RAM.
line2   con    192     ' Position: start of 2nd line.
N9600   con    $4054   ' 9600 baud output to Backpack.
cnt     var    byte    ' Counter for EEPROM addressing.
char    var    byte    ' EEPROM byte.

' Custom-character definitions stored as bit patterns in the
' Stamp's EEPROM. Each character is made up of 8 bytes. Each byte
' represents a row of 5 pixels (only the lowest 5 bits are
' actually used). Comments at the end of each EEPROM entry show
' the appearance of the corresponding graphic.
RA      data   0,4,2,31,2,4,0,0 '   right arrow
LA      data   0,4,8,31,8,4,0,0 '   left arrow
UA      data   4,14,21,4,4,0,0,0        '   up arrow
DA      data   0,0,0,4,4,21,14,4        '   down arrow
OK      data   8,20,20,8,5,6,5,5        '   single-character "OK"
LK      data   14,17,17,31,27,27,31,0   '   locked
UL      data   14,16,16,31,27,27,31,0   '   unlocked
HF      data   17,18,23,9,19,4,7,0      '   single-character "1/2"

' =====Program begins by downloading custom character patterns
' from EEPROM after a 1-second delay for LCD initialization.
pause 1000                      ' Wait a second
serout lcd,n9600,[I,cgRAM]      ' Point to character-generator RAM.
for cnt = 0 to 63               ' For each of 8 chars x 8 bit rows.
  read cnt,char                         ' Get the bit pattern from EEPROM.
  serout lcd,n9600,[char]       ' Write the data to LCD CG RAM.
next

serout lcd,n9600,[I,lcdCls]             ' Clear the LCD.
serout lcd,n9600,["CUSTOM GRAPHICS"]    ' Label top line.

' ============= DEMO PROGRAM ============
' Move to line 2, show custom-character symbols.
serout lcd,n9600,[I,line2]
for cnt = 0 to 7
  serout lcd,n9600,[cnt," "]    ' Print character, then a space.
next
end
```

```
' Listing 2.3: QGRPHX1.BAS
' (Defining custom graphics with LCD Backpack and QBASIC)
' This program demonstrates how to define and use custom
' symbols through the LCD Serial Backpack. The symbol
' patterns are loaded into the LCD from data tables at the
' end of this QBASIC program. Once the symbols are downloaded,
' they can be displayed by sending an ASCII value of 0 - 7.
' Symbols are stored in the LCD's cg RAM, so they stay in
' memory until LCD power is turned off.
' Install a jumper at BPS on the Backpack to configure it
' for 9600 baud. Connect serial input to PC COM1 (or change
' "OPEN" instruction to other COM port below).

CONST I = 254           ' Instruction prefix for LCD.
CONST lcdCls = 1        ' Clear-screen instruction for LCD.
CONST cgRAM = 64        ' Starting point of CG RAM.
CONST line2 = 192       ' Position: start of 2nd line.

CLS : PRINT "Running custom-graphics demo for LCD Serial Backpack..."
' Set up the serial port: COM1, baud, parity, data bits, stop bits,
' disable all handshaking (CD, CS, DS, OP).
OPEN "com1:9600,N,8,1,CD0,CS0,DS0,OP0" FOR OUTPUT AS #1
PRINT #1, CHR$(I); CHR$(cgRAM);         ' Point to CG RAM.
' Write the bit patterns stored in DATA statements below
' into the LCD's CG RAM.
FOR count = 1 TO 64             ' For each of 8 chars x 8 bit rows.
  READ char%                    ' Get the bit pattern from DATA.
  PRINT #1, CHR$(char%);        ' Write the data to LCD CG RAM.
NEXT
PRINT #1, CHR$(I); CHR$(lcdCls);        ' Clear the LCD.
SLEEP 1                                 ' Delay a second.
PRINT #1, "CUSTOM GRAPHICS";            ' Print label.
PRINT #1, CHR$(I); CHR$(line2);         ' Move to line 2.
```

```
FOR count = 0 TO 7
  PRINT #1, CHR$(count); " ";           ' Print custom symbols 0-7.
NEXT
END

' Custom-character definitions stored as bit patterns in the
' DATA statements. Each character is made up of 8 bytes. Each byte
' represents a row of 5 pixels (only the lowest 5 bits are
' actually used). Comments before each DATA entry describe the
' appearance of the corresponding graphic.
' right arrow:
DATA 0,4,2,31,2,4,0,0
' left arrow:
DATA 0,4,8,31,8,4,0,0
' up arrow:
DATA 4,14,21,4,4,0,0,0
' down arrow:
DATA 0,0,0,4,4,21,14,4
' single-character "OK"
DATA 8,20,20,8,5,6,5,5
' locked (padlock closed)
DATA 14,17,17,31,27,27,31,0
' unlocked (padlock open)
DATA 14,16,16,31,27,27,31,0
' single-character "1/2"
DATA 17,18,23,9,19,4,7,0
```

```
' Listing 2.4: ANIM1.BAS (Animation demo with LCD Backpack & BS1)
' This program demonstrates how to create animation by cycling
' through custom graphics characters at a single screen location.

SYMBOL  lcd      = 0    ' Serial output to LCD.
SYMBOL  I        = 254  ' Instruction prefix for LCD.
SYMBOL  lcdCls   = 1    ' Clear-screen instruction for LCD.
SYMBOL  cgRAM    = 64   ' Starting point of CG RAM.
SYMBOL  animPos  = 199  ' Position: middle of 2nd line.
SYMBOL  char     = b11  ' Pointer to animation 'frame.'
SYMBOL  count    = b10  ' Counter for EEPROM addressing.

' Custom-character definitions stored as bit patterns in the
' Stamp's EEPROM. Each character is made up of 8 bytes. Each byte
' represents a row of 5 pixels (only the lowest 5 bits are
' actually used). Comments at the end of each EEPROM entry show
' the appearance of the corresponding graphic.
EEPROM (4,4,4,4,4,0,0,0)        '   |
EEPROM (16,8,4,2,1,0,0,0)       '   \
EEPROM (0,0,31,0,0,0,0,0)       '   -
EEPROM (1,2,4,8,16,0,0,0)       '   /
```

```
' =====Program begins by downloading custom character patterns
' from EEPROM after a 1-second delay for LCD initialization.
pause 1000                      ' Wait a second
serout lcd,n2400,(I,cgRAM)      ' Point to character-generator RAM.
for count = 0 to 31             ' For each of 8 chars x 8 bit rows.
  read count,char               ' Get the bit pattern from EEPROM.
  serout lcd,n2400,(char)       ' Write the data to LCD CG RAM.
next

serout lcd,n2400,(I,lcdCls)     ' Clear the LCD.
serout lcd,n2400,("   ANIMATION")        ' Label top line.

' ============= DEMO PROGRAM LOOP ============
again:
  serout lcd,n2400,(I,animPos,char)      ' Move to line 2, show char.
  char = char + 1 // 4                   ' Next character, range 0-3.
  pause 50                               ' Short pause.
goto again                               ' Repeat endlessly.
```

```
' Listing 2.5: ANIM1.BS2 (Animation demo with LCD Backpack & BS2)
' This program demonstrates how to create animation by cycling
' through custom graphics characters at a single screen location.
' Install a jumper at BPS on Backpack for 9600 baud operation.

lcd      con    0        ' Serial output to LCD.
I        con    254      ' Instruction prefix for LCD.
lcdCls   con    1        ' Clear-screen instruction for LCD.
cgRAM    con    64       ' Starting point of CG RAM.
animPos  con    199      ' Position: middle of 2nd line.
n9600    con    $4054    ' 9600 baud.
char     var    byte     ' Pointer to animation 'frame.'
cnt      var    byte     ' Counter for EEPROM addressing.

' Custom-character definitions stored as bit patterns in the
' Stamp's EEPROM. Each character is made up of 8 bytes. Each byte
' represents a row of 5 pixels (only the lowest 5 bits are
' actually used). Comments at the end of each EEPROM entry show
' the appearance of the corresponding graphic.
one      data   4,4,4,4,4,0,0,0  '   |
two      data   16,8,4,2,1,0,0,0 '   \
three    data   0,0,31,0,0,0,0,0 '   -
four     data   1,2,4,8,16,0,0,0 '   /

' =====Program begins by downloading custom character patterns
' from EEPROM after a 1-second delay for LCD initialization.
pause 1000                      ' Wait a second
serout lcd,n9600,[I,cgRAM]      ' Point to character-generator RAM.
for cnt = 0 to 31               ' For each of 8 chars x 8 bit rows.
```

```
  read cnt,char                 ' Get the bit pattern from EEPROM.
  serout lcd,n9600,[char]       ' Write the data to LCD CG RAM.
next

serout lcd,n9600,[I,lcdCls]             ' Clear the LCD.
serout lcd,n9600,["   ANIMATION"]       ' Label top line.

' ============= DEMO PROGRAM LOOP ============
again:
  serout lcd,n9600,[I,animPos,char]     ' Move to line 2, show char.
  char = char + 1 // 4                  ' Next character, range 0-3.
  pause 75                              ' Short pause.
goto again                              ' Repeat endlessly.
```

```
' Listing 2.6: QANIM1.BAS (Animation demo with LCD Backpack & QBASIC)
' This program demonstrates how to create animation by cycling
' through custom graphics characters at a single screen location.
' Install a jumper at BPS on the Backpack to configure it
' for 9600 baud. Connect serial input to PC COM1 (or change
' "OPEN" instruction to other COM port below).
CONST I = 254          ' Instruction prefix for LCD.
CONST lcdCls = 1       ' Clear-screen instruction for LCD.
CONST cgRAM = 64       ' Starting point of CG RAM.
CONST animPos = 199    ' Position: middle of 2nd line.

CLS : PRINT "Running animation demo for LCD Serial Backpack..."
PRINT "Press ctrl-BREAK to end"
' Set up the serial port: COM1, baud, parity, data bits, stop bits,
' disable all handshaking (CD, CS, DS, OP).
OPEN "com1:9600,N,8,1,CD0,CS0,DS0,OP0" FOR OUTPUT AS #1
PRINT #1, CHR$(I); CHR$(cgRAM);         ' Point to CG RAM.
' Write the bit patterns stored in DATA statements below
' into the LCD's CG RAM.
FOR count = 1 TO 32             ' For each of 8 chars x 8 bit rows.
  READ char%                    ' Get the bit pattern from DATA.
  PRINT #1, CHR$(char%);        ' Write the data to LCD CG RAM.
NEXT
PRINT #1, CHR$(I); CHR$(lcdCls);        ' Clear the LCD.
SLEEP 1                                 ' Delay a second.
PRINT #1, "   ANIMATION";               ' Print label.

again:
' Move to middle of line 2 and show animation symbol.
  PRINT #1, CHR$(I); CHR$(animPos); CHR$(char%);
  char% = (char% + 1) MOD 4       ' Next character, range 0-3.
  PLAY "P16"                      ' Play silent note to pause.
GOTO again                        ' Repeat endlessly (until Break).
END
```

```
' Custom-character definitions stored as bit patterns in the
' DATA statements. Each character is made up of 8 bytes. Each byte
' represents a row of 5 pixels (only the lowest 5 bits are
' actually used). Comments before each DATA entry describe the
' appearance of the corresponding graphic.
' symbol: |
DATA 4,4,4,4,4,0,0,0
' symbol: \
DATA 16,8,4,2,1,0,0,0
' symbol: -
DATA 0,0,31,0,0,0,0,0
' symbol: /
DATA 1,2,4,8,16,0,0,0
```

```
' Listing 2.7: ANIM2.BAS (Animation demo with LCD Backpack & BS1)
' This program demonstrates how to create animation by cycling
' different bit patterns through a single cg RAM location.
' This lets you animate multiple screen locations simultaneously.

SYMBOL   lcd    = 0      ' Serial output to LCD.
SYMBOL   I      = 254    ' Instruction prefix for LCD.
SYMBOL   lcdCls = 1      ' Clear-screen instruction for LCD.
SYMBOL   cgRAM  = 64     ' Starting point of CG RAM.
SYMBOL   line2  = 192    ' Position: middle of 2nd line.
SYMBOL   frame  = b11    ' Pointer to animation 'frame.'
SYMBOL   count  = b10    ' Counter for EEPROM addressing.
SYMBOL   cgStart        = b9      ' Location of cgRAM data in EEPROM.
SYMBOL   cgEnd  = b8     ' End of cgRAM data in EEPROM.
SYMBOL   cgData = b7     ' Individual byte of cgRAM data.

' Custom-character definitions stored as bit patterns in the
' Stamp's EEPROM. Each character is made up of 8 bytes. Each byte
' represents a row of 5 pixels (only the lowest 5 bits are
' actually used). It's hard to simulate these patterns in text,
' but the progression of patterns is meant to look like an
' expanding square that starts as a dot and grows.
EEPROM (0,0,0,4,0,0,0,0)         ' Dot.
EEPROM (0,0,10,4,10,0,0,0)       ' larger..
EEPROM (0,0,14,10,14,0,0,0)      '   larger..
EEPROM (0,21,10,21,10,21,0,0)    '     larger..
EEPROM (0,31,17,17,17,31,0,0)    '       larger..
EEPROM (0,21,0,17,0,21,0,0)      '         larger.

' =====Program begins by clearing the screen after a 1-second
' delay for LCD initialization. It then downloads first graphic,
' labels the top line of the screen, and prints 15 character 0s
' across the second line. The character 0s will be animated.
pause 1000                       ' Wait a second
```

```
serout lcd,n2400,(I,lcdCls)      ' Clear the LCD.
frame = 0: gosub newPattern
serout lcd,n2400,("   ANIMATION")        ' Label top line.
serout lcd,n2400,(I,line2)               ' Move to 2nd line.
for frame = 0 to 15
  serout lcd,n2400,(0)           ' Print character 0 across 2nd line.
next

' ============= DEMO PROGRAM LOOP ============
again:
  frame = frame + 1 // 6         ' Next pattern, range 0-5.
  gosub newPattern               ' Change character-0 pattern.
  pause 100                      ' Short pause.
goto again                       ' Repeat endlessly.


' ============= RELOAD CUSTOM CHARACTER PATTERN  ============
' This subroutine accepts a frame value from 0 to 5 and writes the
' corresponding data from Stamp EEPROM into the cg RAM location for
' LCD character 0 (not "0" but ASCII 0; the pattern that prints
' when you send a byte containing %00000000 to the LCD). This causes
' any screen location containing character 0 to change appearance
' to match the new pattern! The technique lets you animate multiple
' screen locations simultaneously.
newPattern:
  cgStart = frame * 8            ' First bit pattern to load.
  cgEnd = cgStart + 7            ' Last bit pattern to load.
  serout lcd,n2400,(I,cgRAM)     ' Point to character 0 of cgRAM.
  for count = cgStart to cgEnd   ' For each of eight bit patterns..
    read count,cgData            ' Get the bit pattern from EEPROM..
    serout lcd,n2400,(cgData)    ' Write the data to LCD CG RAM.
  next
  serout lcd,n2400,(I,128)       ' Return to display RAM, top line.
return
```

```
' **Listing 2.8**: ANIM2.BS2 (Animation demo with LCD Backpack & BS2)
' This program demonstrates how to create animation by cycling
' different bit patterns through a single cg RAM location.
' This lets you animate multiple screen locations simultaneously.
' Install a jumper at BPS on Backpack for 9600 baud operation.

lcd      con    0        ' Serial output to LCD.
I        con    254      ' Instruction prefix for LCD.
lcdCls   con    1        ' Clear-screen instruction for LCD.
cgRAM    con    64       ' Starting point of CG RAM.
n9600    con    $4054    ' 9600 baud.
line2    con    192      ' Position: middle of 2nd line.
frame    var    byte     ' Pointer to animation 'frame.'
```

```
cnt      var    byte     ' Counter for EEPROM addressing.
cgStart  var    byte     ' Location of cgRAM data in EEPROM.
cgEnd    var    byte     ' End of cgRAM data in EEPROM.
cgData   var    byte     ' Individual byte of cgRAM data.

' Custom-character definitions stored as bit patterns in the
' Stamp's EEPROM. Each character is made up of 8 bytes. Each byte
' represents a row of 5 pixels (only the lowest 5 bits are
' actually used). It's hard to simulate these patterns in text,
' but the progression of patterns is meant to look like an
' expanding square that starts as a dot and grows.
frame1   data   0,0,0,4,0,0,0,0  ' Dot.
frame2   data   0,0,10,4,10,0,0,0         ' larger..
frame3   data   0,0,14,10,14,0,0,0        '   larger..
frame4   data   0,21,10,21,10,21,0,0      '     larger..
frame5   data   0,31,17,17,17,31,0,0      '       larger..
frame6   data   0,21,0,17,0,21,0,0        '         larger.

' =====Program begins by clearing the screen after a 1-second
' delay for LCD initialization. It then downloads first graphic,
' labels the top line of the screen, and prints 15 character 0s
' across the second line. The character 0s will be animated.
pause 1000                      ' Wait a second
serout lcd,n9600,[I,lcdCls]     ' Clear the LCD.
frame = 0: gosub newPattern
serout lcd,n9600,["   ANIMATION"]        ' Label top line.
serout lcd,n9600,[I,line2]               ' Move to 2nd line.
for frame = 0 to 15
  serout lcd,n9600,[0]          ' Print character 0 across 2nd line.
next

' ============= DEMO PROGRAM LOOP ============
again:
  frame = frame + 1 // 6        ' Next pattern, range 0-5.
  gosub newPattern              ' Change character-0 pattern.
  pause 150                     ' Short pause.
goto again                      ' Repeat endlessly.

' ============= RELOAD CUSTOM CHARACTER PATTERN  ============
' This subroutine accepts a frame value from 0 to 5 and writes the
' corresponding data from Stamp EEPROM into the cg RAM location for
' LCD character 0 (not "0" but ASCII 0; the pattern that prints
' when you send a byte containing %00000000 to the LCD). This causes
' any screen location containing character 0 to change appearance
' to match the new pattern! The technique lets you animate multiple
' screen locations simultaneously.
newPattern:
  cgStart = frame * 8           ' First bit pattern to load.
  cgEnd = cgStart + 7           ' Last bit pattern to load.
  serout lcd,n9600,[I,cgRAM]    ' Point to character 0 of cgRAM.
  for cnt = cgStart to cgEnd    ' For each of eight bit patterns..
    read cnt,cgData             ' Get the bit pattern from EEPROM..
    serout lcd,n9600,[cgData]   ' Write the data to LCD CG RAM.
  next
  serout lcd,n9600,[I,128]      ' Return to display RAM, top line.
return
```

```
' Listing 2.9: QANIM2.BAS (Animation demo with LCD Backpack & QBASIC)
' This program demonstrates how to create animation by cycling
' different bit patterns through a single cg RAM location.
' This lets you animate multiple screen locations simultaneously.
' Install a jumper at BPS on the Backpack to configure it
' for 9600 baud. Connect serial input to PC COM1 (or change
' "OPEN" instruction to other COM port below).

DECLARE SUB newPattern (frame%)
CONST lcd = 0           ' Serial output to LCD.
CONST I = 254           ' Instruction prefix for LCD.
CONST lcdCls = 1        ' Clear-screen instruction for LCD.
CONST cgRAM = 64        ' Starting point of CG RAM.
CONST line2 = 192       ' Position: start of 2nd line.
CONST home = 128        ' Position: start of 1st line.
DIM SHARED bitPats(47) AS INTEGER       ' Storage for bit patterns.

CLS : PRINT "Running animation demo for LCD Serial Backpack..."
PRINT "Press ctrl-BREAK to end"
FOR count% = 0 TO 47            ' For each of 6 chars x 8 bit rows.'
  READ bitPats(count%)          ' Get the bit pattern from DATA..
NEXT                            ' ..and store it in bitPats array.

' Set up the serial port: COM1, baud, parity, data bits, stop bits,
' disable all handshaking (CD, CS, DS, OP).
OPEN "com1:9600,N,8,1,CD0,CS0,DS0,OP0" FOR OUTPUT AS #1

PRINT #1, CHR$(I); CHR$(lcdCls);        ' Clear the LCD.
SLEEP 1                                 ' Delay a second.
PRINT #1, "   ANIMATION";               ' Print label.
count% = 0: newPattern (count%)         ' Set bit pattern.
PRINT #1, CHR$(I); CHR$(line2);         ' Move to 2nd line.
FOR count% = 1 TO 16                    ' Print 16 ASCII-0
  PRINT #1, CHR$(0);                    ' ..graphics across
NEXT                                    ' ..line 2.
count% = 0                              ' Clear count.
again:
  count% = (count% + 1) MOD 6   ' Cycle count, 0-5.
  newPattern (count%)           ' Change bit pattern to animate.
  PLAY "P16"                    ' Play silent note to pause.
GOTO again
```

```
' Custom-character definitions stored as bit patterns in the
' DATA statements. Each character is made up of 8 bytes. Each byte
' represents a row of 5 pixels (only the lowest 5 bits are
' actually used). It's hard to simulate these patterns in text,
' but the progression of patterns is meant to look like an
' expanding square that starts as a dot and grows.
DATA 0,0,0,4,0,0,0,0
DATA 0,0,10,4,10,0,0,0
DATA 0,0,14,10,14,0,0,0
DATA 0,21,10,21,10,21,0,0
DATA 0,31,17,17,17,31,0,0
DATA 0,21,0,17,0,21,0,0

SUB newPattern (frame%)
' Given a frame number from 0 to 5, select the appropriate 8 bytes
' of bit patterns stored in bitPats array and write those bytes to
' the LCD's cg RAM.
cgStart% = frame% * 8
cgEnd% = cgStart% + 7
PRINT #1, CHR$(I); CHR$(cgRAM);          ' Point to cg RAM.
FOR count% = cgStart% TO cgEnd%          ' Write the bit pattern.
  PRINT #1, CHR$(bitPats(count%));
NEXT
PRINT #1, CHR$(I); CHR$(home);           ' Cursor to line 1, char 0.
END SUB
```