

What's a Parallax Serial LCD?

It's a two row by 16 character liquid crystal display that's controlled by serial messages that the BASIC Stamp can send with just one I/O pin. There are two versions, standard and backlit:

Version	Parallax Part Number
Standard	27976
Backlit	27977

The items shown in Figure 2 all have liquid crystal displays. They are easy to read, and the smaller ones consume very little power. Think about how many products you own with liquid crystal displays. Think also as you go through these activities about the various BASIC Stamp projects, prototypes and inventions you've got in the works, and how a serial LCD might enhance or help them to completion.

Figure 2 - Products Examples with LCD Displays.

Clockwise from top-left: cell phone, portable GPS unit, calculator, digital multimeter, office clock, office phone, laptop, oscilloscope.





Serial Vs Parallel LCDs

The Parallel LCD is probably the most common of the LCDs. It typically requires a minimum of 6 I/O pins for the BASIC Stamp to control. Also, unless you are using a BASIC Stamp 2p or 2pe, the code for controlling them tends to be more complex.

The serial LCD is a parallel LCD with an extra microcontroller. The extra microcontroller on the LCD converts the serial messages from the BASIC Stamp to the parallel messages that control the LCD. The wiring and coding are both much simpler than the parallel LCD.

The activities in this chapter introduce how to use the Parallax Serial LCD with the BASIC Stamp 2. LCD basics will include displaying text, numbers, placing and animating characters, and creating and animating custom characters.

ACTIVITY #1: CONNECTING AND TESTING THE LCD

Along with the electrical connections and some simple PBASIC test programs for the Parallax serial LCD, this activity will introduce the `SEROUT` command. It will also demonstrate how `DEBUG` is just a special case of `SEROUT`. This will be especially useful for working with your serial LCD because you can take many of the `DEBUG` command arguments and use them with the `SEROUT` command to control your display. So, keep a copy of the Parallax Serial LCD documentation handy as you go through the activities in this chapter.

Serial LCD Parts

In addition to the Parallax serial LCD and three wires, it will be especially important to have either a printed or PDF copy of the Parallax Serial LCD Documentation. Although it's only a few pages, it has a long list of values you can send to your LCD to make it perform functions similar to those you've used with the Debug Terminal. Cursor control, carriage returns, clear screen and so on, all have their own special codes that are not necessarily identical to the codes for the Debug Terminal.

- (1) Parallax Serial LCD: [Standard - 27976](#) or [Backlit - 27977](#).
- (1) [Parallax Serial LCD Documentation \(.pdf\)](#)
- (3) Jumper wires

Building the Serial LCD Circuit

Connecting the serial LCD is amazingly simple. Although there are five pins, you only need to use three, one for power, one for ground, and one for signal. The LCD's RX pin is for the signal and should be connected to a BASIC Stamp I/O pin. In this activity, we will use P14.



CAUTIONS:

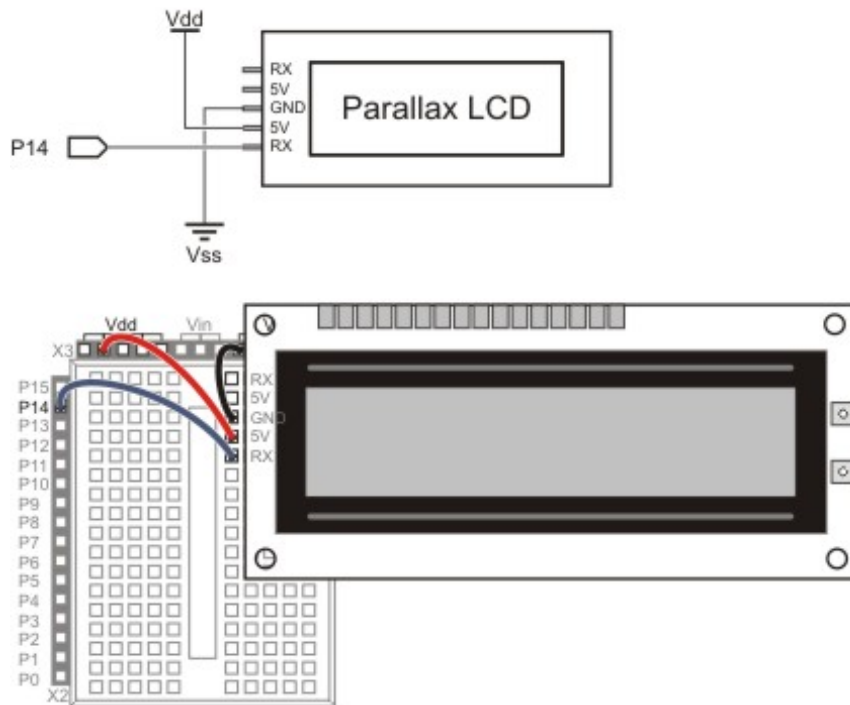
Wiring mistakes can damage this LCD.

This LCD is NOT pin compatible with Scott Edwards or Matrix Orbital models.

Be careful, if you have used other brands of serial LCDs before, the pinout of this LCD is different. Don't make the mistake of using the same wiring that you used for other models.

- √ Disconnect power from your Board of Education.
- √ Connect the Board of Education's P14 socket to the LCD's RX pin. (See figure 3.)
- √ Connect the Board of Education's Vdd socket to the LCD's 5V pin
- √ Connect the Board of Education's Vss socket to the GND pin.
- √ Do not turn the power back on yet.

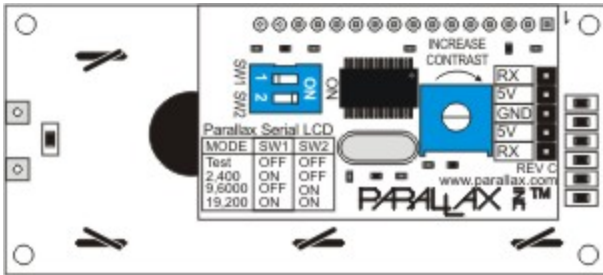
Figure 3 - Schematic and Wiring Diagram



Testing the Serial LCD

The Parallax Serial LCD has a self test mode you can use to make sure it's in working order and that the contrast is properly set. The switches for the self test mode and the contrast adjustment are both on the back of the LCD module, shown in Figure 4.

Figure 4 - LCD Module - Back View



- √ The power to your board should still be off.
- √ Find SW1 and SW2 on the underside of the LCD module shown in Figure 5.
- √ Set SW1 off.
- √ Set SW2 off.
- √ Turn the power back on now.

Figure 5 - Baud Rate Switches to self test mode



When you turn the power back on, the LCD should display the text "Parallax, Inc." on the top line and "www.parallax.com" on the bottom line. (See Figure 1.) If you leave the LCD in this mode for a while, custom a character reminiscent of 1980's video games will appear and eat all the text.

If the display seems dim or looks blank, there is a contrast adjustment potentiometer shown in Figure 6 that you can turn with a screwdriver to make the display clear. If the display is already clear, you probably don't need to adjust it. If the characters are too dim, or they appear in gray boxes, adjusting the potentiometer should help.

- √ Adjust the contrast potentiometer if needed.

Figure 6 - Contrast Adjustment Potentiometer**Adjusting the LCD to Receive Messages from the BASIC Stamp**

Serial communication involves a baud rate. That's the number of bits per second (bps) the sender transmits, and the receiver has to be ready to receive data at the same baud rate. In this chapter's activities, the BASIC Stamp will be programmed to send messages to the LCD at 9600 bps. You can adjust the same switches you used for the LCD self-test to set the baud rate.

- √ Turn the Board of Education's power off.
- √ Leave SW1 in the off position.
- √ Set SW2 to ON as shown in Figure 7.
- √ Turn the power back on now.

The screen will remain blank until you program the BASIC Stamp 2 to control the display.

Figure 7 - Baud Rate Switches to 9600 bps


Figure 8 shows the mode table printed on the back of the Parallax serial LCD. If you need to send messages at other baud rates, like 2400 or 19,200 bps, use this table and adjust SW1 and SW2 accordingly.

Figure 8 - Baud Rate Switch Settings

Parallax Serial LCD		
MODE	SW1	SW2
Test	OFF	OFF
2,400	ON	OFF
9,600	OFF	ON
19,200	ON	ON

ACTIVITY #2: DISPLAYING SIMPLE MESSAGES

As mentioned earlier, the commands that send text, numbers, formatters and control characters to a serial LCD are related to the `DEBUG` command. In fact, the `DEBUG` command is a special version of a more general command called `SEROUT`. The `SEROUT` command has many uses, some of which include sending messages to serial LCDs, other BASIC Stamps, and computers.



Sending messages to the Debug Terminal with the `SEROUT` command - try this:

```
' {$STAMP BS2}
' {$PBASIC 2.5}

SEROUT 16, 16468, ["See this?", CR, "SEROUT works too!"]
```

In this activity, you will program the BASIC Stamp to make the LCD display text messages, and numeric values. As a first step into animation, you will also modify the programs to flash the text and numbers on and off. The `SEROUT` command will be your tool for accomplishing these tasks. You will use the `SEROUT` command to send text, numbers, control characters and formatters to the Parallax serial LCD. As you will soon see, the text, numbers, and formatters are identical to ones you use with the `DEBUG` command. The control characters will be a little different, but with some practice, they'll be just as easy to use as the `DEBUG` command's `CR`, `CLS`, `HOME`, and `CRSRXY` control characters.

Simple Text Messages and Control Characters


Unlike the Debug Terminal, the serial LCD needs to be turned on with a command from the BASIC Stamp. The LCD has to receive the value 22 from the BASIC Stamp to activate its display. Here's the PBASIC command for sending the serial LCD the value 22:

```
SEROUT 14, 84, [22]
```

The value 22 is an example of an LCD control character. Here's a list of some of some more basic control characters:

- 12 clear the display (requires `PAUSE 5` to complete)
- 13 carriage return - cursor to next line
- 21 turn the LCD off

- 22 turn the LCD on



Commands to turn the backlighting on and off (for the backlit LCD only):

Some LCD's have backlighting so that you can read them when it's dark. The backlighting illuminates the display to make the characters visible. It's an especially common feature in digital wristwatches. If you have the backlit version of the Parallax serial LCD (part number 27977), you can control the backlighting with these values:

- 17 turns the backlighting on
- 18 turns the backlighting off

It's usually a good idea to turn the display on and clear it with the control character 12 before displaying characters. Here's how to do that:

```
SEROUT 14, 84, [22, 12]
PAUSE 5
```

Displaying messages with the LCD is very similar to displaying messages on the Debug Terminal. For example, instead of

```
DEBUG "See this?", CR, "DEBUG Works!"
```

you can use the command

```
SEROUT 14, 84, ["See this?", 13, "The LCD works!"]
```

The resulting display should resemble Figure 9. Let's try these commands in an example program.

Figure 9 - Text Display



Example Program - LcdTestMessage.bs2

- ✓ Enter, save, and run LcdTestMessage.bs2
- ✓ Verify that it displayed the message "See this?" on the top line and "The LCD works!" on the bottom line.

If the LCD didn't display properly:

- ✓ Double-check your wiring, program, and the SW settings on the back of the LCD.
- ✓ Also try disconnecting and reconnecting power to your Board of Education
- ✓ Go through the check marked instructions leading up to this program, and verify that each one was completed correctly

```
' Example Program - LcdTestMessage.bs2
' Display a test message on the Parallax serial LCD.

' {$STAMP BS2}
' {$PBASIC 2.5}
' Target device = BASIC Stamp 2
' Language      = PBASIC 2.5

SEROUT 14, 84, [22, 12]
PAUSE 5
' Initialize LCD

SEROUT 14, 84, ["See this?", 13,
               "The LCD works!"]
' Text message, carriage return
' more text on line 2.

END
' Program end
```

More about the SEROUT command:

The minimal version of the **SEROUT** command's syntax looks like this:

SEROUT *Pin*, *BaudMode*, [*DataItem*, {*DataItem*, ...}]

In `LcdTestMessage.bs2`, the *Pin* argument has to be 14 since the LCD's RX (receive data) pin is connected to BASIC Stamp I/O pin P14.



The *BaudMode* argument is a value that instructs the BASIC Stamp on how fast to send the serial data, and it determines some of the serial signal characteristics as well. The BASIC Stamp Editor's Help program has tables that give the *Baudmode* values for common baud rates and signals. It turns out that 84 is the *BaudMode* argument for 9600 bits per second (bps), 8 data bits, no parity, true signal.

The *DataItem* arguments can be the text between quotes like **"Text"**. They can also be control characters like **CR**, **CLS**, or numbers with or without the formatters like **DEC**, **BIN**, and **?**.

You can learn lots more about the **SEROUT** command from either the BASIC Stamp Manual or the BASIC Stamp Editor's Help program (click Help and select Index).

Your Turn - Control Characters to Make the Display Flash On/Off

Remember that 22 turns the display on, and 21 turns it off? You can use these control characters to make the LCD text flash on and off. Here's an example; just replace the **END** command with this loop:

```
DO                                     ' Start DO...LOOP code block
  PAUSE 600                             ' 6/10 second delay
  SEROUT 14, 84, [22]                   ' Turn display on
  PAUSE 400                             ' 4/10 second delay
  SEROUT 14, 84, [21]                   ' Turn display off
LOOP                                    ' Repeat DO...LOOP code block
```

- √ Replace the **END** command in `LcdTestMessage.bs2` with this code block.
- √ Run the modified program and note the effect.

Display Numbers with Formatters

Most of the formatters that worked for displaying numbers with the Debug Terminal can also be used with the Parallax Serial LCD. The **DEC** formatter is probably the most useful, but you can also use **DIG**, **REP**, **ASC**, **BIN**, **SDEC**, and most of the others. For

example, if you want to display the decimal value of a variable named `counter`, you can use commands like this:

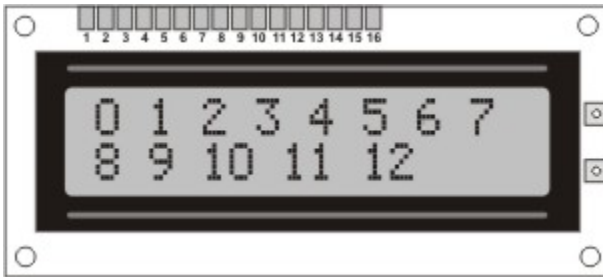
```
SEROUT 14, 84, [DEC counter]
```

Example Program - LcdTestNumbers.bs2

Aside from demonstrating that you can display variable values on the serial LCD, this program also shows what happens if the text displayed tries to go beyond 16 characters in line 1. It wraps to line 2. Also, after printing sixteen more characters and filling line 2, the text will wrap again, to line 1.

- √ Enter, save, and run LcdTestNumbers.bs2
- √ Verify that the display resembles Figure 10.

Figure 10 - Display Numbers



```
' Example Program - LcdTestNumbers.bs2
' Display number values with the Parallax serial LCD.

' {$STAMP BS2}
' {$PBASIC 2.5}

counter          VAR      Byte

SEROUT 14, 84, [22, 12]
PAUSE 5

FOR counter = 0 TO 12

    SEROUT 14, 84, [DEC counter, " "]
    PAUSE 500

' Target device = BASIC Stamp 2
' Language      = PBASIC 2.5
' FOR...NEXT loop index
' Initialize LCD
' 5 ms delay for clearing display
' Count to 12; increment at 1/2 s
```

NEXT

END

' Program end

Your Turn - Other Formatters

- √ Try replacing `DEC` with `DEC2` and observe what happens.
- √ Repeat with the `?` formatter.

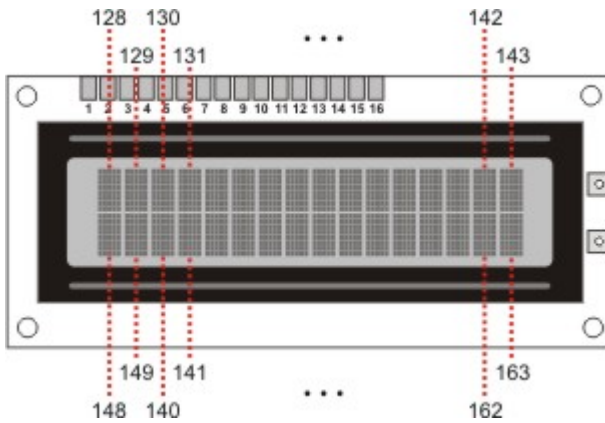
Control Characters for Cursor Positioning

The LCD's control characters are different from the `DEBUG` command's control characters. For example, `HOME`, and `CRSRXY` just don't have the same effect they do with the Debug Terminal. Take a look at the LCD documentation's Command Set section. It lists all the valid control characters for the LCD; here are a few more examples from the list that control cursor position:

- 8 Cursor left
- 9 Cursor right
- 10 Cursor down (bottom line will wrap to top line)
- 128 to 143 Position cursor on line 1, character 0 to 15
- 148 to 163 Position cursor on line 2, character 0 to 15

The values from 128 to 143 and 148 to 163 are particularly useful. Figure 11 shows where these values will place the cursor. You can use values from 128 to 143 to place the cursor at characters 0 to 15 on the top line of the LCD. Likewise, you can use values from 148 to 163 to place the cursor on characters 0 to 15 of the bottom line.

Figure 11 - Text Display



After placing the cursor, the next character you send the LCD will display at that location. For example, here is a `SEROUT` command with an optional pacing value set to 200 ms. This command will display the characters "L", "I", "N", "E", "-", and "1", evenly spaced across the top line, one character every 200 ms.

```
SEROUT 14, 84, 200, [128, "L",
                    131, "I",
                    134, "N",
                    137, "E",
                    140, "-",
                    143, "1"]
```

The LCD will still automatically shift the cursor to the right after each character, making it easy to place the cursor and then display a message. For example, you can also place the cursor on character 7 of the top line and then display "ALL", then move the cursor to character 6 of the bottom line and display "DONE!" like this:

```
SEROUT 14, 84, [135, "ALL", 154, "DONE!"]
```

Here's a code block that will make the text "Line-2" slide across the display's bottom line, from left to right.

```
FOR index = 9 TO 0
  SEROUT 14, 84, [148 + index, "Line-2 "]
```

PAUSE 250
NEXT



Erasing Characters

You can always erase a character by placing the cursor where you want it. Then, send the space " " character to overwrite the character that's there. This is why the text "Line-2 " has a space after it, to erase the characters to the right as the text moves left.

Example Program - Cursor Positions.bs2

This program introduces a few basic cursor placement tricks.

- √ Look over CursorPosition.bs2 and try to predict what the program will make the LCD display along with the sequence and timing.
- √ Enter, save, and run MoreLcdControlCharacters.bs2.
- √ Compare the LCD display's behavior to your predictions.

```
' Example Program - MoreLcdControlCharacters.bs2
' Display number values with the Parallax serial LCD.

' {$STAMP BS2}                ' Target device = BASIC Stamp 2
' {$PBASIC 2.5}              ' Language      = PBASIC 2.5

index          VAR    Nib          ' FOR...NEXT loop index
character      VAR    Byte         ' Character storage
offset         VAR    Byte         ' Offset value

SEROUT 14, 84, [22, 12]         ' Initialize LCD
PAUSE 500                       ' 1/2 second delay

' Display evenly spaced characters on line-1 every 200 ms.
SEROUT 14, 84, 200, [128, "L",
                    131, "I",
                    134, "N",
                    137, "E",
                    140, "-",
                    143, "1"]

PAUSE 1000

' Shift "line-2" across line-2 right to left, then left to right.
FOR index = 9 TO 0
  SEROUT 14, 84, [148 + index, "Line-2 "]
  PAUSE 250
NEXT
```

```

FOR index = 0 TO 9
  SEROUT 14, 84, [148 + index, " Line-2"]
  PAUSE 250
NEXT

PAUSE 1000                                ' 1 second delay

' Clear LCD, then display then Display "ALL DONE" in center and flash 5 times
SEROUT 14, 84, [12]: PAUSE 5              ' Clear LCD
SEROUT 14, 84, [135, "ALL", 13, 154, "DONE!"] ' "ALL" and "DONE" centered

FOR index = 1 TO 4                          ' Flash display 5 times
  SEROUT 14, 84, 500, [21, 22]
NEXT

END                                          ' Program end

```

Your Turn - More Positioning

More elaborate displays can benefit from loops and lookup tables. Here is an example of a "T E S T" display in a loop and with the help of lookup table. Note that you can control the position of each character's placement by adjusting the values in the second **LOOKUP** command's list of values.

```

PAUSE 1000
SEROUT 14, 84, [12]: PAUSE 5              ' Clear display
SEROUT 14, 84, ["This is a", 13]         ' Text & CR

FOR index = 0 TO 3                          ' Character display sequence
  PAUSE 600
  LOOKUP index, ["T", "E", "S", "T"], character
  LOOKUP index, [ 1, 5, 9, 13], offset
  SEROUT 14, 84, [(148 + offset), character]
NEXT

```

√ Try it!

ACTIVITY #3: TIMER APPLICATION

This activity applies the techniques introduced in Activity #2 to an hour, minute, second timer.

Displaying Time Elapsed

Here is a code block that starts the LCD, clears the screen, and places some display characters on the LCD that will not change. The rest of the program can then display hour, minute, and second number values next to the "h", "m", and "s" characters.

```
SEROUT 14, 84, [22, 12]           ' Start LCD & clear display
PAUSE 5                           ' Pause 5 ms for clear display

SEROUT 14, 84, ["Time Elapsed...", 13] ' Text + carriage return
SEROUT 14, 84, [" h m s"]         ' Text on second line
```

For this application, control characters for cursor placement can be particularly useful. For example, the cursor can be placed on line 2, character 0 before sending the two-digit decimal value of hours. The cursor can then be moved to line 2, character 5 to display the minutes, and then to character 10 to display the seconds. Here is a single **SEROUT** command that displays all three values, each at the correct location:

```
SEROUT 14, 84, [ 148, DEC2 hours,
                 153, DEC2 minutes,
                 158, DEC2 seconds ]
```

The next example program applies this concept with just the BASIC Stamp's timing abilities. The accuracy isn't digital wristwatch quality, but it's good enough for an example.



For more accurate timekeeping, try the [DS1302 timekeeping chip](#).

Here is a web address of an application that displays DS1302 time with the Parallax LCD: [DS1302 Demo Code](http://forums.parallax.com/forums/default.aspx?f=21&m=68522) - <http://forums.parallax.com/forums/default.aspx?f=21&m=68522>.

Note: The DS1302 code example sends messages to the LCD at 19,200 bps, so you'll have to update the switch settings on the back of the LCD.

Example Program - LcdTimer.bs2

This example program displays hours, minutes and seconds elapsed with the Parallax Serial LCD. By pressing the RESET button on the Board of Education, you can restart the timer.

- √ Enter, save, and run LcdTimer.bs2.
- √ Verify that the display works as advertised.

```

' Example Program - LcdTimer.bs2
' Display elapsed time with BS2 and Parallax Serial LCD.

' {$STAMP BS2}           ' Stamp directive
' {$PBASIC 2.5}        ' PBASIC Directive

hours      VAR      Byte   ' Stores hours
minutes    VAR      Byte   ' Stores minutes
seconds    VAR      Word   ' Stores 1/10 seconds

SEROUT 14, 84, [22, 12]   ' Start LCD & clear display
PAUSE 5                  ' Pause 5 ms for clear display

SEROUT 14, 84, ["Time Elapsed...", 13] ' Text + carriage return
SEROUT 14, 84, [" h m s"] ' Text on second line

DO                       ' Main routine

' Calculate hours, minutes, seconds
IF seconds = 60 THEN seconds = 0: minutes = minutes + 1
IF minutes = 60 THEN minutes = 0: hours = hours + 1
IF hours = 24 THEN hours = 0

' Display digits on LCD on line 2. The values 148, 153, 158
' place the cursor at character 0, 5, and 10 for the time values.
SEROUT 14, 84, [148, DEC2 hours,
                153, DEC2 minutes,
                158, DEC2 seconds ]

PAUSE 991                ' Pause + overhead ~ 1 second
seconds = seconds + 1    ' Increment 1/10 second counter

LOOP                     ' Repeat main routine

```

Your Turn - Defining Control Characters with Constants

Up to this point, the LCD control characters have been decimal values. It's better to declare a constant for each control character at the beginning of the program. Then, use the constant names instead of the numbers. You can also do the same with the *BaudMode* value, and add a *PIN* directive for I/O pin P14 as well. Here is an example:

```

LcdPin      PIN      14           ' LCD I/O pin
T8N9600     CON      84           ' True, 8-bits, no parity, 9600

LcdCls      CON      12           ' Form feed -> clear screen
LcdCr       CON      13           ' Carriage return
LcdOff      CON      21           ' Turns display off

```

```
LcdOn      CON    22           ' Turns display on
Line1     CON    128          ' Line 1, character 0
Line2     CON    148          ' Line 2, character 0
```

These declarations will make your code easier to understand and less cryptic. For example, the first **SEROUT** command can be rewritten like this:

```
SEROUT LcdPin, T8N9600, [LcdOn, LcdCls]
```

The **SEROUT** command that displays the numbers on line 2 can be rewritten like this:

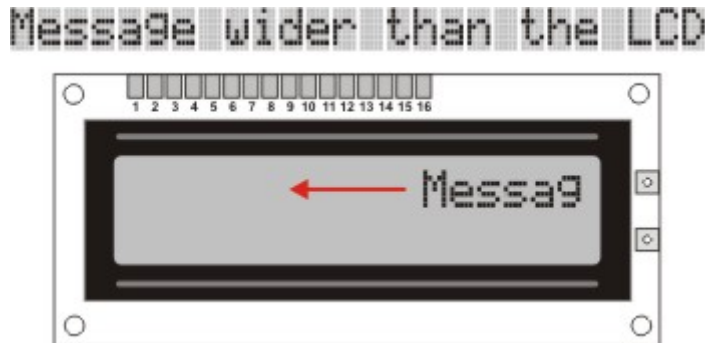
```
SEROUT LcdPin, T8N9600, [(Line2 + 0), DEC2 hours,
                          (Line2 + 5), DEC2 minutes,
                          (Line2 + 10), DEC2 seconds ]
```

- √ Add descriptive constants to your program.
- √ Replace all mystery numbers with their corresponding constant names.
- √ Run your program and trouble-shoot as needed.

ACTIVITY #4: SCROLLING TEXT ACROSS THE DISPLAY

If the message you want to display is too wide for the 16 character display, scrolling the text across the display can make it fit. (See Figure 12.) With scrolling, the message begins at the far right of the display. Then, it the text shifts across the display one letter at a time.

Figure 12 - Scrolling Text



The scrolling code introduced in this activity is quite different from the example program in Activity #2 that made "Line-2" move across the display. The main reason it's different is because the Parallax LCD line-wraps. If you send it more characters than it can fit on the top line, the text wraps down to the bottom line. After 16 more characters, it wraps back up to the top line.

To get text to scroll across just one line, the program has to start with the first character in a message and display it at the rightmost character. After a short delay, the program has to move the cursor to the second character from the right, and print both the first and second characters. It has to continue this process until it places the cursor gets to the left of the display. Then, the cursor has to be repeatedly repositioned to that same location as sixteen character portions of the message shift from right to left, one character at a time.

The programming technique for this process is called sliding-window. Aside from being useful for the Parallax LCD, sliding window is what you see when you scroll text up and down in programs like the BASIC Stamp Editor and your web browser. It's also used in programs for transmitting and collecting TCP/IP packets. So every time you open your web browser, there's more than one instance of sliding-window code at work in the background.

Programming a Sliding Window

You can also use scrolling text to fit a message inside a few characters on the LCD. Let's say your window is only four characters wide on the top row because there are other things that need to be displayed at all times on the LCD. The task at hand is to slide the text through the window without overprinting any other characters. Figure 13 shows the setup and step zero. In the setup step, nothing is displayed in the window. Then, step 0 places the cursor in character 137, and display character 0, the "M".

Figure 13 - Shifting Text Through Window, Setup and Step 0

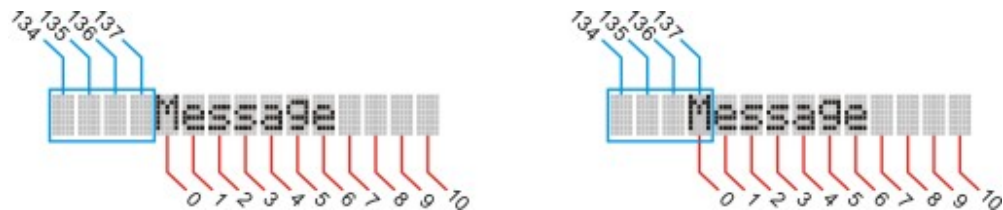


Figure 14 shows steps 1 and 2. After waiting a moment for the "M" to be visible, the cursor has to be positioned at character 136, and then characters 0 and 1, "Me", can be displayed. Next, move the cursor to 135, and display characters 0 to 2, "Mes".

Figure 14 - Shifting Text Through Window, Steps 1 and 2

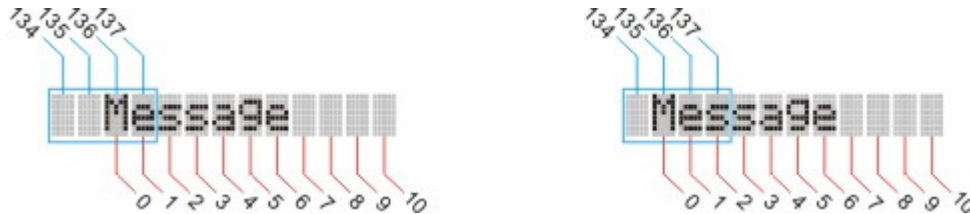


Figure 15 shows steps 3 and 4. Moving the cursor to 134 and displaying characters 0 to 3, "Mess" is still the same sequence, but when the "M" leaves the window, the sequence has to change. The cursor does not advance left; it stays at 134. Also, instead of displaying characters 0 to 3, characters 1 to 4, "essa", have to be displayed.

Figure 15 - Shifting Text Through Window, Steps 3 and 4




The starting cursor position has to remain at 134 while the head and tail characters continue to advance: 2 to 5 - "ssag", 3 to 6 - "sage". The window keeps sliding, and Figure 16 shows the second to last step of characters 6 to 9 - "e" followed by three spaces, and finally the last step, 7 to 10 - four space characters.

Figure 16 - Shifting Text Through Window, Steps 9 and 10



The next example program demonstrates one way of getting this job done. Try it first, and then we'll examine how it works.

Diagrams as Programming Tools

 It's a good idea to draw diagrams on scratch paper to figure out how to program a procedure or algorithm. Case in point, Figures 13 to 17 were drawn on scratch paper to get ScrollMessage.bs2 to work. They were converted to illustrations later, but their first use was as a coding aid.

Example Program - ScrollMessage.bs2

- ✓ Enter and run ScrollMessage.bs2
- ✓ Verify that the message in the **DATA** directive scrolls across the screen.

```
' Example Program - ScrollMessage.bs2
' Scroll a text message across a four character wide window in the LCD.

' {$STAMP BS2}
' {$PBASIC 2.5}
' BASIC Stamp Directive
' PBASIC Directive

MessageStart DATA @ 2, "Message "
MessageEnd DATA

cursorStart VAR Byte
head VAR Byte
tail VAR Byte
pointer VAR Byte
character VAR Byte
' First character location
' Start of displayed text
' End of displayed text
' EEPROM address pointer
' Stores a character

SEROUT 14, 84, [22, 12]
PAUSE 5
' Turn on display & clear
' Delay 5 ms

cursorStart = 137
head = 0
' Rightmost character in window
' Initialize head and tail
```

```

tail = 0                                     ' of message

DO WHILE tail < (MessageEnd - MessageStart) ' Scrolling loop

  SEROUT 14, 84, [cursorStart]               ' Place window's right
  SEROUT 14, 84, [21]                       ' Turn off the display
  PAUSE 140                                 ' Let characters fade away

  FOR pointer = head TO tail                 ' Print new characters
    READ pointer + MessageStart, character
    SEROUT 14, 84, [character]
  NEXT

  SEROUT 14, 84, [22]                       ' Turn display back on
  PAUSE 180                                 ' Give the characters some time

  cursorStart = cursorStart - 1 MIN 134     ' Decrement until at window-left
  tail = tail + 1                           ' Increment tail pointer

  ' Increment head pointer if tail pointer > window width.
  IF (tail > 3) THEN head = head + 1 ELSE head = 0

LOOP                                         ' Repeat scrolling loop

END                                          ' BASIC Stamp -> low power mode

```

How ScrollMessage.bs2 Works

The program starts with a **DATA** directive that stores the message text. The **MessageStart**, symbol stores the EEPROM address where the message begins. This value has been set to 2 with the optional **@Address** operator. The second **DATA** directive is empty; it's just there to make the **MessageEnd** symbol store the address that follows the last character in the message. The message has eleven characters, "**Message**" accounts for seven, and four more characters before the end quote. These characters are stored in EEPROM address 2 to 12. So, the **MessageEnd** symbol points at EEPROM address 13.

```

MessageStart DATA @ 2, "Message           " ' <--- 4 blank spaces after
MessageEnd   DATA

```



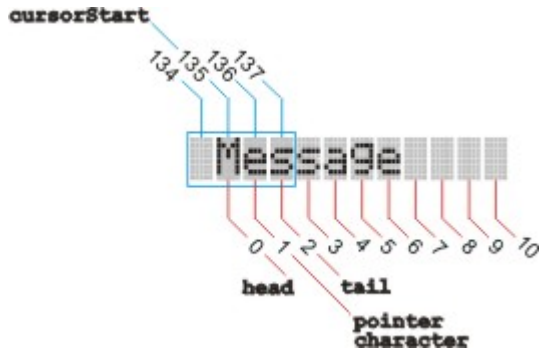
Viewing the Values of MessageStart and MessageEnd

Try adding this to the program to view the values of the MessageStart and MessageEnd constants:

```
DEBUG ? MessageStart, ? MessageEnd, ? (MessageEnd - MessageStart)
```

The next example program uses the variables shown in Figure 17 for the sliding window. `cursorStart` stores the location that the cursor is placed each time before it starts printing the characters in the message. In the figure, `cursorStart` stores the value 135. The next time the text shifts to the left, it will store 134. Two variables, `head` and `tail`, store the beginning and ending addresses of the text that will fit in the message window. In the figure, `head` stores 0, and `tail` stores 2. The `pointer` variable will be used by the `READ` command to get the right character, and the `character` variable will store the character the `READ` command retrieves from EEPROM.

Figure 17 - Shifting Text Through Window, Steps 9 and 10



In Figure 17, `pointer` is pointing at character 1 in the sequence, which is "E". A `FOR...NEXT` loop uses the `pointer` variable to read each of the characters in EEPROM, from `head` to `tail` then display each with `SEROUT`. Each time the text shifts to the right, the new text has to overwrite the old text with the same `head` to `tail` loop.

At the start of the program, the LCD is turned on and cleared.

```
SEROUT 14, 84, [22, 12]           ' Turn on display & clear
PAUSE 5                           ' Delay 5 ms
```

Before the scrolling loop starts, the initial values of the `cursorStart`, `head`, and `tail` variables have to be set. These initial values will have to be changed if you plan to start from the middle of the window or shift more than one character at a time.

```
cursorStart = 137                 ' Rightmost character in window
head = 0                          ' Initialize head and tail
```

```
tail = 0                                ' of message
```

A **DO WHILE...LOOP** waits for **tail** to get through the last character in the message, the 10th character in this case. When **tail** gets to 11 it is no longer less than (**MessageEnd** - **MessageStart**), so the loop terminates.

```
DO WHILE tail < (MessageEnd - MessageStart) ' Scrolling loop
  .
  .
  .
LOOP                                       ' Repeat scrolling loop
END                                         ' BASIC Stamp -> low power mode
```

The first thing that happens inside the scrolling **DO WHILE...LOOP**, is the cursor is placed where the characters will start to appear. In this program, the value of **cursorStart** was initialized to 137 just before the loop started. Before the **DO WHILE...LOOP** repeats, **cursorStart** will be decremented, so the next time through the loop, **cursorStart** will be placed one character to the left, at 136. Each time through the loop, this value will decrease by one, until it gets to 134, at which point, it will remain constant until the end of the message.

```
SEROUT 14, 84, [cursorStart]           ' Place cursor at far right
```

The display has to be turned off and given enough time for the characters to fade. Without this step, the ghost characters in the old locations kind of ruin the scrolling effect.

```
SEROUT 14, 84, [21]                   ' Turn off the display
PAUSE 140                               ' Let characters fade away
```

While the LCD is off, the new line is printed with this **FOR...NEXT** loop. As the **DO WHILE...LOOP** repeats, **cursorStart** moves to the left, and **tail** gets larger to accommodate printing more text as the message shifts to the left.

```
FOR pointer = head TO tail              ' Print characters to display
  READ pointer + MessageStart, character
  SEROUT 14, 84, [character]
NEXT
```

After the new text is displayed, the display can be turned back on so that the text becomes visible.

```
SEROUT 14, 84, [22]           ' Turn display back on
PAUSE 180                     ' Give the characters some time
```

Each time through the **DO WHILE...LOOP**, **cursorStart** has to be decremented, but only until it gets to the left of the sliding window. Then, **cursorStart** has to stay put, which is taken care of by **MIN 134**.

```
cursorStart = cursorStart - 1 MIN 134    ' Decrement until at window-left
```

The tail increases every time through the **DO WHILE...LOOP**. After **cursorStart** gets to the left of the window and characters 0 to 3 are displayed, this command moves both the **head** and the **tail** so that the next time through the **DO WHILE...LOOP**, the **head** and **tail** cause characters 1 to 4 to be displayed. The pattern continues with 2 to 5, 3 to 6, and so on.

```
tail = tail + 1                ' Increment tail pointer

' Increment head pointer if tail pointer > window width.
IF (tail > 3) THEN head = head + 1 ELSE head = 0
```

Your Turn - Modify the Scrolling Behavior

With some changes to the code, you can get a message to scroll across the LCD's top line. Here's how:

- √ Save ScrollMessage.bs2 as ScrollMessageYourTurn1.bs2.
- √ Change

```
MessageStart DATA @ 2, "Message      "           ' Message + four spaces
MessageEnd   DATA
```

to

```
' IMPORTANT - The entry on the second line of this DATA directive
is sixteen spaces between quotes.
MessageStart DATA @ 2, "Message wider than the LCD",
                        "
MessageEnd   DATA
```

- √ Change

```
cursorStart = 137
```

to

```
cursorStart = 143
```

√ Change

```
cursorStart = cursorStart - 1 MIN 134
```

to

```
cursorStart = cursorStart - 1 MIN 128.
```

√ Change

```
IF (tail > 3) THEN head = head + 1 ELSE head = 0
```

to

```
IF (tail > 15) THEN head = head + 1 ELSE head = 0
```

√ Run the modified program and verify that the changes worked.

You can further modify this code to scroll two characters at a time.

√ Save ScrollMessageYourTurn1.bs2 as ScrollMessageYourTurn2.bs2.

√ Add a seventeenth blank space to the spaces between quotes that follow the message text.

```
MessageStart DATA @ 2, "Message wider than the LCD",
                        "                " ' <- 17 blank spaces
MessageEnd   DATA
```

√ Change the **cursorStart** and **tail** variables so that the message starts on the second character from far right and then prints two characters.

```
cursorStart = 142
head = 0
tail = 1
```

√ The code in the **DO...LOOP** has to be changed so that it increments the **head** and **tail** pointers by 2 instead of one. There are three ones below that need to be changed to twos in this code block:

```
cursorStart = cursorStart - 2 MIN 128
```