

**Introduction.** This application note presents a program in PBASIC that enables the BASIC Stamp to operate as a simple user-interface terminal.

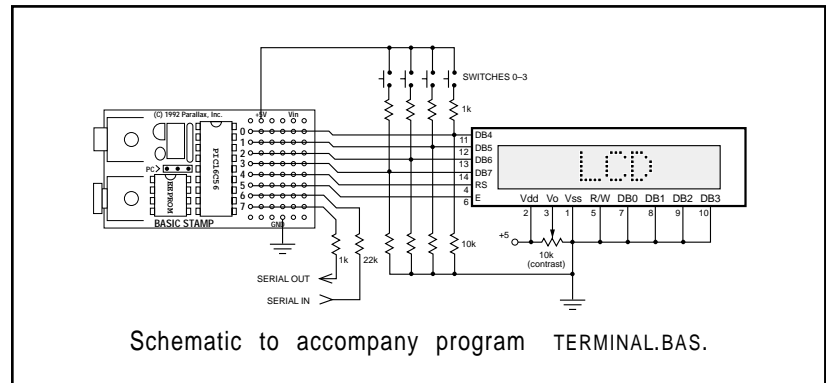
**Background.** Many systems use a central host computer to control remote functions. At various locations, users communicate with the main system via small terminals that display system status and accept inputs. The BASIC Stamp's ease of programming and built-in support for serial communications make it a good candidate for such user-interface applications.

The liquid-crystal display (LCD) used in this project is based on the popular Hitachi 44780 controller IC. These chips are at the heart of LCD's ranging in size from two lines of four characters (2x4) to 2x40.

**How it works.** When power is first applied, the BASIC program initializes the LCD. It sets the display to print from left to right, and enables an underline cursor. To eliminate any stray characters, the program clears the screen.

After initialization, the program enters a loop waiting for the arrival of a character via the 2400-baud RS-232 interface. When a character arrives, it is checked against a short list of special characters (backspace, control-C, and return). If it is not one of these, the program prints it on the display, and re-enters the waiting-for-data loop.

If a backspace is received, the program moves the LCD cursor back one



space, prints a blank (space) character to blot out the character that was there, and then moves back again. The second move-back step is necessary because the LCD automatically advances the cursor.

If a control-C is received, the program issues a clear instruction to the LCD, which responds by filling the screen with blanks, and returning the cursor to the leftmost position.

If a return character is received, the program interprets the message as a query requiring a response from the user. It enters a loop waiting for the user to press one of the four pushbuttons. When he does, the program sends the character ("0" through "3") representing the button number back to the host system. It then re-enters its waiting loop.

Because of all this processing, the user interface cannot receive characters sent rapidly at the full baud rate. The host program must put a little breathing space between characters; perhaps a 3-millisecond delay. If you reduce the baud rate to 300 baud and set the host terminal to 1.5 or 2 stop bits, you may avoid the need to program a delay.

At the beginning of the program, during the initialization of the LCD, you may have noticed that several instructions are repeated, instead of being enclosed in for/next loops. This is not an oversight. Watching the downloading bar graph indicated that the repeated instructions actually resulted in a more compact program from the Stamp's point of view. Keep an eye on that graph when running programs; it a good relative indication of how much program space you've used. The terminal program occupies about two-thirds of the Stamp's EEPROM.

From an electronic standpoint, the circuit employs a couple of tricks. The first involves the RS-232 communication. The Stamp's processor, a PIC 16C56, is equipped with hefty static-protection diodes on its input/output pins. When the Stamp receives RS-232 data, which typically swings between -12 and +12 volts (V), these diodes serve to limit the voltage actually seen by the PIC's internal circuitry to 0 and +5V. The 22k resistor limits the current through the diodes to prevent damage.

Sending serial output without an external driver circuit exploits another loophole in the RS-232 standard. While most RS-232 devices

expect the signal to swing between at least -3 and +3V, most will accept the 0 and +5V output of the PIC without problems.

This setup is less noise-immune than circuits that play by the RS-232 rules. If you add a line driver/receiver such as a Maxim MAX232, remember that these devices also invert the signals. You'll have to change the baud/mode parameter in the instructions `serin` and `serout` to `T2400`, where T stands for true signal polarity. If industrial-strength noise immunity is required, or the interface will be at the end of a mile-long stretch of wire, use an RS-422 driver/receiver. This will require the same changes to `serin` and `serout`.

Another trick allows the sharing of input/output pins between the LCD and the pushbuttons. What happens if the user presses the buttons while the LCD is receiving data? Nothing. The Stamp can sink enough current to prevent the 1k pullup resistors from affecting the state of its active output lines. And when the Stamp is receiving input from the switches, the LCD is disabled, so its data lines are in a high-impedance state that's the next best thing to not being there. These facts allow the LCD and the switches to share the data lines without interference.

Finally, note that the resistors are shown on the data side of the switches, not on the +5V side. This is an inexpensive precaution against damage or interference due to electrostatic discharge from the user's fingertips. It's not an especially effective precaution, but the price is right.

**Program listing.** These programs may be downloaded from our Internet ftp site at [ftp.parallaxinc.com](ftp://ftp.parallaxinc.com). The ftp site may be reached directly or through our web site at <http://www.parallaxinc.com>.

---

```
' PROGRAM: Terminal.bas
' The Stamp serves as a user-interface terminal. It accepts text via RS-232 from a
' host, and provides a way for the user to respond to queries via four pushbuttons.

Symbol S_in   = 6           ' Serial data input pin
Symbol S_out  = 7           ' Serial data output pin
Symbol E      = 5           ' Enable pin, 1 = enabled
Symbol RS     = 4           ' Register select pin, 0 = instruction
Symbol keys   = b0          ' Variable holding # of key pressed.
Symbol char   = b3          ' Character sent to LCD.
```

```
Symbol Sw_0 = pin0          ' User input switches
Symbol Sw_1 = pin1          ' multiplexed w/LCD data lines.
Symbol Sw_2 = pin2
Symbol Sw_3 = pin3
```

' Set up the Stamp's I/O lines and initialize the LCD.

```
begin: let pins = 0          ' Clear the output lines
       let dirs = %01111111 ' One input, 7 outputs.
       pause 200           ' Wait 200 ms for LCD to reset.
```

' Initialize the LCD in accordance with Hitachi's instructions for 4-bit interface.

```
i_LCD: let pins = %00000011 ' Set to 8-bit operation.
       pulsout E,1          ' Send data three times
       pause 10            ' to initialize LCD.
       pulsout E,1
       pause 10
       pulsout E,1
       pause 10
       let pins = %00000010 ' Set to 4-bit operation.
       pulsout E,1          ' Send above data three times.
       pulsout E,1
       pulsout E,1
       let char = 14        ' Set up LCD in accordance with
       gosub wr_LCD         ' Hitachi instruction manual.
       let char = 6         ' Turn on cursor and enable
       gosub wr_LCD         ' left-to-right printing.
       let char = 1         ' Clear the display.
       gosub wr_LCD
       high RS              ' Prepare to send characters.
```

' Main program loop: receive data, check for backspace, and display data on LCD.

```
main:  serin S_in,N2400,char ' Main terminal loop.
       goto bksp
out:   gosub wr_LCD
       goto main
```

' Write the ASCII character in b3 to LCD.

```
wr_LCD: let pins = pins & %00010000
       let b2 = char/16      ' Put high nibble of b3 into b2.
       let pins = pins | b2  ' OR the contents of b2 into pins.
       pulsout E,1          ' Blip enable pin.
       let b2 = char & %00001111 ' Put low nibble of b3 into b2.
       let pins = pins & %00010000 ' Clear 4-bit data bus.
       let pins = pins | b2  ' OR the contents of b2 into pins.
       pulsout E,1          ' Blip enable.
       return
```

' Backspace, rub out character by printing a blank.

```

bksp:  if char > 13 then out          ' Not a bksp or cr? Output character.
        if char = 3 then clear       ' Ctl-C clears LCD screen.
        if char = 13 then cret       ' Carriage return.
        if char <> 8 then main        ' Reject other non-printables.
        gosub back
        let char = 32                ' Send a blank to display
        gosub wr_LCD
        gosub back
        goto main                    ' Back up to counter LCD's auto-
                                        ' increment.
                                        ' Get ready for another transmission.

back:   low RS
        let char = 16
        gosub wr_LCD
        high RS
        return                        ' Change to instruction register.
                                        ' Move cursor left.
                                        ' Write instruction to LCD.
                                        ' Put RS back in character mode.

clear:  low RS
        let b3 = 1
        gosub wr_LCD
        high RS
        goto main                    ' Change to instruction register.
                                        ' Clear the display.
                                        ' Write instruction to LCD.
                                        ' Put RS back in character mode.

' If a carriage return is received, wait for switch input from the user. The host
' program (on the other computer) should cooperate by waiting for a reply before
' sending more data.
cret:   let dirs = %01110000        ' Change LCD data lines to input.
loop:   let keys = 0
        if Sw_0 = 1 then xmit        ' Add one for each skipped key.
        let keys = keys + 1
        if Sw_1 = 1 then xmit
        let keys = keys + 1
        if Sw_2 = 1 then xmit
        let keys = keys + 1
        if Sw_3 = 1 then xmit
        goto loop

xmit:   serout S_out,N2400,(#keys,10,13)
        let dirs = %01111111        ' Restore I/O pins to original state.
        goto main

```

# **BASIC Stamp I Application Notes**

---