LCDbug

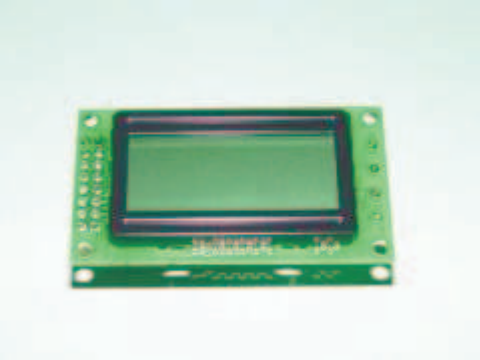


Figure 1. **LCDbug** top view

The **LCDbug** is an alphanumeric, liquid crystal display (LCD) module with a serial interface. Its small size and creative feature set combine to create an exceedingly affordable component that is very easy to use.

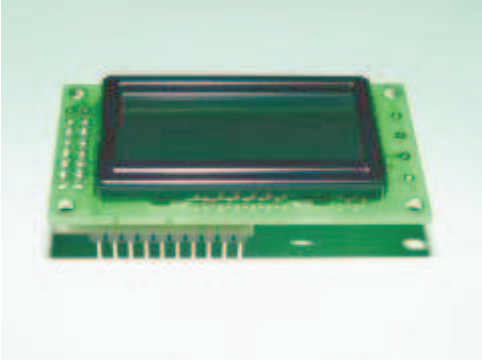


Figure 2. **LCDbug** front view

The modular design of the **LCDbug** allows it to be easily added to almost any circuit. It plugs in just like an IC. The expanded access to all the microcontroller pins allows the resourceful user to modify the basic behavior to suit new applications. Many free development tools are available to facilitate this process.

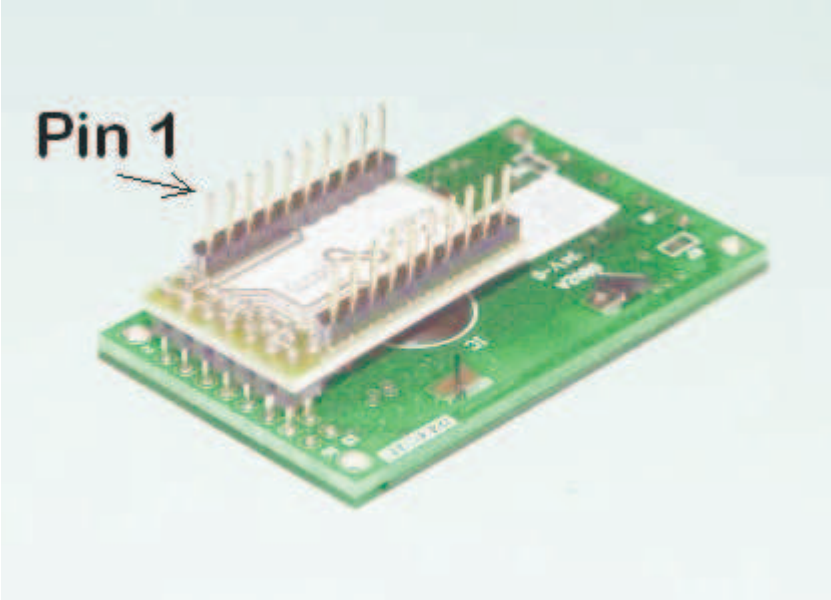


Figure 3. **LCDbug** bottom view

As shipped, the **LCDbug** is a full-featured serial LCD. The **LCDbug** comes completely assembled and programmed.

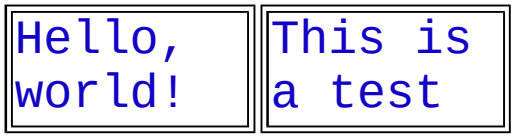
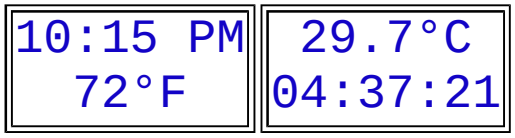
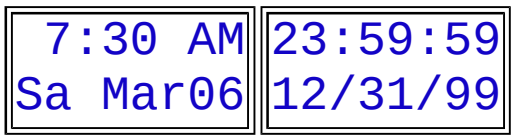
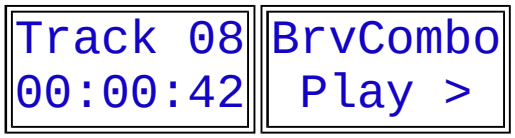
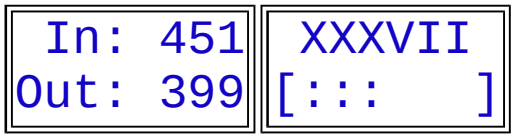
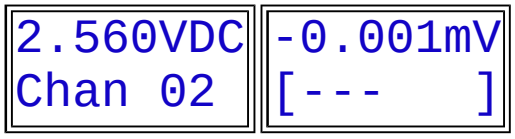
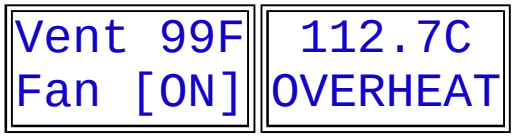
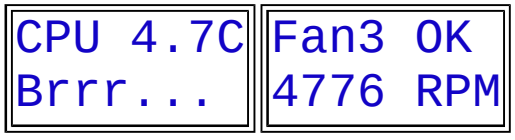
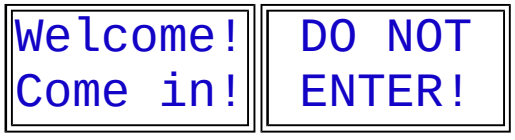
Features

- LCD character display
  - 2 line × 8 character display
  - 5 × 7 dot font
  - Dark blue characters on a light gray background
  - Fixed display contrast
  - No backlight
- Serial interface
  - TTL-level interface
  - 9600 baud, 8 data bits, 1 stop bit
  - No parity
  - No handshaking
  - 96 character buffer
- Processor

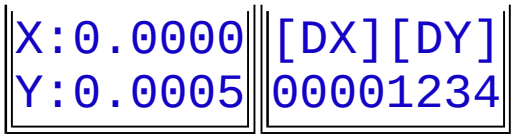
    - Preprogrammed Atmel AVR RISC microcontroller - ATtiny26
    - 8MHz internal calibrated oscillator - almost 8 MIPS
    - 2K reprogrammable flash memory - use existing code, or write your own
    - 7 additional I/O lines available
- Module
    - Low power: +5V @ 10mA max (6mA typical), and 1mA in power save mode
    - Compact size due to chip-on-board and surface mount components
    - All processor pins brought out to headers
    - Module can be easily plugged into wireless breadboards
    - Power, ground and serial input are the only required connections

Applications

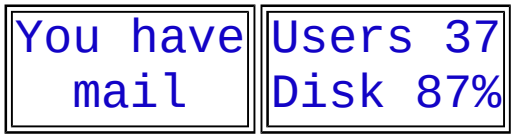
The potential uses of the **LCDbug** are limited only by your imagination. Here are some possible applications that I thought of (most of which have to do with temperature, for some reason). You'll think of more, I'm sure.

Example Applications	**Simulated Output**
Microcontroller debugging	
Time & temperature output	
Clock/calendar display	
Songlist, transport control, current track	
Event counter, progress indicator	
Digital voltage meter	
Thermostat or process readout	
CPU temperature & fan speed indicator	
Status/warning device	

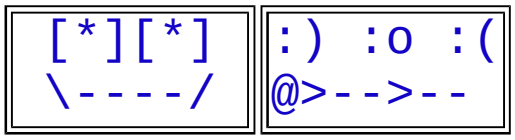
CNC DRO, robot odometer



Screen-saver proof notices



Goofy stuff



Functions

The **LCDbug** displays data as it is sent. It also recognizes and responds to a variety of useful commands. Commands are sent as "control characters".

Function	**Description**	**Decimal**	**Hex**	Keyboard equivalent
Custom Character 0	Displays custom character 0	0	0x00	None
Custom Character 1	Displays custom character 1	1	0x01	Ctrl-A
Custom Character 2	Displays custom character 2	2	0x02	Ctrl-B
Custom Character 3	Displays custom character 3	3	0x03	Ctrl-C
Custom Character 4	Displays custom character 4	4	0x04	Ctrl-D
Custom Character 5	Displays custom character 5	5	0x05	Ctrl-E
Custom Character 6	Displays custom character 6	6	0x06	Ctrl-F
Custom Character 7	Displays custom character 7	7	0x07	Ctrl-G
Backspace	Destructive backspace moves cursor back (left) one space, clearing previous content.	8	0x08	Backspace key Ctrl-H
Horizontal Tab	Moves the cursor to the next tab stop to the right. Tab stops are defined every four (4) columns.	9	0x09	Tab key Ctrl-I
Line Feed (new line)	Moves the cursor down one line. If the cursor goes past the last line, the scrolling mode determines what action is taken. If the scrolling option is enabled, the existing display text is moved up one line, the bottom line is cleared, and the cursor remains in the same place. If the scrolling option is disabled, the cursor wraps around to the same position in the top line.	10	0x0A	Ctrl-J
Vertical Tab	Scrolls the display up one line. The bottom line is cleared and the cursor remains in the same place.	11	0x0B	Ctrl-K
Clear Screen	Clears the display. The cursor returns to the upper left corner.	12	0x0C	Ctrl-L

Carriage Return	Moves the cursor to the left edge of the display. If the CR+LF option is enabled, a line feed is automatically added, moving the cursor to the next line down. If the CR+LF option is disabled, the cursor remains on the same line.	13	0x0D	Ctrl-M
Shift Left	Shifts the entire display to the left one space.	14	0x0E	Ctrl-N
Shift Right	Shifts the entire display to the right one space.	15	0x0F	Ctrl-O
Send LCD data	Sends the next character directly to the LCD data register. This is useful for defining custom characters or displaying any other character that would normally be interpreted as a control character.	16	0x10	Ctrl-P
Send LCD command	Sends the next character directly to the LCD command register. This is useful for defining custom characters, changing the cursor appearance and executing any of the built-in commands of the LCD module itself.	17	0x11	Ctrl-Q
Cursor Home	Moves the cursor to the upper left corner of the display. Resets any previous horizontal display shifting. Does not affect display contents.	18	0x12	Ctrl-R
Cursor Left	Moves the cursor one space to the left. Does not affect display contents.	19	0x13	Ctrl-S
Cursor Right	Moves the cursor one space to the right. Does not affect display contents.	20	0x14	Ctrl-T
Cursor Save	Saves the current position of the cursor, overwriting any previously saved position.	21	0x15	Ctrl-U
Cursor Restore	Restores the previously saved position of the cursor, if any.	22	0x16	Ctrl-V
Flip Page	Five (5) pages of data are stored internally by the LCD module. The lower three (3) bits of the next character received select the desired page.	23	0x17	Ctrl-W
Set Column (X)	Moves the cursor the the column indicated by the lower six (6) bits of the next character received. The current cursor row is unchanged.	24	0x18	Ctrl-X
Set Row (Y)	Moves the cursor the the row indicated by the lowest bit of the next character received. The current cursor column is unchanged.	25	0x19	Ctrl-Y
Set GPO	The next byte received is sent to the general purpose outputs.	26	0x1a	Ctrl-Z
	The next byte received sets an option. **Option Decimal Hex Description** A 65 0x41 Enable CR+LF B 66 0x42 Disable CR+LF			

Set Options	C D E F G H I J W X	67 68 69 70 71 72 73 74 87 88	0x43 0x44 0x45 0x46 0x47 0x48 0x49 0x4a 0x57 0x58	Enable Scrolling Disable Scrolling Enable Underline Cursor Disable Underline Cursor Enable Blinking Cursor Disable Blinking Cursor Select Vertical Bars Select Horizontal Bars Save options Reset options	27	0x1b	Escape key Ctrl-[
Show Revision	The product name and revision level is displayed. LCDbug v 1.00				28	0x1c	Ctrl-\
Low Power Mode	This command reduces power consumption to approximately 1mA. Receipt of the next character returns the unit to full power mode.				29	0x1d	Ctrl-]

Operation

Initialization

When power is first applied to the **LCDbug**, or after a reset, the following events occur:

- The LCD module is cleared
- The cursor is defined as an unblinking underline
- The cursor is placed in the upper, left corner (home position)
- The **LCDbug** is ready to accept data

Options

The appearance and behavior of the **LCDbug** is affected by several options that can be changed.

CR+LF Option

The CR+LF option permits the **LCDbug** to add a line feed to incoming carriage returns. This is convenient if you expect the cursor to move down one line when you type the [Enter] key. This option is disabled by default.

Scrolling Option

The Scrolling option determines what happens when the cursor moves past the bottom of the display. If Scrolling is disabled, the cursor simply wraps around to the top of the display. If Scrolling is enabled, the entire contents of the display is shifted up one line, the bottom line is cleared, and the cursor remains on the new bottom line. This option is disabled by default.

Cursor Appearance Options

The cursor determines the location where the next character will be placed. It can be indicated with an underline, a blinking block, both, or nothing at all. The default cursor appearance is an underline.

Saving and Restoring Options

Once options are changed, they remain active until the **LCDbug** is powered down or reset. The currently active option settings can be saved. These saved settings will be remembered even when power is removed. Options can also be returned to their previously saved state with the Reset Option command.

Connections

The **LCDbug** has twenty (20) pins, arranged as two (2) rows of ten (10) pins each. The pins are 0.100" apart, and the rows are 0.600" apart. This allows you to mount the **LCDbug** in an IC socket or directly onto a solderless breadboard.

Device Pinout

Pin #	Name	Description
1	PB0	Available I/O pin
2	PB1	Available I/O pin
3	PB2	Available I/O pin
4	PB3	Available I/O pin
5	VCC	+5V supply
6	GND	Ground
7	PB4	Available I/O pin
8	PB5	Available I/O pin
9	PB6	Serial input
10	-RESET	Device reset
11	PA7	Available I/O pin
12	PA6	LCD RS - register select
13	PA5	LCD RW - read/write
14	PA4	LCD E - enable
15	VCC	+5V supply
16	GND	Ground
17	PA3	LCD D3 - data line
18	PA2	LCD D2 - data line
19	PA2	LCD D2 - data line
20	PA0	LCD D0 - data line

"Wow!" you say, "that's a lot of connections!"

Have no fear. Only power, ground and serial input are required for normal operation. That's just three (3) wires, and only one (1!) of your valuable I/O lines.

Power connections

Power (4.5V to 5.5V) must be connected at pin 5 or 15, or both. A regulated (±5%) power supply is recommended.

Ground must be connected at pin 6 or 16, or both. Remember that you must connect a common ground between the **LCDbug** and any other electronics that will talk to it.

Serial Input

Pin 9 is the serial input. Please note that this is a TTL level (0-5V) signal and **not RS-232** (±12V) level signal. See the Interface section for more information about connecting the **LCDbug** to RS232.

The input is an asynchronous serial input running at 9600 baud

Device Reset

Pin 10 is configured as a reset input for the device. Momentarily connecting this pin to ground will cause the device to reset. A low pass filter, consisting of a pullup resistor and a small capacitor, provides reliable and consistent reset during power up. No connection is require for normal operation.

LCD connections

The LCD module has its own connector. The lines going to the LCD (LCD data, register select, read/write and enable) are also brought out to the headers. Normally nothing should be connected to these lines to prevent interference with the operation of the LCD module.

Available I/O lines

There are seven (7) additional lines available that are not used in the basic functionality of the **LCDbug**. These lines can be used as general purpose outputs.

The parameter sent after the "Set GPO" command sets the general purpose outputs as follows:

Bit	**D7**	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
Name	PA7	-	PB5	PB4	PB3	PB2	PB1	PB0
Pin	11	-	8	7	4	3	2	1

Interface

The serial input of the **LCDbug** is a TTL level (0-5V) input. **Do not** connect this input directly to RS232 level (±12V) signals, such as come from a PC or modem. To use RS232 level input, a suitable voltage level shifting circuit **must** be used. An example of a simple RS232 to TTL level circuit that will work well with the **LCDbug**, see my 4 Digit 7 Segment LED project. Other circuits can be built using the 1489 or MAX232 chips.

BASIC Stamp

Connecting the **LCDbug** to the BASIC Stamp from Parallax is very simple. Pick an output pin on the BASIC Stamp and connect it directly to the serial input (pin 9) of the **LCDbug**. No level-shifting circuit is required, as both the **LCDbug** and the BASIC Stamp both use TTL-level signals. Be sure to also connect the ground pin (either pin 6 or 16 on the **LCDbug**) to the ground pin (Vss) on the BASIC Stamp. Here is the correct syntax for the serial output command for a BS2:

```
SEROUT 15, 84, ["Hello!"]
```

This example assumes using output pin P15. You can use any pin you want. Note that you only need to use one (1) pin, saving your valuable I/O lines for other purposes. The "84" parameter is specific to the BS2 and defines the baud rate to be 9600. This value varies with different BASIC Stamp models, so be sure to check

your documentation for the correct value.

In some cases, the BASIC Stamp will power on and start sending data before the **LCDbug** has fininshed initializing the LCD. Add a short delay using the NAP command (example `NAP 4`) before sending any data to the **LCDbug**.

Here is a short but complete PBASIC program that talks to the **LCDbug**:

```
'{$STAMP BS2}
NAP 4
SEROUT 15, 84, [12] ' Clear screen
SEROUT 15, 84, ["Hello!"] ' Line 1
SEROUT 15, 84, [CR, 10] ' Carriage return & line feed

SEROUT 15, 84, ["LCDbug"] ' Line 2
```

Programming

As shipped, the **LCDbug** comes pre-programmed with highly optimized firmware, written in AVR assembly language. With a little effort, you can write your own programs for the **LCDbug**. You will need two (2) things to do this:

- Language compiler or assembler software
- AVR ISP programmer

You have many options when it comes to the programming language to use. Here is a partial list of languages that are known to work with the **LCDbug**:

- AVR Assembly
  - Atmel's AVR Assembler (free)
- BASIC
  - FastAVR 4.0 (www.fastavr.com) $59, with free demo version
- C
  - GNU C for AVR (avr-gcc, free)
  - Atman Electronics AtmanAVR C (www.atmanecl.com) 30 day free demo

Likewise, you have a lot of options when it comes to the hardware programmer. You can by Atmel's AVR ISP for ~$29, or build one yourself. It can also be programmed with the STK-200 or STK-500 development boards.

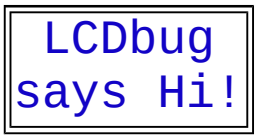
FastAVR BASIC example

Here is an example program that shows how easy it is to program the **LCDbug** yourself in BASIC:

```
$Device= ATtiny26
$Stack = 32
$Clock = 8.000

$Lcd = PORTA.0, RS=PORTA.6, EN=PORTA.4, 8, 2 ' Define the connections to the LCD


DDRA.5 = 1:PORTA.5 = 0 ' The LCD R/W line must be grounded

InitLcd()
```

```
Lcd " LCDbug "
Locate 2, 1
Lcd "says Hi!"
Do : Loop
```

This code produces a display that looks like this:



When writing your own programs, you have access to all the internal resources of the ATtiny26 processor, including:

- 16 programmable I/O lines (7 are permanently connected to the LCD, remember)
- Two 8-bit timer/counters with high frequency PWM capability
- 32 registers
- 128 bytes of SRAM
- 128 bytes of EEPROM
- Analog to digital converter (11 channels, 10 bits)
- Analog comparator
- External interrupt
- Advanced AVR RISC architecture

Note that if you program the **LCDbug** with your own program, the original serial LCD firmware is gone forever.
