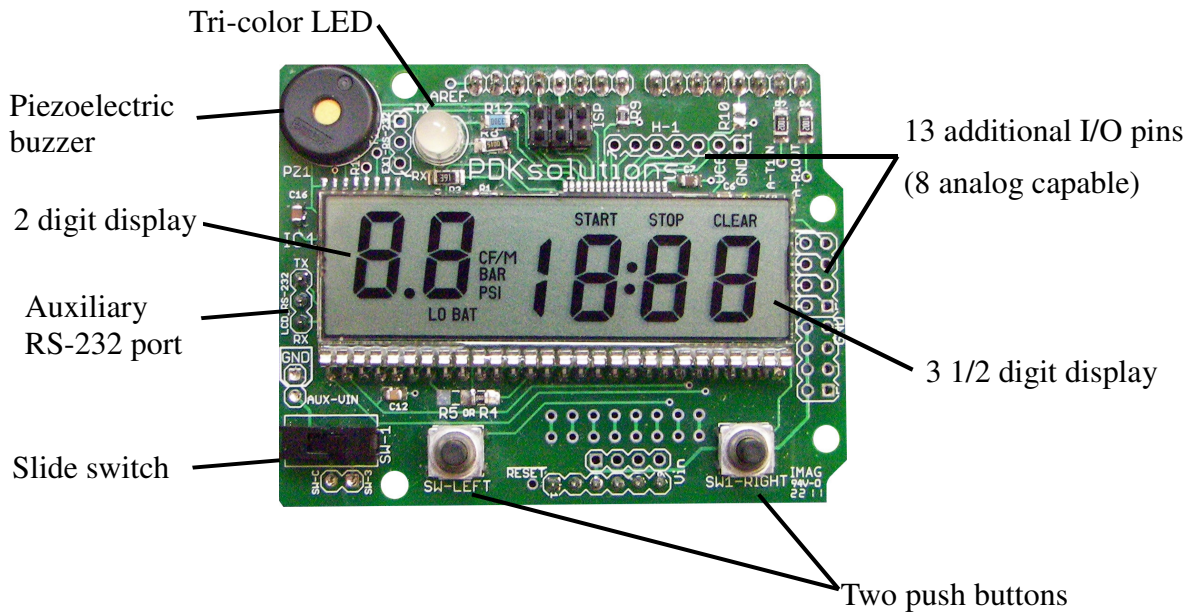


JEMshield LCD-7seg-IO



JEM Innovation Inc
125 Stearman Court
Erie, CO 80516
www.jeminnovation.com

The LCD-7seg-IO JEMshield consumes only 4 I/O pins and provides the following features:

1. Two 7 Segment LCD displays:
 - a. 2 digit with decimal (8.8)
 - b. 3 ½ digit with colon (18:88)
 - c. Numerous icons.
2. Two push buttons
3. Piezoelectric buzzer
4. Tri-color LED (Red, Green, Blue) with PWM brightness control.
5. RS-232 driver chip (one port under GPISO-1 control; the other available for your Arduino.)
6. 13 additional I/O pins: 5 digital; 8 digital I/O or analog.
7. User configurable slide switch

All the above features are controlled via the SPI interface. (And other devices can still share the SPI interface.) The LCD-7seg-IO uses only:

1. MOSI (P11)
2. MISO (P12)
3. SCK (P13)
4. Chip Select

Arduino software drivers are available on our website:

www.pdksolutions.com.

Getting Started:

The LCD-7seg-IO plugs in just like any other shield. Obviously, you'll want it on the top of the stack so you can see the display and use the push buttons.

You'll need to choose the chip select you'll want your Arduino to use to address the LCD-7seg-IO (the default is P8.) See schematic for options.

Drivers:

You MUST copy the LCD-7seg-IO drivers and header file into the “libraries” folder in your Arduino environment exactly as shown:

```
..\libraries\SPIShield\ JEMLCD1.cpp  
..\libraries\SPIShield\ JEMLCD1.h
```

If you don't have a “libraries” folder in your Arduino environment you'll need to create one.

Software:

You must include the following 3 lines of code at the top of your Sketch:

```
#include < JEMLCD1.h>  
#include <SPI.h>
```

```
JEMLCD1 lcd(8);
```

The above line tells the SPI driver that the LCD-7seg-IO is on chip select P8 (default but you can use any of the other I/O pins. The board has jumpers for P4, P5, P6 and P7.)

This also tells the compiler that the shield will be referred to as “lcd”. You can rename it any valid name: myLCD, AUXIO, displayboard or even Fred or Mary. You'll have to use that name whenever you want to access the features on the board, so make it something you'll remember and is easy to type.

In setup(), you must include the line:

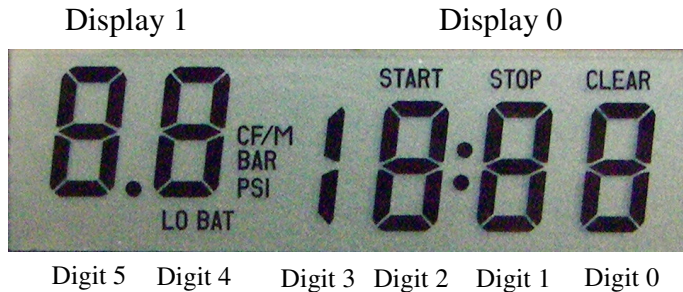
```
lcd.JEM_init(); // this initializes the LCD-7seg-IO shield
```

Note that if you named your board something other than “lcd” you have to use that name instead.

LCD Display

Hardware

The LCD consists a two digit display and a 3½ display as shown below. The right display is *display0*; the left is *display1*.



The displays are offset vertically to help the viewer distinguish between them. For example, you could display pressure on the left display and time on the right one.

Software

void display(int value, [byte display= 0], [byte format= 0])

This function will display *value* on the specified display.

Value:

valid range: -99 to 1999 – (display0)
 0-99 – (display1)

Display (optional):

0 – put the data on the right side (display1)
1 – put the data on the left side (display0)

Format (optional):

0 – do not include leading zeros
1 – pad the display with leading zeros

Examples:

```
lcd.display(21);            // display the value 21 on display0 (right side)
lcd.display(21,1);        // display the value 21 on display1 (left side)
lcd.display(21,0,1);      // display "021" on display0
lcd.display(3,1,1);       // display "03" on display1
lcd.display(-21);         // display "-21" on display0
```

void setDigit(byte position, byte value)

This function allows you to write to an individual digit. For example:

```
lcd.setDigit (4, 7);
lcd.setDigit (5, 3);
```

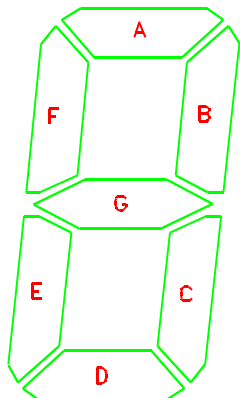
This will display the number '37' on the left display.

Valid values are: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, ' ', '-', '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '@', 'A', 'B', 'C', 'D', 'E', 'F', 'H', 'L', 'N', 'O', 'P', 'U', 'u'.

Writing the value '255' (0xFF) will blank that digit.

void setSegments(byte position, byte segments)

Allows you to set/clear individual segments. The standard designation for a 7 segment display is:



On this board the segments are mapped as:

data bit:	7	6	5	4	3	2	1	0
display segment:	-	E	D	C	F	G	B	A

(The mapping is a result of the routing on the LCD glass itself.)

This mapping does NOT apply to digit3, which is actually an icon; not a digit. Thus writing any value other than '0', to digit3 will turn it on.

Example:

```
lcd.setSegments (0, 0x04); // turns on segment G (minus sign) in digit0
```

void setIcons(int set, int clear)

Turn the icons on or off. Setting a bit to '1' will turn on the corresponding icon; setting a bit to '0' will turn in off. This allow software to set and clear icons in one instruction.

data bit:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Icon	FBA	-	-	-	-	BAR	PSI	CFM	-	-	.	:	LoBat	CLEAR	STOP	START

Note: if you remove R4 and install a short on R5, then bit 9 will display “AGE” (and not PSI).

Examples:

```
lcd.setIcons(0x01, 0); // turns on “START”
lcd.setIcons(0x02, 0); // turns on “STOP” but leaves “START” on
lcd.setIcons(0x02, ~(0x02)); // turns on “STOP” and clears all other icons
```

void setContrast(byte contrast)

This function allows the software to write to the LCDCCR register in the Atmega169. This sets the drive time and drive voltage for the LCD. Users may want to adjust these values to optimize the appearance of the display over temperature or for different viewing angles.

You'll want to read the Atmel data sheet before using this function!

RGB LED

The shield is equipped with a 5mm RGB (Red, Green, Blue) LED that is under software control.

void setRGB(byte red, byte green, byte blue)

Turns on the RGB LED.

Valid range: 0-255 (higher the number, the brighter the corresponding color)

Example:

```
lcd.setRGB (128,0,0);    // red only
lcd.setRGB (128,128,0); // red and green, no blue
lcd.setRGB (0,0,64);    // light blue
```

Note: the human eye is not linear. For example, it is more sensitive to blue than green. Of course, this sensitivity varies from person to person as well. Thus, you'll have to experiment to get just the right shade of chartreuse that you're looking for.

Push Buttons

The board has two push buttons that are monitored by the shield software.

*byte buttons([byte *currentPtr])*

Returns a byte that indicates if a button has been pressed since the last call to *buttons()*.

Bit 0: right button was pressed

bit 7: left button was pressed

Optionally: writes the current state of the buttons to the address specified by *currentPtr*. This allows the user to see if the button is down at that instant.

```
byte button_status, button_events;
```

```
button_events = buttons (&button_status);
```

If, after the above line of code is executed:

`button_events == 0x8` - both buttons have been pressed since the last time *buttons* was called

`button_status == 0x01` - the right button is still being pressed

If you just want to know if a button was pressed since the last call to the function, you can use:

```
button_events = buttons ();
```

Piezoelectric Buzzer

The purpose of the buzzer is to provide audio feedback to the user. It was not intended to play Beethoven's 5th symphony. Note that piezoelectric buzzers are notoriously non-linear as shown in the chart below (from the TDK datasheet, part number PS1240P02BT).

`void beep(byte pitch, byte duration);`

Generates a “beep” from the onboard piezo electric buzzer. Similar to the Arduino “tone” command except that it's completely different. Mainly, the parameters are relative indexes and not absolute values.

pitch: specifies the relative pitch of the tone. Valid range is 0-255; practical range is 1-20.

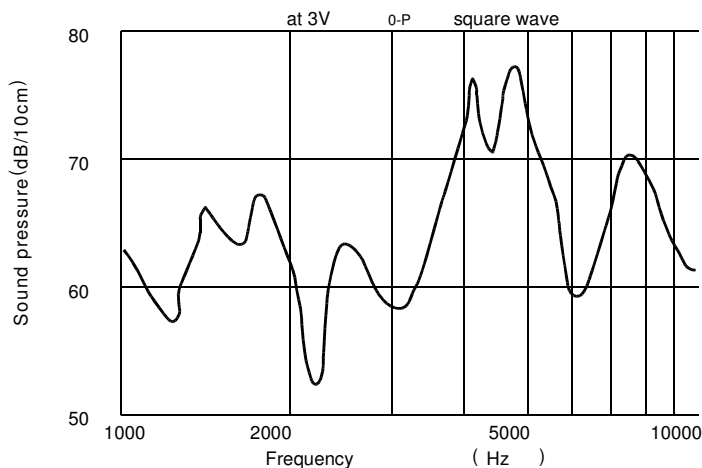
duration: specifies the duration of the tone. Each count is *about* 8 mS.

You can take advantage of the non-linear aspect of the buzzer and use it to *sort-of* regulate volume.

Example:

```
lcd.beep (3, 25); // loudest beep, about 200mS  
lcd.beep (4, 250); // very quiet but very long
```

Even with these limitations, you can make some pretty cool sounding noise. Who knows, maybe you can play Beethoven on it.



Pitch value	Approx frequency	Relative volume
1	7800 (Hz)	Low
3	4000 (Hz)	Loudest!
5	2600 (Hz)	Very quiet
15	980 (Hz)	medium

Serial Port

The board has a dual RS-232 driver chip. One driver is connected to the UART in the Atmega169; the other can be connected to the UART from the Arduino board.

Note: none of these commands affect the Arduino serial port.

void write(byte data)

Sends a byte of data out the TX port.

Example:

```
lcd.write ('J'); // transmits a 'J'
```

byte available(void)

Returns the number of bytes in the serial receive buffer on the LCD7seg shield, (max is 128). Use this to determine if you need to read from the buffer.

byte read(void)

Reads the next byte from the buffer on the LCD7seg shield.

Example:

```
if (lcd.available()){  
    val = lcd.read();  
    lcd.write(val);  
} // */
```

This little code snippet reads the next available byte and then sends it out the transmit port. You would probably want to expand on this and do something useful with the data.

Note: all other serial functions (print etc) can be built from these three functions.

void SetBitRate(unsigned long bitrate)

Just as it says, sets the bit rate for UART. The UART will default to 19,200 if this function isn't called. This can be done two different ways:

bitrate

0 – 299: The LCD-7seg-IO board will treat any number in this range as the *divisor* for the Atmega169 UART and program it into the UBRR0L register. Better read the data sheet before using this syntax.

300 – 115200: The LCD-7seg-IO software will attempt to calculate the closest divisor to the specified bit rate and program the UART accordingly.

WARNING: while you can request 115,200, the Atmega will be so far off frequency that it probably won't sync up with anything (except maybe another LCD7seg shield board); 57,600 is the practical limit. This is related to using an 8 MHz clock (see that Atmega169 datasheet for details.)

Auxiliary I/O pins

The shield provides an additional 13 I/O pins that can be accessed via the serial interface. Eight of these can be used as analog inputs; the other 5 are digital only.

NOTE: These functions work the same as the standard Arduino library calls. However, you MUST include "lcd." (or whatever name you assigned the board), so that the compiler can distinguish between Arduino functions and JEMshield functions.

void pinMode(byte pin, byte onoff)

Sets the mode for the specified pin. Valid range is 0-12. Onoff = 0 (input) or 1 (output);

```
lcd.pinMode (1, OUTPUT); // sets the LCD7seg pin 1 to an output.
```

byte digitalRead(byte pin)

Reads the state of the specified pin. Valid range is 0-12; make sure you set the pin as an input before calling this function. Note: running the JEM_init sets all pins to inputs.

Example:

```
byte sensor1;
```

```
sensor1 = lcd.digitalRead (0); // sensor1 will equal 1 if pin is high; 0 if it's low
```

void digitalWrite(byte pin, byte pinstate)

Sets the specified pin to the value specified by pinstate. Make sure that you've set the pin to an output before calling this function or you'll just turn on the pullup resistor for that pin.

pin: Specifies which pin to use. Valid range: 0-12; see schematic for details.

Pinstate: 0 for LOW; 1 for HIGH.

```
lcd.digitalWrite (1, HIGH); // sets pin 1 high
```

unsigned int analogRead(byte pin)

Returns the analog value present on the specified pin. The 5 volt power supply is used as reference. The pin must be set as an input pin using pinMode. Returns an unsigned integer between 0 and 1023.

valid range: 0-7

The voltage present on the pin can be found from the following formula:

$$V = (\text{adc_count} * 5) / 1024$$

Example:

```
float voltage;
```

```
voltage = (lcd.analogRead(0) * 5) / 1024; // voltage on LCD-7seg-IO pin 0
```

Slide Switch

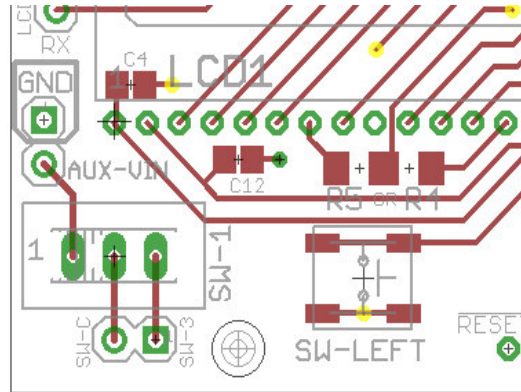
Over the years, we've found that a general purpose switch often comes in handy; so we added one to the shield. By default, it's connected to nothing so that you can hook it up however you want. A blowup of the circuit board is shown below.

Examples:

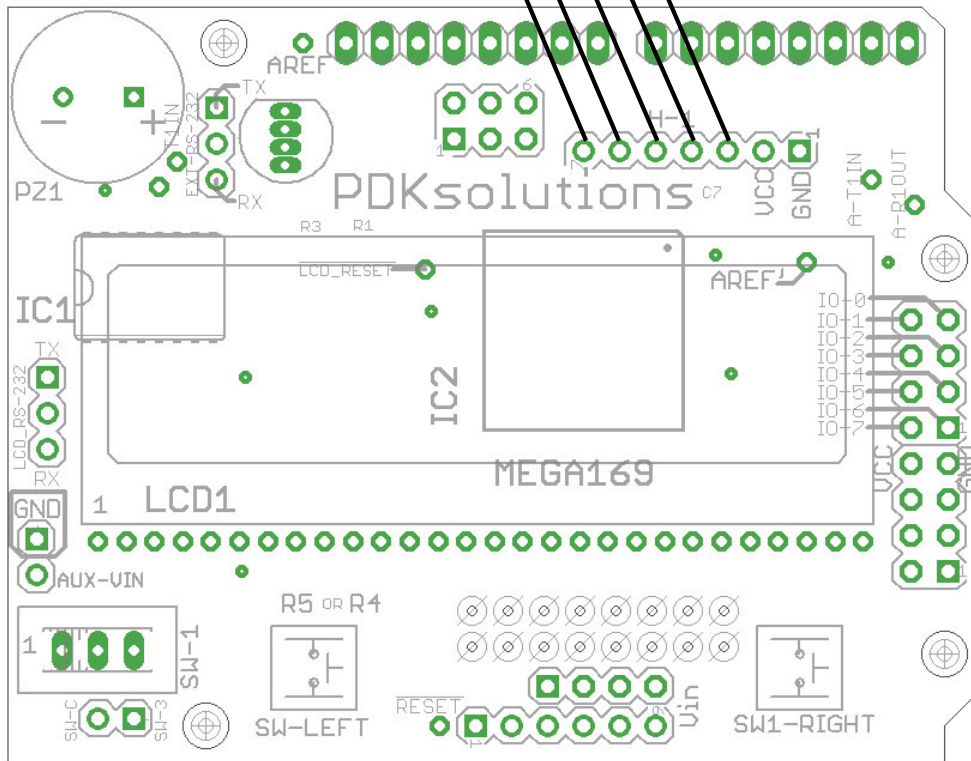
Tie pin 1 to ground (conveniently available via the two pin connector), tie SW-C to an unused digital I/O pin and you've just added a "mode" switch to your board.

Connect power to "AUX-VIN" and connect the SW-C to your Arduino Vin and you've got a power switch.

We're sure you'll think of something to use it for that we didn't. Just be careful.



- Digital I/O 8
- Digital I/O 9
- Digital I/O 10
- Digital I/O 11
- Digital I/O 12



Analog capable I/O