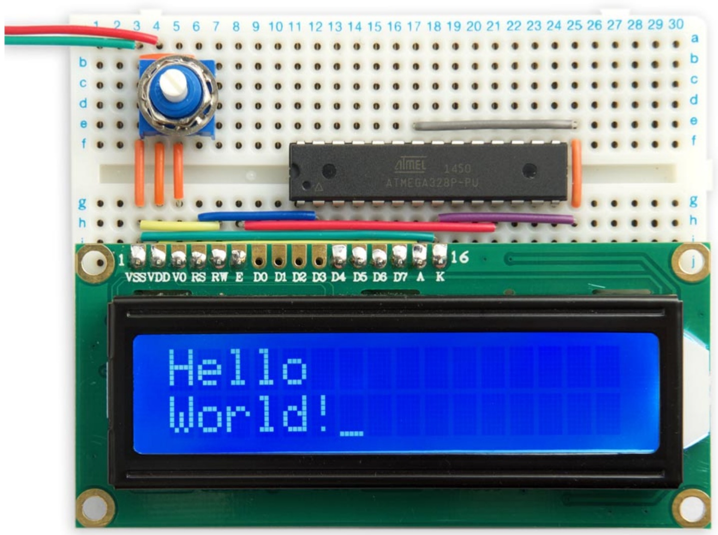
Technoblogy

Arduino and AVR projects

# Simple LCD Character Display

15th September 2018

LCD character displays are a convenient low-cost way to provide readable text output for a project. This project shows a simple way to interface a 16x2 character display to an ATmega328 on a prototyping board with the minimum of wiring:



*Simple LCD character display, interfaced to an ATmega328.*

## Introduction

LCD character displays are typically based on the Hitachi HD44780 display driver [1], and are available in a variety of colours and formats, including 16 characters x 2 lines, 20 characters x 4 lines, 16 characters x 4 lines, and 8 characters x 2 lines. The displays usually include an analogue contrast adjustment signal, and an LED backlight.

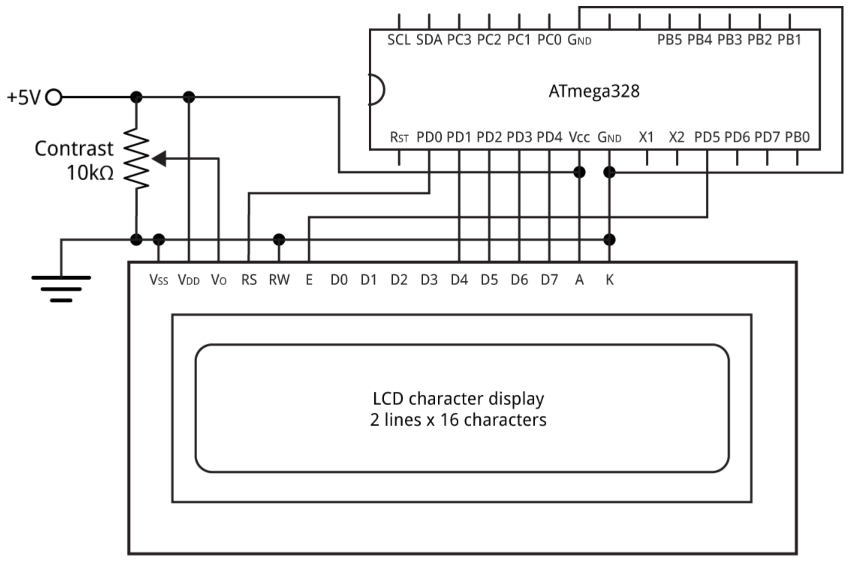
The following example is based on a 5V 16x2 LCD display [2] [3] but should work with other displays. Alternatively if you need more advanced features, or have a different type of display, there's an Arduino Liquid Crystal library.

The displays include a built-in ASCII character set, so the interface program is relatively simple. The displays offer two interface modes: 8-bit, or 4-bit. This application uses the 4-bit mode which saves I/O lines by sending the data in two 4-bit parts, using just D4 to D7.

I first tested this program in uLisp, my Lisp for the Arduino, before converting it into C; see LCD character display.

## The circuit

You need to connect six I/O lines from the Arduino to the display; four for the data lines D4 to D7, and two for the enable and RS lines. This application keeps the wiring and program simpler by using four consecutive pins in one ATmega328 register for these four data pins. By aligning the display with the ATmega328 on the prototyping board we save having to connect these wires:


SCL SDA PC3 PC2 PC1 PC0 GND PB5 PB4 PB3 PB2 PB1
ATmega328
RST PD0 PD1 PD2 PD3 PD4 VCC GND X1 X2 PD5 PD6 PD7 PB0
+5V
Contrast 10kΩ
VSS VDD VO RS RW E D0 D1 D2 D3 D4 D5 D6 D7 A K
LCD character display
2 lines x 16 characters

*Circuit of the LCD character display interface.*

The pins you've used should be specified by these constants:

```
const int data = 1;    // PD1 to PD4 connect to D4 to D7 on the display
const int enable = 5;  // PD5
const int rs = 0;      // PD0
```

**Tip:** Don't fit pin headers to the pins D0 to D3 on the display module; these aren't used, and leaving them unconnected may allow you to save some wiring between the Arduino board and the display module, as in my prototype in the photograph above.

### Defining the LCD class

First we define a class called **LCD** based on the **Print** class. This enables it to inherit the behaviour of the standard I/O functions such as **print()** to print strings and integers:

```
class LCD : public Print {
  public:
    void init();
    void cmd(uint8_t);
    size_t write(uint8_t);
  private:
    void nib(uint8_t);
};
```

### Initialising the display

First the routine **init()** defines all the pins used by the display as outputs, and sends four commands to initialise the display:

```
void LCD::init () {
  DDRD = DDRD | 0xF<<data | 1<<enable | 1<<rs;   // Make data E and RS pins outputs
  PORTD = PORTD & ~(1<<rs);                      // Take RS low
  cmd(0x33);                                     // Ensures display is in 8-bit mode
  cmd(0x32);                                     // Puts display in 4-bit mode
  cmd(0x0e);                                     // Display and cursor on
```

## About me

About me
Contact me

Follow @technoblogy

## Feeds

RSS feed

```
  cmd(0x01);                                    // Clear display
}
```

The commands are:

- 0x33 and 0x32 put the display into 4-bit mode.
- 0x0e turns the display and cursor on. Change this to 0x0c if you don't want a cursor.
- 0x01 clears the display.

The first two commands are designed to work whether the display is initially in 8-bit or 4-bit mode, as follows.

If the display is initially in 8-bit mode, as it is after power-on, the first byte 0x33 is interpreted as two commands:

0b0011xxxx, 0b0011xxxx

where xxxx are the 'don't care' states of the lower-four data lines, D0 to D3. These leave the display in 8-bit mode.

If, however, the display is initially in 4-bit mode, as it would be if you just reset the Arduino, the first byte 0x33 is interpreted as a single command:

0b00110011

which puts the display into 8-bit mode. In either case the second byte 0x32 is interpreted in 8-bit mode as the two commands:

0b0011xxxx, 0b0010xxxx

which put the display into 4-bit mode.

### Sending data

The routine **nib()** sends a four-bit nibble to the four data pins, and pulses the enable pin:

```
void LCD::nib (uint8_t n) {
  PORTD = (PORTD & ~(0xF<<data)) | n<<data | 1<<enable; // Send data and enable high
  PORTD = PORTD & ~(1<<enable);                 // Take enable low
  delay(1);                                     // Allow data to execute on display
}
```

The routine **write()** calls **nib** twice to send a byte:

```
size_t LCD::write (uint8_t b) {
  nib(b>>4); nib(b&0xf);
  return 1;
}
```

Finally, the routine **cmd()** calls **write()** to send a command. The RS line is taken low for commands:

```
void LCD::cmd (uint8_t c) {
  PORTD = PORTD & ~(1<<rs);                     // Take RS low
  write(c);
  PORTD = PORTD | 1<<rs;                        // Take RS high
  delay(1);                                     // Allow to execute on display
}
```

## Demo program

First create an instance of the **LCD** class, called **lcd**:

```
LCD lcd;
```

Then to setup the display, and write **Hello World!** on the two lines of the display run:

```
void setup() {
  lcd.init();
  lcd.print("Hello");
  lcd.cmd(0xc0);                                  // Cursor to start of second line
  lcd.print("World!");
}
```

Some useful commands are:

- 0x01 clears the display.
- 0x80 moves the cursor to the start of the first line.
- 0xc0 moves the cursor to the start of the second line.
- 0x0c turns off the cursor.
- 0x0e turns on the cursor.

## Compiling the program

Compile the program with my **ATmegaBreadboard** hardware configuration on GitHub, following the instructions there to add it to your Arduino **hardware** folder: ATmegaBreadboard.

Select the **ATmega328** option under the **ATmegaBreadboard** heading on the **Boards** menu. Then choose **ATmega328P**, **8 MHz (internal)**, and **B.O.D Disabled** from the subsequent menus. Choose the correct programmer from the **Programmer** option on the **Tools** menu; for example, **USBtinyISP** if you're using the Sparkfun Tiny AVR Programmer. Choose **Burn Bootloader** to set the fuses appropriately; then choose **Upload** to upload the program.

Here's the whole program: Simple LCD Character Display Program

1. ^ HD44780 datasheet on Sparkfun.
2. ^ 16x2 LCD Display White/Blue LED Backlight on HobbyTronics.
3. ^ Standard LCD 16x2 + extras - white on blue on Adafruit.

**Next:** I2C GPS Module **Previous:** Alcohol Unit Counter

**10 Comments** Technoblogy Disqus' Privacy Policy 1 Login

Favorite Tweet Share Sort by Best

Join the discussion…

LOG IN WITH OR SIGN UP WITH DISQUS ?

Name

**MirouP** • 2 years ago • edited

As a newbie, who just overcome "buttons" and "led flashing" codes with Atmega328 board, I've

decided to get more experienced :-) and try this example. I'm using WinAVR 20100110. Unfortunately I'm getting several error during compiling program file. It looks it doesn't know classes:
line 16 in your program: class LCD : public Print {
error: main.c:16: error: expected '=', ',', ';', 'asm' or '__attribute__' before 'LCD'

saving main.c as main.cpp :-) I resolved issue with knowing class, but It looks WinAVR doesn't support print class in its header files. So, I have to switch to Arduino IDE :-)

Moved to stage where WinAVR compiler created .o file (I used print.h, Wstring.h and printable.h from Arduino IDE) , but still not in final stage:
main.cpp:60: undefined reference to `Print::print(char const*)'
main.cpp:62: undefined reference to `Print::print(char const*)'
line 60: lcd.print("Hello");
line 62: lcd.print("World!");
main.o:(.data+0x12): undefined reference to `Print::write(unsigned char const*, unsigned int)'

Reply • Share ›

**johnsondavies** Mod ➦ MirouP • 2 years ago

Sorry I don't really know anything about WinAVR, but it should work fine under the Arduino IDE, which is what I used.

Reply • Share ›

**MirouP** ➦ johnsondavies • 2 years ago • edited

Yes, I've compiled it and exported compiled binary to create hex file, it looks pretty simply in the Arduino IDE. Thanx :-)
Compiled for Arduino UNO (uses same controller) and work fine :-):-)
Just contrast setting is very sensitive :-)
Many thanks for this kind of lectures :-)

Reply • Share ›

**Mann Hansen** • 4 years ago

Thanks for this tutorial. Very interesting and educational.
Is it possible to change or modify this example to support 4X20 and 2X8 LCD's ?
I'm not so good at reading datasheets. That why I'm asking for advice.

Reply • Share ›

**johnsondavies** Mod ➦ Mann Hansen • 4 years ago

I haven't test other display formats, but I think they should work fine with this code. The 8x2 should work the same as the 16x2. The 20x4 needs the following codes to get to the start of each line:

0x80 - start of line 1
0xc0 - start of line 2
0x94 - start of line 3
0xd4 - start of line 4

Reply • Share ›

**Phil-S** • 4 years ago

I like this.
Bought-in serial backpacks (PIC-based) have not proved very robust in my experience and are, of course, "secret". The PCF8574's work fine but add quite a bit more wiring. The "bare" 328P chips are cheap enough to build in as a dedicated backpack. Thanks for sharing and nice presentation as ever.

Reply • Share ›

**Mike, K8LH** • 4 years ago • edited

Based on your example, I modified my little 2-pin 8-bit LCD Backpack Demo' sketch to use the

Print class and it only increased program size by 184 bytes (from 442 to 626 bytes). The original sketch simply had init(), cmd(), char(), and str() functions, as well as an overload str() function for "flash string helper" strings. I'm very 'geeked'! Thank you, David.

^ | ⌄ • Reply • Share ›

johnsondavies Mod ➦ Mike, K8LH • 4 years ago

Glad it was helpful!

^ | ⌄ • Reply • Share ›

Mike, K8LH • 4 years ago • edited

Another stimulating and thoughtful demo'. Thank you, David. Your connection between the '328 and LCD is quite clever and It's exciting to see how to attach functions/drivers to a class in a simple and concise example like this.

The Print class supports "flash string helper" strings, too, doesn't it?

Cheerful regards, Mike

^ | ⌄ • Reply • Share ›

johnsondavies Mod ➦ Mike, K8LH • 4 years ago

Yes, that's a good point.

^ | ⌄ • Reply • Share ›

Subscribe Add Disqus to your siteAdd DisqusAdd Do Not Sell My Data DISQUS