Last Minute ENGINEERS

Interfacing 16×2 Character LCD Module with Arduino

Want your Arduino projects to display status messages or sensor readings? Then these LCD displays might be the perfect fit. They are extremely common and a fast way to add a readable interface to your project.

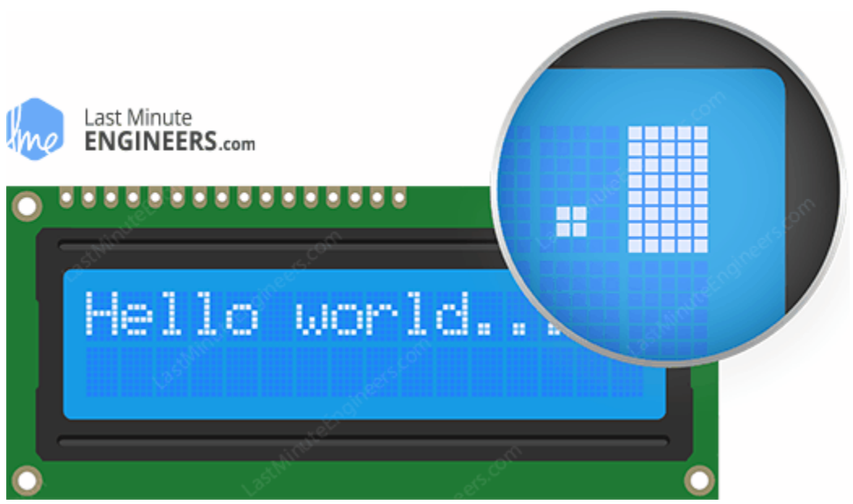
This tutorial will cover everything you need to know to get up and running with Character LCDs. Not just 16×2(1602) but any character LCDs (for example, 16×4, 16×1, 20×4 etc.) that are based on parallel interface LCD controller chip from Hitachi called the HD44780. Because, the Arduino community has already developed a library to handle HD44780 LCDs; so we'll have them

interfaced in no time.

> **Did you know?**
>
> An LCD is short for Liquid Crystal Display. It is basically a display unit which uses liquid crystals to produce a visible image.
>
> When current is applied to this special kind of crystal, it turns opaque blocking the backlight that lives behind the screen. As a result that particular area will become dark compared to other. And that's how characters are displayed on the screen.

Hardware Overview

These LCDs are ideal for displaying text/characters only, hence the name 'Character LCD'. The display has an LED backlight and can display 32 ASCII characters in two rows with 16 characters on each row.



Each rectangle contains grid of 5×8 pixels

If you look closely, you can actually see the little rectangles for each character on the display and the pixels that make up a character. Each of these rectangles is a grid of 5×8 pixels.

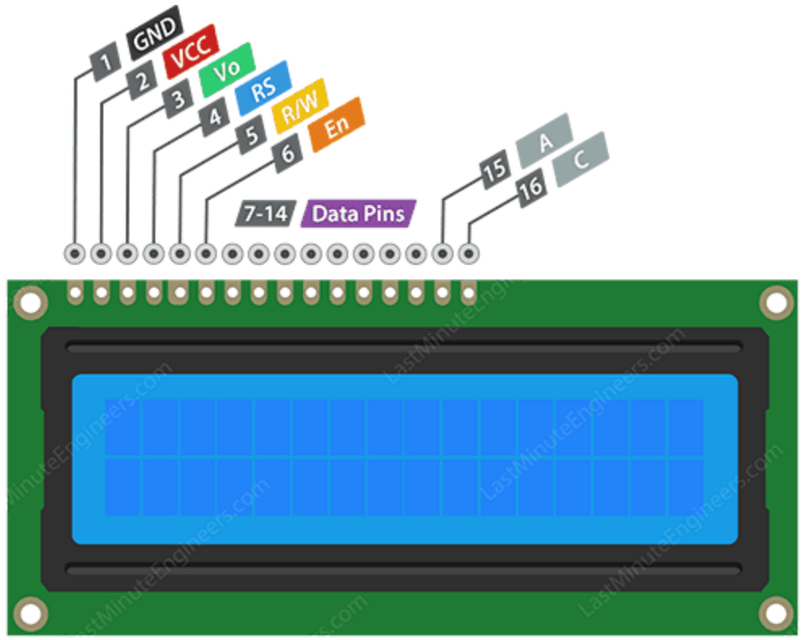
Although they display only text, they do come in many sizes and colors: for example, 16×1, 16×4,

20×4, with white text on blue background, with black text on green and many more.

The good news is that all of these displays are 'swappable' – if you build your project with one you can just unplug it and use another size/color LCD of your choice. Your code may have to adjust to the larger size but at least the wiring is the same!

16×2 Character LCD Pinout

Before diving into hookup and example code, let's first take a look at the LCD Pinout.



GND should be connected to the ground of Arduino.

VCC is the power supply for the LCD which we connect the 5 volts pin on the Arduino.

Vo (LCD Contrast) controls the contrast and brightness of the LCD. Using a simple voltage divider with a potentiometer, we can make fine adjustments to the contrast.

RS (Register Select) pin lets the Arduino tell the LCD whether it is sending commands or the data.

Basically this pin is used to differentiate commands from the data.

For example, when RS pin is set to LOW, then we are sending commands to the LCD (like set the cursor to a specific location, clear the display, scroll the display to the right and so on). And when RS pin is set on HIGH we are sending data/characters to the LCD.

R/W (Read/Write) pin on the LCD is to control whether or not you're reading data from the LCD or writing data to the LCD. Since we're just using this LCD as an OUTPUT device, we're going to tie this pin LOW. This forces it into the WRITE mode.

E (Enable) pin is used to enable the display. Meaning, when this pin is set to LOW, the LCD does not care what is happening with R/W, RS, and the data bus lines; when this pin is set to HIGH, the LCD is processing the incoming data.

D0-D7 (Data Bus) are the pins that carries the 8 bit data we send to the display. For example, if we want to see the uppercase 'A' character on the display we will set these pins to 0100 0001(according to the ASCII table) to the LCD.

A-K (Anode & Cathode) pins are used to control the backlight of the LCD.

Testing Character LCD

Now we're onto the interesting stuff. Let's test your LCD.

First, connect the 5V and GND pins from the Arduino Uno to the breadboard power rails and get your LCD plugged into the breadboard.

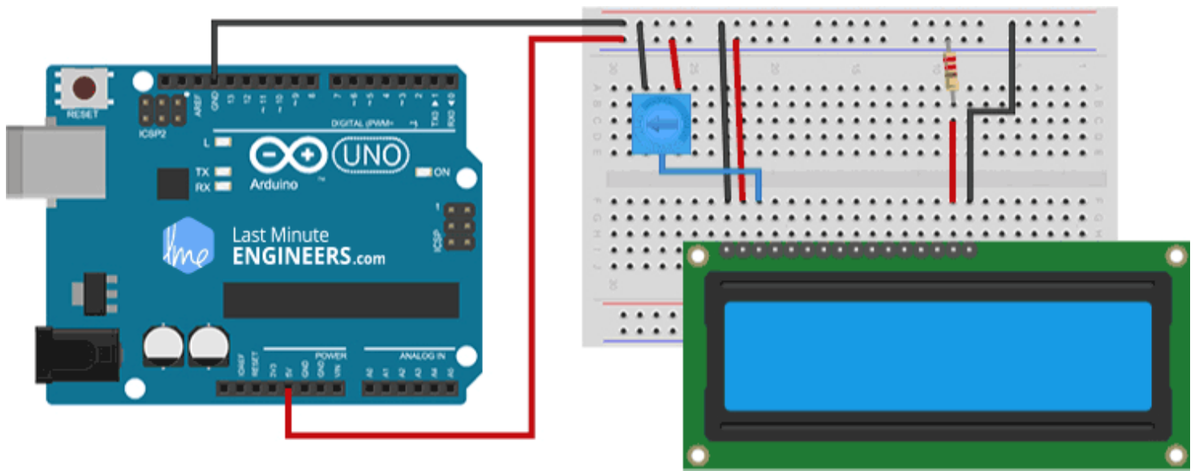
Now we'll power up the LCD. The LCD has two separate power connections; One (Pin 1 and Pin 2) for the LCD itself and another one (Pin 15 and Pin 16) for the LCD backlight. Connect pins 1 and 16 on the LCD to GND and pins 2 and 15 on the LCD to 5V.

> Vast majority of LCDs have a built-in series resistor for the LED backlight. If you happen to have a LCD that does not include a resistor, you'll need to add one between 5V and pin 15.
>
> To calculate the value of the series resistor, look up the maximum backlight current and the typical backlight voltage drop from the data sheet. And using simple ohm's law you

can calculate the resistor value.

If you can't find the data sheet, then it should be safe to use a 220 ohm resistor, however a value this high may make the backlight slightly dim.

Next we'll make connections for Pin 3 on the LCD which controls the contrast and brightness of the display. In order for fine contrast adjustments we'll connect a 10K potentiometer between 5V and GND; connect the center pin (wiper) of the potentiometer to pin 3 on the LCD.



Adjusting LCD contrast by rotating potentiometer knob

That's it! Now turn on the Arduino, you'll see the backlight light up. And as you rotate the knob on the potentiometer, you should notice the first line of rectangles appear. If this happens, Congratulations! Your LCD is doing just fine.

Wiring – Connecting 16×2 Character LCD with Arduino Uno

Before we get to uploading code and sending data to the display, let's hook the LCD up to the Arduino.

The LCD has a lot of pins (16 pins in total) that we'll show you how to wire up. But, the good news is that not all these pins are necessary for us to connect to the Arduino.

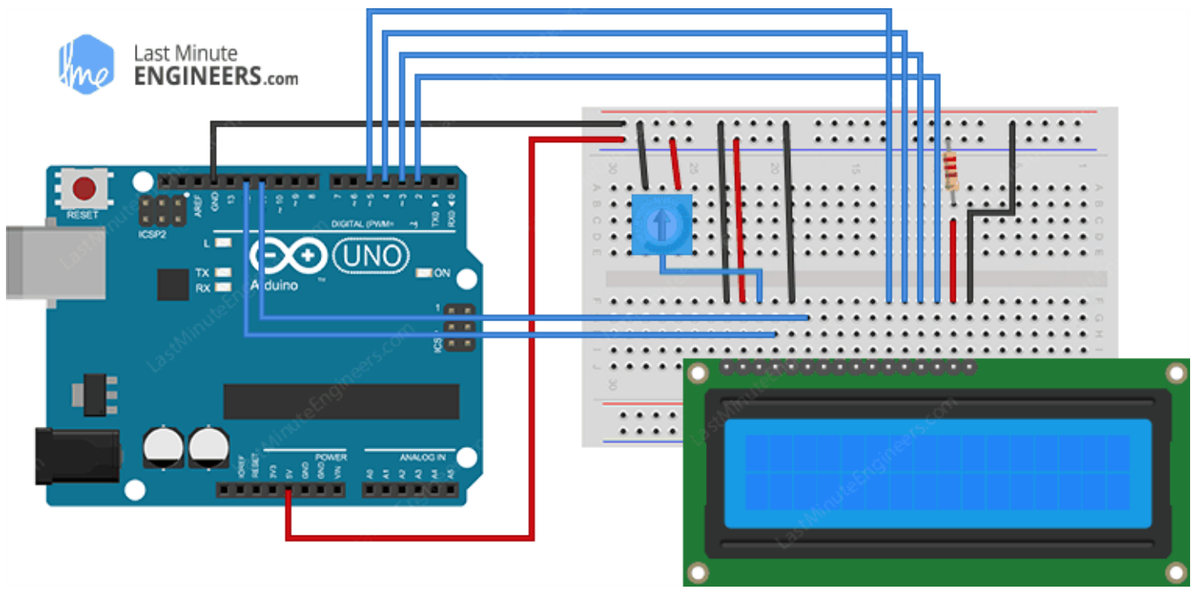
We know that there are 8 Data lines that carry raw data to the display. But, HD44780 LCDs are designed in a way that we can talk to the LCD using only 4 data pins(4-bit mode) instead of 8(8-bit mode). This saves us 4 pins!

> **Difference between 4-bit and 8-bit mode**
>
> It's faster to use 8-bit mode as it takes half as long to use 4-bit mode. Because in 8-bit mode you write the data in just one go. However, in 4-bit mode you have to split a byte in 2 nibbles, shift one of them 4 bits to the right, and perform 2 write operations.
>
> So, The 4-bit mode is often used to save I/O pins. But, 8-bit mode is best used when speed is required in an application and at least 10 I/O pins are available.

So to recap, we will be interfacing LCD using 4-bit mode and hence we need only 6 pins: RS, EN, D7, D6, D5, and D4 to talk to the LCD.

Now, let's connect the LCD Display to the Arduino. Four data pins (D4-D7) from the LCD will be connected to Arduino's digital pins from #4-7. The Enable pin on LCD will be connected to Arduino #2 and the RS pin on LCD will be connected to Arduino #1.

The following diagram shows you how to wire everything.



Wiring connections of 16×2 character LCD and Arduino UNO

With that, you're now ready to upload some code and get the display printing.

Arduino Code

The following test sketch will print 'Hello World!' message on the LCD. Try the sketch out and then we will dissect it in some detail.

```
// include the library code:
#include <LiquidCrystal.h>

// Creates an LCD object. Parameters: (rs, enable, d4, d5, d6, d7)
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

void setup()
{
    // set up the LCD's number of columns and rows:
    lcd.begin(16, 2);

    // Clears the LCD screen
    lcd.clear();
}

void loop()
{
    // Print a message to the LCD.
    lcd.print(" Hello world!");

    // set the cursor to column 0, line 1
    // (note: line 1 is the second row, since counting begins with 0):
    lcd.setCursor(0, 1);
    // Print a message to the LCD.
    lcd.print(" LCD Tutorial");
}
```

If everything goes right, you should see something like this on the display.

Code Explanation:

The sketch starts by including LiquidCrystal library. As mentioned earlier in this tutorial, the Arduino community has a library called LiquidCrystal that makes programming the LCD module less difficult. You can explore more about the library on Arduino's official website.

```
// include the library code:
#include <LiquidCrystal.h>
```

Next we have to create an LiquidCrystal object. This object uses 6 parameters and specifies which Arduino pins are connected to the LCD's RS pin, Enable pin, and data pins: d4, d5, d6, and d7.

```
// Creates an LCD object. Parameters: (rs, enable, d4, d5, d6, d7)
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
```

Now that you have declared a LiquidCrystal object, you can access special methods (aka functions) that are specific to the LCD.

In the 'setup' function: we will use two functions: The first function is `begin()` . This is used to specify the dimensions of the display i.e. how many columns and rows the display has. If you are using 16×2 character LCD, pass the parameters 16 & 2; If you are using 20×4 LCD, pass the

parameters 20 & 4. You got the point!

The second function is `clear()` .This clears the LCD screen and moves the cursor to the upper left corner.

```
lcd.begin(16, 2);
lcd.clear();
```

In the 'loop' function: we will use the `print()` function which displays the message that we see on the first line of the screen.

```
// Print a message to the LCD.
lcd.print(" Hello world!");
```

Following that we will set the cursor position to second row, by calling function `setCursor()` The cursor position specifies the location where you need the new text to be displayed on the LCD. The upper left corner is considered col=0, row=0

```
lcd.setCursor(0, 1);
lcd.print(" LCD Tutorial");
```

Other useful functions in LiquidCrystal Library

There are a few useful functions you can use with **LiquidCrystal** object. Few of them are listed below:

- If you just want to position the cursor in the upper-left of the LCD without clearing the display, use `home()`
- There are many applications like turbo C++ compiler or notepad++, in which pressing 'insert' key on the keyboard changes cursor. Just like that you can change the cursor on the LCD using `blink()` or `lcd.cursor()` .

- `blink()` function displays the blinking block of 5×8 pixels, while `lcd.cursor()` displays an underscore (line) at the position to which the next character will be written.
- You can use the `noBlink()` function to turns off the blinking LCD cursor and `lcd.noCursor()` to hide the LCD cursor.
- You can scroll the contents of the display one space to the right using `lcd.scrollDisplayRight()` or one space left using `lcd.scrollDisplayLeft()` . If you want to scroll the text continuously, you need to use these functions inside a 'for loop'.

Custom character generation for 16×2 character LCD

If you are finding characters on the display dull and unexciting, you can create your own custom characters (glyph) and symbols for your LCD. They are extremely useful when you want to display a character that is not part of the standard ASCII character set.

CGROM and CGRAM

All LCD displays based on Hitachi HD44780 controller have two types of memories that store defined characters called CGROM and CGRAM (Character Generator ROM & RAM). CGROM memory is non-volatile and can't be modified whereas; CGRAM memory is volatile and can be modified any time.

CGROM is used for storing all permanent fonts that can be displayed by using their ASCII code. For example, if we write 0x41 then on the display we get character 'A'. CGRAM is another memory that can be used for storing user defined characters.

This RAM is limited to 64 bytes. Meaning, for 5×8 pixel based LCD; up to 8 user-defined characters can be stored in the CGRAM. And for 5×10 pixel based LCD, only 4 user-defined characters are can be stored.

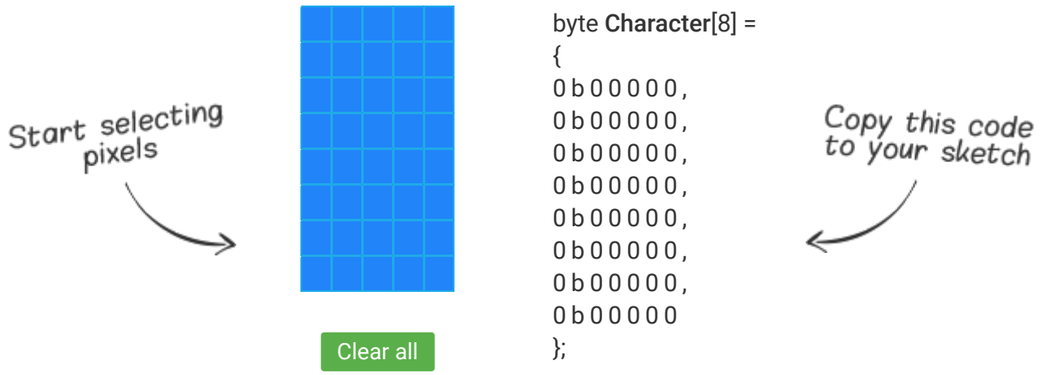
As we discussed earlier in this tutorial that a character on the display is formed in a 5×8 matrix of pixels so you need to define your custom character within that matrix. To define the character you'll use the createChar() function of the LiquidCrystal library.

To use `createChar()` you first set up an array of 8 bytes. Each byte (only 5 bits are considered) in the array defines one row of the character in the 5×8 matrix. Whereas, the zeros and ones in

the byte indicate which pixels in the row should be on and which ones should be off.

Custom Character Generator

Creating custom character was not easy until now! We have created a small application called **Custom character generator for character LCD**. Can you see the blue grid below? That's it. That's the application. You can click on any of the 5×8 pixels to set/clear that particular pixel. And as you click on pixels, the code for the character is generated next to the grid. This code can directly be used in your Arduino sketch.



Your imagination is limitless. The only limitation is that the LiquidCrystal library supports only eight custom characters. But don't get disappointed, look at the bright side, at least we have eight characters.

The following sketch demonstrates how you can use these custom characters on the display.

```
// include the library code:
#include <LiquidCrystal.h>

// initialize the library with the numbers of the interface pins
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
```

```
// make some custom characters:
byte Heart[8] = {
0b00000,
0b01010,
0b11111,
0b11111,
0b01110,
0b00100,
0b00000,
0b00000
};

byte Bell[8] = {
0b00100,
0b01110,
0b01110,
0b01110,
0b11111,
0b00000,
```

After including the library, we need initialize the custom character array of eight bytes.

```
byte Heart[8] = {
0b00000,
0b01010,
0b11111,
0b11111,
0b01110,
0b00100,
0b00000,
0b00000
};
```

In the setup we have to create the custom character using the `createChar()` function. This function takes two parameters. The first one is a number between 0 and 7 in order to reserve one of the 8 supported custom characters. The second parameter is the name of the array of bytes.

```
// create a new character
```

```
lcd.createChar(0, Heart);
```

Next in the loop, to display the custom character we use `write()` function and as a parameter we use the number of the character we reserved.

```
// byte(0) represents Heart character.
lcd.write(byte(0));
```