
Learn Arduino – LCD Interfacing



A Comprehensive Guide on LCD interfacing to Arduino Board with different code examples.

Interfacing with Liquid Crystal Displays V1.0

Table of Contents

Solution1:Setting Up the LCD.....	4
1.1: Parts You'll Need for This Chapter:	4
1.2: Using the LiquidCrystal Library to Write to the LCD	7
1.3: Adding Text to the Display	8
1.4: Creating Special Characters and Animations	9
Solution 2: Temperature Control	13
2.1: Building a Personal Thermostat	13
2.2: Setting Up the Hardware	13
2.3: Displaying Data on the LCD	14
2.4: Adjusting the Set Point with a Button.....	16
2.5: Adding an Audible Warning and a Fan.....	17
2.6: Bringing It All Together: The Complete Program	18
Solution 3: Display Text and Character	21
3.1: String variables: String type vs. char type	21
3.2: Parallel character LCDs: the Hitachi HD44780	23
3.3: 4-bit or 8-bit?	23
3.4: Library and functions	23
3.5: Circuit diagram	24
3.6: Connecting everything up in 4-bit mode	24
3.7: Sketch for writing to the Hitachi HD44780	26
3.8: Upload and test.....	27
Solution 4:UsingLCD+Keypad Shield for Arduino.....	28
4.1: How to use 16×2 Character LCD + 6-buttons Keypad Shield ?	28
4.2: Pins Assignment	29
4.3: Application Ideas.....	29
Solution 5:Arduino Code Examples withLCD+Keypad Shield.....	32
5.1: Formatting Text.....	32
5.2: Displaying Special Symbols	35
5.3: Creating Custom Characters	38
5.4: Displaying Symbols Larger Than a Single Character.....	39
Solution 6: Using an Arduino as an LCD clock.....	42
6.1: The Components	43
6.2: Connecting It All Together	43

6.2: LCD Connection where Pin 1 on LCD is far left	43
6.3: Wiring up the RTC Module:.....	44
6.4: Checking It Works	45
6.5: Setting the Time in the RTC.....	45
6.6: Uploading the Time Sketch	47
7. Appendix	49
7.1: LCD Application Manual.....	49
7.2: HandsOn Technology Products Quality Commitments	49

Solution1:Setting Up the LCD

1.1: Parts You'll Need for This Chapter:

- Arduino Uno
- USB cable (A to B for Uno)
- Speaker
- Pushbuttons (n2)
- Small DC fan
- 16x2 character LCD
- 4.7k Ω resistors (n2)
- 10k Ω resistors (n2)
- 150 Ω resistor
- 10k Ω potentiometer
- TC74A0-5.0VAT I2C temperature sensor
- Jumper wires
- Breadboard

In this chapter, you learn how to connect an LCD to your Arduino, and you learn how to use the *ArduinoLiquidCrystal library* to write text and arbitrary custom characters to your LCD. After you have the basics down, you add some components to make a simple thermostat capable of obtaining local temperature data, reporting it to you, and controlling a fan to compensate for heat. An LCD will give you live information, a speaker will alert you when the temperature is getting too hot, and the fan will turn on to automatically cool you down.

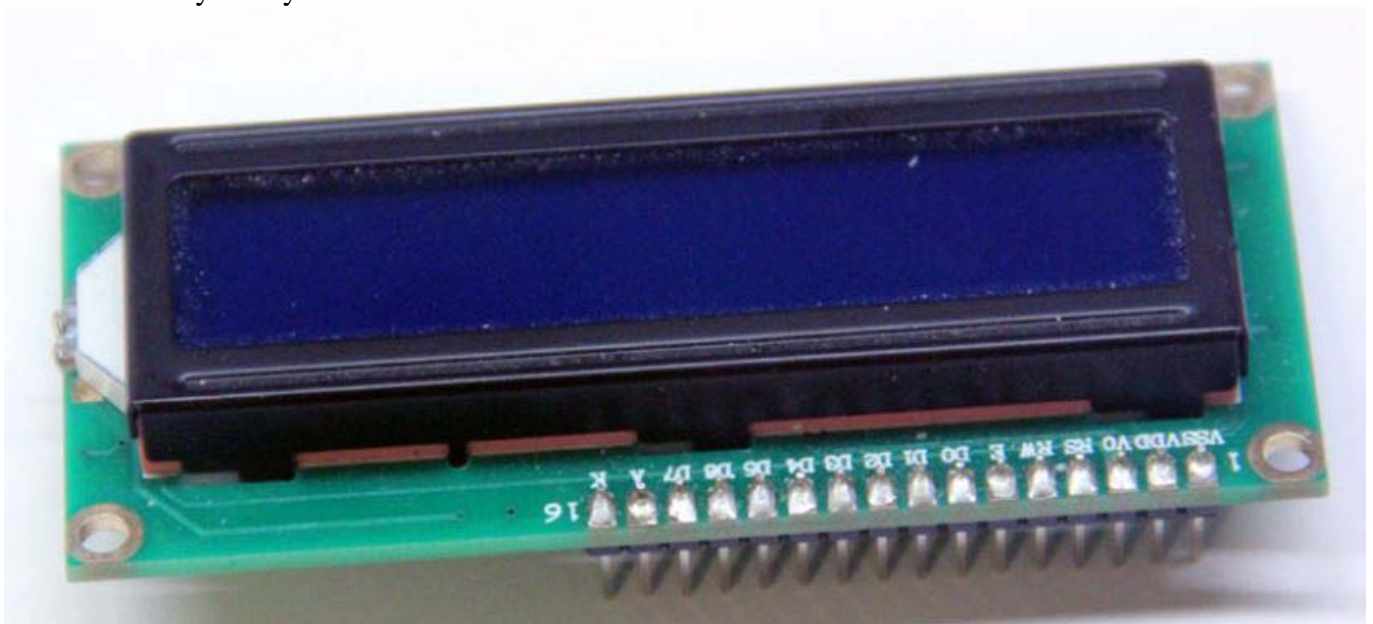


Figure 1-1: LCD with Headers soldered on

To complete the examples in this chapter, we use a parallel LCD screen. These are extremely common and come in all kinds of shapes and sizes. The most common is a 16 \times 2 character display with a single row of 16 pins (14 if it does not have a backlight). In this chapter, we use a 16-pin LCD display that can show a total of 32 characters (16 columns and 2 rows). If your display didn't come with a 16-pin header already soldered on, you need to solder one on so that you can easily install it in your breadboard. With the header successfully soldered on, your LCD should look like the one shown in Figure 1-1, and you can insert it into your breadboard. Refer to Figure 1-2 board layout for complete pins assignment.

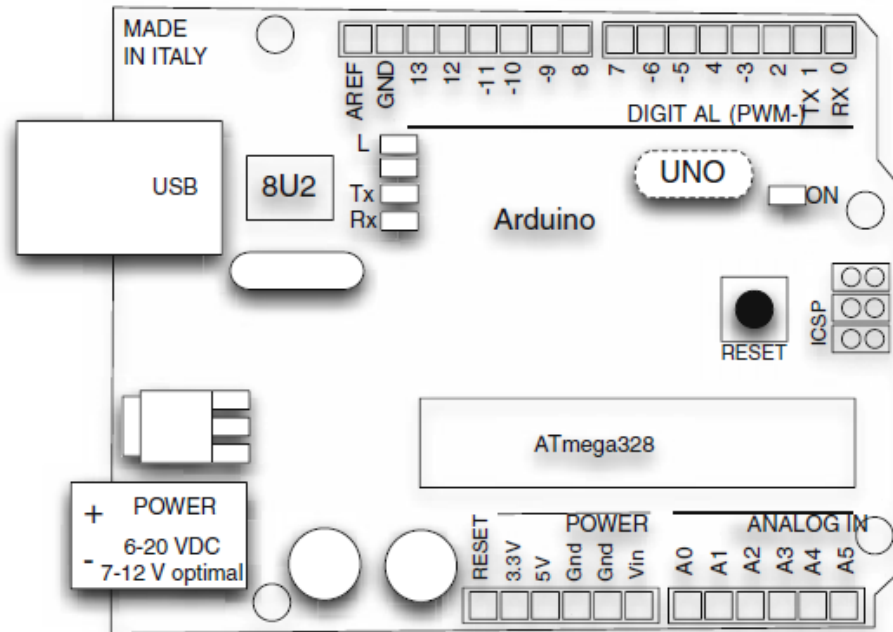


Fig 1.2: Figure 1.2 Board layout and pins of the Arduino Uno

Next, you wire up your LCD to a breadboard and to your Arduino. All of these parallel LCD modules have the same pin-out and can be wired in one of two modes: 4-pin or 8-pin mode. You can accomplish everything you might want to do using just 4 pins for communication; that's how you'll wire it up. There are also pins for enabling the display, setting the display to command mode or character mode, and for setting it to read/write mode. Table 1-1 describes all of these pins.

PIN NUMBER	PIN NAME	PIN PURPOSE
1	VSS	Ground connection
2	VDD	+5V connection power supply
3	VO	Contrast adjustment (to potentiometer)
4	RS	Register selection (Character vs. Command)
5	RW	Read/write
6	EN	Enable
7	D0	Data line 0
8	D1	Data line 1
9	D2	Data line 2
10	D3	Data line 3
11	D4	Data line 4
12	D5	Data line 5
13	D6	Data line 6
14	D7	Data line 7
15	A	Backlight anode
16	K	Backlight cathode

Table 1-1: Parallel LCD Pins assignment

Here's a breakdown of the pin connections:

- The contrast adjustment pin changes how dark the display is. It connects to the center pin of a potentiometer.
- The register selection pin sets the LCD to command or character mode, so it knows how to interpret the next set of data that is transmitted via the data lines. Based on the state of this pin, data sent to the LCD is either interpreted as a command (for example, move the cursor) or characters (for example, the letter a).
- The RW pin is always tied to ground in this implementation, meaning that you are only writing to the display and never reading from it.
- The EN pin is used to tell the LCD when data is ready.
- Data pins 4~7 are used for actually transmitting data, and data pins 0~3 are left unconnected.
- You can illuminate the backlight by connecting the anode pin to 5V and the cathode pin to ground if you are using an LCD with a built-in resistor for the backlight. If you are not, you must put a current-limiting resistor in-line with the anode or cathode pin. The datasheet for your device will generally tell you if you need to do this.

We can connect the communication pins of the LCD to any I/O pins on the Arduino. In this chapter, they are connected as shown in Table 1-2.

LCD PIN	ARDUINO Uno PIN NUMBER
RS	Pin 2
EN	Pin 3
D4	Pin 4
D5	Pin 5
D6	Pin 6
D7	Pin 7

Table 1-2: Communication Pins Connections

Reference the wiring diagram shown in Figure 1-3 and hook up your LCD accordingly.

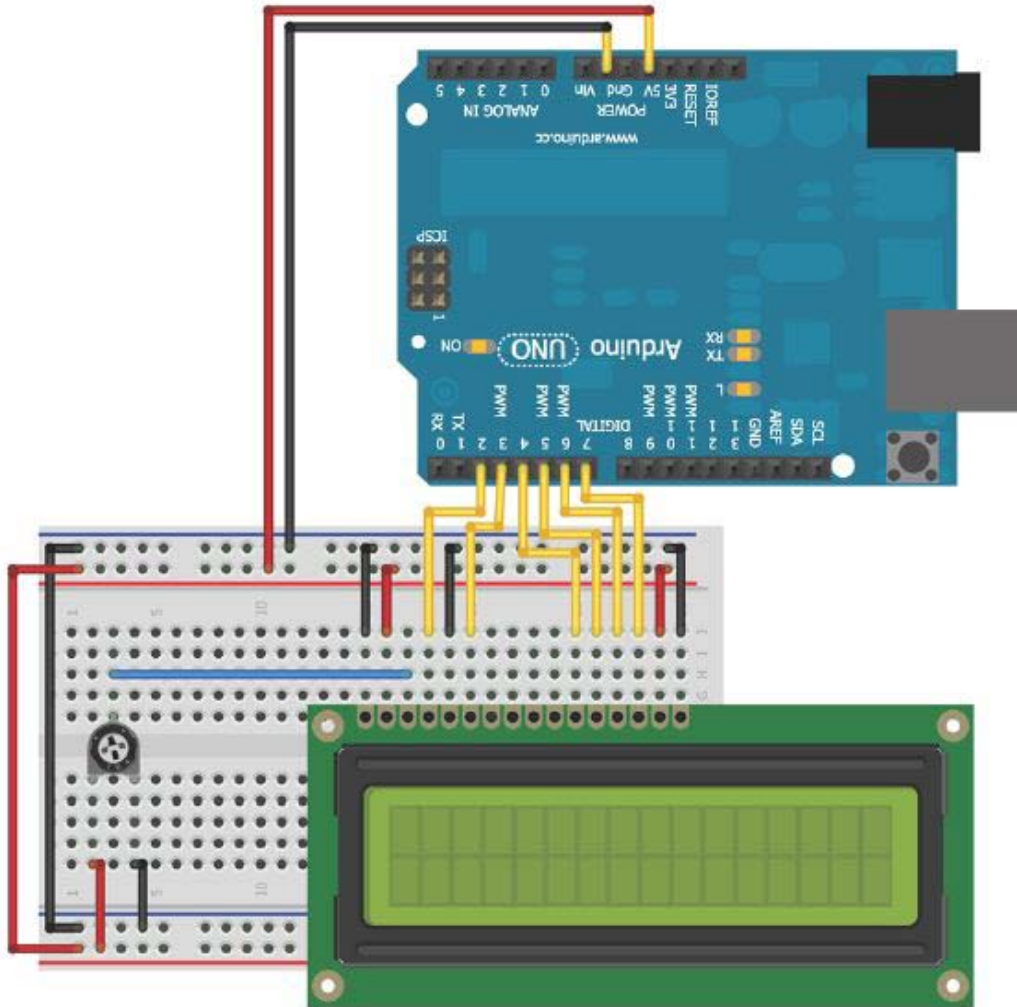


Figure 1-3: LCD wired to breadboard and Arduino

Now your LCD is ready for action! Once you get the code loaded in the next section, you can start displaying text on the screen. The potentiometer will adjust the contrast between the text and the background color of the screen.

1.2: Using the `LiquidCrystal` Library to Write to the LCD

The Arduino IDE includes the *LiquidCrystal library*, a set of functions that makes it very easy to interface with the parallel LCD that you are using. The *LiquidCrystal library* has an impressive amount of functionality, including blinking the cursor, automatically scrolling text, creating custom characters, and changing the direction of text printing. This chapter does not cover every function, but instead gives you the tools you need to understand to interface with the display using the most important functions. You can find descriptions of the library functions and examples illustrating their use on the Arduino website:

<https://www.arduino.cc/en/Reference/LiquidCrystal>

1.3: Adding Text to the Display

In this first example, we add some text and an incrementing number to the display. This exercise demonstrates how to initialize the display, how to write text, and how to move the cursor. First, include the *LiquidCrystal* library:

```
#include <LiquidCrystal.h>
```

Then, initialize an LCD object, as follows:

```
LiquidCrystal lcd (2, 3, 4, 5, 6, 7);
```

The arguments for the LCD initialization represent the Arduino pins connected to RS, EN, D4, D5, D6, and D7, in that order. In the setup, you call the library's *begin()* function to set up the LCD display with the character size.

(The one we are using is a 16×2 display, but you might be using another size, such as a 20×4). The arguments for this command represent the number of columns and the number of rows, respectively:

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

After doing that, you can call the library's *print()* and *setCursor()* commands to print text to a given location on the display. For example, if you want to print your name on the second line, you issue these commands:

```
lcd.setCursor(0, 1);  
lcd.print("Mr.Arduino Uno");
```

The positions on the screen are indexed starting with (0,0) in the top-left position. The first argument of *setCursor()* specifies which column number, and the second specifies which row number. By default, the starting location is (0,0). So, if you call *print()* without first changing the cursor location, the text starts in the top-left corner.

WARNING: The library does not check for strings that are too long. So, if you try to print a string starting at position 0 that is longer than the number of characters in the row you are addressing, you might notice strange behavior. Make sure to check that whatever you are printing will fit on the display!

Using this knowledge, we can now write a simple program that displays some text on the first row and that prints a counter that increments once every second on the second row. Listing 3-1 shows the complete program to accomplish this. Load it on to your Arduino and confirm that it works as expected. If you don't see anything, adjust the contrast with the potentiometer.

There are more ways of connecting the LCD Module to Arduino. In above example, we implemented the simplest of all versions. It requires the minimum amount of cables. However, there are four types in total:

```
LiquidCrystal(rs, enable, d4, d5, d6, d7)  
LiquidCrystal(rs,rw, enable, d4, d5, d6, d7)  
LiquidCrystal(rs, enable, d0, d1, d2, d3, d4, d5, d6, d7)  
LiquidCrystal(rs,rw, enable, d0, d1, d2, d3, d4, d5, d6, d7)
```

Having access to the RW pin will allow us to read from the monitor, which is not really useful as we are the ones who also write on it. Usually, this pin is just connected to GND to make it stay in receive mode.

Using all eight data lines is not necessary but is twice as fast as only using four. There are also some I2C-based LCD displays that only require the two I2C pins. If we are tight on pins, that is the solution.

Listing 1-1: LCD Text with an Incrementing Number—LCD_text.ino

```
/*
   This program is free software: you can redistribute it and/or modify
   it under the terms of the GNU General Public License v3 as published by
   the Free Software Foundation.
   */

//LCD text with incrementing number
//Include the library code
#include <LiquidCrystal.h>

//Start the time at 0
int time =0;

//Initialize the library with the numbers of the interface pins
LiquidCrystal lcd(2,3,4,5,6,7);

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

    // Print a message to the LCD.
    lcd.print("Mr. Arduino Uno");
}

void loop()
{
    //Move cursor to second line, first position
    lcd.setCursor(0,1);
    //Print Current Time
    lcd.print(time);
    //Wait 1 second
    delay(1000);
    //Increment the time
    time++;
}
```

This program combines all the steps that you learned about earlier. The library is first included at the top of the program. A time variable is initialized to 0, so that it can be incremented once per second during the *loop()*. A *LiquidCrystal* object called *lcd* is created with the proper pins assigned based on the circuit we've already wired up. In the setup, the LCD is configured as having 16 columns and 2 rows, by calling *lcd.begin(16,2)*. Because the first line never changes, it can be written in the setup. This is accomplished with a call to *lcd.print()*. Note that the cursor position does not need to be set first, because we want the text to be printed to position (0,0), which is already the default starting location. In the loop, the cursor is always set back to position (0,1) so that the number we print every second overwrites the previous number. The display updates once per second with the incremented *time* value.

1.4: Creating Special Characters and Animations

What if we want to display information that cannot be expressed using normal text? Maybe we want to add a Greek letter, a degree sign, or some progress bars. Thankfully, the *LiquidCrystal* library supports the definition of custom characters that can be written to the display. In the next example, we use this

capability to make an animated progress bar that scrolls across the display. After that, we take advantage of custom characters to add a degree sign when measuring and displaying temperature.

Creating a custom character is pretty straightforward. If we take a close look at the LCD, you'll see that each character block is actually made up of a 5×8 grid of pixels. To create a custom character, we simply have to define the value of each of these pixels and send that information to the display. To try this out, we make a series of characters that will fill the second row of the display with an animated progress bar. Because each character space is 5 pixels wide, there will be a total of five custom characters: one with one column filled, one with two columns filled, and so on.

At the top of our sketch where we want to use the custom characters, create a byte array with 1s representing pixels that will be turned on and with 0s representing pixels that will be turned off. The byte array representing the character that fills the first column (or the first 20% of the character) looks like this:

```
byte p20[8] = {  
  B10000,  
  B10000,  
  B10000,  
  B10000,  
  B10000,  
  B10000,  
  B10000,  
  B10000,  
  B10000,  
};
```

We chose to call this byte array `p20`, to represent that it is filling 20 percent of one character block (the `p` stands for percent). In the `setup()` function, call the `createChar()` function to assign our byte array to a custom character ID. Custom character IDs start at 0 and go up to 7, so we can have a total of eight custom characters. To map the 20% character byte array to custom character 0, type the following within our `setup()` function:

```
lcd.createChar(0, p20);
```

When we're ready to write a custom character to the display, place the cursor in the right location and use the library's `write()` function with the ID number:

```
lcd.write((byte)0);
```

In the preceding line, `(byte)` casts, or changes, the 0 to a byte value. This is necessary only when writing character ID 0 directly (without a variable that is defined to 0), to prevent the Arduino compiler from throwing an error caused by the variable type being ambiguous. Try removing the "byte cast" and observe the error that the Arduino IDE displays. We can write other character IDs without it, like this:

```
lcd.write(1);
```

Putting this all together, we can add the rest of the characters and put two nested `for()` loops in our program loop to handle updating the progress bar.

The completed code looks like the code shown in Listing 1-2.

Listing 1-2: LCD Progressing Bar-graph

```
/*=====
// Author      : Handson Technology
// Project     : Arduino Uno
// Description  : LCD with Progress Bar
// LiquidCrystal Library - Special Chars
// Source-Code : bar\_graph.ino
//=====
*/

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

//Initialize the library with the numbers of the interface pins
LiquidCrystallcd(8,9,4,5,6,7);

//Create the progress bar characters
byte p20[8]={
  B10000,
  B10000,
  B10000,
  B10000,
  B10000,
  B10000,
  B10000,
  B10000,
};
byte p40[8]={
  B11000,
  B11000,
  B11000,
  B11000,
  B11000,
  B11000,
  B11000,
  B11000,
};
byte p60[8]={
  B11100,
  B11100,
  B11100,
  B11100,
  B11100,
  B11100,
  B11100,
  B11100,
};
byte p80[8]={
  B11110,
  B11110,
  B11110,
  B11110,
  B11110,
  B11110,
  B11110,
  B11110,
};
byte p100[8]={
  B11111,
  B11111,
  B11111,
  B11111,
};
```

```

    B11111,
    B11111,
    B11111,
    B11111,
};

void setup()
{
  //Set up the LCDs number of columns and rows:
  lcd.begin(16,2);

  // Print a message to the LCD.
  lcd.print("  HandsOn Tech  ");

  //Make progress characters
  lcd.createChar(0, p20);
  lcd.createChar(1, p40);
  lcd.createChar(2, p60);
  lcd.createChar(3, p80);
  lcd.createChar(4, p100);
}

void loop()
{
  //Move cursor to second line
  lcd.setCursor(0,1);
  //Clear the line each time it reaches the end
  //with 16 " " (spaces)
  lcd.print("                ");

  //Iterate through each character on the second line
  for(int i=0;i<16;i++)
  {
    //Iterate through each progress value for each character
    for(int j=0; j<5;j++)
    {
      lcd.setCursor(i,1); //Move the cursor to this location
      lcd.write(j); //update progress bar
      delay(60); //wait
    }
  }
}

```

At the beginning of each pass through the loop, the 16-character-long string of spaces is written to the display, clearing the progress bar before it starts again. The outer *for()* loop iterates through all 16 positions. At each character position, the inner *for()* loop keeps the cursor there and writes an incrementing progress bar custom character to that location. The byte cast is not required here because the ID 0 is defined by the *j* variable in the *for()* loop.

When successfully compiled and load into Arduino board, the LCD display will look similar as shown in Figure 1-4.



Figure 1-4: Display running the Listing 1.2 code.

Solution 2: Temperature Control

2.1: Building a Personal Thermostat

Now, let's make this display a bit more useful. To do so, we add the temperature sensor. The display shows the temperature and the current fan state. When it gets too hot, the speaker makes a noise to alert you, and the fan turns on. When it gets sufficiently cool again, the fan turns off. Using two push buttons and the de-bounce code, we add the ability to increment or decrement the desired temperature.

2.2: Setting Up the Hardware

A low-power DC fan hooked directly to a 5V I/O pin will suffice to show that it spins when it should. It will be accelerating slowly enough that you don't need to worry too much about inductive spikes. The two buttons have one side connected to power; the other side is connected to ground through 10k Ω pull-down resistors and to the Arduino. The speaker is connected to an I/O pin through a 150 Ω resistor and to ground. The frequency of the sound will be set in the program.

Placing it in front of the LCD's contrast potentiometer allows us to conserve some breadboard space and to fit everything onto the same half-size breadboard that we've been using so far. The diagram in Figure 2-1 shows the complete wiring setup with everything you need to create this project. The symbol for the TC74 temperature sensor has been made partially transparent so that you can see the potentiometer behind it.

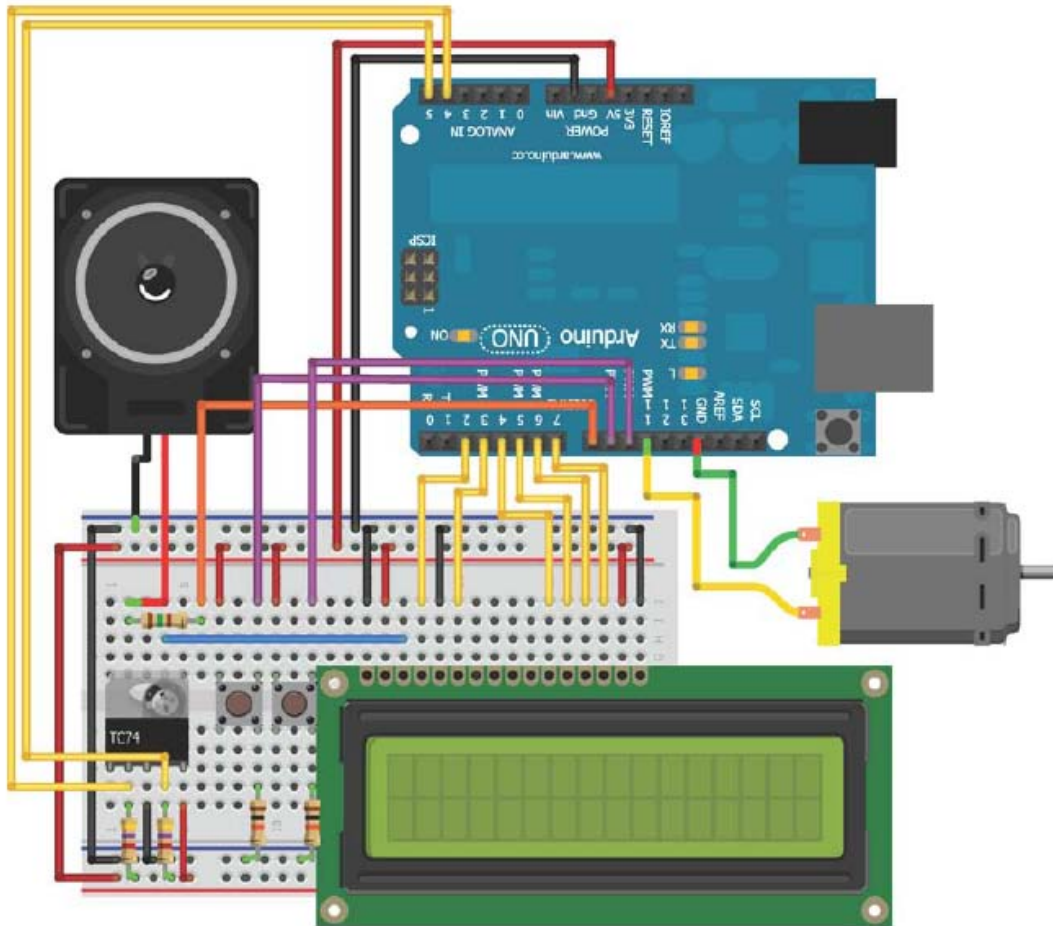


Figure 2-1: LCD thermostat system

2.3: Displaying Data on the LCD

Having some parameters in place before hand makes writing information to the LCD screen easier. First, use degrees Celsius for the display, and second, assume that we'll always be showing two digits for the temperature. Once the software is running, the LCD display will look something like Figure 2.2.



Figure 2-2: LCD display

The "Current:" and "Set:" strings are static; they can be written to the screen once at the beginning and left there. Similarly, because the temperatures are assumed to be two digits, we can statically place both "°C" strings into the correct locations. The current reading will be displayed in position (8,0) and will be updated on every run through the *loop()*. The desired, or set, temperature will be placed in position (8,1)

and updated every time a button is used to adjust its value. The fan indicator in the lower right of the display will be at position (15,1). It should update to reflect the fan's state every time it changes.

The degree symbol, fan off indicator, and fan on indicator are not part of the LCD character set. Before using them in our sketch, we need to create them as type arrays at the beginning of our program, as shown in the following snippet.

```
//Custom degree character
byte degree[8]={
B00110,
B01001,
B01001,
B00110,
B00000,
B00000,
B00000,
B00000,
};

//Custom "fan on" indicator
byte fan_on[8]={
B00100,
B10101,
B01110,
B11111,
B01110,
B10101,
B00100,
B00000,
};

//Custom "fan off" indicator
byte fan_off[8]={
B00100,
B00100,
B00100,
B11111,
B00100,
B00100,
B00100,
B00000,
};
```

Writing these characters will be done in *setup()*. Move the cursor to the right locations, and with the LCD library's *write()* and *print()* functions, update the screen, as shown in the following snippet.

```
//Make custom characters
lcd.createChar(0, degree);
lcd.createChar(1, fan_off);
lcd.createChar(2, fan_on);

//Print a static message to the LCD
lcd.setCursor(0,0);
lcd.print("Current:");
lcd.setCursor(10,0);
lcd.write((byte)0);
lcd.setCursor(11,0);
lcd.print("C");
lcd.setCursor(0,1);
lcd.print("Set:");
lcd.setCursor(10,1);
lcd.write((byte)0);
lcd.setCursor(11,1);
```



```

lcd.print("C");
lcd.setCursor(15,1);
lcd.write(1);

```

You also update the fan indicator and temperature values each time through *loop()*. You need to move the cursor to the right location each time before you update these characters.

2.4: Adjusting the Set Point with a Button

We used a *debounce()* function for key press. One button will increase the set point, and the other will decrease it. We need to define variables for holding the previous and current button states:

```

//Variables for debouncing
booleanlastDownTempButton= LOW;
booleancurrentDownTempButton= LOW;
booleanlastUpTempButton= LOW;
booleancurrentUpTempButton= LOW;

```

We can write the *debounce()* function to support multiple buttons :

```

//A debouncing function that can be used by both buttons

booleandebounce(booleanlast,int pin)
{
    boolean current =digitalRead(pin);
    if(last != current)
    {
        delay(5);
        current=digitalRead(pin);
    }
    return current;
}

```

In *loop()*, we want to check both buttons using the *debounce()* function, change the *set_temp* variable as needed, and update the set value that is displayed on the LCD:

```

//Debounce both buttons
currentDownTempButton=debounce(lastDownTempButton, DOWN_BUTTON);
currentUpTempButton=debounce(lastUpTempButton, UP_BUTTON);

//Turn down the set temp
if(lastDownTempButton== LOW &&currentDownTempButton== HIGH)
{
    set_temp--;
}

//Turn up the set temp
elseif(lastUpTempButton== LOW &&currentUpTempButton== HIGH)
{
    set_temp++;
}

//Print the set temp
lcd.setCursor(8,1);
lcd.print(set_temp);

//Update the button state with the current
lastDownTempButton=currentDownTempButton;
lastUpTempButton=currentUpTempButton;

```

The preceding code snippet first runs the *debounce()* function for each button, and then adjusts the set temperature variable if one of the buttons has been pressed. Afterward, the temperature displayed on the LCD is updated, as are the button state variables.

2.5: Adding an Audible Warning and a Fan

In this section, we add code to control the fan and the speaker. Although the LCD showing us live information is nice, we'll often find it useful to have an additional form of feedback to tell us when something is happening. For example, the speaker beeps when the fan turns on. In this example, we use *tone()* paired with *delay()* and a *noTone()* command. We could instead add a duration argument to *tone()* to determine the duration of the sound. We want to make sure that the tone plays only one time so (and does not beep forever when above the set temperature).

Using a state variable, we can detect when the speaker has beeped and thus keep it from beeping again until after the temperature dips below the set temperature and resets the state variable. When the fan turns on, an indicator changes on the LCD (represented by the custom character we defined at the top of the program). The following code snippet checks the temperature and controls the speaker, the fan indicator on the LCD, and the fan:

```
//It's too hot!

if(c >=set_temp)
{
    //So that the speaker will only beep one time...

    if(!one_time)
    {
        tone(SPEAKER,400);
        delay(500);
        one_time= true;
    }
    //Turn off the speaker if it's done
    else
    {
        noTone(SPEAKER);
    }
    //Turn the fan on and update display

    digitalWrite(FAN, HIGH);
    lcd.setCursor(15,1);
    lcd.write(2);
}
//It's not too hot!
else
{
    //Make sure the speaker is off, reset the "one beep" variable
    //Update the fan state, and LCD display

    noTone(SPEAKER);
    one_time= false;
    digitalWrite(FAN, LOW);
    lcd.setCursor(15,1);
    lcd.write(1);
}
}
```

The *one_time* variable is used to make sure that the beep plays only one time instead of continuously. Once the speaker has beeped for 500ms at 400Hz, the variable is set to true and is reset to false only when the temperature drops back below the desired temperature.

2.6: Bringing It All Together: The Complete Program

It's time to bring all the parts together into a cohesive whole. We need to make sure that you include the appropriate libraries, define the pins, and initialize the state variables at the top of the sketch. Listing 2-1 shows the complete program. Load it on to Arduino and observe the result.

Listing 2-1: Personal Thermostat Program—LCD_thermostat.ino

```
/*=====
// Author      : Handson Technology
// Project     : Arduino Uno
// Description : Building Thermostat with LCD
// LiquidCrystal Library
// Download the complete source code: LCD\_Thermostat
//=====
*/

//Keep yourself cool! This is a thermostat.
//This assumes temperatures are always two digits

//Include Wire I2C library and set the address
#include <Wire.h>
#define TEMP_ADDR 72

//Include the LCD library and initialize:
#include <LiquidCrystal.h>
LiquidCrystal lcd(2,3,4,5,6,7);

//Custom degree character
byte degree[8]={
    B00110,
    B01001,
    B01001,
    B00110,
    B00000,
    B00000,
    B00000,
    B00000,
};

//Custom "Fan On" indicator
byte fan_on[8]={
    B00100,
    B10101,
    B01110,
    B11111,
    B01110,
    B10101,
    B00100,
    B00000,
};

//Custom "Fan Off" indicator
byte fan_off[8]={
    B00100,
```

```

    B00100,
    B00100,
    B11111,
    B00100,
    B00100,
    B00100,
    B00000,
};

//Pin Connections
constint SPEAKER      =8;
constint DOWN_BUTTON  =9;
constint UP_BUTTON    =10;
constint FAN           =11;

//Variables for debouncing
booleanlastDownTempButton= LOW;
booleancurrentDownTempButton= LOW;
booleanlastUpTempButton= LOW;
booleancurrentUpTempButton= LOW;

intset_temp=23;//The Default desired temperature
booleanone_time= false;//Used for making the speaker beep only one time

void setup()
{
  pinMode(FAN, OUTPUT);

  //Create a wire object for the temp sensor
  Wire.begin();

  //Set up the LCD's number of columns and rows
  lcd.begin(16,2);

  //Make custom characters
  lcd.createChar(0, degree);
  lcd.createChar(1, fan_off);
  lcd.createChar(2, fan_on);

  //Print a static message to the LCD
  lcd.setCursor(0,0);
  lcd.print("Current:");
  lcd.setCursor(10,0);
  lcd.write((byte)0);
  lcd.setCursor(11,0);
  lcd.print("C");
  lcd.setCursor(0,1);
  lcd.print("Set:");
  lcd.setCursor(10,1);
  lcd.write((byte)0);
  lcd.setCursor(11,1);
  lcd.print("C");
  lcd.setCursor(15,1);
  lcd.write(1);
}

//A debouncing function that can be used by multiple buttons
booleandebounce(boolean last,int pin)
{
  boolean current =digitalRead(pin);
  if(last != current)
  {
    delay(5);
    current=digitalRead(pin);
  }
}

```

```

}
return current;
}

void loop()
{
//Get the Temperature
Wire.beginTransmission(TEMP_ADDR);//Start talking
Wire.write(0);//Ask for register zero
Wire.endTransmission();//Complete transmission
Wire.requestFrom(TEMP_ADDR,1);//Request 1 byte
while(Wire.available()==0);//Wait for response
int c =Wire.read();//Get the temp in C
lcd.setCursor(8,0);//Move the cursor
lcd.print(c);//Print this new value

//Debounce both buttons
currentDownTempButton=debounce(lastDownTempButton, DOWN_BUTTON);
currentUpTempButton=debounce(lastUpTempButton, UP_BUTTON);

//Turn down the set temp
if(lastDownTempButton== LOW &&currentDownTempButton== HIGH)
{
set_temp--;
}
//Turn up the set temp
elseif(lastUpTempButton== LOW &&currentUpTempButton== HIGH)
{
set_temp++;
}
//Print the set temp
lcd.setCursor(8,1);
lcd.print(set_temp);
lastDownTempButton=currentDownTempButton;
lastUpTempButton=currentUpTempButton;

//It's too hot!
if(c >=set_temp)
{
//So that the speaker will only beep one time...
if(!one_time)
{
tone(SPEAKER,400);
delay(500);
one_time= true;
}
//Turn off the speaker if it's done
else
{
noTone(SPEAKER);
}

//Turn the fan on and update display
digitalWrite(FAN, HIGH);
lcd.setCursor(15,1);
lcd.write(2);
}
//It't not to hot!
else
{
//Make sure the speaker is off, reset the "one beep" variable
//Update the fan state, and LCD display
noTone(SPEAKER);
one_time= false;
}
}

```

```
digitalWrite(FAN, LOW);
lcd.setCursor(15,1);
lcd.write(1);
}
}
```

We no longer need to have the Arduino and components tethered to the computer to see what the temperature is. If you like, you can plug in a battery or wall power supply and place it anywhere in your room.

Solution 3: Display Text and Character

3.1: String variables: String type vs. char type

Pairing an Arduino with an LCD is a great way to display text feedback and add menu-browsing capabilities to our project. So how exactly does one send text to an LCD? In the Arduino and other computer programming languages, text is thought of as a sequence of characters. For example, if we were to put together the four characters “T, e, x, t” we would make the word Text. In this way, all text is represented in code as an array of characters.

There are two main ways to create our text in code (or, more generally, to create any sequence of characters). One way is to use the *String* data type, and the other is to use a null-terminated array of type char. If that makes complete sense to us, great! If not, don’t worry; it’s much simpler than it sounds. Let’s take a look and see what the fuss is all about.

As previously mentioned, all text in our code will actually be an array of characters. For more control, a container class called the *String* class was created, which enables us to manipulate this array of characters in many complex ways. We can compare two Strings against each other to see if they’re the same, search a String of text for a substring, and append characters to the end of a String or even concatenate multiple Strings. Table 3.1 provides an overview of the various functions that can be performed on a String variable. As you can see, the String class is extremely powerful and can be useful for preparing text on our character display.

Table 3.1: The functions of the Arduino *String* class

Function	Description
charAt()	Accesses a particular character of a String
compareTo(String two)	Tests if two Strings are equal, or if one comes before or after the other
concat(String two)	Combines two Strings into one new String
endsWith(String two)	Tests whether a String ends with the char of another
equals(String two)	Performs case-sensitive comparison of two Strings’ equality
equalsIgnoreCase(String two)	Performs non-case-sensitive comparison of two Strings’ equality
getBytes(char [], int length)	Copies a String’s characters to the supplied buffer
indexOf(val)	Locates a character or String within another String
indexOf(val, int index)	starting from the front (val can be a char or String)
length()	Returns the length of the String in characters
replace(String one, String two)	Replaces all instances of a character or substring with another

setCharAt(int index, char c)	Sets or changes a particular character of a String
startsWith(String s)	Returns a Boolean (true/false) indicating whether aString starts withthe characters of another String
substring(int start)	Returns a substring of a String
toCharArray(char [], intlength)	Copies a String's characters to the supplied array
toLowerCase()	Returns a copy of the original String with allcharacters lowercase
toUpperCase()	Returns a copy of the original String with allcharacters uppercase
Trim()	Returns a copy of the original String with allwhitespace before and after the String removed

Declaring a String is easy. Here are a couple of examples:

```
String s = "Arduino in Action Rocks!";
String s = String(13);
```

Both of these lines will create a String called *s*, the first from a constant string of characters, and the second from an integer number (defaults to base 10). The String functions outlined in table 7.1 provide many utilities. For example, to combine these two lines,

```
String first = "Hello";
String second = " World";
```

into a new String *third*, simply call this function:

```
String third = first.concat(second);
```

String third would now be "Hello World". But as is often the case, this added functionality of the *String* class comes at the price of memory, and because memory can be precious on the Arduino, we may want to bypass the *String* type and use the more lightweight array of type *char* directly.

NOTE: You may have noticed that up until now we've been referring to Strings with a capital S; for char strings, we'll use a lowercase s.

There are many ways to represent a string as a char array, as you can see in table 3.2. Character strings normally end with a null character (ASCII code 0), which ensures that Arduino functions such as *Serial.print()* know exactly where the end of the string is. This is why the arrays *two[5]* and *three[5]* in table 3.2 are five characters long, even though text is technically only four characters long; the Arduino compiler automatically inserts the extra null character at the end. One last important reminder: constant strings are always declared inside double-quotes, whereas single chars are declared in single-quotes. See table 3.2 for examples.

Table 3.2 Possible **char** type **string** array initializations

Declaration	Description
<code>char one[10];</code>	Declares a non-initialized char array
<code>char two[5] = { 't', 'e', 'x', 't' };</code>	Declares an array with an extra char so the compiler can automatically add the null char
<code>char three[5] = { 't', 'e', 'x', 't', '\0' };</code>	Same as the previous example with the null char added explicitly
<code>char four[] = "text";</code>	Compiler automatically sizes to the string constant plus the null character
<code>char five[5] = "text";</code>	Initializes to explicit size and string constant


```
char six[10] = "text";
```

Initializes the array with extra space for a larger string

At this point, we've hopefully demystified the differences between the String type and the array of char type strings that we'll encounter when working with the Arduino and LCDs. Next we'll look at wiring up our first LCD, so without further ado, please welcome the Hitachi HD44780.

3.2: Parallel character LCDs: the Hitachi HD44780

The Hitachi HD44780 is one of the most common LCD controller chips designed for embedded systems and microcontrollers. The chip supports many shapes and sizes of displays, and in this example, we'll use one to drive a 16 x 2 LCD (2 rows, 16 characters long). The pervasiveness of the Hitachi HD44780 controller (and other similar LCD chips) is great news for us, because they can usually be purchased cheaply or salvaged from your old machines. Some are even pretty fancy, offering single and multicolor (RGB) backlighting. Backlit LCDs have lights (LEDs) embedded in the screen, which can be turned on to make the screen glow. This isn't only great for low-lit situations, but also for visual feedback. For example, an LCD such as the Hitachi HD44780 that has an RGB backlight can light the screen with different colors, reporting the status of your Arduino. You might turn the screen red to let the user know something is wrong, or green to signal that things are OK.

3.3: 4-bit or 8-bit?

Hitachi HD44780-based LCDs come in many different configurations, but there are two ways in which we can interface with the Hitachi HD44780 LCD: 4-bit and 8-bit. The main trade-off between the 4-bit and 8-bit configurations is the number of pins needed on the Arduino versus speed of execution. Because it's a parallel LCD, the simplest means of communication would be to send the full byte (8-bits) of data all at once (in a 1-byte message). To do this would require at least 10 I/O pins on the Arduino. In contrast, the 4-bit mode requires just 6 I/O pins and splits the byte into two 4-bit nibbles. This saves pins but takes a little more time (two messages versus one message). The 4-bit mode is still "parallel" in the sense that we receive 4 bits at a time, but it's split into two messages that are sent one after another.

3.4: Library and functions

Luckily for us, working with LCDs based on the Hitachi HD44780 chipset (or other similar chipsets) is a breeze. One of the standard libraries preinstalled in the Arduino IDE is the *LiquidCrystal LCD library*. It's compatible with both 4-bit and 8-bit configurations and provides us with many useful functions for controlling our LCD. Table 3.3 details the functions available in the *LiquidCrystal library*.

Table 3.3 The functions available in the LiquidCrystal LCD library

Function	Description
<code>begin(int column, int row)</code>	Sets the dimensions of the screen
<code>clear()</code>	Resets and clears everything on the display
<code>home()</code>	Sets the cursor to the upper left of the LCD
<code>setCursor(int column, int row)</code>	Sets the cursor to the position passed in
<code>write(byte value)</code>	Writes a character to the current cursor position
<code>print(data)</code>	Prints text to the string; can be a char, byte, int, long, or string/String
<code>cursor()</code>	Displays an underscore at the current position
<code>noCursor()</code>	Hides the cursor character
<code>blink()</code>	Blinks the cursor character
<code>noBlink()</code>	Disables the cursor character from blinking

display()	Turns the display on and restores text if turned off by noDisplay()
noDisplay()	Turns off the display, saving current text
scrollDisplayLeft()	Scrolls text one space to the left
scrollDisplayRight()	Scrolls text one space to the right
autoscroll()	Automatically scrolls text, pushing previous character one position to the left or right
noAutoscroll()	Disables autoscrolling
leftToRight()	Sets the direction of text being displayed
rightToLeft()	Sets the direction of text being displayed
createChar(int num, byte[] charData)	Defines a custom 5 x 8 character

3.5: Circuit diagram

Now that we have a good understanding of how the Hitachi HD44780 communicates on both the hardware and software level, we're ready to start connecting everything. For this, we're going to need the following components:

- An Arduino (such as Arduino Uno or Mega).
- A Hitachi HD44780-based LCD screen.
- A 10k ohm potentiometer or trimpot (R1).
- A resistor. (R2. This is only needed if your LCD has a backlight, and the value of the resistor will depend on the backlight of the LCD)

NOTE: In figure 3.1, the backlight LED+ (if present) is connected to 5V on the Arduino via a 68-ohm current-limiting resistor. This value may differ and should be calculated based on the specification of your screen backlight. There are many tutorials and online calculators to help you determine the optimal value for your screen.

3.6: Connecting everything up in 4-bit mode

To save our precious I/O pins for other sensors and devices, we can connect everything in 4-bit mode. The *LiquidCrystal* library automatically takes care of the logic required to communicate in 4-bit mode, so there's no difference in terms of coding. Our completed wiring will look similar to figure 3-1. If possible, the first thing we should do is look up the data sheet for our particular LCD module. This is useful for identifying the pin layout of our LCD if the layout isn't already silk-screened onto the LCD circuit board itself. If there isn't a printed label on your board, or a sticker with the model number, and you purchased the LCD from an online hobby store, check the retailer's website because they often provide links to the data sheets. If you salvaged your LCD from an old printer or other machine, and you can't find any information about the pin layout, be careful. Most 16-pin Hitachi 77480-compatible LCDs will follow a similar pin layout, but incorrectly wiring up the LCD can damage both the LCD and your Arduino.

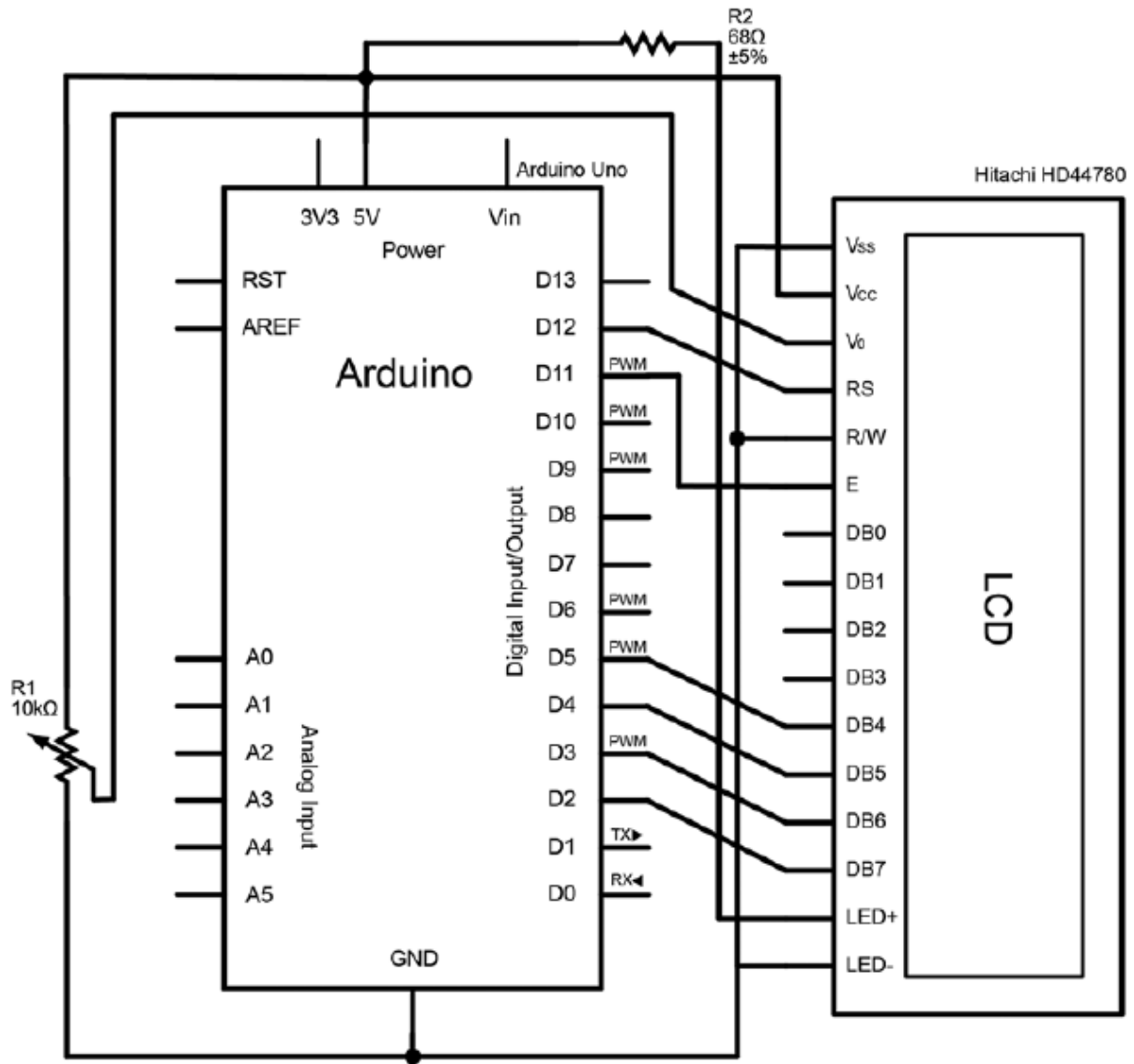


Figure 3.1: The connections between a Hitachi HD44780-based LCD display and the Arduino

As per the circuit diagram in figure 3.1, first connect the Vss pin to GND. The next pin, Vcc, supplies power to the LCD and should be connected to +5V on the Arduino (or 3.3V depending on your LCD). Next, we'll want to connect the V0 pin on the LCD to the wiper (middle leg) of a 10k linear potentiometer. Connect the left leg to +5V and the right to GND. This is used to set the contrast of your LCD, and you may prefer to use a trimpot if the contrast is something you want to set and forget.

NOTE Ground and power rails have been created using the vertical columns labelled "+" and "-". Now that power and contrast are all set, we can move on to your communication lines. Connect the Register Select (RS) pin on the LCD to Arduino digital pin 12. This RS pin is used to control where in the LCD's internal memory the Arduino will write the current character. Next, wire the Enable (E) pin to Arduino digital pin 11. Enable is what actually allows writing to those registers. Read/Write (RW) can be connected directly to GND or to Arduino digital pin 10 (optional). Tying RW to digital pin 10 instead of ground gives you the added functionality of being able to send information back from the LCD if you so choose. If you don't wish to read from the LCD (which you won't in most cases), save yourself the digital pin and connect RW directly to ground.

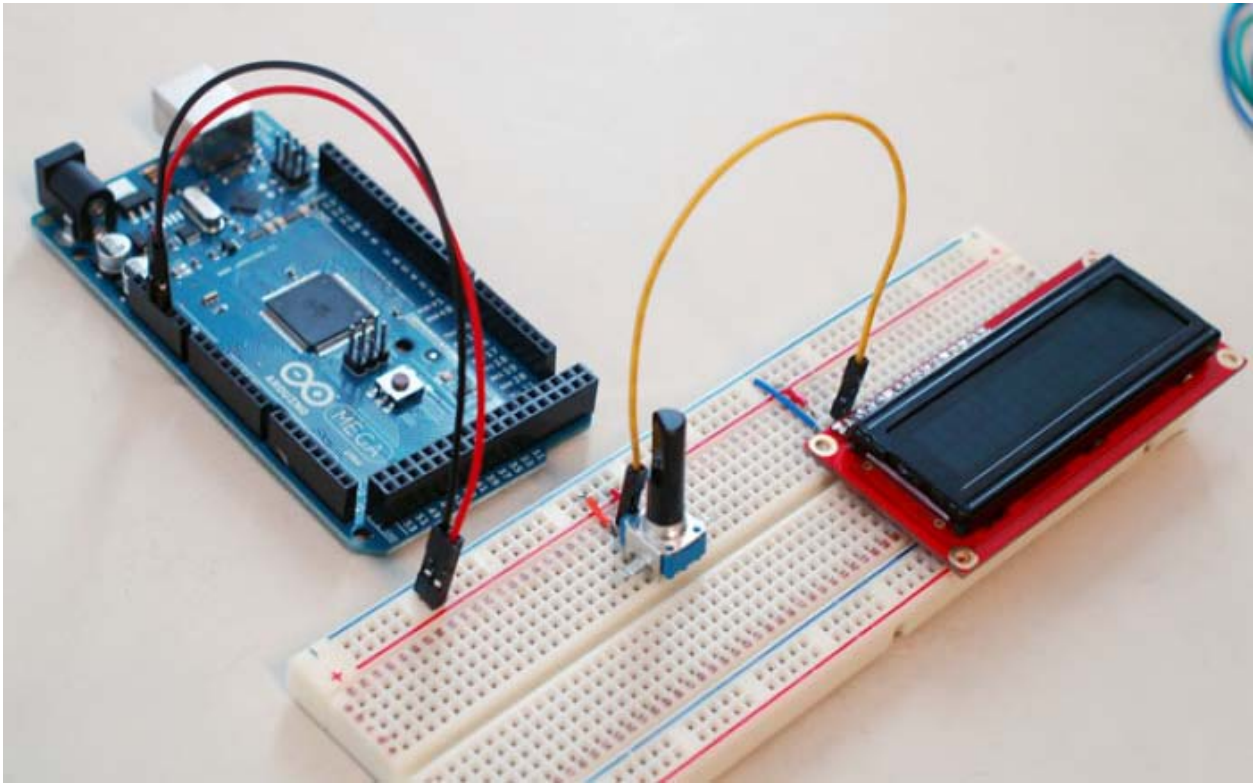


Figure 3.2 Power and contrast wiring for the Hitachi HD44780 parallel LCD

Since we're running in 4-bit mode, the next four pins on the LCD (DB0–DB3) don't need to be connected. That brings us to LCD bits DB4–DB7: DB4 should be connected to Arduino digital pin 5, DB5 to digital pin 4, DB6 to digital pin 3, and DB7 to digital pin 2. If your LCD has a backlight, now would be the time to connect the backlight LED+ to +5V on the Arduino through a series resistor (see the earlier note), and the LED- to GND on your Arduino.

Voilà! We're now ready to test our first sketch and communicate with your LCD. If you want to triple check your connections, please refer to table 3.4.

Table 3.4 The required circuit connections between the Hitachi HD44780 LCD and the Arduino

Arduino pin	LCD pin
GND	Vss
+5V	Vcc
Pin 2 (wiper) of 10k linear potentiometer	V0
D12	Register Select (RS)
D11	Enable (E)
GND or D10 (optional)	Read/Write (RW)
D5	DB4 (bit 4)
D4	DB5 (bit 5)
D3	DB6 (bit 6)
D2	DB7 (bit 7)
+5V through a series resistor (for example, 68 ohm)	LED+
GND	LED-

3.7: Sketch for writing to the Hitachi HD44780

Now that everything is wired up, let's print something to the screen. Open up the Arduino IDE and carefully copy the code from the following listing into your empty sketch (or simply run the provided sketch).

Listing 7.1 Writing text onto the LCD

```
#include <LiquidCrystal.h>

LiquidCrystal lcd(12,11,5,4,3,2);

void setup(){
  lcd.begin(16,2);
  lcd.print("Arduino in");
  lcd.setCursor(0,1);
  lcd.print("Action Rocks!");
}

void loop(){// do nothing}
```

First, we must include the *LiquidCrystal* library header file to tell Arduino that we wish to use the *LiquidCrystal* library. Next, we create an instance of the *Liquid-Crystal* class, *lcd*, and pass in the Arduino pins to which our LCD is connected. The number of arguments we pass in to the constructor automatically configures the *lcd* object to run in 4-bit or 8-bit mode.

In our setup routine, configure the size of our screen, and then print some text to the LCD. Because the text we want to print is longer than one row, we print the first half, then go to the second line using the *setCursor* method and then print the second half of the *text*.

3.8: Upload and test

Connect the USB cable to your Arduino and verify that the sketch compiles. Make sure your board and serial port are selected in the Tools menu, and click the Upload icon. Shortly after the sketch completes, your LCD should display the words "Arduino in Action Rocks!" (It should look similar to the image in figure 7.3.). Exciting! If you don't see anything displayed on the screen, or if the text is too dim, turn the potentiometer wired to V0 to adjust the contrast of the screen. Seeing text on your LCD for the first time is always very exciting, especially when the display is giving you useful information.

Now that you've tackled displaying static text in your first LCD project.

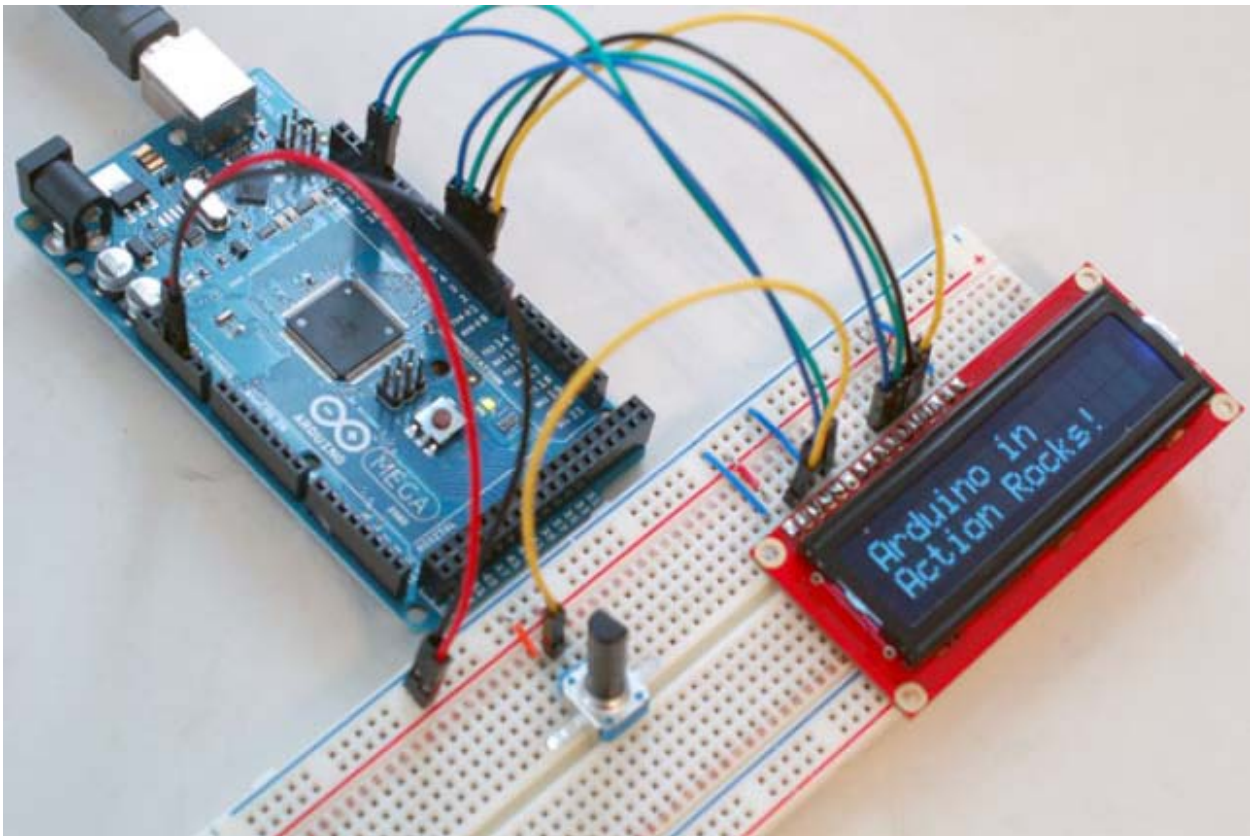


Figure 3.3: Completing the wiring for the Hitachi HD44780 parallel LCD

Solution 4: Using LCD+Keypad Shield for Arduino

4.1: How to use 16×2 Character LCD + 6-buttons Keypad Shield ?

The LCD-Keypad Shield attaches to your Arduino board to provide a 16-character by 2-line display, white character, blue back light LCD with a keypad consisting of 5 keys > “select”, “up”, “right”, “down” and “left”. With this shield you will be able to move through menus and make selections straight from one board attached to your Arduino without requiring a massive tower of shields or wiring tangling around. The LCD-Keypad Shield works perfectly in 4-bit mode with the “*LiquidCrystal*” library found in the Arduino IDE, using this library will allow you to control the LCD with only 6 digital I/O lines. This shield provides you with the capability of pushing multiple buttons at once and combining the results. No longer will you be restrained to only 5 inputs, now you have the ability to make use of 32 different button combinations!

This LCD Keypad Shield use total of 6-pins to control the LCD display which is pin-4, 5, 6, 7, 8, 9. For LCD Data, it use pin-4, 5, 6, 7, while for the RS and Enable pin, it use pin-8 and 9. The Arduino-LCD Keypad Shield is only required to plug into the Arduino main board and there was no soldering are required such as shown in figure below.



4.2: Pins Assignment

Arduino Pin	Function
A0	Buttons (Select, Up, Down, Left, Right)
4	LCD DB4
5	LCD DB5
6	LCD DB6
7	LCD DB7
8	LCD Register Select, RS
9	LCD Enable, En
10	PWM control for Backlight brightness

4.3: Application Ideas

The below code listing demonstrate the use of this LCD+Keypad shield.

Listing 4.1: Key_press Sketch

```
//=====
// Author      : Handson Technology
// Project     : Arduino Uno
// Description  : Display "Hello World" and 6-keys press
// Source-Code : Key\_press.ino
```



```
//=====

//Include the LiquidCrystal header file which is inside the Arduino IDE
#include <LiquidCrystal.h>

/* ==== PINS ASSIGNMNET =====
 * LCD RS pin to digital pin 8
 * LCD EN pin to digital pin 9
 * LCD D4 pin to digital pin 4
 * LCD D5 pin to digital pin 5
 * LCD D6 pin to digital pin 6
 * LCD D7 pin to digital pin 7
 * Backlight PWM control to Pin 10
 * LCD R/W pin to ground
 */

// Set the I/O pin for LCD 4-bit mode following the library assignment:
// LiquidCrystal(rs, en, d4, d5, d6, d7).
LiquidCrystal lcd(8,9,4,5,6,7);

int analogPin = A0;//Define the A0 as analogPin as integer type.
int adc_key_old;
int adc_key_in;
int NUM_KEYS =5;
int key=-1;
int adc_key_val[5]={30,150,360,535,760};//Define the value at A0 pin
// when a key is pressed.

// Define each key as character string type for keypress display.
char msgs[5][15]={"Right Key OK ",
"Up Key OK    ",
"Down Key OK  ",
"Left Key OK  ",
"Select Key OK"};

/*****
 * PRIVATE FUNCTION: setup()
 * PARAMETERS:void
 * RETURN:void
 * DESCRIPTIONS:
 * Define of I/O pin as Input or Output
 *
 *****/

// The setup() method runs once, when the sketch starts

void setup ()
{
  lcd.begin(16,2);// set the lcd type: 16-character by 2-lines
  lcd.clear();// LCD screen clear
  lcd.print(" HandsOn Tech ");// Send the ASCII code to the LCD for
// displaying the message

  pinMode(10, OUTPUT);// sets backlight pin-10 as PWM output
  analogWrite(10,125);// Set backlight to 50% brightness

  lcd.setCursor(0,1);// set the position of next message string:
// 1st position at 2nd line
  lcd.print("LCD-Keypad Sh");
  delay(5000);// delay for 3000ms
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print(" HandsOn Tech ");

```

```

    lcd.setCursor(0,1);
    lcd.print("Pls press any");
    adc_key_old = analogRead(analogPin); // store the unpress key value
}

/*****
* PRIVATE FUNCTION: loop()
* PARAMETERS: void
* RETURN: void
* DESCRIPTIONS:
* Non-Stop looping
*****/
void loop()
{
    adc_key_in = analogRead(analogPin); // Read the value at analogPin A0 and store
// the value in the adc_key_in register
    adc_key_in = get_key(adc_key_in); // Send the adc_key_in value to
// get_key() subroutine.
    lcd.setCursor(0,1);
    lcd.print(msgs[adc_key_in]); // Display message with msgs[]
// according to adc_key_in value
}

/*****
* PRIVATE FUNCTION: get_key
* PARAMETERS: integer
* RETURN: unsigned int input
* DESCRIPTIONS:
* convert the ADC value to number between 0 to 4
*****/
int get_key(unsignedint input)
{
    int k;

    for(k =0; k < NUM_KEYS; k++)
    {
        if(input < adc_key_val[k])
        {
            return k;
        }
    }

    if(k >= NUM_KEYS)
        k =-1; // No valid key pressed

    return k;
}

```



Fig 4.1: Display with code Listing4.1: LCD+Keypad Sketch running.

Once the code is successfully compiled and upload into Arduino board, you should see the display as above shown. The code is self explanatory with the detail comments. Compile and upload the code into Arduino board and watch the result.

Solution 5: Arduino Code Examples with LCD+Keypad Shield

5.1: Formatting Text

You want to control the position of text displayed on the LCD screen; for example, to display values in specific positions.

This sketch displays a countdown from 9 to 0. It then displays a sequence of digits in three columns of four characters. Change numRows and numCols to match the rows and columns in your LCD:

Listing 5.1: Formatting Text

```

/*=====
// Author      : Handson Technology
// Project     : Arduino Uno
// Description  : FormatingText
//=====
*/

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

//constants for the number of rows and columns in the LCD

```

```

const int numRows=2;
const int numCols=16;
int count;

// initialize the library with the numbers of the interface pins
LiquidCrystal lcd(8,9,4,5,6,7);

void setup()
{
    lcd.begin(numCols,numRows);

    pinMode(10, OUTPUT); // sets backlight pin-10 as PWM output
    analogWrite(10,125); // Set backlight to 50% brightness
    lcd.print("Starting in "); // this string is 12 characters long
    lcd.setCursor(0,1); // set the position of next message string
    lcd.print("  HandsOn Tech  "); // Send the ASCII code to the LCD

    for(int i=9;i>0;i--) // count down from 9
    {
        // the top line is row 0
        lcd.setCursor(12,0); // move the cursor to the end of the string
        lcd.print(i);
        delay(1000);
    }
}

void loop()
{
    int columnWidth=4; //spacing for the columns
    int displayColumns=3; //how many columns of numbers

    lcd.clear();

    for(int col=0; col < displayColumns; col++)
    {
        lcd.setCursor(col * columnWidth,0);
        count= count+1;
        lcd.print(count);
    }
    delay(1000);
}

```

Discussion:

The *lcd.print()* functions are similar to *Serial.print()*. In addition, the LCD library has commands that control the cursor location (the row and column where text will be printed).

The *lcd.print()* statement displays each new character after the previous one. Text printed beyond the end of a line may not be displayed or may be displayed on another line. The *lcd.setCursor()* command enables you to specify where the next *lcd.print()* will start. You specify the column and row position (the top-left corner is 0,0). Once the cursor is positioned, the next *lcd.print()* will start from that point, and it will overwrite existing text. The sketch in this recipe's Solution uses this to print numbers in fixed locations.

For example, in setup:

```

    lcd.setCursor(12,0); // move the cursor to the 13th position
    lcd.print(i);

```

lcd.setCursor(12,0) ensures that each number is printed in the same position, the thirteenth column, first row, producing the digit shown at a fixed position, rather than each number being displayed after the previous number.

Note: Rows and columns start from zero, so `setCursor(4,0)` would set the cursor to the fifth column on the first row. This is because there are five characters located in positions 0 through 4. If that is not clear, it may help you if you count this out on your fingers starting from zero.

The following lines use `setCursor()` to space out the start of each column to provide `columnWidth` spaces from the start of the previous column:

```
lcd.setCursor(col * columnWidth, 0);  
count = count + 1;  
lcd.print(count);  
lcd.clear();
```

`lcd.clear()` clears the screen and moves the cursor back to the top-left corner.

To adjust how many numbers are displayed in a row to fit the LCD, calculate the `displayColumns` value rather than setting it.

Change:

```
int displayColumns = 3;
```

to:

```
int displayColumns = numCols / columnWidth;
```



Fig 5.1: Display with code Listing 5.1: Formatting Text running

5.2: Displaying Special Symbols

You want to display special symbols: ° (degrees), ¢, √, π (pi), or any other symbol stored in the LCD character memory.

Identify the character code you want to display by locating the symbol in the character pattern table in the LCD data sheet. This sketch prints some common symbols in *setup()*. It then shows all displayable symbols in *loop()*:

Listing 5-2: Display Special Symbols

```
/*=====
// Author      : Handson Technology
// Project     : Arduino Uno
// Description  : Displaying Special Symbols
// LiquidCrystal Library
// Source Code : special\_symbols.ino
//=====
*/

#include <LiquidCrystal.h>

//set constants for number of rows and columns to match your LCD
const int numRows=2;
const int numCols=16;

// defines for some useful symbols
const byte degreeSymbol= B11011111;
const byte piSymbol= B11110111;
const byte centsSymbol= B11101100;
const byte sqrtSymbol= B11101000;
const byte omegaSymbol= B11110100; // the symbol used for ohms
byte charCode=32; // the first printable ascii character

int col;
int row;

// initialize the library with the numbers of the interface pins
LiquidCrystal lcd(8,9,4,5,6,7);

void setup()
{
    lcd.begin(numRows,numCols);

    // Turn-on display Backlight
    pinMode(10, OUTPUT); // sets backlight pin-10 as PWM output
    analogWrite(10,125); // Set backlight to 50% brightness

    showSymbol(degreeSymbol,"degrees");
    showSymbol(piSymbol,"pi");
    showSymbol(centsSymbol,"cents");
    showSymbol(sqrtSymbol,"sqrt");
    showSymbol(omegaSymbol,"ohms");
    lcd.clear();
}

void loop()
{
    lcd.print(charCode);
    calculatePosition();
}
```

```

        if(charCode==255)
        {
// finished all characters so wait another few seconds and start over
        delay(2000);
        lcd.clear();
        row= col =0;
        charCode=32;
    }

charCode=charCode+1;

}

void calculatePosition()
{
    col= col +1;

    if( col ==numCols)
    {
        col=0;
        row= row +1;

        if( row ==numRows)
        {
            row=0;
            delay(2000); // pause
            lcd.clear();
        }
        lcd.setCursor(col, row);
    }
}

// function to display a symbol and its description
void showSymbol( byte symbol, char* description)
{
    lcd.clear();
    lcd.write(symbol);
    lcd.print(' '); // add a space before the description
    lcd.print(description);
    delay(3000);
}
}

```

Discussion:

A table showing the available character patterns is in the data sheet for the LCD controller chip (you can find it on page-9 of the data sheet at http://www.handson tec.com/pdf_files/GDM2002G.pdf). To use the table, locate the symbol you want to display. The code for that character is determined by combining the binary values for the column and row for the desired symbol (see Figure 5-2).

Character Pattern Codes from Data Sheet

Upper 4 Bits / Lower 4 Bits	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
xxxx0000	CG RAM (1)			ø	@	P	`	P				-	9	ε	ρ	
xxxx0001	(2)		!	1	A	Q	a	9			。	ア	チ	△	ä	ç
xxxx1110	(7)		.	>	N	^	n	→			ヨ	セ	ホ	”	ん	
xxxx1111	(8)		/	?	0	_	o	+			ッ	ソ	マ	°	ö	■

Lower 4 bits Upper 4 bits Degree symbol

Fig 5-2: Using data sheet to derive character codes

For example, the degree symbol (°) is the third-from-last entry at the bottom row of the table shown in Figure 5-2. Its column indicates the upper four bits are 1101 and its row indicates the lower four bits are 1111. Combining these gives the code for this symbol: B11011111. You can use this binary value or convert this to its hex value (0xDF) or decimal value (223). Note that Figure 5-2 shows only 4 of the 16 actual rows in the data sheet. The LCD screen can also show any of the displayable ASCII characters by using the ASCII value in `lcd.print()`. The sketch uses a function named `showSymbol` to print the symbol and its description:

`Void showSymbol(byte symbol, char * description)`

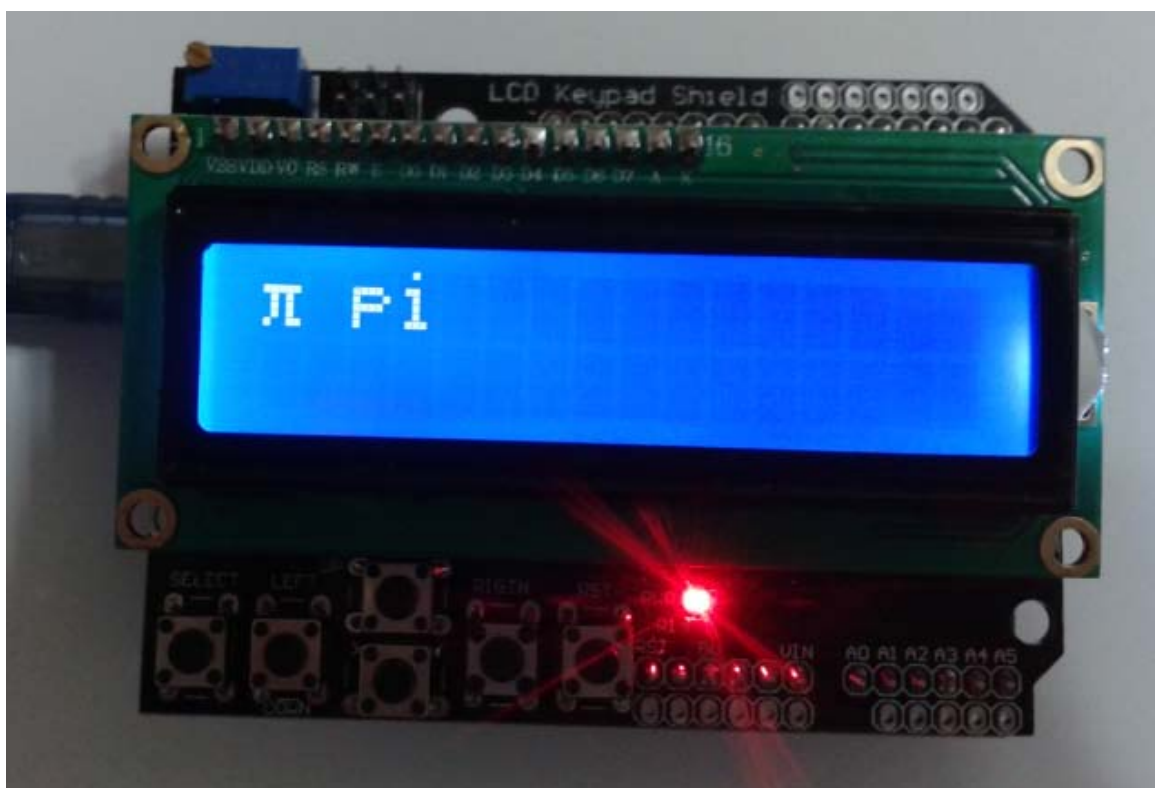


Fig 5-3: Display special Symbols π running code Listing 5-2.

5.3: Creating Custom Characters

You want to define and display characters or symbols (glyphs) that you have created. The symbols you want are not predefined in the LCD character memory. Uploading the following code will create an animation of a face, switching between smiling and frowning:

Listing 5-3: Create Custom Character

```
/*=====
// Author      : Handson Technology
// Project     : Arduino Uno
// Description  : Animated face using custom characters
// LiquidCrystal Library - Special Chars
//=====
*/

#include <LiquidCrystal.h>

LiquidCrystallcd(8,9,4,5,6,7);

byte happy[8]=
{
  B00000,
  B10001,
  B00000,
  B00000,
  B10001,
  B01110,
  B00000,
  B00000
};

bytesaddy[8]=
{
  B00000,
  B10001,
  B00000,
  B00000,
  B01110,
  B10001,
  B00000,
  B00000
};

void setup(){
  lcd.createChar(0, happy);
  lcd.createChar(1, saddy);
  lcd.begin(16,2);
  // Turn on LCD backlight
  pinMode(10, OUTPUT); // sets backlight pin-10 as PWM output
  analogWrite(10,125); // Set backlight to 50% brightness
}

void loop(){
  for(int i=0;i<2;i++)
  {
    lcd.setCursor(0,0);
    lcd.write(i);
    delay(1000);
  }
}
```

Discussion

The *LiquidCrystal library* enables you to create up to eight custom characters, which can be printed as character codes 0 through 7. Each character on the screen is drawn on a grid of 5×8 pixels. To define a character, you need to create an array of eight bytes. Each byte defines one of the rows in the character. When written as a binary number, the “1” indicates a pixel is “On”, “0” is “Off” (any values after the fifth bit are ignored). The sketch example creates two characters, named happy and saddy (see Figure 5-4).

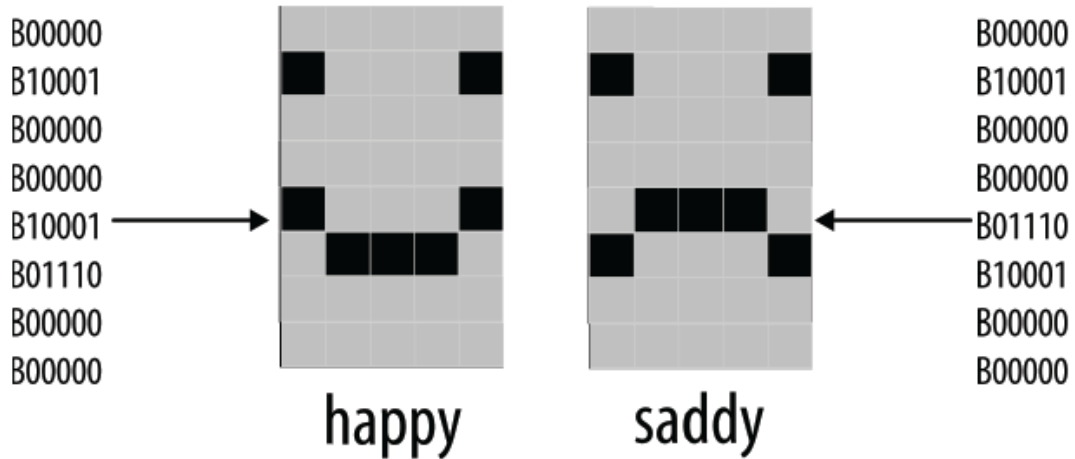


Figure 5-4: Defining custom characters

The following line in setup creates the character using data defined in the happy array that is assigned to character 0:

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

To print the custom character to the screen you would use this line:

```
lcd.write(0);
```

Note the difference between writing a character with or without an apostrophe. The following will print a zero, not the happy symbol:

```
lcd.write('0'); // this prints a zero
```

Code in the *for loop* switches between character 0 and character 1 to produce an animation.

5.4: Displaying Symbols Larger Than a Single Character

You want to combine two or more custom characters to print symbols larger than a single character; for example, double-height numbers on the screen. The following sketch writes double-height numbers using custom characters:

Listing 5-4: Displaying Double Height Characters

```
/*=====
// Author       : Handson Technology
// Project      : Arduino Uno
// Description   : Double Height Characters
// LiquidCrystal Library
```

```

// Source Code : double\_height.ino
//=====
*/

#include <LiquidCrystal.h>

LiquidCrystallcd(8,9,4,5,6,7);

byte glyphs[5][8]={
{ B11111,B11111,B00000,B00000,B00000,B00000,B00000,B00000},
{ B00000,B00000,B00000,B00000,B00000,B00000,B11111,B11111},
{ B11111,B11111,B00000,B00000,B00000,B00000,B11111,B11111},
{ B11111,B11111,B11111,B11111,B11111,B11111,B11111,B11111},
{ B00000,B00000,B00000,B00000,B00000,B01110,B01110,B01110}};

constint digitWidth=3;// the width in characters of a big digit
// (excludes space between characters)

//arrays to index into custom characters that will comprise the big numbers
// digits 0 - 4 0 1 2 3 4 5 6 7 8 9
constchar bigDigitsTop[10][digitWidth]={3,0,3,0,3,32,2,2,3,0,2,3,3,1,3,3,2,2,3,2,2,0,0,
,3,3,2,3,3,2,3};

constchar bigDigitsBot[10][digitWidth]={3,1,3,1,3,1,3,1,1,1,1,3,32,32,3,1,1,3,3,1,3,32,
,32,3,3,1,3,1,1,3};
char buffer[12];// used to convert a number into a string

void setup ()
{
    lcd.begin(20,4);

    pinMode(10, OUTPUT);// sets backlight pin-10 as PWM output
    analogWrite(10,125);// Set backlight to 50% brightness

    // create the custom glyphs
    for(int i=0;i<5;i++){
        lcd.createChar(i, glyphs[i]);// create the 5 custom glyphs
    }

    // show a countdown timer
    for(int digit =9; digit >=0; digit--){
        showDigit(digit,2);// show the digit
        delay(1000);
    }
    lcd.clear();
}

void loop ()
{
    // now show the number of seconds since the sketch started
    int number =millis()/1000;
    showNumber( number,0);
    delay(1000);
}

void showDigit(int digit,int position)
{
    lcd.setCursor(position *(digitWidth+1),0);

    for(int i=0;i<digitWidth;i++){
        lcd.write(bigDigitsTop[digit][i]);
    }
}

```

```

    lcd.setCursor(position *(digitWidth+1),1);

    for(int i=0;i<digitWidth;i++){
        lcd.write(bigDigitsBot[digit][i]);
    }
}

Void showNumber(int value,int position)
{
    int index;// index to the digit being printed, 0 is the leftmost digit
    itoa(value, buffer,10);// see Recipe 2.8 for more on using itoa

    // display each digit in sequence

    for(index =0; index <10; index++)// display up to ten digits
    {
        char c = buffer[index];
        if( c ==0)// check for null (not the same as '0')
            return;// the end of string character is a null, see Chapter 2
        c = c -48;// convert ascii value to a numeric value (see Recipe 2.9)
        showDigit(c, position + index);
    }
}

```

Discussion:

The LCD display has fixed-size characters, but you can create larger symbols by combining characters. This solution creates five custom characters using the technique described in 5.3. These symbols (see Figure 5-5) can be combined to create double-sized digits (see Figure 5-6).

The sketch displays a countdown from 9 to 0 on the LCD using the big digits. It then displays the number of seconds since the sketch started. The *glyphs* array defines pixels for the five custom characters. The array has two dimensions given in the square brackets:

```
byte glyphs[5][8] = {
```

[5] is the number of glyphs and [8] is the number of rows in each glyph. Each element contains 1s and 0s to indicate whether a pixel is on or off in that row. If you compare the values in `glyph[0]` (the first glyph) with Figure 5-2, you can see that the 1s correspond to dark pixels:

```
{ B11111,B11111,B00000,B00000,B00000,B00000,B00000,B00000 } ,
```

Each big number is built from six of these glyphs, three forming the upper half of the big digit and three forming the lower half. *bigDigitsTop* and *bigDigitsBot* are arrays defining which custom glyph is used for the top and bottom rows on the LCD screen.

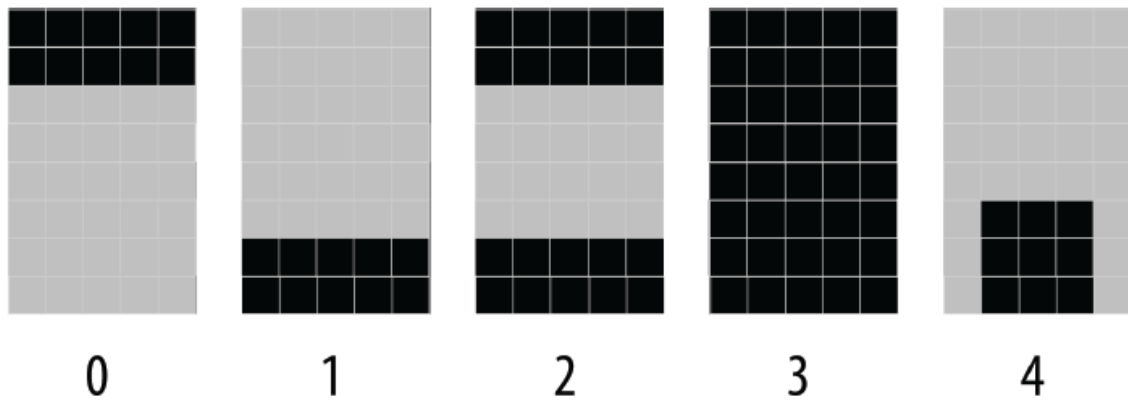


Figure 5-5: Custom characters used to form big digits

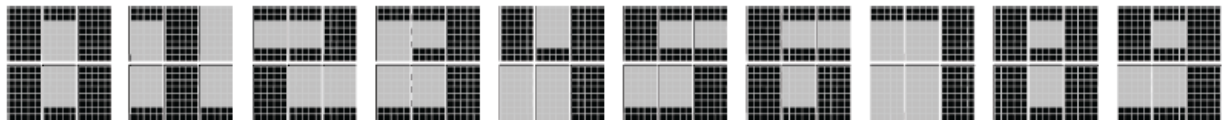


Figure 5-6: Ten big digits composed of custom glyphs

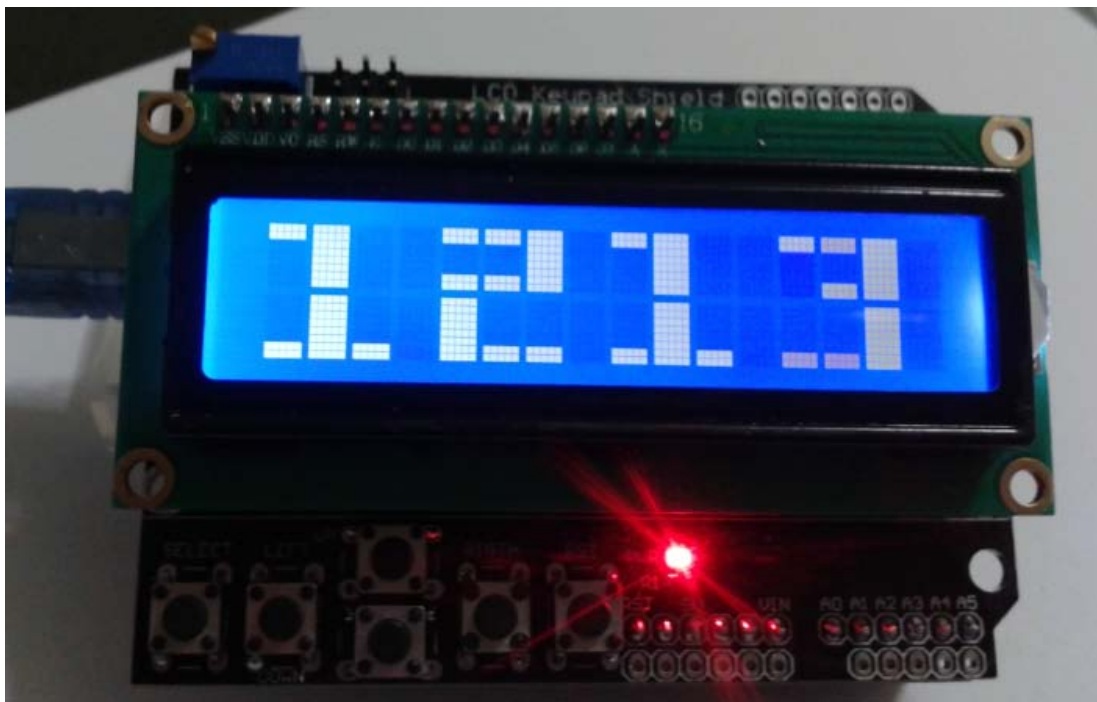


Figure 5-6: Display double digit height with code Listing5-4.

Solution 6: Using an Arduino as an LCD clock

In this tutorial we'll show you how to turn an Arduino into a clock using a 1602 LCD module, an Arduino, a hand full of jumpers and a Real Time Clock chip/module. Great as an accessory for your desk at home or work.

6.1: The Components

- Arduino Uno
- 16x02 LCD Module or any HD44780 based screen works
- 1307 RTC Chip module
- Jumper

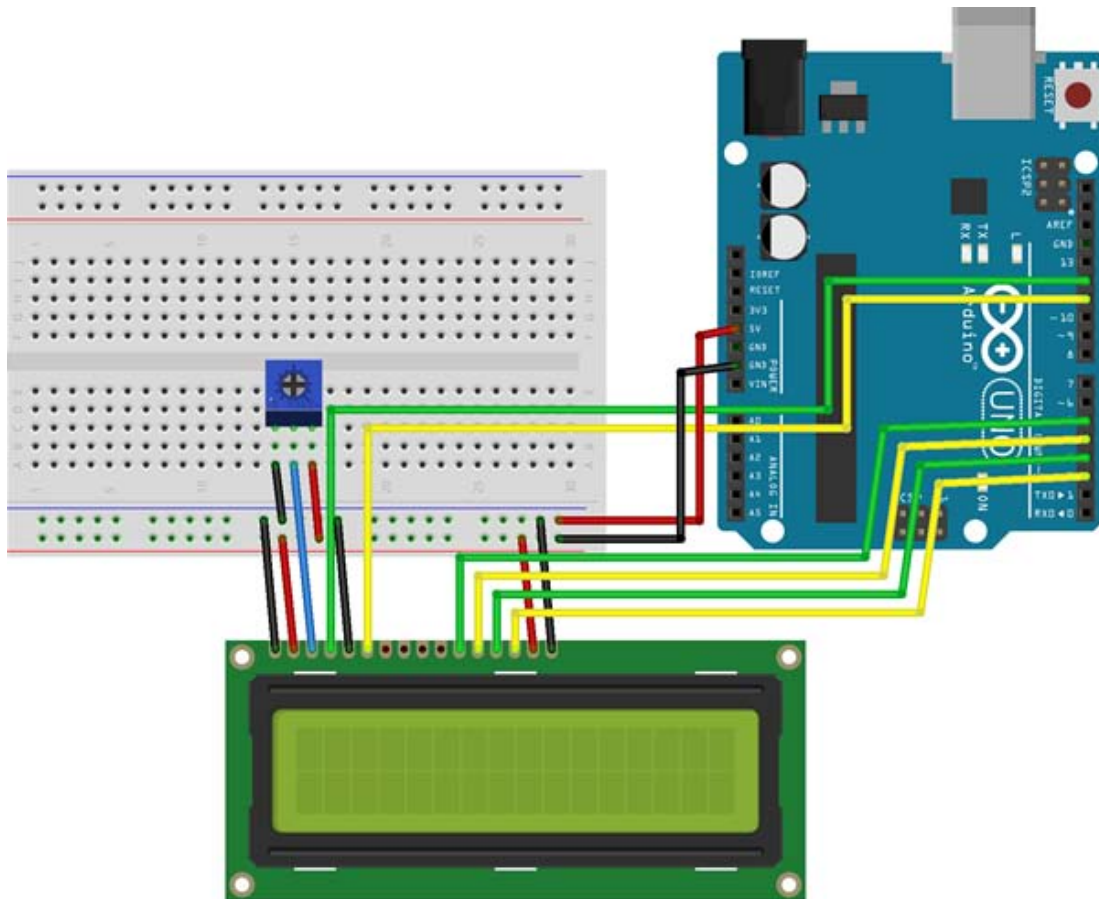
6.2: Connecting It All Together

This project will use a number of jumper cables within a very small space so take your time to check and recheck your placements so as not to produce any magic blue smoke.

Connect the LCD to the Arduino, just to check everything works as it should. Start with the red and black power cables from the Arduino to the breadboard, and then power to the LCD screen. If your display doesn't have a backlight, ignore pins 15 and 16 on the LCD screen (the far right connectors).

6.2: LCD Connection where Pin 1 on LCD is far left

- 1 - Ground Rail
- 2 - 5v Rail
- 3 - To potentiometer
- 4 - Digital 12
- 5 - Ground Rail
- 6 - Digital 11
- 11 - Digital 5
- 12 - Digital 4
- 13 - Digital 3
- 14 - Digital 2
- 15 (Backlight) - 5V Rail
- 16 (Backlight) - Ground Rail

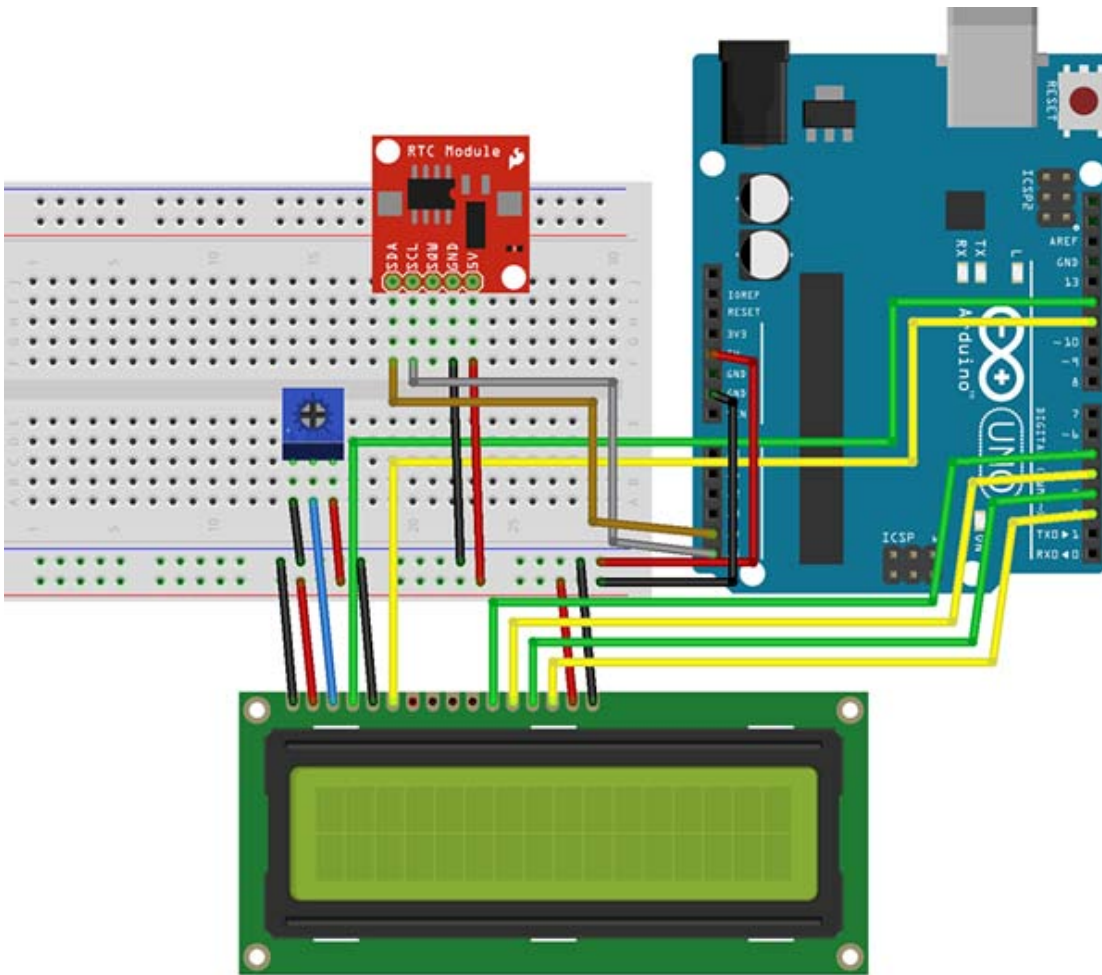


The screen is based on the HD44780 chip, a very versatile chip that is used in a number of different screen sizes, ranging from 16x1 up to 40x2. All of these will work fine, the only minor adjustment required is to ensure everything lines up correctly within the code, but this is annotated to make it easy to suit to your needs.

Once the screen is wired up, you can connect the RTC module. Different modules may have a different order of pins, so check closely with your module before moving on to the next step.

6.3: Wiring up the RTC Module:

- SDA - A4
- SCL - A5
- 5V/VCC - 5V Rail
- GND - Ground Rai



6.4: Checking It Works

Before adding some power, it's best to double check the wiring, ensuring the power and ground cables are in the right places.

Once you're happy, connect the Arduino. You'll see a line of black boxes on the screen. If you don't, try adjusting the potentiometer until something appears.

You should have the Arduino IDE (latest at time of writing is v1.0.5) installed on your computer. If not, visit the [Arduino website](http://www.arduino.cc) to download and install.

Within the Arduino IDE, click **File > Examples > LiquidCrystal > HelloWorld**. A new window full of code will open. Click **File** and then **Upload**. In less than a minute the LCD will spring to life, showing **Hello World** and counting up every second.

You have a working screen.

6.5: Setting the Time in the RTC

You'll need to tell the RTC chip the correct time. As the chip uses a battery to keep count, it can only do this once it has a time to start counting from.

Starting with a fresh Arduino screen, copy and paste the following code, scrolling down a little to change the time to the current time. Listing 6-1 only run once to set the correct time or when you wish to change the time.

Listing 6-1: Routine for time setting

```
/*=====
// Author      : Handson Technology
// Project     : Arduino Uno
// Description  : DS1307 Real Time Clock RTC with LCD
//             : Routine for time setting
// LiquidCrystal Library
// Source-Code : ds1307\_RTC.ino
//=====
*/

#include "Wire.h"
#define DS1307_I2C_ADDRESS 0x68

byte decToBcd(byte val)
{
  return((val/10*16)+(val%10));
}

void setDateDs1307(
byte second, // 0-59
byte minute, // 0-59
byte hour,   // 1-23
byte dayOfWeek, // 1-7 1=Mon, 7=Sun
byte dayOfMonth, // 1-28/29/30/31
byte month, // 1-12
byte year      // 0-99
)
{
  Wire.beginTransmission(DS1307_I2C_ADDRESS);
  Wire.write(0);
  Wire.write(decToBcd(second));
  Wire.write(decToBcd(minute));
  Wire.write(decToBcd(hour));
  Wire.write(decToBcd(dayOfWeek));
  Wire.write(decToBcd(dayOfMonth));
  Wire.write(decToBcd(month));
  Wire.write(decToBcd(year));
  Wire.endTransmission();
}

void setup()
{
  byte second, minute, hour, dayOfWeek, dayOfMonth, month, year;
  Wire.begin();
  pinMode(13, OUTPUT);

  // Change these values to what you want to set your clock to.
  // It is best to add 30 seconds to a minute to allow time for your
  // computer to compile and upload the current time.
  // Only run this script once, as running it again will
  // overwrite the time set in the RTC chip!
  // Hours are in 24 hour format
  // Day of week starts with Monday = 1 up to Sunday = 7
  // Year is in YY format, so only use the last 2 digits of the year
  //
  // Once you have run the program, the LED on pin 13 will flash
  // to say it has finished, DO NOT UNPLUG OR RESET.
}
```

```

// Simply follow the tutorial and upload the LCD code
// to avoid overwriting the correct time with this time again.
//
second=0;
minute=36;
hour=20;
dayOfWeek=6;
dayOfMonth=12;
month=12;
year=14;
setDateDs1307(second, minute, hour,dayOfWeek,dayOfMonth, month, year);
}

void loop()
{
digitalWrite(13, HIGH);// turn the LED on (HIGH is the voltage level)
delay(1000);// wait for a second
digitalWrite(13, LOW);// turn the LED off by making the voltage LOW
delay(1000);
}

```

6.6: Uploading the Time Sketch

The last step is the actual sketch that will run the clock. The only changes you will need to make is to set the date to how you would normally read it. The current script shows the date as **DD/MM/20YY**, to change this scroll down to the very end of the code, I have marked the 2 lines that can be changed as well as the screen

Listing 6.2: Display the real time on LCD

```

/*=====
// Author      : Handson Technology
// Project     : Arduino Uno
// Description  : DS1307 Real Time Clock RTC with LCD
//             : Routine to display the time
// LiquidCrystal Library
// Source Code : ds1307\_RTC\_Display.ino
//=====
*/

#include "Wire.h"
#include <LiquidCrystal.h>

#define DS1307_I2C_ADDRESS 0x68

LiquidCrystallcd(8,9,4,5,6,7);

bytebcdToDec(byte val)
{
return((val/16*10)+(val%16));
}

void getDateDs1307( byte *second,
byte*minute,
byte*hour,
byte*dayOfWeek,
byte*dayOfMonth,
byte*month,
byte*year)
{

```

```

Wire.beginTransmission(DS1307_I2C_ADDRESS);
Wire.write(0);
Wire.endTransmission();
Wire.requestFrom(DS1307_I2C_ADDRESS,7);

*second=bcdToDec(Wire.read()&0x7f);
*minute    =bcdToDec(Wire.read());
*hour      =bcdToDec(Wire.read()&0x3f);
*dayOfWeek=bcdToDec(Wire.read());
*dayOfMonth=bcdToDec(Wire.read());
*month     =bcdToDec(Wire.read());
*year      =bcdToDec(Wire.read());
}

void setup()
{
byte second, minute, hour;
bytedayOfWeek,dayOfMonth, month, year;
Wire.begin();

// AMEND IF YOUR USING A DIFFERENT LCD SCREEN
lcd.begin(16,2);
pinMode(10, OUTPUT);// sets backlight pin-10 as PWM output
analogWrite(10,200);// Set backlight to 50% brightness

}

void loop()
{
byte second, minute, hour;
bytedayOfWeek,dayOfMonth, month, year;
String s, m, d,mth, h;

getDateDs1307(&second,&minute,&hour,&dayOfWeek,&dayOfMonth,&month,&year);

if(second <10){ s ="0"+ String(second);}else{ s = String(second);}
if(minute <10){ m ="0"+ String(minute);}else{ m = String(minute);}
h =String(hour);
if(dayOfMonth<10){ d ="0"+ String(dayOfMonth);}else{ d = String(dayOfMonth);}
if(month <10){mth="0"+ String(month);}else{mth= String(month);}

char* days[]={"NA", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"};

lcd.clear();

// JUMP TO CENTER ON A 16X2 SCREEN //
lcd.setCursor(4,0);

// CHANGE THE FOLLOWING TO SET THE DATE IN TO YOUR PREFERED ORDER //
lcd.print(h + ":" + m + ":" + s);

// NEXT LINE, 1 SPACE IN FROM THE LEFT //
lcd.setCursor(1,1);

// PREFIX THE 20 AS THE RTC CHIP ONLY USES 2 DIGITS FOR THE YEAR //
lcd.print(String(days[dayOfWeek])+" "+ d + "/" +mth+"/20"+ year);
delay(1000);// Wait 1 second
}

```

When you are happy with your changes, click **Upload**. Your screen will now display the current date and time, even after being unplugged.

You will have a brilliant homemade clock for your desk or bedside table that won't require setting again. As it is made using an Arduino, and having 7 extra pins available you can add a buzzer, LEDs or buttons to add an alarm easily.

7. Appendix

7.1: LCD Application Manual

- [Download this Manual](#)
- [LCD+KeypadShield Schematic](#)

7.2: HandsOn Technology Products Quality Commitments

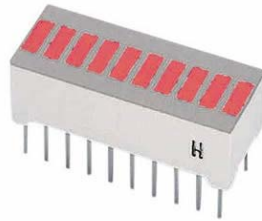
HandsOn Technology wish to be perceived as simple and affordable by our customers. However the joy over a low price is never greater than the disappointment over poor quality products. All our parts are original genuine parts with proper data specifications from manufacturers. This is to ensure you always get the high quality genuine original part as stated in our products information.

www.handsontec.com

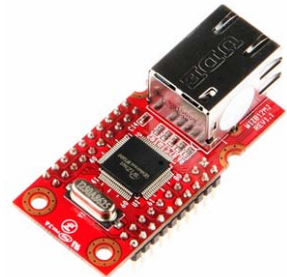
Welcome to Handsontec Store



LCD+Keyboard Shield



10-Segments LED Bar Display



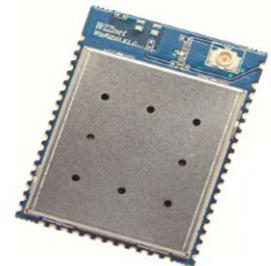
Ethernet Module



Arduino Uno



MicroSD Breakout Board



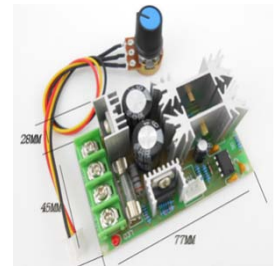
WiFi Module



20x4 LCD Display Module



Stepper Motor Driver



PWM Motor Speed Controller



www.handsontec.com

Breakout Board & Modules



www.handsontec.com

Integrated Circuits



www.handsontec.com

Discrete Parts



www.handsontec.com

Assembled Kits



www.handsontec.com

Connectors

This page is intentional left blank.