**Lab 3 : INTERFACE LCD - KEYPAD**

1. **Objectives: After this lab, the learner will be able to:**
   - Interface LCD module with Arduino board
   - Interface key pad modle with Arduino board
2. **Facilities**
   - Proteus software
   - Arduino IDE
3. **Prerequest**
   - Basic electronic
   - C programming
4. **Introduction LCD**

LCD (Liquid Crystal Display) is typically used in embedded systems to display text and numbers for the end user as an output device. The 16×2 alphanumeric display is based on the Hitachi HD44780 driver IC. Which is the small black circular chip on the back of the LCD module itself. This is the controller that controls the LCD unit and we communicate with using Arduino to send commands and text messages.

The LCD module consists of 16×2 character cells (2 rows x 16 columns), each cell of which is 5×8 dots. Controlling all of these individual dots is a tedious task for our Arduino. However, it doesn't have to do so. As there is a specific function controller on the LCD itself controlling the display while reading the user's commands & data (the Hitachi HD44780 controller).

The simplest way to communicate with the Hitachi HD44780 LCD driver is to use the Arduino LiquidCrystal library to handle this communication. This is going to provide us with an easy way to perform different operations with the LCD display without getting into the details of the LCD driver itself. However, if you want to learn more about the HD44780 LCD driver and how the LCD works on a low level, you can check this tutorial.

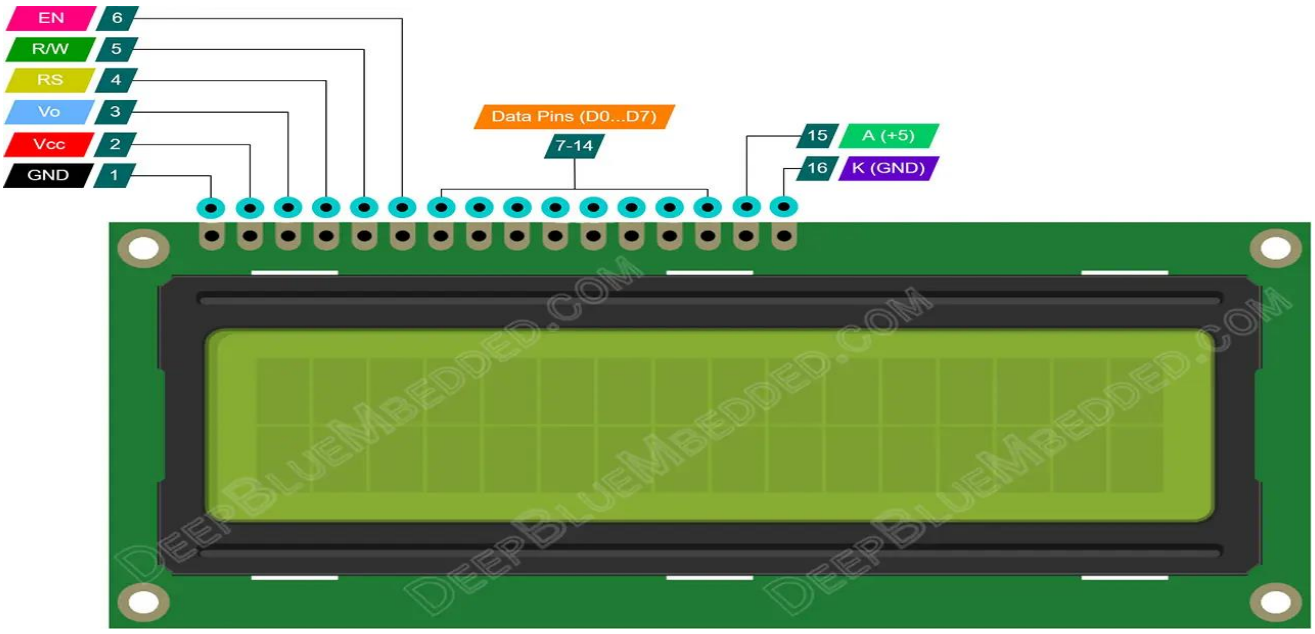
Alphanumeric LCDs are available in different units with different sizes (e.g. 16×1, 16×2, 16×4, 20×4, and so on). The backlight color of the LCD display is also available in multiple variants (green, yellow, blue, white, and red).

### 4.1. 16×2 LCD Pinout

This is the pinout for a typical LCD 16×2 display unit. It's got 8 data lines (you can use only 4 of them or all of the 8). And keep in mind that it needs to be powered from a stable +5v source.

***GND*** is the ground pin.
***Vcc*** is the LCD's power supply input pin (connects to +5v).

***Vo (Contrast Control)*** is the LCD contrast control pin. We typically use a voltage divider network to establish the control voltage that sets the desired contrast level or simply use a potentiometer to have a controllable contrast level for the LCD display.



***RS*** is the register select pin whose function is to separate the data (text and numbers) from the commands to LCD (like set cursor position, clear LCD display, shift text, etc).
***RW*** is the Read/Write pin. Which determines the type of operation we'll be doing with the LCD (read or write). As we're typically using the LCD for displaying output data, it's rarely set in read mode and this pin is usually held LOW to operate in the WRITE mode.
***EN*** is the enable pin. The enable signal is required for the parallel communication interface of the LCD in order to latch in the incoming data. An enable short pulse is required for the completion of each data transfer operation.
***D0-D7 (Data Pins)*** are the data bus pins (8-bit). The LCD can operate in 4-bit bus mode which means we can send each byte of data in two steps using the 4-bit mode. This will save us 4 IO pins that would have been wasted to create the full 8-pins bus.
***LED+ (Anode)*** is the LCD's backlight LED's anode (+) pin. This pin connects to the +5v power supply through a current limiting resistor (220Ω or 330Ω).
***LED- (Kathode)*** is the LCD's backlight LED's cathode (-) pin. This pin connects to the ground.

### 4.2. LCD 16×2 Pin Diagram

| Pin | Label | Function |
|---|---|---|
| 1 | VSS | Ground connection (GND) |
| 2 | VDD | Power supply (+5V) |
| 3 | V0 | Contrast control (connect to potentiometer) |
| 4 | RS | Register Select (HIGH = data, LOW = command) |
| 5 | RW | Read/Write (GROUND for write mode) |

| Pin | Label | Function |
|---|---|---|
| 6 | E | Enable signal (trigger to read data) |
| 7-14 | D0-D7 | Data pins (use 4 or 8 pins depending on mode) |
| 15 | LED+ | Backlight anode (+5V via 220Ω resistor) |
| 16 | LED- | Backlight cathode (GND) |

### 4.3. Working

- LCD can be used in two modes- 4 bit mode or 8 bit mode. In 8 bit mode we require 8 data pins and 3 control pins whereas in 4 bit mode, data is sent using 4 data pins and 3 control pins.
- R/W pin is always grounded so we require only 6 pins in 4 bit mode, thus saving no of pins.
- First initialize the library and then define pins using the command LiquidCrystal lcd(RS, E, D4, D5, D6, D7), pins are assigned in this order.
- LiquidCrystal lcd(12, 11, 5, 4, 3, 2), here RS pin to 12, Enable pin to 11, D4 pin to 5, D5 pin to 4, D6 pin to 3 and D7 pin to 2 respectively.
- Then in setup function write the message to display as lcd.print(“message”).
- We can print message anywhere in the LCD by selecting column and row, it’s done by writing lcd.setCursor(column, row). However there is one thing to consider, that’s the number of columns and rows start from zero. For example, to print a message on 2nd row 1st column, write “lcd.setCursor(0,1);” before the print command. Similarly for 5th column and 3rd row, we write lcd.setCursor(4,2).
- You can use “lcd.write()” to send characters. To print zero on 2nd colum 2nd row, type lcd.setCursor(1,1); lcd.write(48); where 48 is the decimal equivalent for ACII ‘0’.

### 4.4. Interfacing LCD with Arduino programming

Let’s summarize our program into the following steps.

- Step1: Initialize the library for LCD.
- Step2: Define LCD columns and rows in setup function.
- Step3: Write the data to display.
- Step4: If you want to display variables on LCD, write it in loop function. Loop function is a must for all arduino sketches.

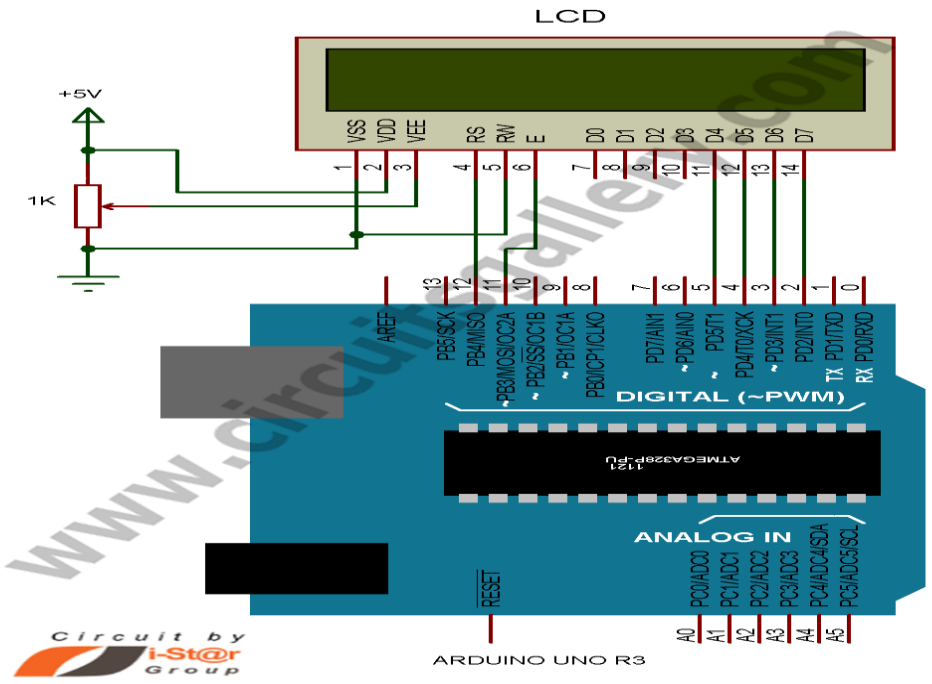
### 4.5. Functions

- **LiquidCrystal()**: function sets the pins the Arduino uses to connect to the LCD. You can use any of the Arduino’s digital pins to control the LCD. Just put the Arduino pin numbers inside the parentheses in this order: LiquidCrystal(RS, E, D4, D5, D6, D7). RS, E, D4, D5, D6, D7 are the LCD pins. For example, say you want LCD pin D7 to connect to Arduino pin 12. Just put “12” in place of D7 in the function like this: LiquidCrystal(RS, E, D4, D5, D6, 12). This function needs to be placed before the void setup() section of the program.
- **lcd.begin()**: This function sets the dimensions of the LCD. It needs to be placed before any other LiquidCrystal function in the void setup() section of the program. The number of rows and columns are specified as lcd.begin(columns, rows). For a 16×2 LCD, you would use lcd.begin(16, 2), and for a 20×4 LCD you would use lcd.begin(20, 4).

- **lcd.clear()**: This function clears any text or data already displayed on the LCD. If you use lcd.clear() with lcd.print() and the delay() function in the void loop() section
- **lcd.home()**: This function places the cursor in the upper left hand corner of the screen, and prints any subsequent text from that position. For example, this code replaces the first three letters of “hello world!” with X’s:
- **lcd.setCursor()**: Similar, but more useful than lcd.home() is lcd.setCursor(). This function places the cursor (and any printed text) at any position on the screen. It can be used in the void setup() or void loop() section of your program.
- **lcd.write()**: You can use this function to write different types of data to the LCD, for example the reading from a temperature sensor, or the coordinates from a GPS module. You can also use it to print custom characters that you create yourself (more on this below). Use lcd.write() in the void setup() or void loop() section of your program.
- **lcd.print()**: This function is used to print text to the LCD. It can be used in the void setup() section or the void loop() section of the program.To print letters and words, place quotation marks (” “) around the text. For example, to print *hello, world!*, use lcd.print(“hello, world!”).
- **lcd.Cursor()**: This function creates a visible cursor. The cursor is a horizontal line placed below the next character to be printed to the LCD.
- **lcd.blink()**: This function creates a block style cursor that blinks on and off at approximately 500 milliseconds per cycle. Use it in the void loop() section. The function lcd.noBlink() disables the blinking block cursor.
- **lcd.display()**: This function turns on any text or cursors that have been printed to the LCD screen. The function lcd.noDisplay() turns off any text or cursors printed to the LCD, without clearing it from the LCD’s memory.These two functions can be used together in the void loop() section to create a blinking text effect. This code will make the “hello, world!” text blink on and off
- **lcd.scrollDisplayLeft()**: This function takes anything printed to the LCD and moves it to the left. It should be used in the void loop() section with a delay command following it. The function will move the text 40 spaces to the left before it loops back to the first character. This code moves the “hello, world!” text to the left, at a rate of one second per character:
- **lcd.scrollDisplayRight()**: This function behaves like lcd.scrollDisplayLeft(), but moves the text to the right.
- **lcd.autoscroll()**: This function takes a string of text and scrolls it from right to left in increments of the character count of the string. For example, if you have a string of text that is 3 characters long, it will shift the text 3 spaces to the left with each step:
- **lcd.noAutoscroll()**: lcd.noAutoscroll() turns the lcd.autoscroll() function off. Use this function before or after lcd.autoscroll() in the void loop() section to create sequences of scrolling text or animations.
- **lcd.rightToLeft()**: This function sets the direction that text is printed to the screen. The default mode is from left to right using the command lcd.leftToRight(), but you may find some cases where it’s useful to output text in the reverse direction:
- **lcd.createChar()**: This command allows you to create your own custom characters. Each character of a 16×2 LCD has a 5 pixel width and an 8 pixel height. Up to 8 different custom characters can be defined in a single program

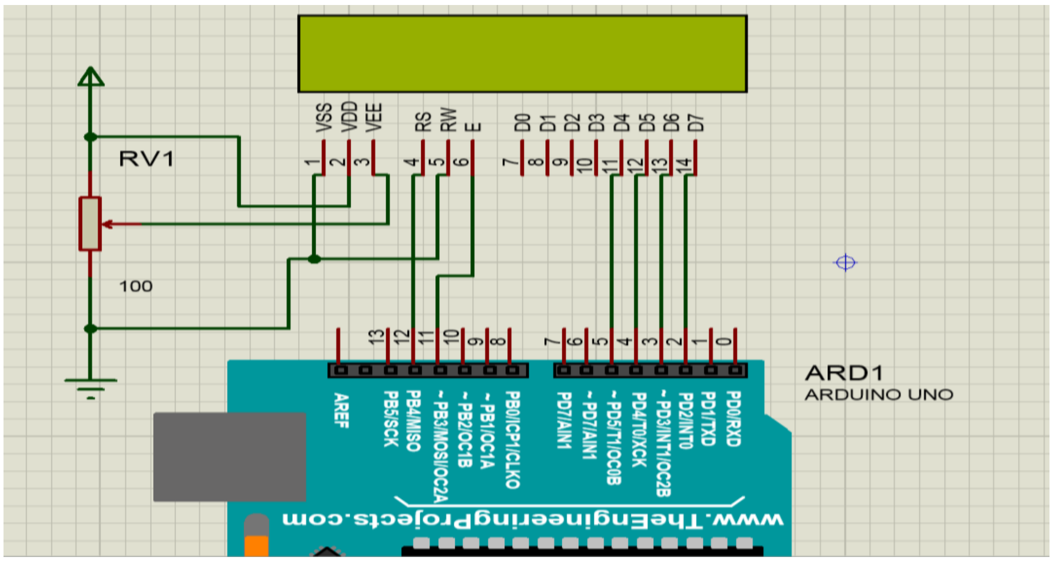
## 5. Examples 1

### 5.1. Circuit Schematic



## 5.2. Connecting Arduino to 16×2 LCD (Wiring Diagram)

Here is the wiring diagram for the 16×2 LCD display with Arduino that we'll be using in all examples hereafter in this tutorial.



And here is a summary table for Arduino -> LCD connections.

| Arduino UNO | 16×2 LCD |
|---|---|
| 2 | D7 |
| 3 | D6 |
| 4 | D5 |
| 5 | D4 |

| | |
|---|---|
| 11 | EN |
| 12 | RS |

! Note

We'll be using the 4-bit data mode instead of the 8-bit data mode to same up some IO pins. The LCD is taking up 6 pins already, so we can't waste 4 more pins just to operate in the 8-bit mode.

**Arduino LCD Contrast Adjustment**

Before attempting to program the LCD and start actually using it, we need first to test it on a hardware level and also set a proper level of display contrast. This can easily be done after connecting the LCD to Arduino's +5v power, then using the potentiometer we can set the LCD contrast level as shown in the short demo video below.

### 5.3. Arduino 16×2 LCD Code Example

Now, let's test what we've learned so far about the Arduino LCD and create our first project to display some text messages on the LCD display. On the first row of the LCD, we'll print the "Hello World!" message. And we'll move the cursor to the second row and write a "DeepBlueMbedded" text message.

**Arduino LCD Text Display Example**

In this first example, we'll include the Arduino's LiquidCrystal library and use it to display some text messages on the LCD 16×2 display.

```
#include <LiquidCrystal.h>
LiquidCrystal MyLCD(12, 11, 5, 4, 3, 2);
// Creates an LCD object, Parameters: (RS, EN, D4, D5, D6, D7)
void setup()
{
        MyLCD.begin(16, 2);
        // Set up the number of columns and rows on the LCD.
        // Move The Cursor To Char:1,Line:1
        MyLCD.setCursor(0, 0);
        // Print The First Message
        MyLCD.print("Hello World!");
        // Move The Cursor To Char:1,Line:2
        MyLCD.setCursor(0, 1);
        // Print The Second Message
        MyLCD.print("DeepBlueMbedded");
}
void loop()
{
  // DO NOTHING
}
```

## 5.4. Code explaination

First of all, we need to include the Arduino LiquidCrystal.h library which we'll be using to control the LCD driver.

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

Next, we'll create an object of the LiquidCrystal class and define its parameters. The parameters for the LiquidCrystal object are the pin numbers for the following signals: RS, EN, D4, D5, D5, D7. If you want to add another LCD, you'll have to create another object with another 6 IO pins for its signals.

```
LiquidCrystal MyLCD(12, 11, 5, 4, 3, 2); // Creates an LCD
object, Parameters: (RS, EN, D4, D5, D6, D7)
```

**setup()**

in the setup() function, we initialize the LCD object ( MyLCD) using the .begin() function. Which takes only 2 parameters (number of columns, and number of rows). For **16×2** LCD, we call MyLCD.begin(16, 2); For **20×4** LCD, we call MyLCD.begin(20, 4); And so on.

```
MyLCD.begin(16, 2); // Set up the number of columns and rows
on the LCD.
```

Then, we set the LCD cursor position to point to the first line, the first character cell. Any write operation to the LCD will start from the current cursor position value, that's why it's important to set it manually before attempting any write operation. This is to make sure that the text will be displayed exactly where we want it to be.

```
MyLCD.setCursor(0, 0); // (CharIndex, LineIndex)
```

There are 2 rows and 16 columns in our LCD as we've stated earlier. The cursor index value starts from 0, therefore, the first row has an index of 0, and the second row has an index of 1. And this is also the case for the columns' index value which also starts from 0 up to 15.

Next, we'll print the first text message "Hello World!" to the LCD starting from the current cursor position (0, 0). Using the .print() function.

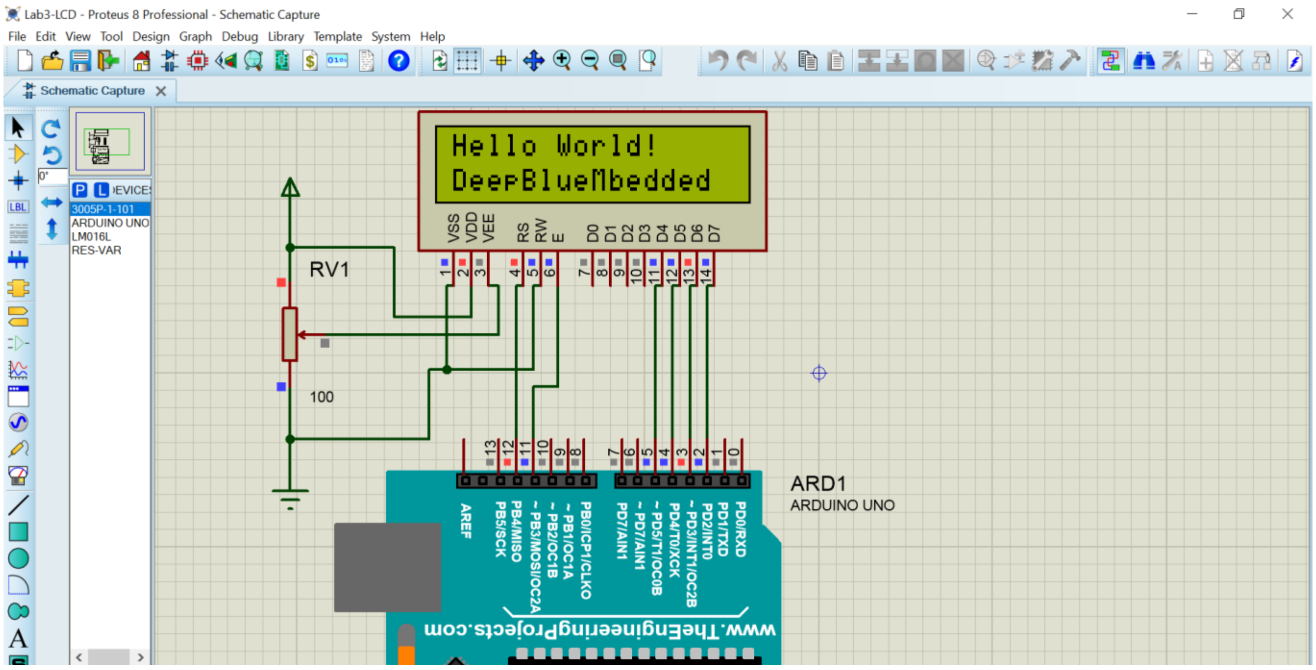
```
MyLCD.print("Hello World!");
```

Similarly, we'll point to the beginning of the second LCD line (row) and write the second message.

**loop()**

in the loop() function, nothing needs to be done

## 5.5. Simulation


Lab3-LCD - Proteus 8 Professional - Schematic Capture
File Edit View Tool Design Graph Debug Library Template System Help
Schematic Capture
DEVICES
3005P-1-101
ARDUINO UNO
LM016L
RES-VAR
Hello World!
DeepBlueMbedded
RV1
100
ARD1
ARDUINO UNO
www.TheEngineeringProjects.com

6. **Examples 2:**

In this example, we'll print numeric variables to the LCD display. We'll create a counter variable and print it to the LCD using the same .print() function. Which can also accept a lot of variable data types (strings, integers, float, double, etc). So luckily, we won't be in need to do string manipulations and data type conversion to achieve the goals of this example project.

**6.1. Example code**

```
#include <LiquidCrystal.h>
uint16_t Counter = 0;
LiquidCrystal MyLCD(12, 11, 5, 4, 3, 2);
// Creates an LCD object, Parameters: (RS, EN, D4, D5, D6, D7)
void setup()
{
        MyLCD.begin(16, 2); // Set up the number of columns and rows on the LCD.
        MyLCD.setCursor(0, 0);
        MyLCD.print("Counter Value:");
}
void loop()
{
        MyLCD.setCursor(0, 1);
        MyLCD.print(Counter++);
        delay(250);
}
```

**6.2. Code Explanation**

First of all, we need to include the Arduino LiquidCrystal.h library which we'll be using to control the LCD driver.

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

Next, we'll create an object of the LiquidCrystal class and define its parameters. The parameters for the LiquidCrystal object are the pin numbers for the following signals: RS, EN, D4, D5, D5, D7. If you want to add another LCD, you'll have to create another object with another 6 IO pins for its signals.

```
LiquidCrystal MyLCD(12, 11, 5, 4, 3, 2); // Creates an LCD object, Parameters: (RS, EN, D4, D5, D6, D7)
```

**setup()**

in the setup() function, we initialize the LCD object ( MyLCD) using the .begin() function.

```
MyLCD.begin(16, 2); // Set up the number of columns and rows on the LCD.
```

Then, we set the LCD cursor position to point to the first line, the first character cell. Any write operation to the LCD will start from the current cursor position value, that's why it's important to set it manually before attempting any write operation. To make sure that the text will be displayed exactly where we want it to be.

```
MyLCD.setCursor(0, 0); // (CharIndex, LineIndex)
```

Next, we'll print the first text message "Counter Value:" to the LCD starting from the current cursor position (0, 0). Using the .print() function.

```
MyLCD.print("Counter Value:");
```

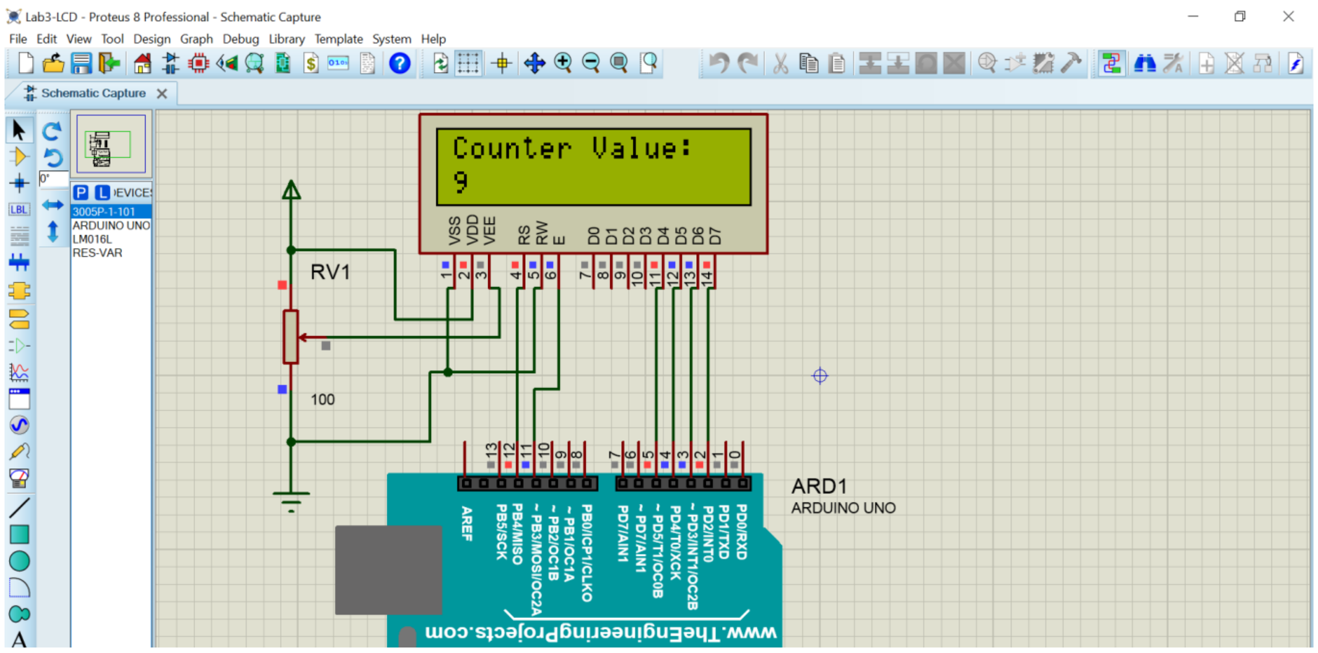
**loop()**

in the loop() function, we'll point to the first character location on the second row (line) of the LCD. And write the Counter variable to the LCD using the .print() function.

```
MyLCD.print(Counter++);
```

We also increment the Counter variable and insert a small time delay before repeating the same instructions over and over again.

### 6.3. Result


Lab3-LCD - Proteus 8 Professional - Schematic Capture
File Edit View Tool Design Graph Debug Library Template System Help
Schematic Capture
DEVICES
3005P-1-101
ARDUINO UNO
LM016L
RES-VAR
Counter Value:
9
RV1
100
ARD1
ARDUINO UNO
www.TheEngineeringProjects.com

## 7. Examples 3: Arduino LCD Scrolling Text Example

In this example, we'll print some text messages and shift the entire LCD display to create a text-scrolling effect. We'll use the scrollDisplayLeft() and scrollDisplayRight() functions to create the scrolling effect.

Note

}

The LCD’s internal Display data RAM (**DDRAM**) stores display data represented in 8-bit character codes. Its extended capacity is 80 × 8 bits or 80 characters. The area in display data RAM (DDRAM) that is not used for display can be used as general data RAM.

Therefore, whatever data you send to the DDRAM, it’ll get displayed on the LCD. As long as the characters count is below 32 (for 16×2 LCD), it’ll be visible. Otherwise, written characters are stored in the DDRAM but not visible.

Shifting the LCD display moves the data in the DDRAM in a circular way, when it reaches the end of the RAM space, you’ll find your text coming from the other LCD end. To test this, keep shifting the LCD text to the left maybe 100 times. When the buffer reaches the limit, you’ll find your text message coming from the right side of the LCD. Why? because it’s circulating through the DDRAM.

### 7.1. Example Code

```
    #include <LiquidCrystal.h>
    // Create An LCD Object. Signals: [ RS, EN, D4, D5, D6, D7 ]
    LiquidCrystal MyLCD(12, 11, 5, 4, 3, 2);
void setup()
{
        // Initialize The LCD. Parameters: [ Columns, Rows ]
        MyLCD.begin(16, 2);
        // Display The First Message In Home Position (0, 0)
        MyLCD.print("  Arduino LCD");
        // Display The Second Message In Position (0, 1)
        MyLCD.setCursor(0, 1);
        MyLCD.print("DeepBlueMbedded");
}
void loop()
{
        // Shift The Entire Display To Right 10 Times
        for(int i=0; i<10; i++)
        {
        MyLCD.scrollDisplayRight();
        delay(350);
        }
        // Shift The Entire Display To Left 10 Times
        for(int i=0; i<10; i++)
        {
        MyLCD.scrollDisplayLeft();
        delay(350);
        }
}
```

### 7.2. Code Explanation

First of all, we need to include the Arduino LiquidCrystal.h library which we’ll be using to control the LCD driver.

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

Next, we'll create an object of the LiquidCrystal class and define its parameters. The parameters for the LiquidCrystal object are the pin numbers for the following signals: RS, EN, D4, D5, D5, D7. If you want to add another LCD, you'll have to create another object with another 6 IO pins for its signals.

```
LiquidCrystal MyLCD(12, 11, 5, 4, 3, 2); // Creates an LCD
object, Parameters: (RS, EN, D4, D5, D6, D7)
```

**setup()**

in the setup() function, we initialize the LCD object ( MyLCD) using the .begin() function.

```
MyLCD.begin(16, 2); // Set up the number of columns and rows
on the LCD.
```

Then, we set the LCD cursor position to point to the first line, the first character cell. Any write operation to the LCD will start from the current cursor position value, that's why it's important to set it manually before attempting any write operation. To make sure that the text will be displayed exactly where we want it to be.

```
MyLCD.setCursor(0, 0); // (CharIndex, LineIndex)
```

Next, we'll print the first text messages " Arduino LCD " and " DeepBlueMbedded" to the LCD on lines 1 and 2 respectively. Using the .print() function.

```
MyLCD.print("  Arduino LCD");
// Display The Second Message In Position (0, 1)
MyLCD.setCursor(0, 1);
```

**loop()**

in the loop() function, we'll shift the entire display to the right side 10 times using a for loop and the scrollDisplayRight() function from the LCD LiquidCrystal library. The inserted delay here is to allow us to see the scrolling effect, and you can definitely change it to make the scrolling effect even faster or slower.

```
for(int i=0; i<10; i++)
{
  MyLCD.scrollDisplayRight();
  delay(350);
}
```

Next, we'll repeat the same step but the shifting will be in the opposite direction (left) and the code logic will be repeated forever.

## 8. Example: Scrolling Text on 16×2 LCD using Arduino

In the last section, we have learned to display simple text on LCD using Arduino. Now let's move to some advanced examples. In this section, we will discuss examples of scrolling text on LCD. That means moving text towards left and right direction.

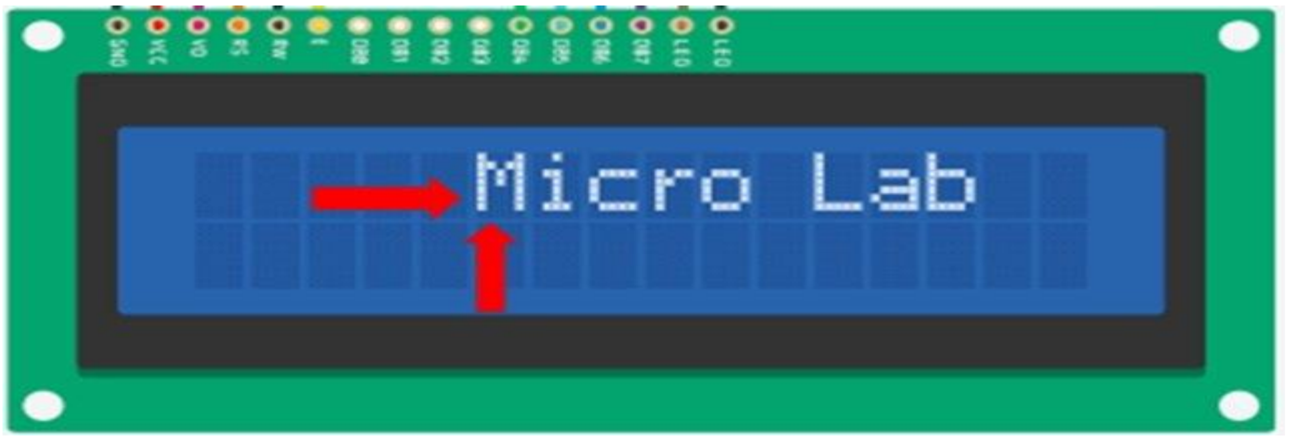
### 8.1.Arduino LCD Text Scrolling Functions

Arduino LCD library supports two functions that are used to scroll text on LCD. These are routines are:

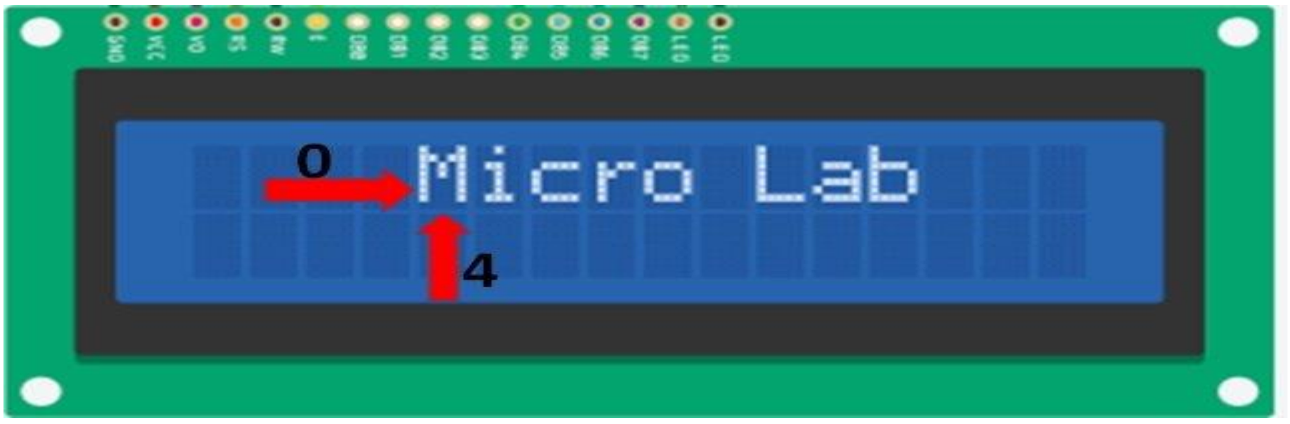
- lcd.scrollDisplayLeft();
- lcd.scrollDisplayRight();

As their name suggests, lcd.scrollDisplayRight(); moves the text one cursor position towards right from current text position and  lcd.scrollDisplayLeft(); moves the text one cursor position towards left.

For example, in this picture LCD shows text “Micro Lab” on the 6th column and 1st row location.



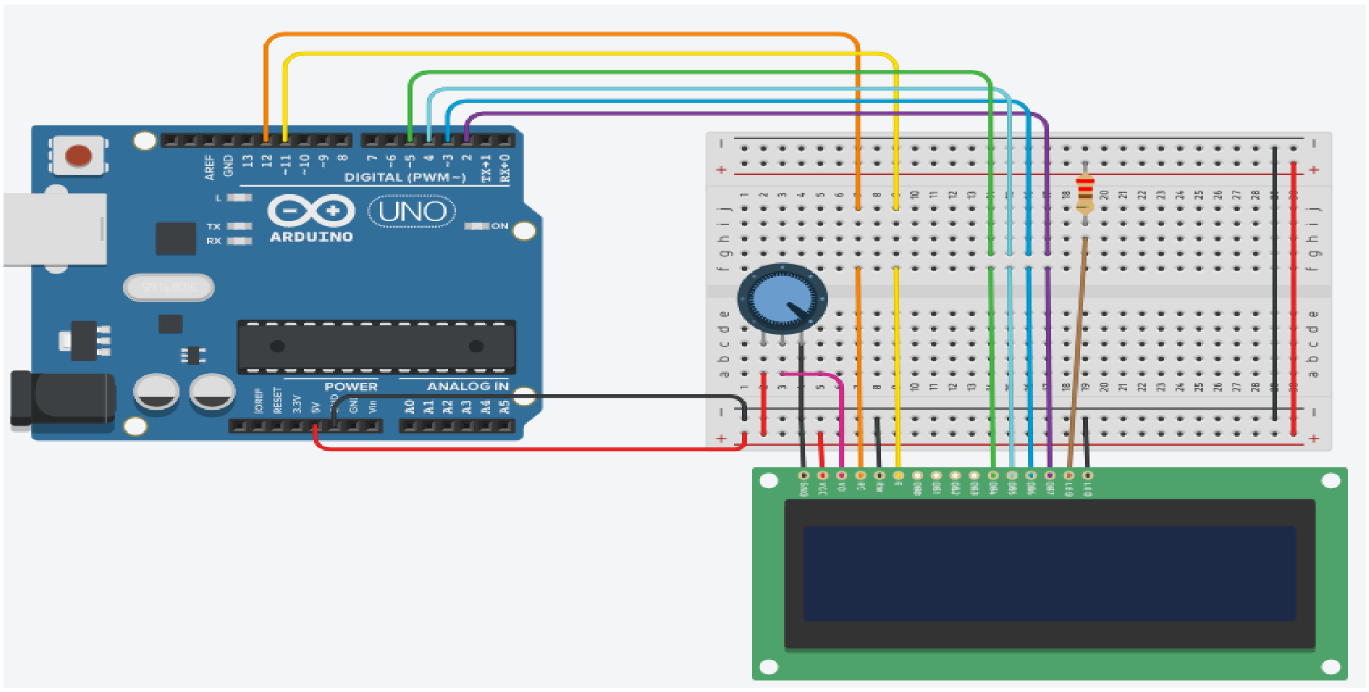
Now if we call, lcd.scrollDisplayLeft(); function inside the code, it will move the text one position left and output will look like this:



Similarly, if we call lcd.scrollDisplayRight(), it will move the text one position right and output will look like this:

## 8.2.Schematic Diagram

Make connection with Arduino and 16×2 LCD according to this schematic diagram:



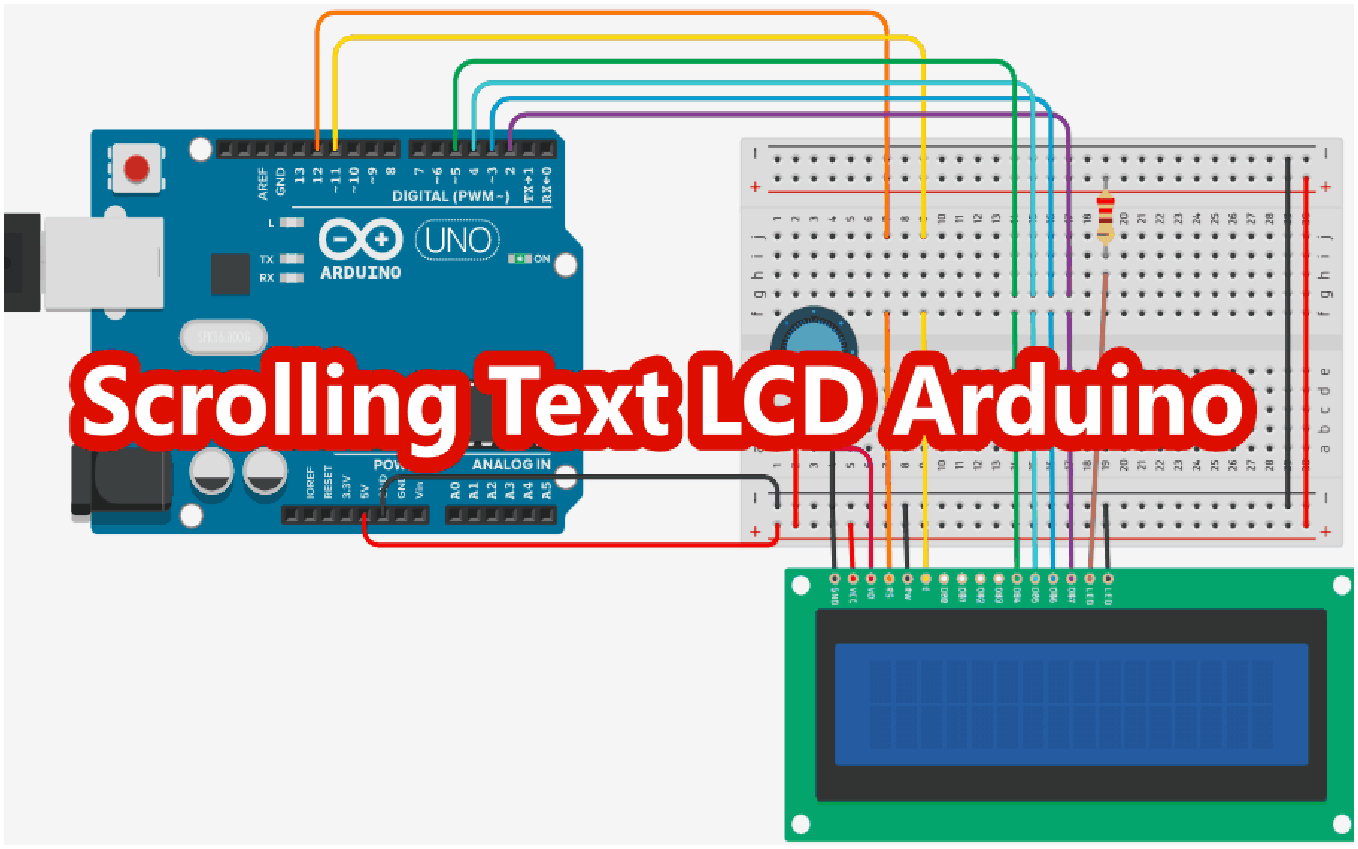
### 8.3.Scolling Text Arduino Code

Upload this code to Arduino IDE.

```
#include<LiquidCrystal.h>
LiquidCrystal lcd(12,11,5,4,3,2);
void setup() {
  lcd.begin(16,2);
  lcd.setCursor(5, 0);
  lcd.print("Micro Lab");
}
void loop()
{
  for(int i=0; i<5; i++)
  {
    lcd.scrollDisplayLeft();
    delay(600);
  }
  for(int i=0; i<5; i++)
 {
    lcd.scrollDisplayRight();
    delay(600);
  }

}
```

The output of code would be like this:



### 8.4.How Code Works?

This code is quite similar to the last example except for the LCD scrolling part inside the loop() function.

First, we set the cursor position to location (5,0) using cursor setting routine.

```
lcd.setCursor(5, 0);
```

After that, it prints the text “Micro Lab” on the current cursor position.

```
 lcd.print("Micro Lab");
```

As you know that if we call scrolling text left or right function of Arduino, they will move the text one one position towards left or right. Therefore, we used a loop to call these functions more than one time.

First, we call  lcd.scrollDisplayLeft() 5 times using for loop iteration 5 times. Therefore, it moves the text towards left for 5 positions.

```
 for(int i=0; i<5; i++)
{
    lcd.scrollDisplayLeft();
    delay(600);
  }
```

After that, we call  lcd.scrollDisplayRight() 5 times using for loop iteration 5 times. Therefore, it moves the text towards the right for 5 positions.

```
for(int i=0; i<5; i++)
{
    lcd.scrollDisplayRight();
    delay(600);
  }
```

## 9. Displaying Custom Characters on 16×2 LCD LCD using Arduino

In this section, we will learn to generate custom characters and learn to display custom characters on LCD using Arduino. Creating custom characters are useful when we want to display any symbol on LCD which is not available in the standard ASCII character set.

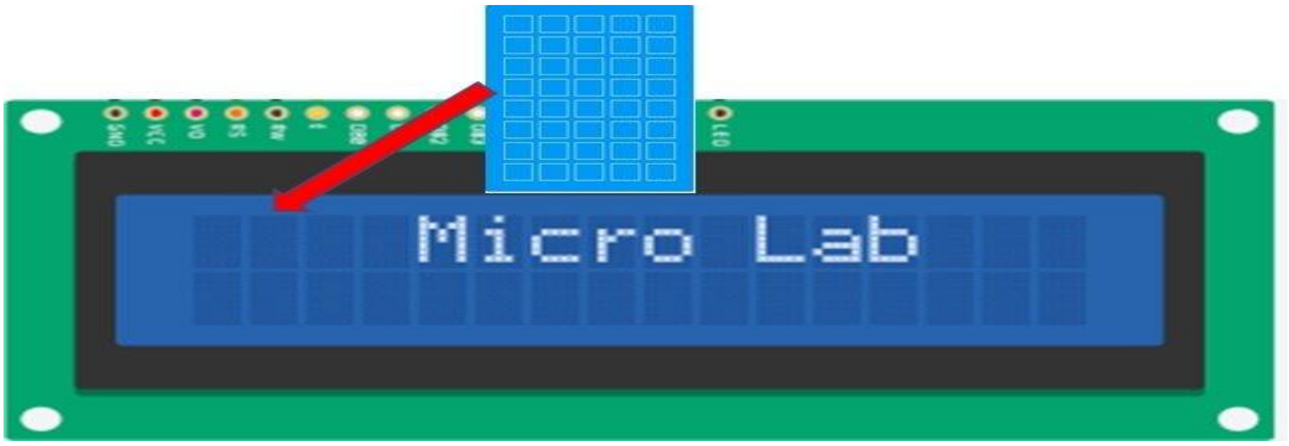
### 9.1. LCD Memory to store Custom Characters

As you know, the 16×2 LCD is based on the HD44780 controller. This controller has a butil-in character generation random access memory. CGRAM is used to display store user defined symbols or characters. Only 64 bytes of CGRAM is available. That means, due to limited memory, we can store a limited number of custom characters inside this memory.

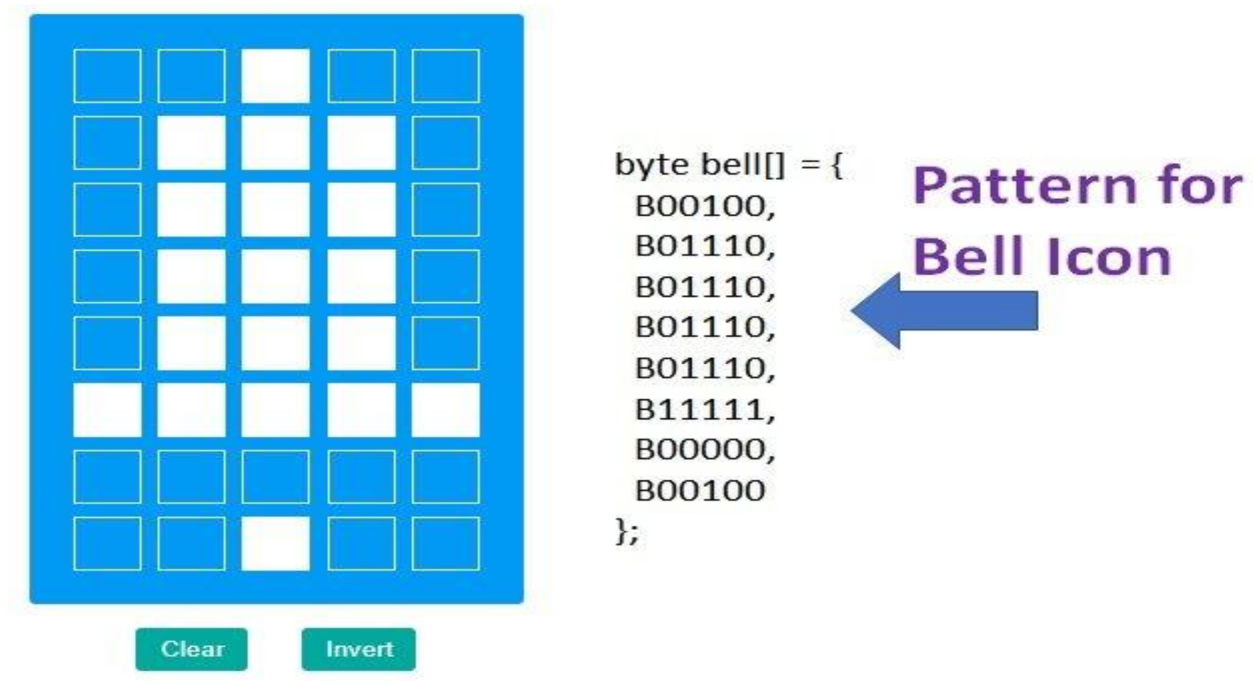
For example, a 16×2 LCD with 5×8 pixels for each position, only 8 custom characters can be stored and similarly for 5×10 pixel, type, only four can be stored. But still, this is enough memory. Because usually we need 1-2 user defined characters only such as battery. Sound, lock, bell symbol, etc.

### 9.2. Custom Characters Generation

Each location in the 16×2 LCD consists of 5×8 pixels. Therefore, to generate custom patterns, we must turn on and turn off individual dots according to the character that we want to display.



To create custom symbols patterns, we can use an application available on this link. Go to the above link and make any user-defined character by setting or clearing specific pixels from the 5×8 grid of pixels. Let’s say, we want to make a bell icon. Select pixels according to this picture and you will see the pattern code will be generated.



```
byte bell[] = {
  B00100,
  B01110,
  B01110,
  B01110,
  B01110,
  B11111,
  B00000,
  B00100
};
```

Similarly, you can generate as many user defined characters you want. But due to limited CGRAM, only 8 can be stored in the memory.

To display custom characters, first we must generate code for each character and after that store this value inside the CGROM of 16×2 LCD.We can store our custom generated characters inside CGRAM and load them to display on LCD.

## 9.3. Storing Character Inside CGRAM with Arduino

In order to store these patterns inside CGRAM, Arduino IDE provides a createChar() function. The input argument to this function is a pattern ( array of bytes) and an address. The address will be from 0-7. For example, we generated a pattern for a bell icon that is

```
byte bel[] = {
  B00100,
  B01110,
  B01110,
  B01110,
  B01110,
  B11111,
  B00000,
  B00100
};
```

To store this, we call lcd.Create() function like this

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

It will store the user-defined character bell on location '0' in CGRAM. Now if you want to write this letter on LCD, you can use lcd.Write() like this:

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

The important thing to note here is that we read character from the same location where lcd.Create() writes it.

## 9.4. Custom Characters Generation Arduino Code

Upload this code to Arduino. It displays user-defined symbols such as lock, sound, heart and bell on 16×2 LCD.

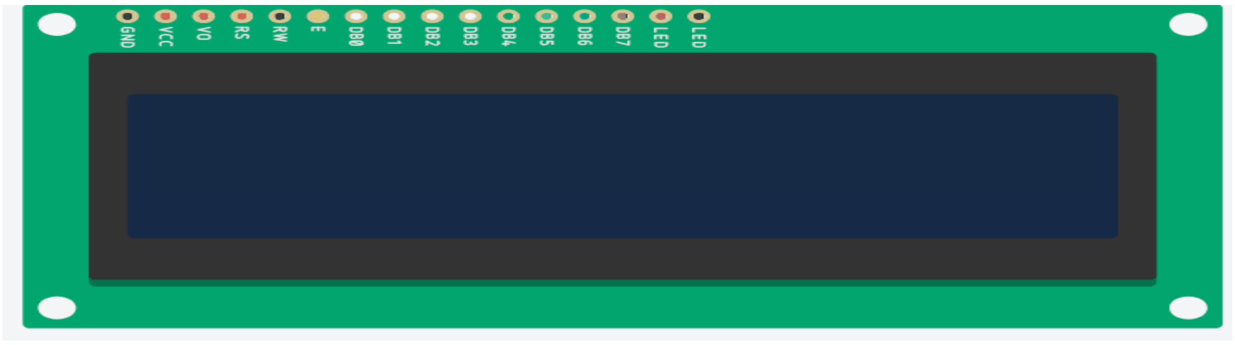
```
#include<LiquidCrystal.h>
LiquidCrystal lcd(12,11,5,4,3,2);
byte lock[8] = {
0b01110,
0b10001,
0b10001,
0b11111,
0b11011,
0b11011,
0b11111,
0b00000,};
byte sound[8] = {
0b00001,
0b00011,
0b00101,
0b01001,
0b01001,
0b01011,
0b11011,
0b11000} ;
byte speaker[8] = {0b00001,
0b00011,
0b01111,
0b01111,
0b01111,
0b00011,
0b00001,
0b00000,};
byte heart[8] = {0b00000,
0b01010,
0b11111,
0b11111,
0b01110,
0b00100,
0b00000,
0b00000 };
void setup() {
  lcd.clear();
  lcd.begin(16,2);
  lcd.createChar(0,lock);
  lcd.setCursor(0,0);
  lcd.write(byte(0));
  delay(600);
  lcd.createChar(1,sound);
  lcd.setCursor(4,0);
  lcd.write(byte(1));
  delay(600);
  lcd.createChar(2,speaker);
  lcd.setCursor(8,0);
  lcd.write(byte(2));
  delay(600);
  lcd.createChar(3,heart);
  lcd.setCursor(12,0);
  lcd.write(byte(3));
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

The output of this code will look like this:



### 9.5. Write Serial Data on 16×2 LCD using Arduino

In this section, we will see how to write data on 16×2 LCD which receives on a serial pin of Arduino. For example, we will send a text from the serial monitor to Arduino and as soon as the Arduino serial pin receives this data, it will be displayed on LCD.

### 9.6. Arduino Code

This LCD interfacing example with Arduino demonstrates how to write serial data from Arduino serial monitor directly on 16×2 LCD.

```
#include<LiquidCrystal.h>
LiquidCrystal lcd(12,11,5,4,3,2);
byte ch;
int col=0;
int row=0;
void setup() {
  Serial.begin(9600);
  lcd.begin(16,2);
  lcd.clear();
  // put your setup code here, to run once:

}
```

```
void loop() {
  if(Serial.available()){
    char ch=Serial.read();
    Serial.write(ch);
    Serial.println();
    lcd.setCursor(col,row);
    lcd.write(ch);
    col++;

  if(col>15){
    row++;
    col=0;
    lcd.write(ch);
  }
  // put your main code here, to run repeatedly:

}
if(ch=='*' ||row==1&&col>=15){
  lcd.clear();
  col=0;
  row=0;
}
}
```

The above code works similar to the previous examples except for the serial communication section. Here, the UART communication of Arduino is used to receive characters from a user via a serial terminal. You can use a serial terminal of Arduino to send the character to the LCD that you want to display on the LCD. If you want to read more about the UART communication library of Arduino, you can read this tutorial:

Applications

Liquid crystal display has many applications in embedded systems and digital electronics projects. Engineering students used LCDs in their projects to display various types of parameters. I have posted many articles on embedded system projects.

In all these projects I have used liquid crystal display to display various physical parameters like temperature, analog current, analog voltage, humidity, the intensity of light, moisture, and solar panel parameters. In all these projects, I have used LCD to display these parameters.

In summary, this tutotrial, we learned:

- Display text on LCD
- LCD Cursor Setting with Arduini
- Scrolling Text on 16×2  LCD using Arduino
- Displaying Custom Characters on 16×2 LCD LCD using Arduino
- A way Custom Characters to store LCD Memory
- Custom Characters Generation for 16×2 LCD and Arduino

- Write Serial Data on 16×2 LCD using Arduino

**10.**