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.

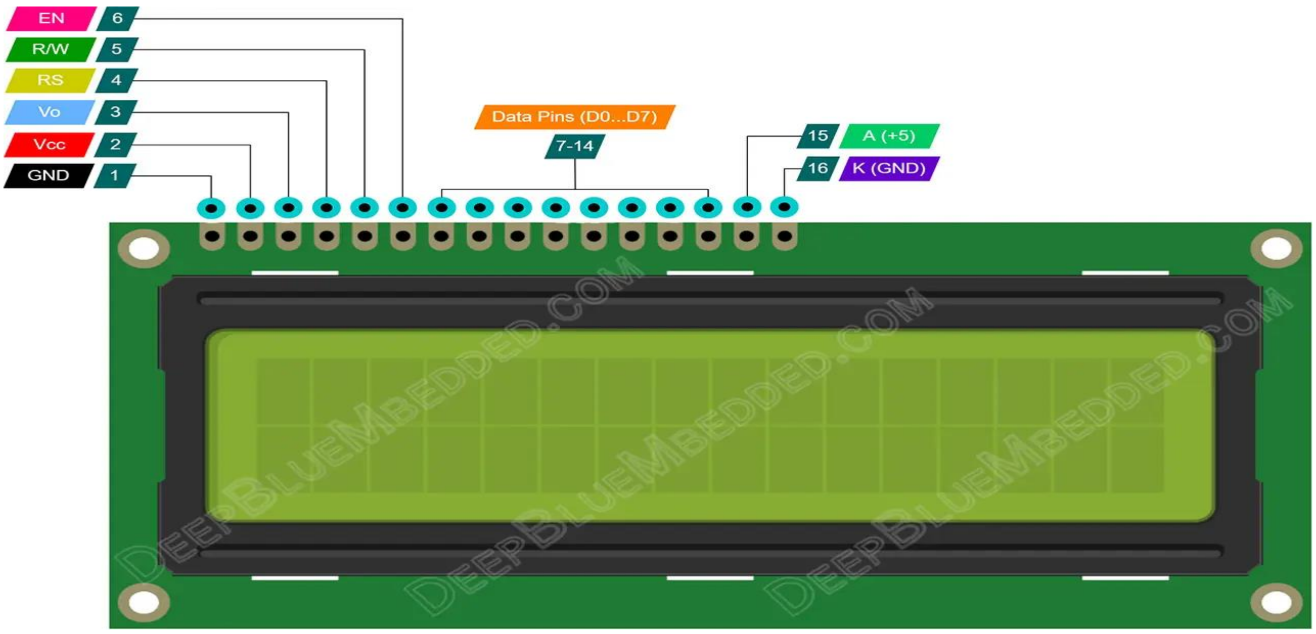
Alphanumeric LCDs are available in different units with different sizes (e.g. 16×1, 16×2, 16×4, 20×4, and so on). The backlight color of the LCD display is also available in multiple variants (green, yellow, blue, white, and red).

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.

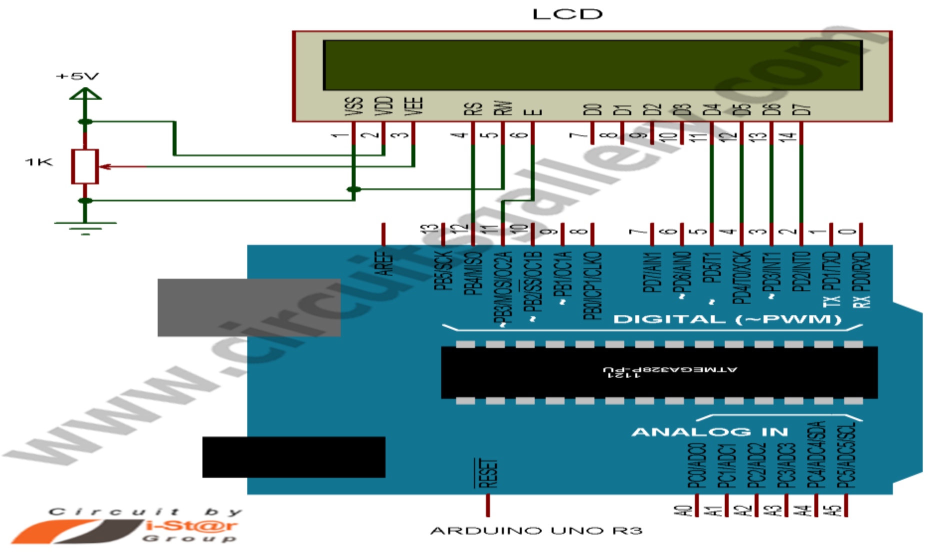
GND is the ground pin.
Vcc is the LCD's power supply input pin (connects to +5v).

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.



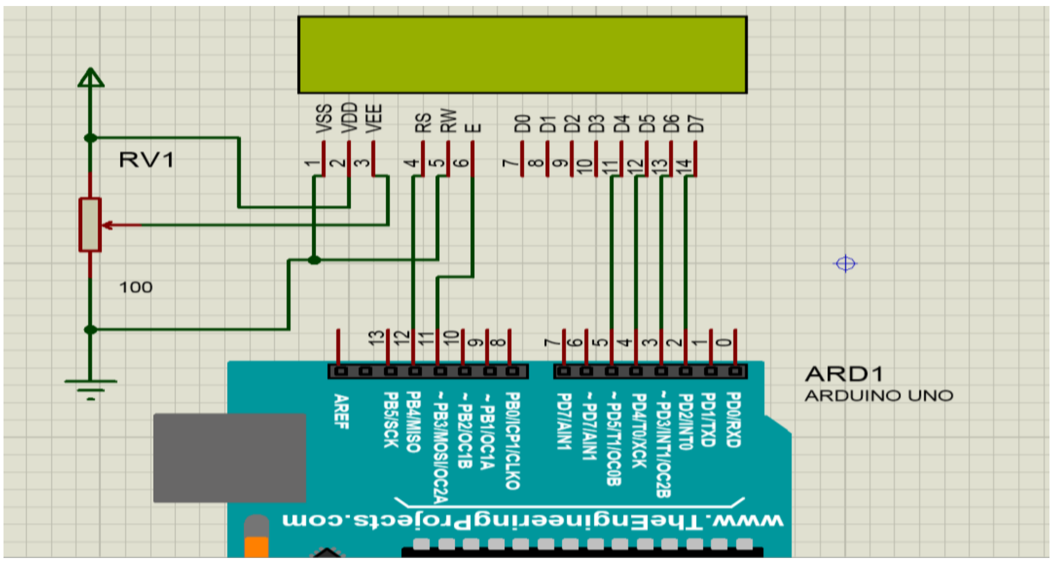
RS is the register select pin whose function is to separate the data (text and numbers) from the commands to LCD (like set cursor position, clear LCD display, shift text, etc).
RW is the Read/Write pin. Which determines the type of operation we'll be doing with the LCD (read or write). As we're typically 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. Circuit Schematic



4.3. 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.

4.4. 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.

```
/*LAB Name: Arduino LCD 16x2 Text Example
* Author: Khaled Magdy
* For More Info Visit: www.DeepBlueMbedded.com*/
#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
}
```

4.5. 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.

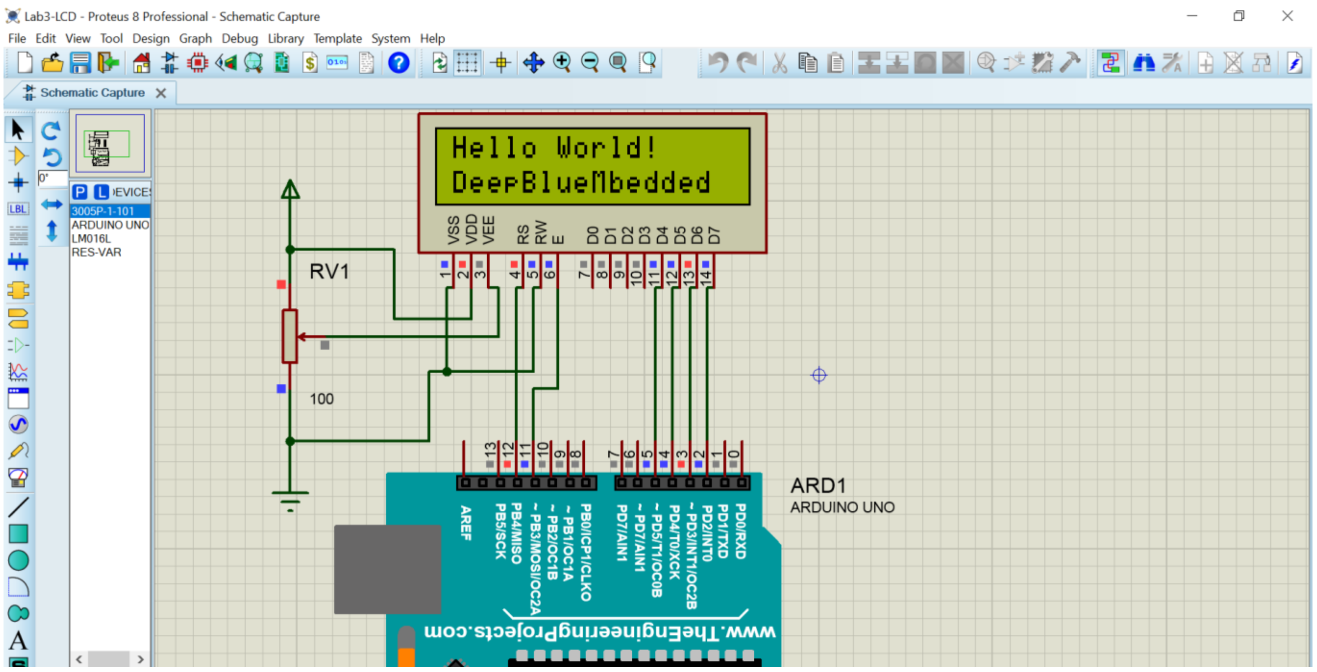
```
MyLCD.print("Hello World!");
```

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

4.6. 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

5. Arduino 16×2 LCD Display Variables Example

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.

5.1. Example code

```
    /* LAB Name: Arduino LCD 16x2 Variable Display
    * Author: Khaled Magdy
    * For More Info Visit: www.DeepBlueMbedded.com*/
    #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);
}
```

5.2. 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:");
```

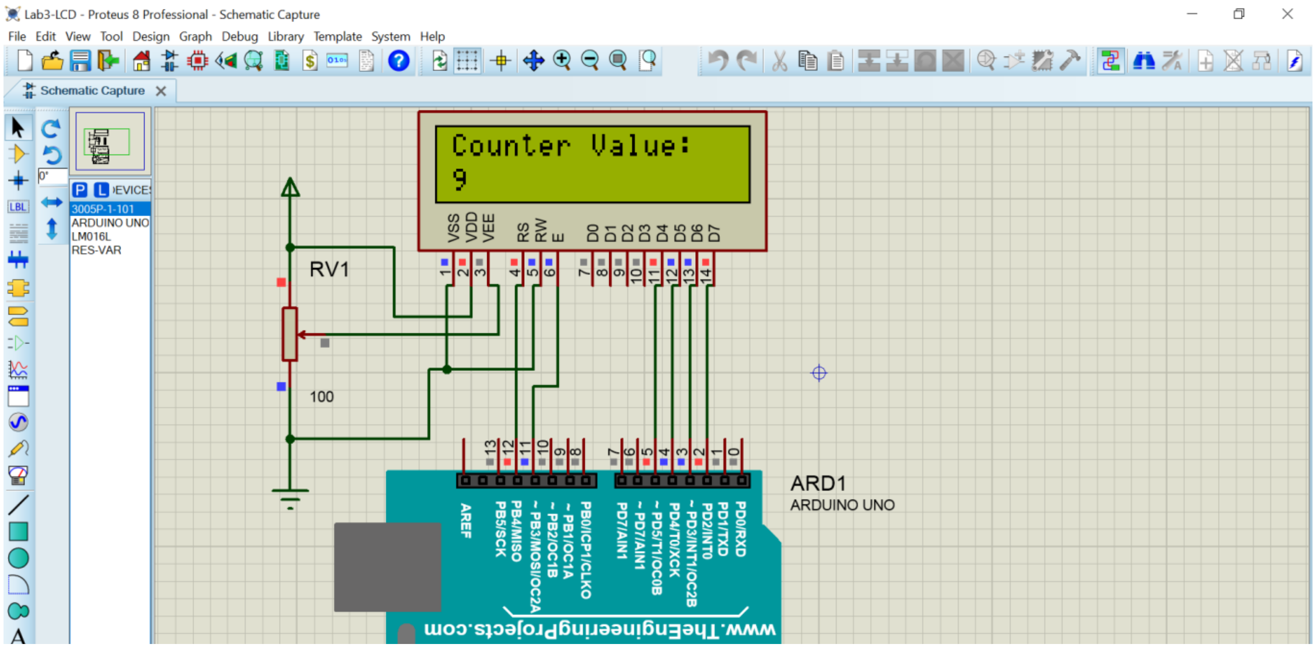
loop()

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.

5.3. 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
RV1
100
ARD1
ARDUINO UNO
www.TheEngineeringProjects.com

```
Counter Value:
9
```

6. 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.

6.1. Example Code

```
    /* LAB Name: Arduino LCD 16x2 Scrolling Text
    * Author: Khaled Magdy
    * For More Info Visit: www.DeepBlueMbedded.com*/
    #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);
        }
}
```

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 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.

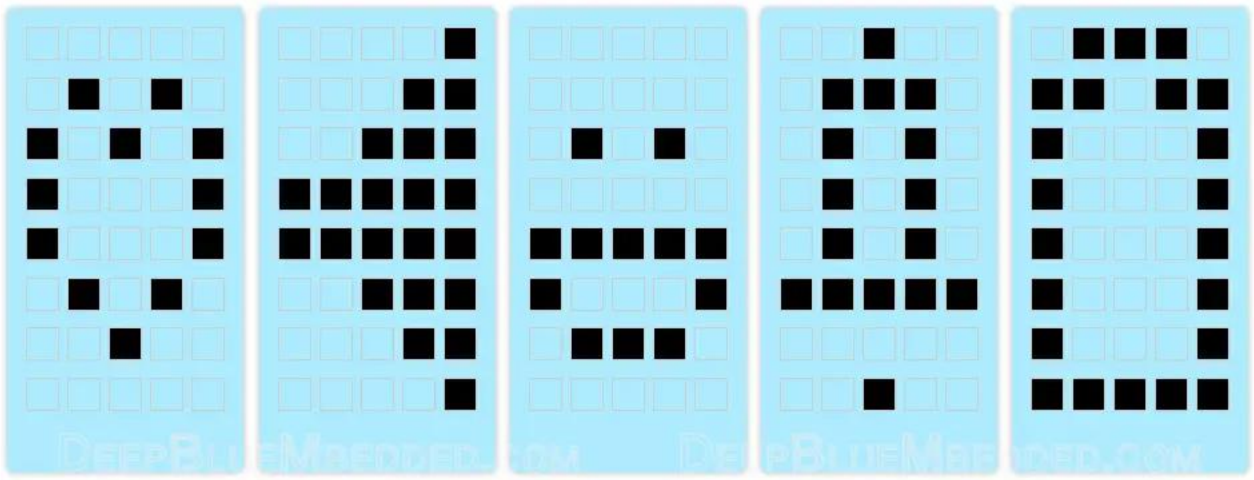
7. Arduino LCD Custom Character Generation & Display

In this example project, we'll create some custom characters (emojis and icons) and send them to the LCD display. It has an internal CGRAM that can hold up to 8 custom characters at maximum and you'll learn how to use it in this example project.

Arduino LCD Custom Character Generator

You can use this online LCD Custom Character Generator Tool and it'll give you the Arduino C-Code for it, which you can easily copy and paste into your project code. And here is how to use it:

Click on the pixels to draw your custom LCD character, you can invert or clear the entire display cell if you want with the buttons below. If you're satisfied with how your icon/emoji looks, you can copy the code and you're good to go. Here are some example custom characters generated by this tool.



I used my custom LCD character generator tool to create the above icons/emojis (heart, speaker, smiley face, notification bell, battery level indicator). We'll display all of those icons in this example project to show you how it's done in Arduino code.

```
/* LAB Name: Arduino LCD 16x2 Custom Characters Display
* Author: Khaled Magdy
* For More Info Visit: www.DeepBlueMbedded.com*/
#include <LiquidCrystal.h>
// Create An LCD Object. Signals: [ RS, EN, D4, D5, D6, D7 ]
LiquidCrystal MyLCD(12, 11, 5, 4, 3, 2);
// LCD Custom Characters
uint8_t HeartChar[] = {0x00, 0x00, 0x0a, 0x15, 0x11, 0x0a, 0x04, 0x00};
uint8_t SpeakerChar[] = {0x01, 0x03, 0x07, 0x1f, 0x1f, 0x07, 0x03, 0x01};
uint8_t SmileyFaceChar[] = {0x00, 0x00, 0x0a, 0x00, 0x1f, 0x11, 0x0e, 0x00};
uint8_t BellChar[] = {0x04, 0x0e, 0x0a, 0x0a, 0x0a, 0x1f, 0x00, 0x04};
uint8_t Battery1Char[] = {0x0e, 0x1b, 0x11, 0x11, 0x11, 0x11, 0x11, 0x1f};
uint8_t Battery2Char[] = {0x0e, 0x1b, 0x11, 0x11, 0x11, 0x11, 0x1f, 0x1f};
uint8_t Battery3Char[] = {0x0e, 0x1b, 0x11, 0x11, 0x11, 0x1f, 0x1f, 0x1f};
uint8_t Battery4Char[] = {0x0e, 0x1b, 0x11, 0x1f, 0x1f, 0x1f, 0x1f, 0x1f};
void setup()
{
  // Initialize The LCD. Parameters: [ Columns, Rows ]
  MyLCD.begin(16, 2);
  // Send The Custom Characters To LCD's CGRAM
  MyLCD.createChar(0, HeartChar);
  MyLCD.createChar(1, SpeakerChar);
  MyLCD.createChar(2, SmileyFaceChar);
  MyLCD.createChar(3, BellChar);
  MyLCD.createChar(4, Battery1Char);
  MyLCD.createChar(5, Battery2Char);
  MyLCD.createChar(6, Battery3Char);
  MyLCD.createChar(7, Battery4Char);
  // Clear The LCD Dispaly
  MyLCD.clear();
  MyLCD.print("Custom Characters:");
  // Print The Custom Characters
  MyLCD.setCursor(0, 1);
  MyLCD.write(byte(0));
  MyLCD.setCursor(2, 1);
  MyLCD.write(byte(1));
  MyLCD.setCursor(4, 1);
  MyLCD.write(byte(2));
  MyLCD.setCursor(6, 1);
  MyLCD.write(byte(3));
  MyLCD.setCursor(8, 1);
  MyLCD.write(byte(4));
  MyLCD.setCursor(10, 1);
  MyLCD.write(byte(5));
  MyLCD.setCursor(12, 1);
  MyLCD.write(byte(6));
  MyLCD.setCursor(14, 1);
  MyLCD.write(byte(7));
}
void loop()
{
  // DO NOTHING!
}
```

8. **Arduino LiquidCrystal Library Useful Function**
   Here is a summarized list for the most important and useful functions in the LCD LiquidCrystal.h library that you'd need to use in your Arduino projects.

LiquidCrystal Library Function	**Function Description**
lcd.begin(cols, rows);	Initializes the interface to the LCD screen, and specifies the dimensions (width and height) of the display. the .begin() function needs to be called before any other LCD library commands.
lcd.clear();	Clears the LCD screen and points the cursor to the home position (in the upper-left corner).
lcd.home();	Sets the cursor to the home position (in the upper-left of the LCD) without clearing the text on the display.

lcd.setCursor(col, row);	Position the LCD cursor to a desired location.
lcd.noCursor();	Hides the LCD cursor.
lcd.cursor();	Display the LCD cursor if it has been hidden.
lcd.blink();	Display the blinking LCD cursor. If used in combination with the cursor() function, the result will depend on the particular display device.
lcd.noBlink();	Turns off the blinking LCD cursor.
lcd.noDisplay();	Turns off the LCD display, without losing the text currently shown on it. This can be done to save some power if needed.
lcd.display();	Turns on the LCD display, after it's been turned off with the noDisplay() function. This will restore the text (and cursor) that was on the display.
lcd.scrollDisplayLeft();	Scrolls the contents of the display (text and cursor) one space to the left.
lcd.scrollDisplayRight();	Scrolls the contents of the display (text and cursor) one space to the right.
lcd.autoscroll();	Turns on automatic scrolling of the LCD. This causes each character output to the display to push previous characters over by one space. If the current text direction is left-to-right (the default), the display scrolls to the left; if the current direction is right-to-left, the display scrolls to the right. This has the effect of outputting each new character to the same location on the LCD.
lcd.noAutoscroll();	Turns off the automatic scrolling feature of the LCD.
lcd.leftToRight();	Set the direction for text written to the LCD to left-to-right, the default. This means that subsequent characters written to the display will go from left to right, but does not affect previously-output text.
lcd.rightToLeft();	Set the direction for text written to the LCD to right-to-left (the default is left-to-right). This means that subsequent characters written to the display will go from right to left, but does not affect previously-output text.

9. Introduciton to Keypad

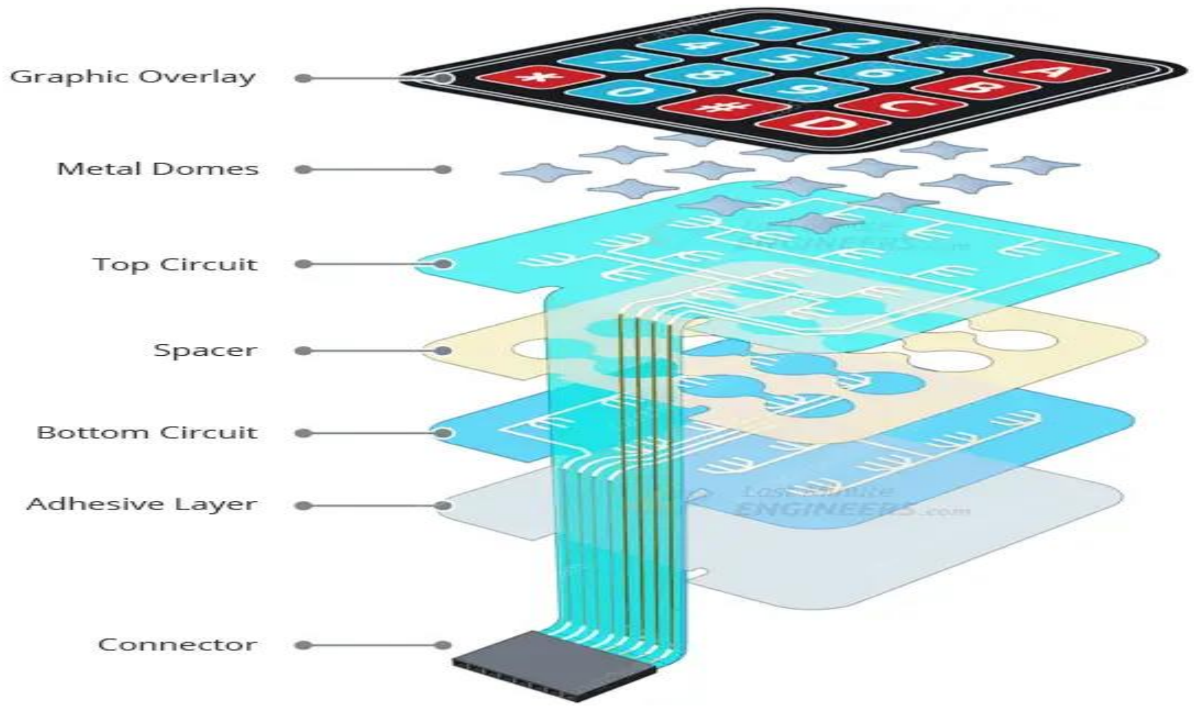
Membrane keypads are an excellent starting point for adding key input to a project because they are inexpensive, long-lasting, and resistant to water. And knowing how to interface them with an Arduino is extremely useful for building a variety of projects that require user input for menu selection, password entry, or robot operation.

Membrane keypads come in a variety of sizes, the most common of which are the 4×3 keypad (12 keys) and the 4×4 keypad (16 keys). They have a layout similar to that of a standard telephone keypad, making them easy to use for anyone.

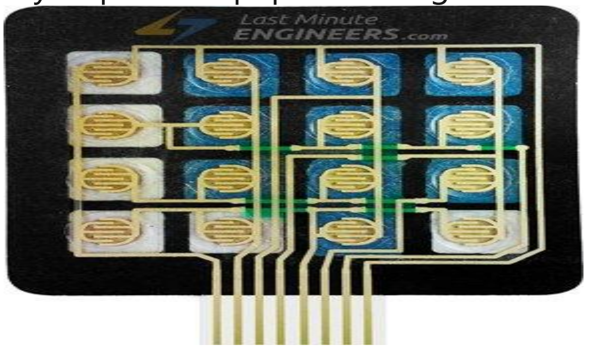
No matter what size they are, they all function identically. Let's learn more about Membrane keypads.

Membrane Keypad Construction

Membrane keypads are made of a thin, flexible membrane material and typically have six layers:



Graphic Overlay – Graphic overlays are typically made of polyester because it has better flex life than polycarbonate.
Metal Domes – This layer houses metal domes or polydomes that provide tactile feedback.
Top Circuit Layer – This is typically a polyester printed layer with silver-filled electrically conductive inks. This layer terminates as a flexible tail that connects to the outside world.
Spacer – This layer separates the top and bottom circuits, allowing the switch to remain normally open until the keypad is pressed.
Bottom Circuit Layer – This is typically a polyester printed layer with silver-filled electrically conductive inks. This layer also terminates as a flexible tail.
Rear Adhesive Layer – This layer sticks the keypad to almost anything, which is convenient.
If you peel the paper backing off the keypad, you can see how it is made.

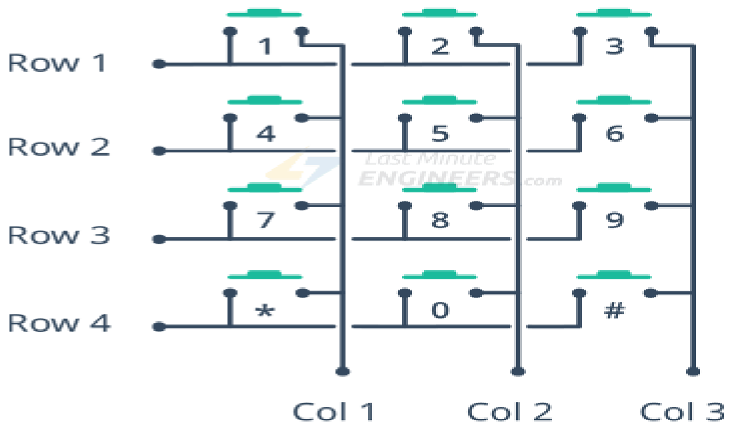


In order to reduce the number of I/O connections, as you can see, all rows and columns are wired together. If this were not the case, interfacing 16 individual pushbuttons, for example, would require 17 I/O pins, one for each pushbutton and one for a common ground. By connecting rows and columns, only 8 pins are required to control the entire 4×4 keypad. This technique of controlling a large number of inputs using fewer pins is known as Multiplexing.

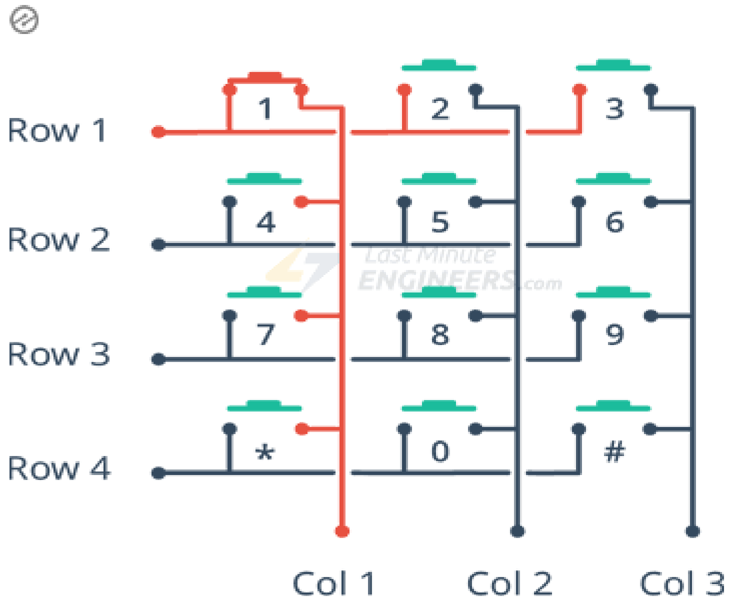
9.1. How Does the Matrix Keypad Work?

The matrix keypad consists of pushbutton contacts that are connected to the row and column lines. There is one pin for each column and one pin for each row. So the 4×4 keypad has 4 + 4 = 8 pins, while the 4×3 keypad has 4 + 3 = 7 pins.

This illustration of a basic 4×3 keypad arrangement demonstrates how the internal conductors connect the rows and columns.



When the button is pressed, one of the rows is connected to one of the columns, allowing current to flow between them. When the key '4' is pressed, for instance, column 1 and row 2 are connected.



By identifying which column and row are connected, we can determine which button has been pressed.

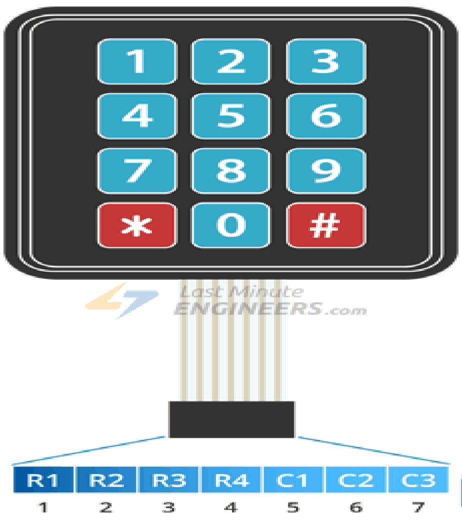
9.2. Keypad Scanning

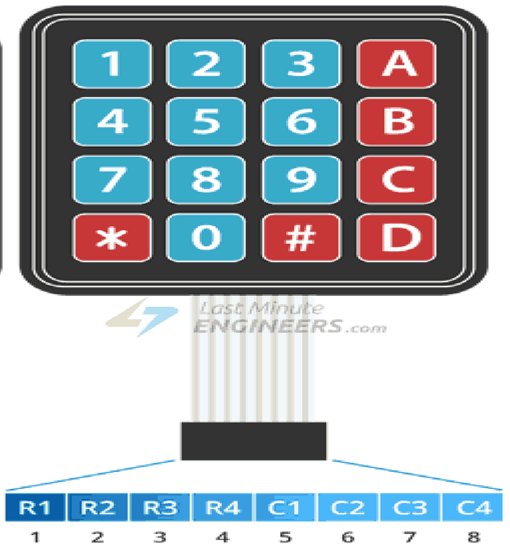
Here is how a microcontroller scans rows and columns to identify which button has been pressed.

1. Each row is connected to an input pin, and each column is connected to an output pin.
2. Input pins are pulled HIGH by enabling internal pull-up resistors.
3. The microcontroller then sequentially sets the pin for each column LOW and then checks to see if any of the row pins are LOW. Because pull-up resistors are used, the rows will be high unless a button is pressed.
4. If a row pin is LOW, it indicates that the button for that row and column is pressed.
5. The microcontroller then waits for the switch to be released. It then searches the keymap array for the character that corresponds to that button.

4×3 and 4×4 Membrane Keypad Pinout

The keypad has a female Dupont connector. When looking at the front of the keypad, the row pins are on the left, and they usually have a dark strip near the connector to help identify them. The pinouts are as follows:



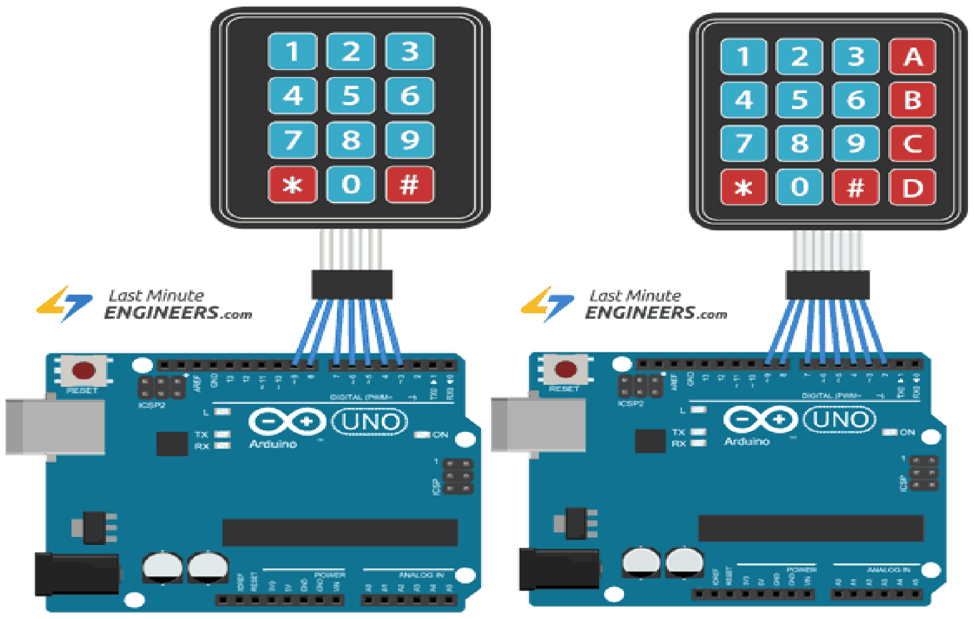


Connecting a 4×3 and a 4×4 Membrane Keypad to an Arduino

Now that we know everything about the membrane keypad, we can start connecting it to Arduino.

The connection is quite straightforward, as the Arduino connections are made in the same order as the keypad connector. Begin by connecting keypad pin 1 to Arduino digital pin 9. And continue doing the same with the subsequent pins (2 to 8, 3 to 7, and so on).

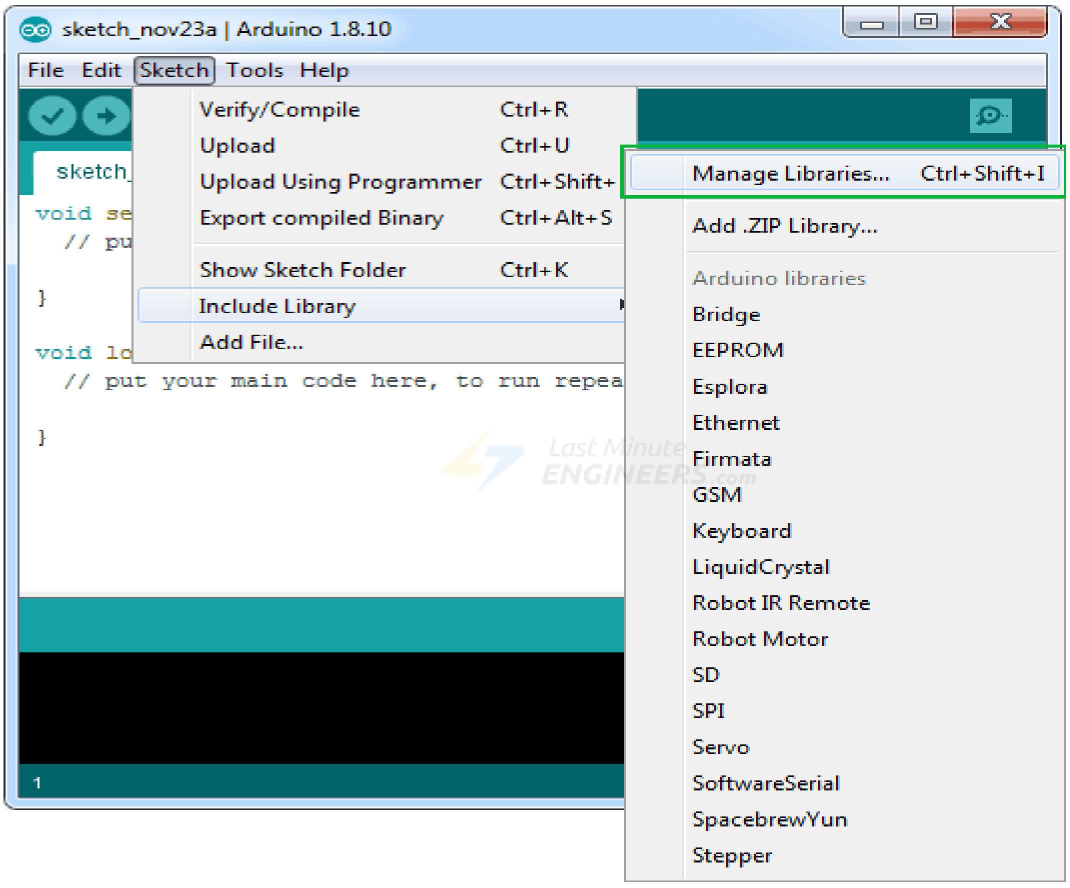
The most convenient way to connect everything is to use an 8-pin male-to-male Dupont ribbon cable.



Installing Keypad Library

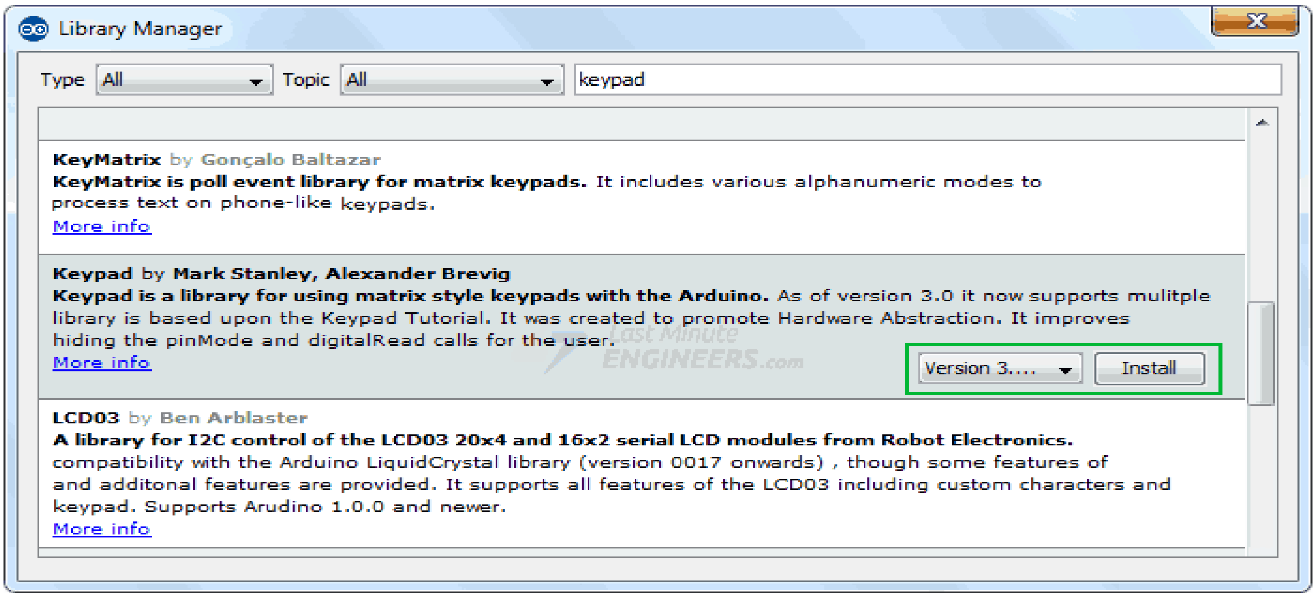
To determine which key was pressed, we must continuously scan rows and columns. Fortunately, [Keypad.h]() was written to abstract away this unnecessary complexity.

To install the library, navigate to Sketch > Include Library > Manage Libraries... Wait for the Library Manager to download the libraries index and update the list of installed libraries.



```
void se
  // pu

}

void lo
  // put your main code here, to run repea

}
```

Filter your search by entering ‘keypad’. Look for Keypad by Mark Stanley, Alexander Brevig. Click on that entry and then choose Install.


Library Manager

Type All Topic All keypad

KeyMatrix by Gonçalo Baltazar
KeyMatrix is poll event library for matrix keypads. It includes various alphanumeric modes to process text on phone-like keypads.
More info

Keypad by **Mark Stanley, Alexander Brevig**
Keypad is a library for using matrix style keypads with the Arduino. As of version 3.0 it now supports mulitple library is based upon the Keypad Tutorial. It was created to promote Hardware Abstraction. It improves hiding the pinMode and digitalRead calls for the user.
More info

LCD03 by Ben Arblaster
A library for I2C control of the LCD03 20x4 and 16x2 serial LCD modules from Robot Electronics. compatibility with the Arduino LiquidCrystal library (version 0017 onwards) , though some features of and additonal features are provided. It supports all features of the LCD03 including custom characters and keypad. Supports Arudino 1.0.0 and newer.
More info

9.3. Arduino Example Code

The basic sketch below will print key presses to the Serial Monitor.

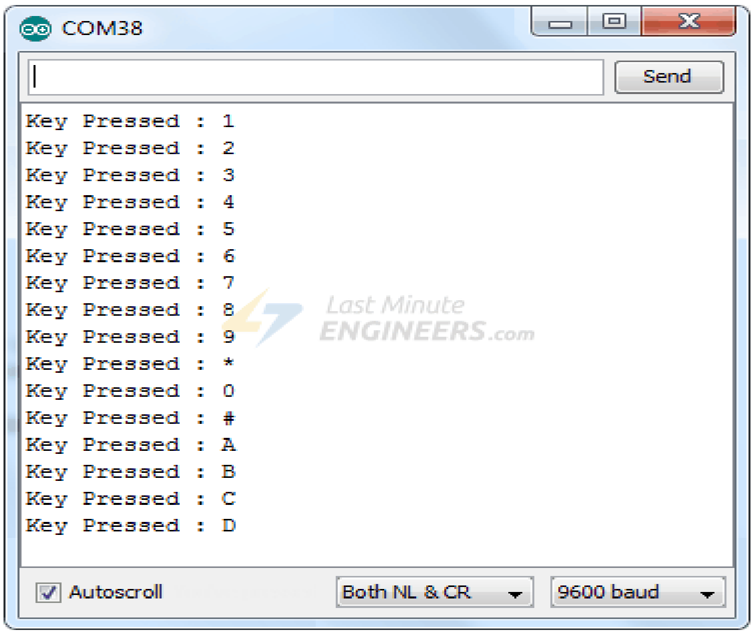
Code for 4×3 Keypad

```
#include <Keypad.h>
const byte ROWS = 4; //four rows
const byte COLS = 3; //three columns
char keys[ROWS][COLS] = {
  {'1','2','3'},
  {'4','5','6'},
  {'7','8','9'},
  {'*','0','#'}
};
byte rowPins[ROWS] = {9, 8, 7, 6}; //connect to the row pinouts of
the keypad
byte colPins[COLS] = {5, 4, 3}; //connect to the column pinouts of
the keypad
//Create an object of keypad
Keypad keypad = Keypad( makeKeymap(keys), rowPins, colPins, ROWS,
COLS );
void setup(){
  Serial.begin(9600);
}
void loop(){
  char key = keypad.getKey();// Read the key
  // Print if key pressed
  if (key){
    Serial.print("Key Pressed : ");
    Serial.println(key);
  }
}
```

Code for 4×4 Keypad

```
#include <Keypad.h>
const byte ROWS = 4; //four rows
const byte COLS = 4; //four columns
char keys[ROWS][COLS] = {
  {'1','2','3','A'},
  {'4','5','6','B'},
  {'7','8','9','C'},
  {'*','0','#','D'}
};
byte rowPins[ROWS] = {9, 8, 7, 6}; //connect to the row pinouts of
the keypad
byte colPins[COLS] = {5, 4, 3, 2}; //connect to the column pinouts
of the keypad
//Create an object of keypad
Keypad keypad = Keypad( makeKeymap(keys), rowPins, colPins, ROWS,
COLS );
void setup(){
  Serial.begin(9600);
}
void loop(){
  char key = keypad.getKey();// Read the key
  // Print if key pressed
  if (key){
    Serial.print("Key Pressed : ");
    Serial.println(key);
  }
}
```

After loading the sketch, open your serial monitor at 9600 baud. Now, press some keys on the keypad; the serial monitor should display the key values.



```
Key Pressed : 1
Key Pressed : 2
Key Pressed : 3
Key Pressed : 4
Key Pressed : 5
Key Pressed : 6
Key Pressed : 7
Key Pressed : 8
Key Pressed : 9
Key Pressed : *
Key Pressed : 0
Key Pressed : #
Key Pressed : A
Key Pressed : B
Key Pressed : C
Key Pressed : D
```

Code Explanation

The sketch begins by including the Keypad.h library and defining constants for the number of rows and columns on the keypad. If you're using a different keypad, modify these constants accordingly.

```
#include <Keypad.h>
const byte ROWS = 4; //four rows
const byte COLS = 4; //four columns
```

Following that, we define a 2-dimensional keymap array `keys[ROWS][COLS]` that contains characters to be printed when a specific keypad button is pressed. In our sketch, the characters are laid out exactly as they appear on the keypad.

```
char keys[ROWS][COLS] = {
  {'1','2','3','A'},
  {'4','5','6','B'},
  {'7','8','9','C'},
  {'*','0','#','D'}
};
```

However, you can define these to be anything you want. For example, if you intend to create a calculator project, simply change the array definition to this:

```
char keys[ROWS][COLS] = {
  {'1','2','3','4'},
  {'5','6','7','8'},
  {'9','0','+','-'},
  {'.','*','/','='}
};
```

Two more arrays are defined. These arrays specify the Arduino connections to the keypad's row and column pins.

```
byte rowPins[ROWS] = {9, 8, 7, 6}; //connect to the row pinouts of
the keypad
```

```
byte colPins[COLS] = {5, 4, 3, 2}; //connect to the column pinouts
of the keypad
```
Next, we create a keypad library object. The constructor accepts five arguments.

- `makeKeymap(keys)` initializes the internal keymap to be equal to the user defined keymap.
- `rowPins` and `colPins` specify the Arduino connections to the keypad's row and column pins.
- `ROWS` and `COLS` represent the keypad's rows and columns.

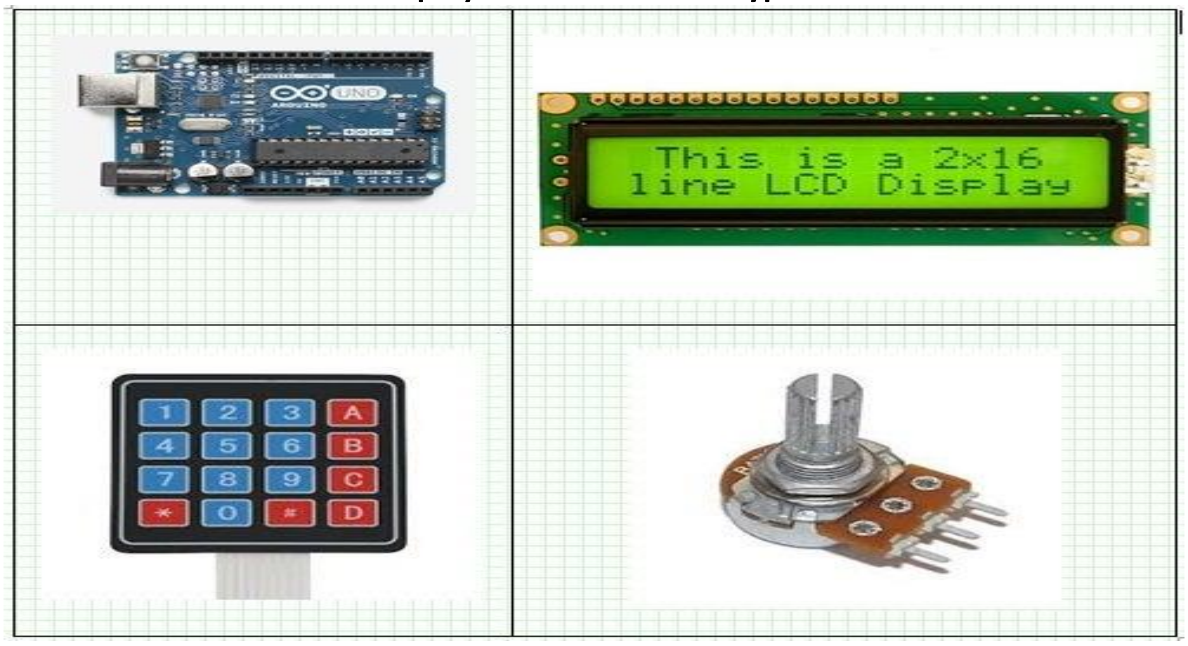
```
//Create an object of keypad
Keypad keypad = Keypad( makeKeymap(keys), rowPins, colPins, ROWS,
COLS );
```
In the Setup, we initialize the serial port.
```
void setup(){
  Serial.begin(9600);
}
```
In the Loop, we use the `getKey()` method to get a key value when a keypress is detected. Then we print it to the serial monitor.
```
void loop(){
  char key = keypad.getKey();// Read the key

  // Print if key pressed
  if (key){
    Serial.print("Key Pressed : ");
    Serial.println(key);
  }
}
```
9.4. Some useful functions of the Keypad library

There are many useful functions you can use with the Keypad object. Some of them are listed below:

- char waitForKey() waits forever until someone presses a key. Warning – It blocks all other code until a key is pressed. That means no blinking LED's, no LCD screen updates, no nothing with the exception of interrupt routines.
- KeyState getState() returns the current state of any of the keys. The four states are IDLE, PRESSED, RELEASED and HOLD.
- boolean keyStateChanged() let's you know when the key has changed from one state to another. For example, instead of just testing for a valid key you can test for when a key was pressed.
- setHoldTime(unsigned int time) sets the amount of milliseconds the user will have to hold a button until the HOLD state is triggered.
- setDebounceTime(unsigned int time) sets the amount of milliseconds the keypad will wait until it accepts a new keypress/keyEvent.

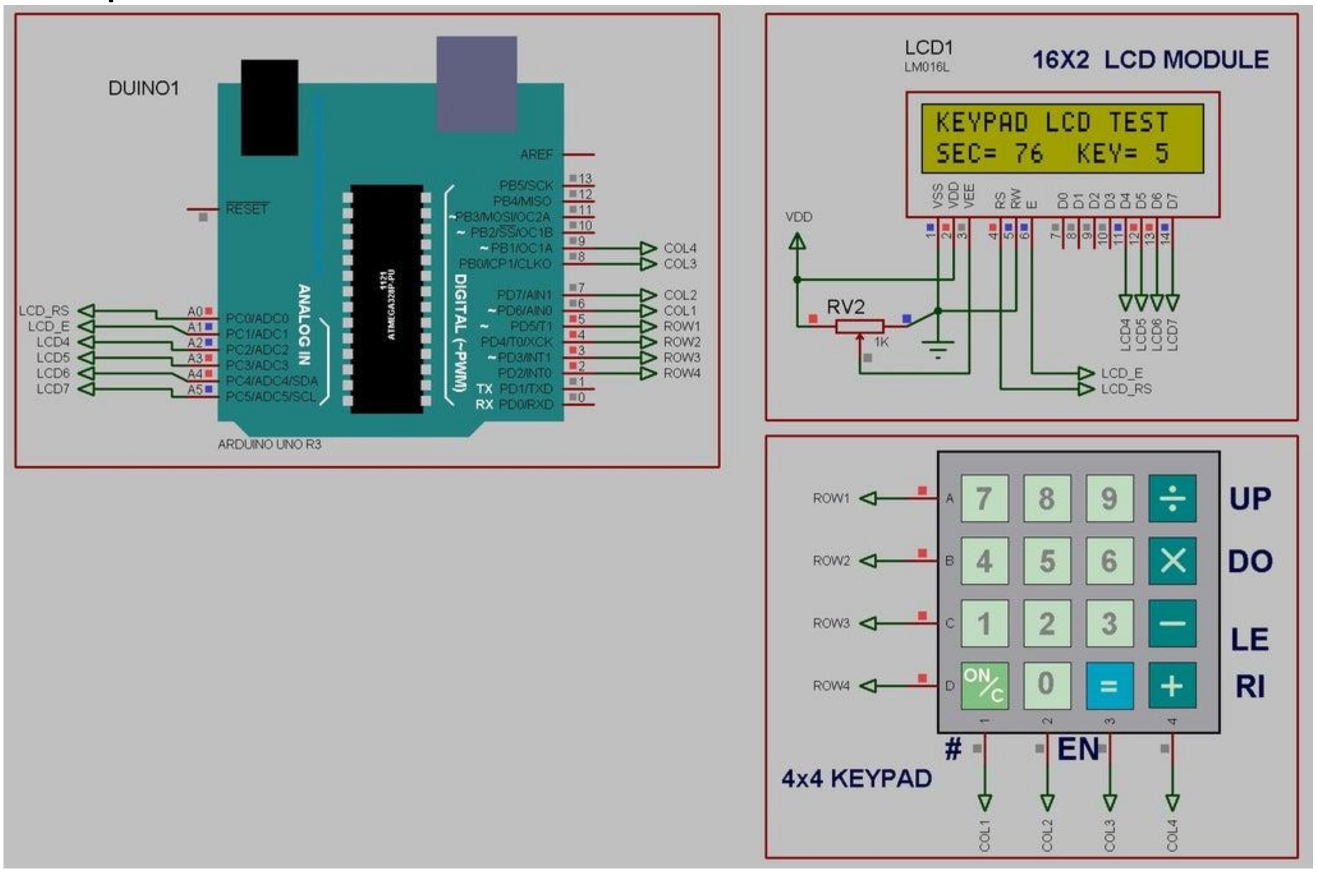
- addEventListener(keypadEvent) triggers an event if the keypad is used.

9.5. Arduino 16x2 LCD Display and 4x4 Matrix Keypad



In Embedded system design, matrix keypad (4x4, 4x3, 3x3 or 5x5) is used for key in the user inputs. Similarly character LCD display [16x2, 16x4, 20x2 or 20x4 LCDs] is used for indicating the system status / parameters. This intractable is about interfacing 16×2 LCD and 4x4 matrix keypad with Arduino microcontroller.

9.6. Step 1: Schematic



- The 16x2 is very common type LCD, with two rows, and each row displays 16 characters of either 5x7 or 5x8 dot matrix characters.
- The LCD is available in a 16 pin package. It consists of back light and contrast adjustment function and each dot matrix has 5×8 dot resolution.
- The 16x2 LCD display is connected to the Arduino (A0,A1,A2,A3,A4,A5) analog IO pins, where those pins are configured as digital in / out, where LCD operates at 4 bit data mode.
- If the display is not visible, adjust the Contrast pot (1K), to make it visible.
- Matrix key pad is arranged by push button switches in rows and columns.
- In a simple technique, the 16 keys of matrix keypad is connected with 8 digital IOpins of Arduino.
- Usually the keypad scan procedure is
- The key is decoded through Column selection / Row read.
- Write HIGH to Column One. And keep rest of the Column to LOW.
- Scan (Read) the Row One to Row Four, to find the key.
- Repeat until a key press (are multiple) is identified.
- The Row pins are connected to 5,4,3 and 2nd digital IO pins of Arduino.
- The Column pins are connected to 6,7,8 and 9th digital IO pins of Arduino.

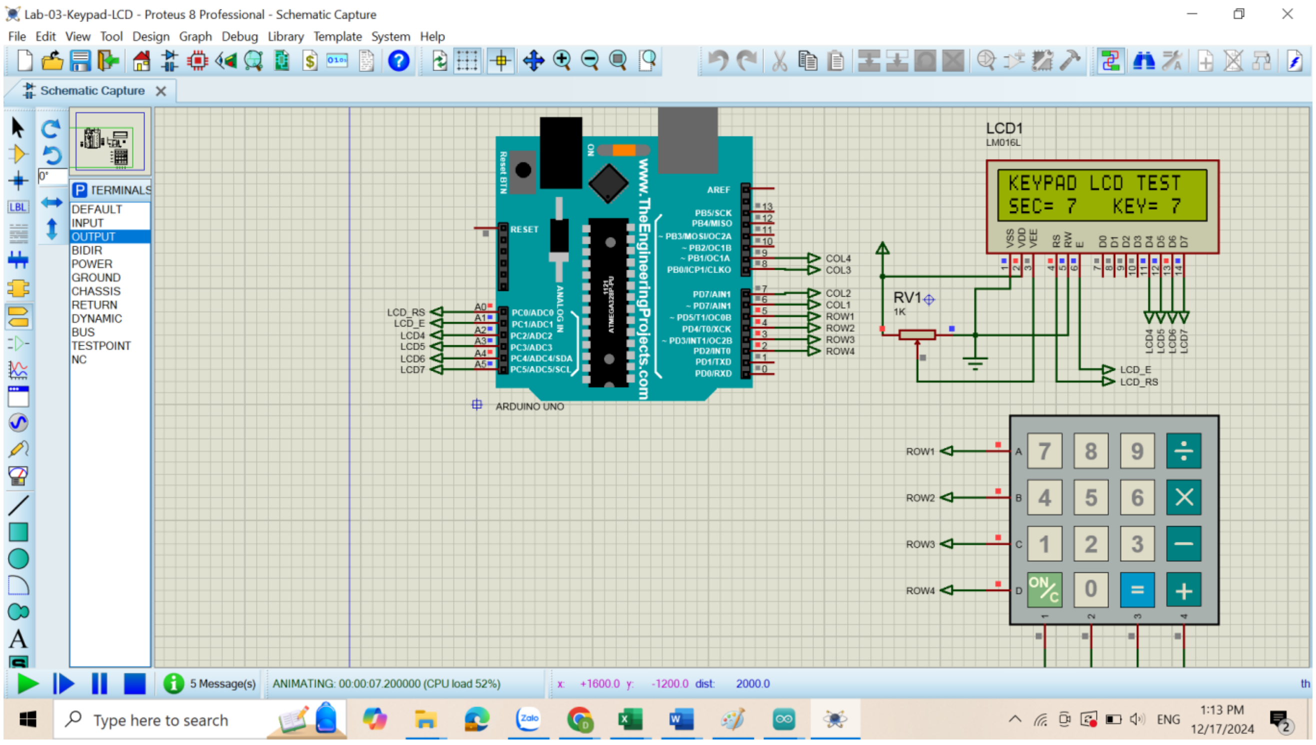
9.7. Step 2: Software

The Arduino LiquidCrystal and Keypad library is used for displaying the status and detecting key press.
The LCD displays the “KEYPAD LCD TEST” as welcome screen at the first line
Number of seconds since start “SEC= 123” is displayed at the second line column one.
The detected key is displayed at (“KEY= 5”) second line column 9 on wards.

```
/*Demonstrates the use a 16x2 LCD display and 4x4 LCD display.  T
   The  Arduino circuit connection for LCD:
   * LCD RS  pin to analog pin A0
   * LCD  Enable pin to analog pin A1
   * LCD D4  pin to analog pin A2
   * LCD D5  pin to analog pin A3
   * LCD D6  pin to analog pin A4
   * LCD D7  pin to analog pin A5
   The  Arduino circuit connection for MAtrix Key Pad:
   * ROW1 pin  to digital pin 5
   * ROW2 pin  to digital pin 4
   * ROW3 pin  to digital pin 3
   * ROW4 pin  to digital pin 2
   * COLUMN1  pin to digital pin 6
   * COLUMN2  pin to digital pin 7
   * COLUMN3  pin to digital pin 8
   * COLUMN4  pin to digital pin 9
   Name:-  M.Pugazhendi
   Date:-   04thJul2016
   Version:-  V0.1
   e-mail:-  muthuswamy.pugazhendi@gmail.com
```

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

   #include <Keypad.h>
   //initialize the library with the numbers of the interface pins
   LiquidCrystal lcd(A0,A1,A2,A3,A4,A5);
   //4x4 Matrix key pad
   const byte ROWS = 4; // Four rows
   const byte COLS = 4; // Four columns
   // Define the Keymap
   char keys[ROWS][COLS] =
   {
   {'7','8','9','/'},
   {'4','5','6','X'},
   {'1','2','3','-'},
   {'#','0','=','+'}
   };
   // Connect keypad ROW0, ROW1, ROW2 and ROW3 to Arduino pins.
   byte rowPins[ROWS] = { 5, 4, 3, 2 };
   // Connect keypad COL0, COL1, COL2 and COL3 to Arduino pins.
   byte colPins[COLS] = { 6, 7, 8, 9 };
   // Create the Keypad
   Keypad kpd = Keypad( makeKeymap(keys), rowPins,colPins, ROWS, COLS
 );
   void setup()
   {
     // set up the LCD's number of columns and rows:
   lcd.begin(16, 2);
     // Print a message to the LCD.
   lcd.print("KEYPAD LCD TEST");
   }
   void loop()
   {
     char key = kpd.getKey();
     // set the cursor to column 0, line 1
     // (note: line 1 is the second row, since counting begins with
 0):
   lcd.setCursor(0, 1);
     // print the number of seconds since reset:
   lcd.print("SEC= ");
   lcd.print(millis() / 1000);
     // Check for a valid key
   if(key)
     {        // set  the cursor to column 9, line 1
                      // (note: line 1 is the second row, since
 counting begins with 0):
   lcd.setCursor(9, 1);
   //Print the detected key
   lcd.print("KEY= ");
   lcd.print(key);
     }
   }
```

9.8. Result



9.9.