كلية الهندسة
قسم هندسة الروبوت والأنظمة الذكية

جَامعة المَنارة
MANARA UNIVERSITY

تطبيقات ميكاترونيك1

محاضرة 3

Interfacing Arduino with Liquid Crystal Displays

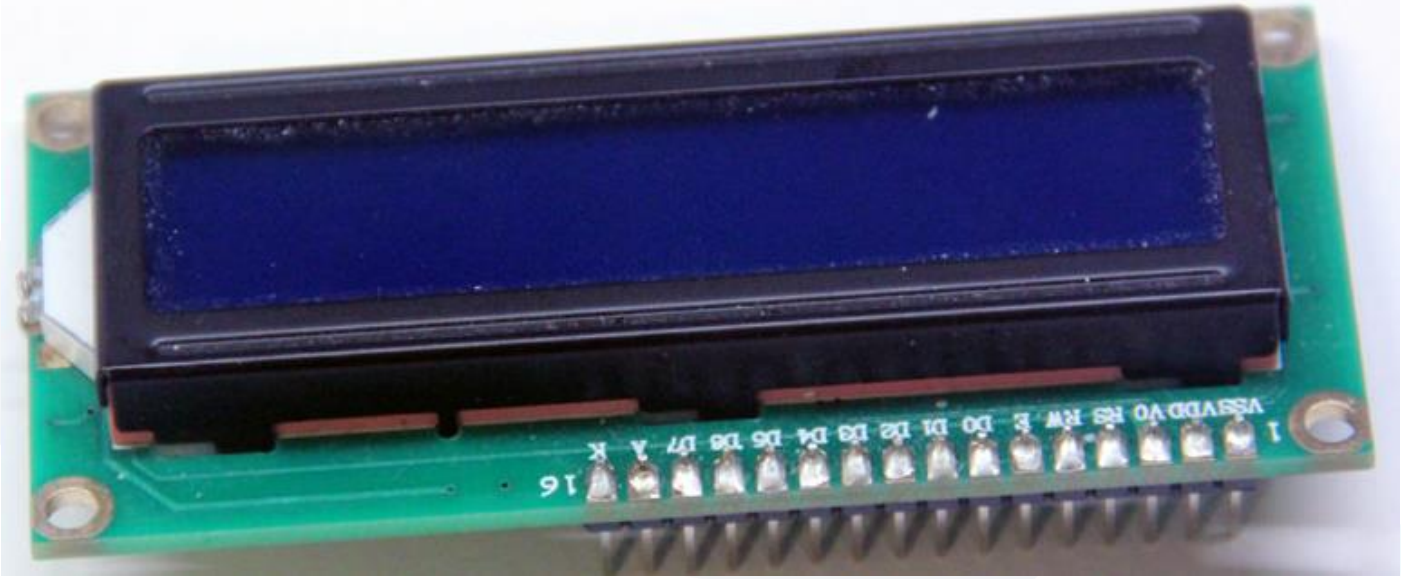
د. فادي متوج د.عيسى الغنام

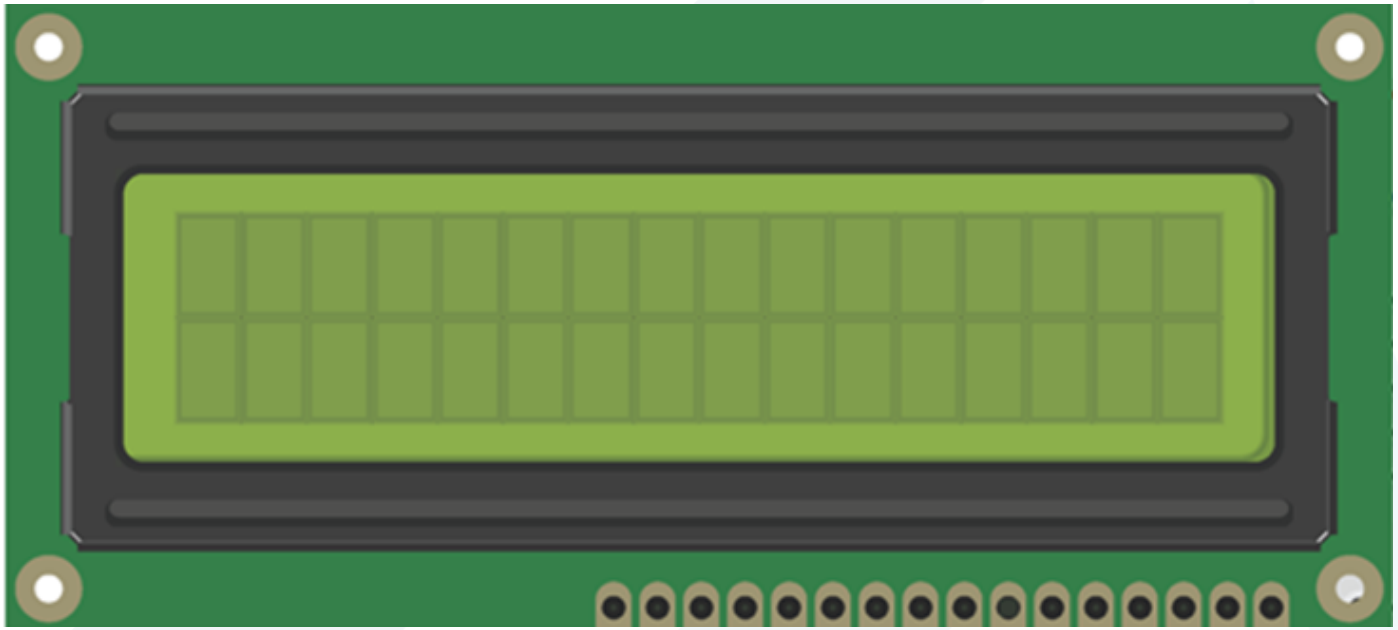
2024-2023

- One of the best things about designing **embedded systems** is the fact that they can operate independently of a computer.
- Until now, we have been tethered to the computer if we want to display any kind of information more complicated than an illuminated LED.
- By adding a **liquid crystal display (LCD)** to our Arduino, we can more easily display complex information (sensor values, timing information, settings, progress bars, etc.) directly on our Arduino project without having to interface with the serial monitor through the computer.

Setting Up the LCD

- We will use a parallel LCD screen. These are extremely common and come in all kinds of shapes and sizes.
- The most common is a **16x2** character display with a single row of **16 pins** (14 if it does not have a backlight).
- **16-pin LCD** display that can show a total of **32 characters** (16 columns and 2 rows).


16
A K
D7 D6 D5 D4 D3 D2 D1 D0 E RW RS V0 VDD VSS
1



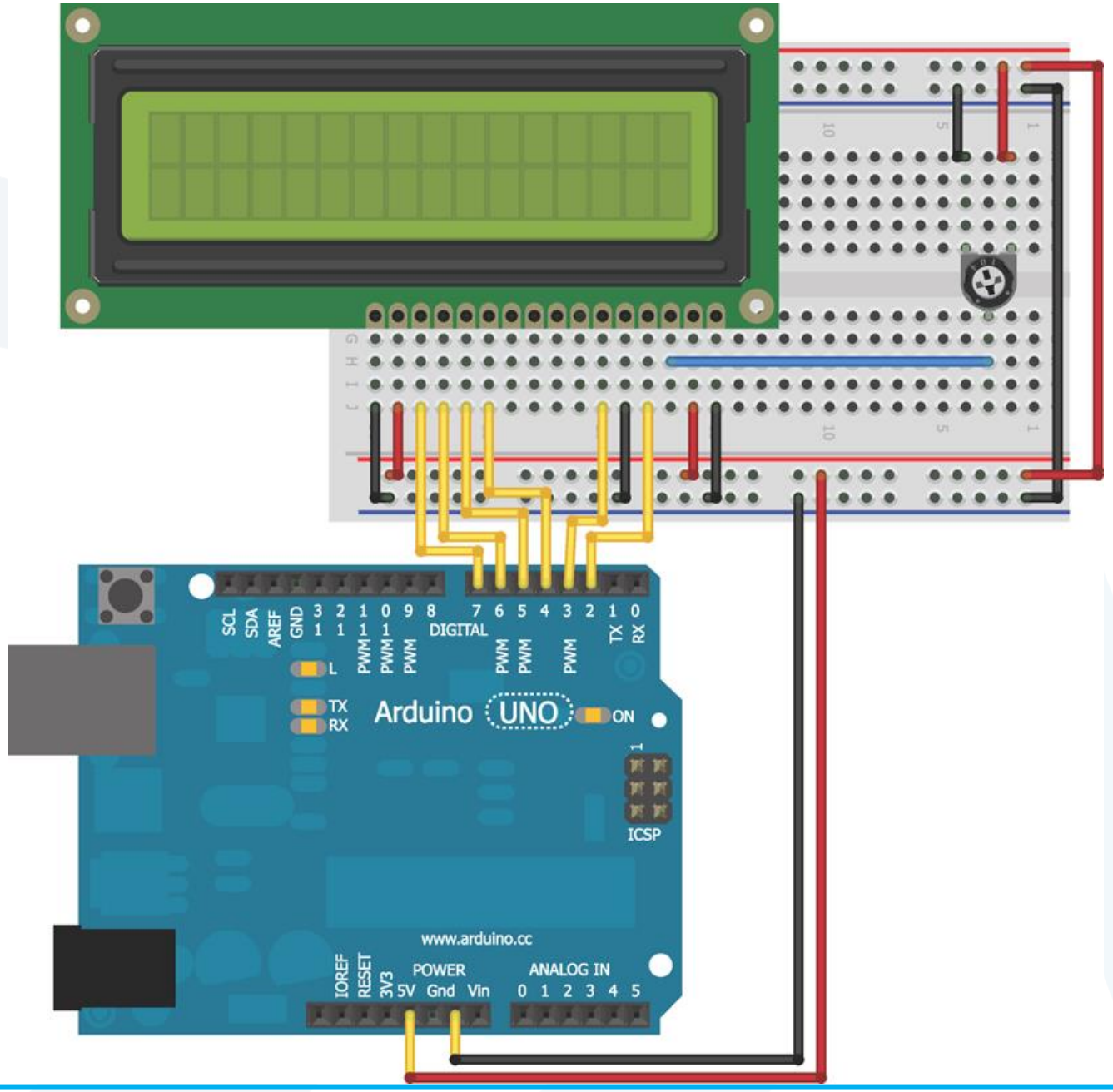
Parallel LCD Pins

PIN NUMBER	PIN NAME	PIN PURPOSE
1	VSS	Ground connection
2	VDD	+5V connection
3	V0	Contrast adjustment (to potentiometer)
4	RS	Register selection (Character vs. Command)
5	RW	Read/write
6	EN	Enable
7	D0	Data line 0 (unused)
8	D1	Data line 1 (unused)
9	D2	Data line 2 (unused)
10	D3	Data line 3 (unused)
11	D4	Data line 4
12	D5	Data line 3
13	D6	Data line 6
14	D7	Data line 7
15	A	Backlight anode
16	K	Backlight cathode

- **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 we 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.
- We can illuminate the backlight by connecting the anode pin to 5V and the cathode pin to ground if we are using an LCD with a built-in resistor for the backlight. If we are not, we must put a current-limiting resistor in-line with the anode or cathode pin.

- we can connect the communication pins of the LCD to any I/O pins on the Arduino.

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


SCL
SDA
AREF
GND
DIGITAL
PWM
TX
RX
L
TX
RX
Arduino
UNO
ON
ICSP
www.arduino.cc
IOREF
RESET
3V3
5V
Gnd
Vin
POWER
ANALOG IN
0 1 2 3 4 5

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 we 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.

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, we include the **LiquidCrystal** library: `#include <LiquidCrystal.h>`
- Then, we 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, we call the library's **begin()** function to set up the LCD display with the character size. The arguments for this command represent the number of columns and the number of rows, respectively: `lcd.begin(16, 2);`
- After doing that, we can call the library's **print()** and **setCursor()** commands to print text to a given location on the display. For example, if we want to print Hello world on the second line, we write these commands:

```
lcd.setCursor(0,1);
lcd.print("Hello world");
```

- 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 we call **print()** without first changing the cursor location, the text starts in the top-left corner.

Displaying Text with an Incrementing Number

- 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.

```
//Include the library :
#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(" Mechatronic Eng ");
}

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++;
}
```

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.
- 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.
- If we take a close look at our LCD, we will see that each character block is actually made up of a 5x8 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.

```
//LCD with Progress Bar

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

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

//Create the progress bar characters
byte p20[8] = {
B10000,
B10000,
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(" Mechatronic Eng ");

//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(100);                   //Wait
      }
   }
}
```

- At the top of our sketch, we 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 p20 is filling 20 percent of one character block (the p stands for percent).
- In the setup() function, we 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, we type the following within our setup()function: **lcd.createChar(0, p20);**
- When we are ready to write a custom character to the display, we 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.
- We can write other character IDs without it, like this: **lcd.write(1);**

- 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.

Building a Personal Thermostat

- To make this display more useful, we add the temperature sensor ,a fan, and a speaker.
- The display shows the temperature and the current fan state.
- When it gets too hot, the speaker makes a noise to alert us, and the fan turns on. When it gets sufficiently cool again, the fan turns off.
- Using two pushbuttons, we add the ability to increment or decrement the desired temperature.

```
//Keep yourself cool! This is a thermostat.

//This assumes temperatures are always two digits

//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
const int DOWN_BUTTON =9;
const int UP_BUTTON =10;
const int FAN =11;
int set_temp = 23; //The Default desired temperature
```

```
void setup()
{
pinMode(FAN, OUTPUT);
//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);
}
```

```
void loop()
{
//Get the Temperature
.....................
lcd.setCursor(8,0);      //Move the cursor
lcd.print(c);            //Print this new value
```

```
//Turn down the set temp
if (digitalRead(DOWN_BUTTON)== LOW)
{
set_temp--;
}
//Turn up the set temp
else if (digitalRead(UP_BUTTON)== LOW)
{
set_temp++;
}
//Print the set temp
lcd.setCursor(8,1);
lcd.print(set_temp);
```

Adjusting the Set Point with a Button

```
//It's too hot!
if (c >= set_temp)
{
//So that the speaker will beep...
.....
//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
.....
//Update the fan state, and LCD display
digitalWrite(FAN, LOW);
lcd.setCursor(15,1);
lcd.write(1);
}
}
```