Co-funded by the
Erasmus+ Programme
of the European Union

UNIT 6: LCD (LIQUID-CRYSTAL DISPLAY) SCREENS

AIMS

As you can imagine there's a broad range of digital input and output peripherals. Up until now you've just used the most common ones – simple, economical pushbuttons or switches – as devices for setting "1" and "0" logic levels and the LEDs and the diodes for representing them.

But now it's time to talk about other more important and better known digital peripherals. In this unit so we're going to take a look at the LCD screen as an output peripheral: it enables you to display any kind of output information including numbers, letters and symbols.

THEORY SECTION

- LCD SCREENS
- THE CHARACTER SET
- GRAPHIC CHARACTERS
- THE LiquidCrystal.h LIBRARY
- THE EEPROM DATA MEMORY

PRACTICE SECTION

- CONNECTING LCD SCREENS
- EXAMPLE 1: Hello world!!
- EXAMPLE 2: The screen
- EXAMPLE 3: The cursor
- EXAMPLE 4: Blinking
- EXAMPLE 5: Text direction
- EXAMPLE 6: Scrolling
- EXAMPLE 7: AutoScroll
- EXAMPLE 8: Custom characters
- EXAMPLE 9: Display
- EXAMPLE 10: Displaying whole numbers
- EXAMPLE 11: Displaying floating-point numbers
- EXAMPLE 12: Menu

- EXAMPLE 13: Choice of options
- EXAMPLE 14: One last improvement

PRACTICE MATERIALS

-Lap top or desk top computer

-Arduino IDE work environment; this should include the supplementary material already installed and configured.

-Arduino UNO controller board

-A USB cable

TABLE OF CONTENTS

THEORY SECTION **4**

1. LCD SCREENS 4
2. THE CHARACHER SET 6
3. THE GRAPHIC CHARACTERS 7
4. THE LiquidCrystal.h LIBRARY 9
5. THE EEPROM DATA MEMORY 16

PRACTICE SECTION **18**

6. CONNECTING THE LCD SCREEN 18
7. EXAMPLE 3: Hello world! 21
8. EXAMPLE 2: The Screen 21
9. EXAMPLE 3: The Cursor 21
10. EXAMPLE 4: Blinking 21
11. EXAMPLE 5: Text Direction 22
12. EXAMPLE 6: Scrolling 22
13. EXAMPLE 7: AutoScroll 22
14. EXAMPLE 8: Custom characters 23
15. EXAMPLE 9: Display 24
16. EXAMPLE 10: Displaying whole numbers 24
17. EXAMPLE 11: Displaying floating-point numbers 25
18. EXAMPLE 12: Menu 25
19. EXAMPLE 13: Option selection 27
20. EXAMPLE 14: One last improvement 28

:REFERENCES **29**

THEORY SECTION

1. LCD SCREENS

This is an output peripheral that not only shows numbers but also all kinds of characters, texts, symbols and even simple graphics. It’s got thousands of uses and you must have seen it tons of times. You’ll see them with different numbers of lines and characters per line. There are also ones with different coloured backlighting and characters of different colours and sizes.

They’ve all got their own controller that administers all the internal operations. Most of them are compatible with the popular Hitachi HD44780. That’s why it doesn’t make much difference which screen you use: 2 lines by 16 characters one or 4 lines by 20 characters.

You’re going to use a 2 x 16 LCD screen. There’s a picture of one in the figure with a simplified representation and an electrical diagram (Figure 1).

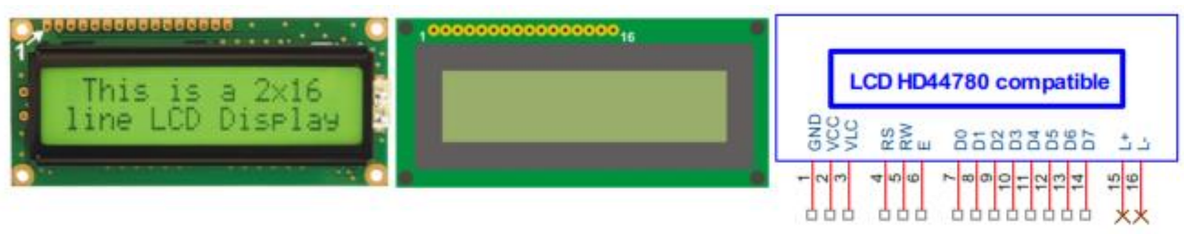


Figure 1

An LCD screen is a digital peripheral. Its signals can be connected directly to the controller’s input and output pins. They’re divided into three groups:

- **<u>Power supply</u>**: The input and output pins deliver the voltage required by the screen and they’re usually + 5V GND and VCC. There’s also a third pin, a VLC; it delivers a voltage alternating between 0 V y +5 V that can be used to adjust screen contrast.
- **<u>Control</u>**: These are the signals that decide whether the screen receives a command or some data, if it’s going to be read or written by the controller or if the screen is enabled or not.
- **<u>Data</u>**: The controller uses these signals to send commands and data proper to the screen.

Pin nº 1 is the first on the left, have a look at where pin number 1 is positioned (Figure 2).

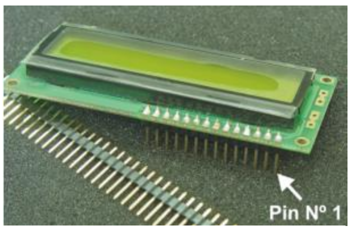


Figure 2

Have a look at the following table that contains a description of each pins.

PIN Nº	NAME	TYPE	DESCRIPTION
1	**Vss**	Voltage supply	Ground power supply (0 V)
2	**Vdd**	Voltage supply	Positive +5 Vcc power supply
3	**VLC**	Voltage supply	Contrast adjustment: voltages alternating between 0 and +5 Vcc.

4	**RS**	Input	Output from the Arduino controller. Selects between instructions and data:
			RS=0 Arduino transfers instructions
			RS=1 Arduino transfers data (ASCII codes)

5	**R/W**	Input	Output from Arduino. Controls reading and writing: R/W=0 Arduino performs data write on the LCD screen R/W=1 Arduino performs data read of the LCD screen

6	**E**	Input	Output from Arduino. Enables the screen:
			E=0 LCD screen disabled (on high impedance)
			E=1 LCD screen enabled

7-14	**DB0:DB7**	Input/Output	Data and instruction bus lines. Arduino transfers instructions or data to the screen according to the RS signal. The DB0:DB7 lines are used with an 8 bit interface The DB4:DB7 lines are used with an 4 bit interface
15	**L+**	Voltage supply	Positive voltage for background light (+5 Vcc)
16	**L-**	Voltage supply	Negative voltage for background light (0 V)

We're not going to use pin numbers 15 and 16: they're optional. It's quite possible they don't even appear on the screen; only the ones with background light use them.

Screens are amongst the most powerful and versatile peripherals. As we've already said, they're able to display all kinds of messages comprised of text, numbers or symbols as well as provide a variety of viewing effects such as movements to the left or right, flickering, scrolling etc. An LCD screen has a separate controller.

The LCD screen you're going to use and which is included in the kit of suggested materials appears in the above figure. This screen comprises two rows of 16 characters each (2 x16). You'll also notice how you have to solder a strip of 14/16 male pins; this will make it easier to insert them on the practice module board later. You'll be able to make the connexions with the controller much more quickly..

2. THE CHARACHER SET

Communication between Arduino and LCD screens is achieved basically through DB0-DB7 digital pins. You can use eight pins or, as in our case, just four. This is called working with a “4 bit interface”.

Arduino send commands or instructions to the screen through the pins. The screen can then carry out a range of display effects: scrolling, flickering, deleting, positioning the cursor etc… It also sends the ASCII character codes that you want to display. These are 8 bit codes. If you use an “8 bit interface”, Arduino needs only a single transfer to display each character; a “4 bit interface” needs two transfers to display each character. It’s a bit slower but you use fewer connection cables. But in any case, don’t worry too much about it: the functions you’re going to study will make the task a great deal easier.

The following **¡Error! No se encuentra el origen de la referencia.** shows a sample of the character set that LCD screens display; this set is established by the manufacturer. The internal ROM memory contains the definition of each one of the characters and is able to vary between different models and versions.

The four lightest bits, B3:B0, are represented in the rows to the left. The four heaviest ones, B7:B4, are represented at the top of the table in the columns in binary code. All you have to do to code a character is select it and then find which column or row it’s in. Here’s an example: the character ‘F’ is in column 4 (0100) and row 6 (0110). Its binary code is therefore 0100 0110 (0x46) which corresponds exactly with the ASCII code for the character “F”. We mentioned before that not all screens have the same character set. It depends on the manufacturer, the model, the version etc. However, the codes corresponding to the standard ASCII characters are common to all screens. These are the ones in columns 2 (0010) to 7 (0111).

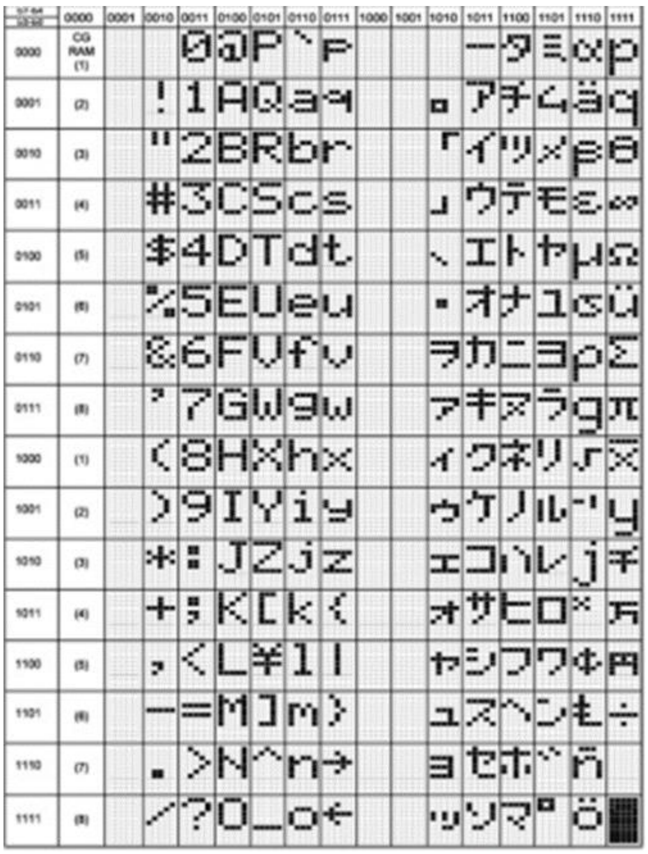


Figure 3

3. THE GRAPHIC CHARACTERS

You can create a total of up to eight graphic characters of 5 x 8 points or “pixels”. Each character is numbered 0 to 7 and needs a total of 8 bytes to be defined. The LCD screen has an internal RAM memory called CGRAM to do this task. Once they’ve been defined, you can display them by sending the number that corresponds to the character (between 0 and 7). Don’t forget: the graphic characters are stored in the RAM memory. If you unplug the system you lose them and Arduino has to define them again.

The graphic characters are defined by inserting bytes into successive positions of the CGRAM memory; the binary patterns of the bytes define each character. The CGRAM is a volatile memory capable of storing a total of 64 bytes. A 5 x 8 point character needs 8 bytes to define it. So you can define up to eight different characters (8 x 8) whatever way you like.

The following table (Figure 4) contains four 5 x 8 defined characters as an example. They are inserted in the first 32 positions of the CGRAM. The first one goes in positions 0 to 7, the second in positions 8 to 15 and so on and so forth.

Each bit of each one of the bytes that have a value of "1" activates its corresponding point or pixel on the LCD screen. The first graphic character of the CGRAM is displayed by sending the number 0 as if it were an ASCII code. The second character is viewed by sending the number 1 and so on with all the ones you have defined.

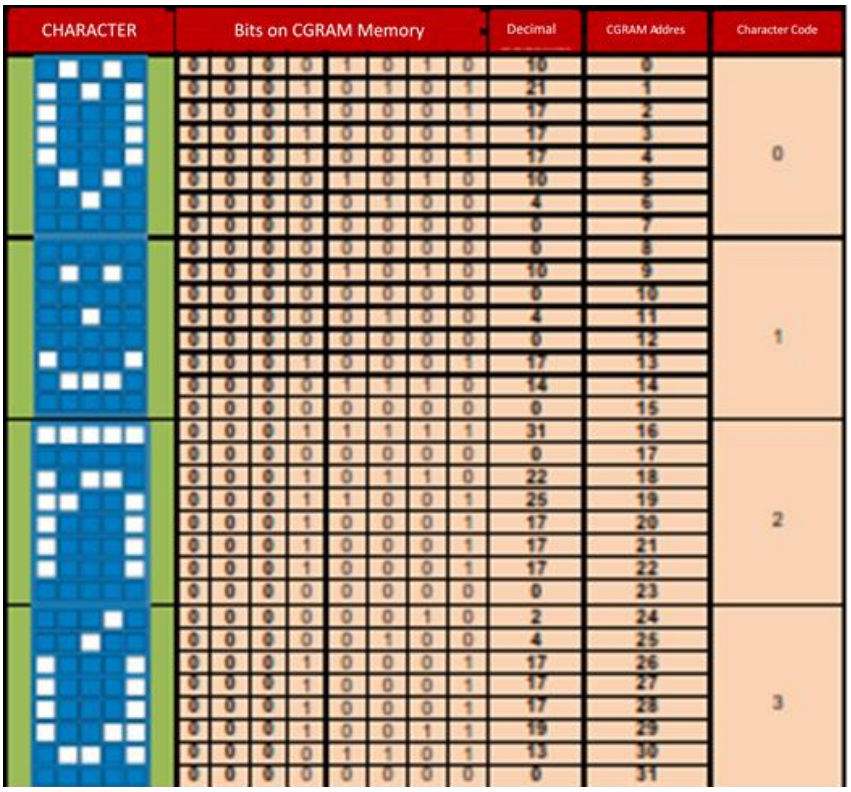
All you have to do to define them is to create an array or matrix for each character. Next, the contents of each array is copied onto the screen CGRAM using the **createChar()** function - you'll be studying this straight away. You have to create the same number of arrays as characters; there four in this example:

1. byte heart[8] = {10,21,17,17,17,10,4,0};

2. byte smile[8] = {B0,B01010,B0,B00100,B00000,B10001,B01110,B0};

3. byte letter_ñ [8] = {31,0,24,27,17,17,17,0};

4. byte letter_ú [8] = {2,4,17,17,17,19,13,0};



CHARACTER	Bits on CGRAM Memory								Decimal	CGRAM Addres	Character Code
	0	0	0	0	1	0	1	0	10	0	0
	0	0	0	1	0	1	0	1	21	1	
	0	0	0	1	0	0	0	1	17	2	
	0	0	0	1	0	0	0	1	17	3	
	0	0	0	1	0	0	0	1	17	4	
	0	0	0	0	1	0	1	0	10	5	
	0	0	0	0	0	1	0	0	4	6	
	0	0	0	0	0	0	0	0	0	7	
	0	0	0	0	0	0	0	0	0	8	1
	0	0	0	0	1	0	1	0	10	9	
	0	0	0	0	0	0	0	0	0	10	
	0	0	0	0	0	1	0	0	4	11	
	0	0	0	0	0	0	0	0	0	12	
	0	0	0	1	0	0	0	1	17	13	
	0	0	0	0	1	1	1	0	14	14	
	0	0	0	0	0	0	0	0	0	15	
	0	0	0	1	1	1	1	1	31	16	2
	0	0	0	0	0	0	0	0	0	17	
	0	0	0	1	0	1	1	0	22	18	
	0	0	0	1	1	0	0	1	25	19	
	0	0	0	1	0	0	0	1	17	20	
	0	0	0	1	0	0	0	1	17	21	
	0	0	0	1	0	0	0	1	17	22	
	0	0	0	0	0	0	0	0	0	23	
	0	0	0	0	0	0	1	0	2	24	3
	0	0	0	0	0	1	0	0	4	25	
	0	0	0	1	0	0	0	1	17	26	
	0	0	0	1	0	0	0	1	17	27	
	0	0	0	1	0	0	0	1	17	28	
	0	0	0	1	0	0	1	1	19	29	
	0	0	0	0	1	1	0	1	13	30	
	0	0	0	0	0	0	0	0	0	31	

Figure 4

4. THE LIQUIDCRYSTAL.H LIBRARY

You'll be familiar with the concept of a "library" by now and how to handle the range of functions it contains. Arduino has created a large number of libraries but we're just going to talk about one: "LiquidCrystal.h". It contains a large number of functions designed to control and operate LCD screens based on the Hitachi HD44780 controller or compatible with it.

We're going to study the most representative and important functions. In any case, I recommend you visit www.arduino.cc where you'll find lots of information and examples of all of them.

- **Function: LiquidCrystal()**

This function creates a "LiquidCrystal" type variable and establishes the connections between the LCD screen and the Arduino controller; you can use an 8 or 4 bit interface between them. In this case, don't use the D0-D3 lines. You can also decide whether or not to use the R/W screen signal. If you don't use it, as in this case, connect the signal to GND. You'll do this in the practice section.

Syntax:

LiquidCrystal var(RS,E,D4,D5,D6,D7); //For a 4 bit interface without R/W signal
LiquidCrystal var(RS,RW,E,D4,D5,D6,D7); //For a 4 bit interface with R/W signal
LiquidCrystal var(RS,E,D0,D1,D2,D3,D4,D5,D6,D7); //For an 8 bit interface without R/W signal

LiquidCrystal var(RS,RW,E,D0,D1,D2,D3,D4,D5,D6,D7); //For an 8 bit interface with R/W signal

var: the name for the variable assigned to the LCD screen that you're going to control.

RS: the Arduino pin connected to the screen RS signal.

RW: the Arduino pin connected to the screen R/W signal (if it's going to be used).

E: the Arduino pin connected to the screen E signal.

D0-D7: Arduino pins that are connected to the screen DB0-DB7 data lines. If there are no pins indicated for DB0-DB3, we assume a 4 bit interface and only employ DB4-DB7 signals.

Example:

LiquidCrystal lcd(7,8,9,10,11,12); *//Establishes the connection of a screen called "lcd". Arduino //pins D7 to D12 are connected to the screen RS, E, DB4, DB5, //DB6 and DB7 signals. The R/W signal must be connected to //GND.*

- **Function: Begin()**

This function starts up the LCD screen and assigns it the number of rows and the number of characters per row according to the model in question. In this case you're going to use a 16 x 2 character screen.

Syntax:

var.begin(c,f);

var:	this is the name that defines the screen in question (established in LyquidCrystal()).
c:	the number of columns.
f:	the number of rows.

Example:

LiquidCrystal lcd(7,8,9,10,11,12);	//Connections to the LCD screen *lcd.begin(16,2);* this // is a 16 x 2 "lcd" screen

- **Function: setCursor()**

This function positions the LCD screen cursor as desired. From then on the preceding characters will be displayed.

Syntax:

var.setCursor(c,f);

var:	This is the name that defines the screen in question (established as LyquidCrystal()).
c:	the number of columns (starting from 0).
f:	the number of rows (starting from 0).

Example:

LiquidCrystal lcd(7,8,9,10,11,12);	//Connections to the LCD screen
lcd.setCursor(3,0);	//Places the cursor in position 3 (4th //character) of row //0 (the 1st)

- **Function: home()**

This function locates the cursor in the top left hand corner (position 0 of row 0) of the screen's first position. It doesn't delete what was being previously displayed.

Syntax:

var.home();

var:	This is the name that defines the screen in question (established as LyquidCrystal()).

- **Function: clear()**

This function clears the LCD screen and locates the cursor in the top left hand corner (row 0 position 0).

Syntax:

var.clear();

var:	This is the name that defines the screen in question (established as LyquidCrystal()).

Example:

```
LiquidCrystal lcd(7,8,9,10,11,12);          //Connections to the LCD screen

lcd.clear();                                //Clears the screen
```

- **Function: write()**

This function writes a character in the cursor's current position.

Syntax:

var.write(char);

var:	This is the name that defines the screen in question (established as LyquidCrystal()).
char:	The character to be displayed

Example:

```
lcd.write('A');                             //It writes 'A'
```

- **Function: print()**

This function prints on the LCD screen starting from the current position of the cursor.

Syntax:

var.print(data);

var.print(data,base);

var:	This is the name that defines the screen in question (established as LyquidCrystal()).
data:	This is the data to be printed. This might be char, int, long, float or string.
base:	This is optional and shows the desired numerical base: BIN=Binary; DEC=Decimal (by default); OCT=Octal; HEX=Hexadecimal; or N=nº in decimals for floating-point numbers (2 by default).

Examples:

int A=19;

float PI=3.1416;

lcd.print(A,HEX);	*//Prints A in hexadecimal (1910 = 1316)*
lcd.print("Hello");	*//Prints "Hello"*
lcd.print(PI*2,4);	*//Prints 6.2832 (four decimals)*

- **Function: cursor()**

This function displays the cursor on the LCD screen in its current position as an underscore (_). This is where it'll start to write the next character.

Syntax:

var.cursor();

var: This is the name that defines the screen in question (established as LyquidCrystal()).

- **Function: noCursor()**

This function hides the LCD cursor.

Syntax:

var.noCursor();

var: This is the name that defines the screen in question (established as LyquidCrystal()).

- **Function: blink()**

This function displays the LCD cursor on the screen in its current position as a solid intermittent symbol (▓).This is where it'll start to write the next character.

Syntax:

var.blink();

var: This is the name that defines the screen in question (established as LyquidCrystal()).

- **Function: noBlink()**

This function hides the solid intermittent cursor (▓).

Syntax:

var.noBlink();

var: This is the name that defines the screen in question (established as LyquidCrystal()).

- **Function: noDisplay()**

This function turns the LCD screen off without losing whatever content there may be on it or the cursor's position.

Syntax:

var.noDisplay();

var: This is the name that defines the screen in question (established as LyquidCrystal()).

- **Function: display()**

This function connects the LCD screen and recovers the content displayed on it before noDisplay() was executed.

Syntax:

var.display();

var: This is the name that defines the screen in question (established as LyquidCrystal()).

- **Function: scrollDisplayLeft()**

This function displaces the content (the text and the position of the cursor) displayed on the screen at any given moment one place to the left.

Syntax:

var.scrollDisplayLeft();

var: This is the name that defines the screen in question (established as LyquidCrystal()).

- **Function: scrollDisplayRight()**

This function displaces the content (the text and the position of the cursor) displayed on the screen at any given moment one place to the right.

Syntax:

var.scrollDisplayRight();

var: This is the name that defines the screen in question (established as LyquidCrystal()).

- **Function: leftToRight()**

This function automatically establishes which direction the cursor writes on the screen: from left to right. This means that the characters are written from left to right without affecting the ones that have already been written.

Syntax:

var.LeftToRight();

var: This is the name that defines the screen in question (established as LyquidCrystal()).

- **Function:rightToLeft()**

This function reverses the direction the cursor writes on the screen: the right to left. This means that the characters are written from right to left without affecting the ones that have already been written.

Syntax:

var.RightToLeft();

var: This is the name that defines the screen in question (established as LyquidCrystal()).

- **Function: autoscroll()**

This function activates the scrolling or automatic display movement. Each time a character is sent to the screen this function displays it and then moves the rest of the contents one place. If the direction at the time is left to right (leftToRight()), the contents moves to the left. If the direction at the time is right to left (rightToLeft()), the content moves to the right.

Syntax:

var.autoscroll();

var: This is the name that defines the screen in question (established as LyquidCrystal()).

- **Function: noAutoscroll()**

This function deactivates the scrolling or automatic display movement.

Syntax:

var.noAutoscroll();

var: This is the name that defines the screen in question (established as LyquidCrystal()).

- **Function: createChar()**

This function creates a character defined by the user. It is capable of creating a total of eight 5 x 8 pixel characters numbered from 0 to 7. A matrix of 8 bytes, one per row, determines a character's appearance or design. The five lightest bits of each byte correspond to the 5 pixels that make up each row of the character. Once this function has created a character, the user can display it on the screen simply by indicating its number.

Syntax:

var.createChar(n,dato);

var: This is the name that defines the screen in question (established as LyquidCrystal()).

n: this represents the number of the character to be defined (from 0 to 7).

data: This is the name of the matrix that contains the bytes that define the custom character.

Example:

bytearrowhead[8]={B00100,B01110,B10101,B00100,B00100,B00100, B00100, B00100};
//Creates the 8 byte "arrowhead" matrix which defines the new graphic character
lcd.createChar(0, arrowhead);
//Creates the new nº 0 character from the defined content in the "arrowhead" array….
lcd.write(byte(0));
//displays the nº 0 graphic character (the arrowhead)

5. THE EEPROM DATA MEMORY

If you thought the following subject had nothing to do with what this unit is about, you'd be absolutely right: it doesn't. But it just happens to be a good time to talk about the Arduino UNO controller EEPROM data memory. It'll help you to enhance lots of your present and future projects.

The controllers that comprise the various Arduino boards form part of a special memory: the EEPROM memory. It enables you to read or write 8 bit data between 0 and 255. This memory has a special feature: it retains all the information you've saved even if the power goes off; it doesn't delete it.

That's why it's ideal for storing non volatile information that can be modified such as access codes, configuration parameters, telephone numbers, passwords etc… As you are already aware, the NANO Arduino board includes an ATmega328 model controller that has a 1KB (1024 bytes) EEPROM memory.

By this time you know more than enough to use it. You've got access to a library as well: "EEPROM.h". It's got just two functions and you can use them to read and write data onto this memory. As usual, you should include this library in your programs by using **#include <EEPROM.h>**.

- **Function:read()**

This function reads the 8 bit data that's on the EEPROM memory.

Syntax:

EEPROM.read(dir);

dir: This determines which direction you read in from the EEPROM. It has 1024 positions (KB) so the possible range is from 0 to 1023.

Example:

```
#include EEPROM.h             //Includes the library

byte value;                   //8 bit variable

value = EEPROM.read(279);     //Reads the byte in position 279
```

- **Function:write()**

This function writes an 8 bit value (between 0 and 255) in any of the positions available in the EEPROM.

Syntax:

EEPROM.write(dir, value);

dir: This determines which direction you write in from the EEPROM. It has 1024 positions (KB) so the possible range is from 0 to 1023.

value: This is the 8 bit value (between 0 and 255) which you want to write in the direction indicated.

Example:

```
EEPROM.write(982, 33);                    //Write the value 33 in position 982
```

IMPORTANT: The EEPROM memory has an estimated capacity of 100,000 write/erase cycles. An EEPROM write may take up to 3.3 mS to complete so you may need to be careful about how often you write to it; if you go over the limit it could be disenabled. Keep it in mind.

PRACTICE SECTION

6. CONNECTING THE LCD SCREEN

You're going to start off the practical section by connecting the LCD screen to the Arduino controller on the module board. Look carefully at the connections and make sure you assemble everything properly. Here's the circuit diagram:

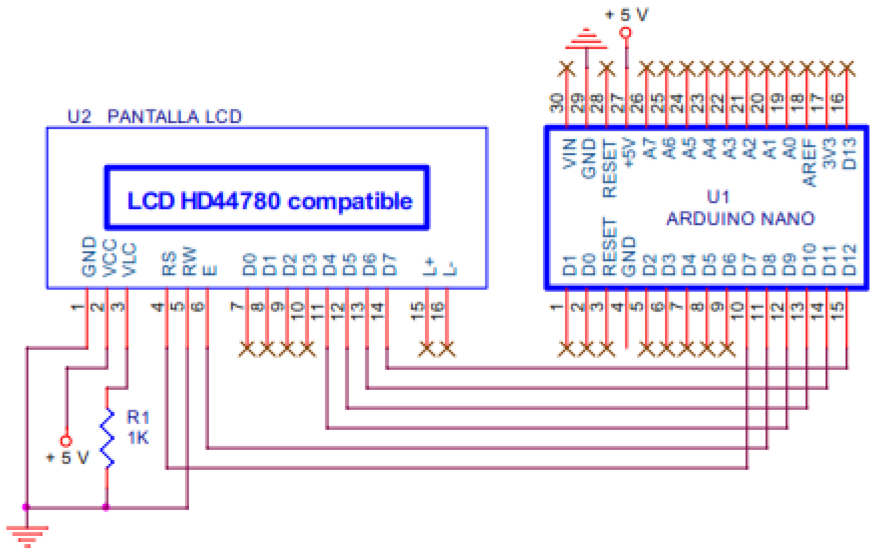

U2 PANTALLA LCD
LCD HD44780 compatible
GND VCC VLC RS RW E D0 D1 D2 D3 D4 D5 D6 D7 L+ L-
U1
ARDUINO NANO
VIN GND RESET +5V A7 A6 A5 A4 A3 A2 A1 A0 AREF 3V3 D13
D1 D0 RESET GND D2 D3 D4 D5 D6 D7 D8 D9 D10 D11 D12
R1
1K
+ 5 V
+ 5 V

Figure 5

The D9:D12 Arduino pins are connected to the DB4:DB7 signals from the screen. Monitored by your program Arduino will send the instructions and the characters to be displayed. D7 and D8 control the RS and E signals respectively. RW must be connected to GND.

The screen is powered by +5 V through the GND and the VCC pin. We send a variable voltage of between 0 and 5 through the VLC pin, number 3, to adjust the contrast. We usually wire it to GND with R1, a resistor. The resistor value may vary according to the screen model.

As it's impossible to predict exactly what kind you'll come across, we'll start off with a 1K Ω resistor. If you notice that the characters seem too stark and you can hardly make them out, change the resistor value for a higher one like 4K7 Ω, for example. You're sure to hit on the right contrast somewhere between these two figures.

We use pins 15 and 16 of the LCD screen to control the backlight although not all models have this feature. We're not going to use them and it is possible that your screen doesn't even have these two pins. If it does have them, don't connect them.

We recommend you start the assembly with the connection wires just like in the figure and the photo. You'll be installing the screen later and it just might cover one of these wires. Have a close look.

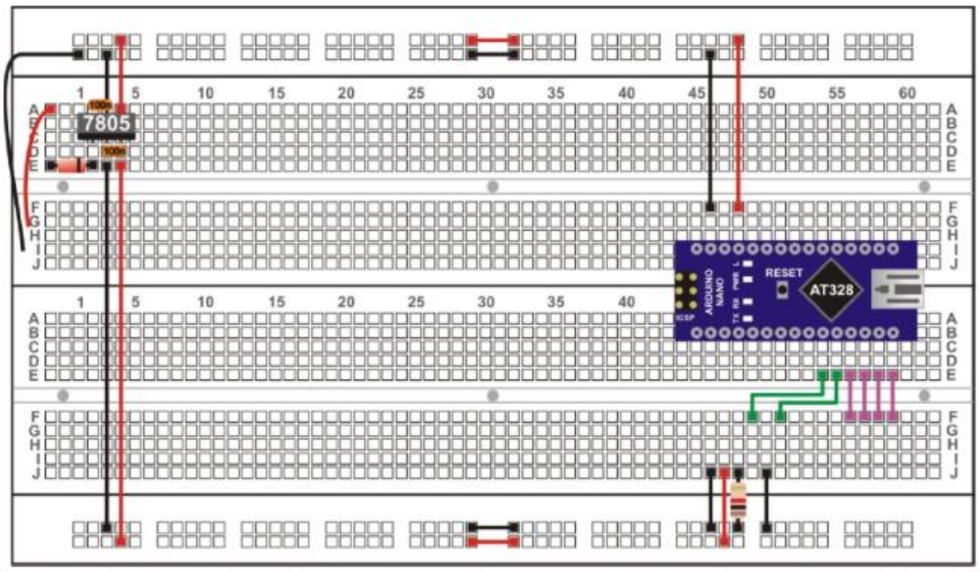


Figure 6

Now position the LCD screen; make sure you insert pin 1 on the screen in the same column of holes as the black GND wire, pin 2 in the same column as the red +5 V wire, pin 3 in the same column as the resistor and so on and so forth.

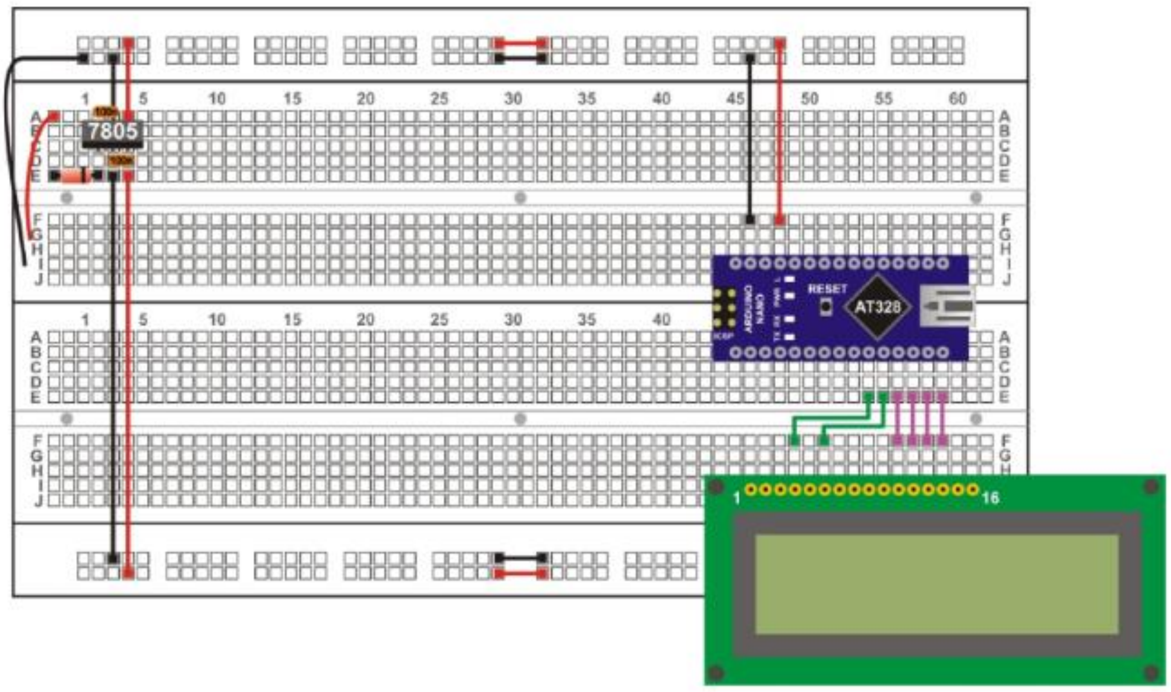


Figure 7

You're sure to think of some other way of connecting the components and it might even be better. You should remember however that the LCD screen is going to stay connected for almost all of the rest of the course. For this reason you should leave space on the module board for the components and peripherals you'll be using in the future.

7. EXAMPLE 3: Hello world!

Here’s the first program you’re going to use to display the famous message on the LCD screen.

You can include the LCD function library by using **#include**. You can establish the lcd variable and configure the connections using **LiquidCrystal().** Make sure the connections correspond with the ones in the circuit diagram for the assembly you’ve just done.

Select the 16 x 2 screen model using the **lcd.begin(16,2)** function in **setup().**

Last of all, the main **loop()** program transmits the message to be displayed using the **lcd.print()** function. Next the message is placed on a continuous loop called **while(1)** that doesn’t really do anything useful; in fact, you could call it the end of the execution.

8. EXAMPLE 2: The Screen

We suggest you do the following examples to familiarize yourself with some of the functions included in the **“LiquidCrystal.h”** library and the various effects they produce.

You’re going to use the **noDisplay()** and **display()** functions to enable or disenable the screen: the screen will go blank or display whatever content it has at the time.

9. EXAMPLE 3: The Cursor

The cursor indicates where the next character it receives will be written on the screen. This example uses the **cursor()** and **noCursor()** functions to enable the character to be displayed or not.

In theory the cursor is viewed as an underscore (_).

10. EXAMPLE 4: Blinking

The cursor can also be viewed as an intermittent solid symbol (▓) that indicates where the next character it receives will be written on the screen. That’s what the **blink()** and **noBlink()** functions are for.

The cursor flashes on and off spontaneously; the LCD screen controls it automatically. In the example cursor is enabled and kept flashing for three seconds and then disenabled for another three so you can see what these functions do.

11. EXAMPLE 5: Text Direction

By using the leftToRight() and rightToLeft() functions you can determine how you write and display the text on the screen. You can write and display from left to right or from right to left. The cursor writes from left to right by default, the same way we do when we write something.

This example shows you another way to display a message. This time the text is displayed in an array:

char Mens1[]={"ILove Arduino"};   //Message to be displayed

Two **for()** loops send each character from the array to the screen at intervals of 0.150 seconds using the **lcd.write()** function. The cursor goes to row 0 position 0 (row 0 is the first one on the LCD) in the first **for()** loop and starts writing the characters from left to right as usual. The cursor goes to row 1 position 15 (the last one in second row) in the second **for()** loop and starts writing the characters from right to left.

Next we change the time sequence to three seconds and the **lcd.clear()** function clears the screen. Then there's a final sequence of one second and then the cycle starts again.

12. EXAMPLE 6: Scrolling

The **scrollDisplayLeft()** and **scrollDisplayRight()** functions also enable you to achieve some nice display effects – you must have seen them on billboards.

These functions move whatever is on the screen at any given moment to the left or the right. This example shows a two line text on the screen. Subsequently the entire text starts moving to the left from one position to the next at 3 second intervals.

When the last character disappears on the left the text starts to move again until it disappears on the right. Then it starts to move to the left again until it finally returns to its original position. After that the screen goes on and off for a while.

13. EXAMPLE 7: AutoScroll

This is another example that will enable you to create striking display effects. The **autoscroll()** function moves all the text one space to the left each time a letter is added and **noAutoscroll()** disenables the scrolling.

When the auto scroll is enabled the function automatically moves each character written on the screen. In the previous example you had to use the **scrollDisplayLeft()** or **scrollDisplayRight()** functions to achieve this effect.

The function moves the characters automatically to the left by default or to the right. It all depends on the last function you used: **leftToRight()** or **RightToLeft().**

The following example shows two messages that were previously defined in two separate arrays:

```
char Mens1[]={"ARDUINO"};                //Message 1

char Mens2[]={"I'Love you"};             //Message 2
```

The first **for()** loop writes the contents of the "Mess1" array on the screen character by character. It automatically writes the characters from left to right at three second intervals.

The second **for()** loop writes the contents of the "Mess2" array on the screen character by character. However, prior to executing this loop, the cursor locates itself in the last position of the second row of the screen and executes the lcd.autoscroll() function. It writes the characters from left to right but each time it writes one it automatically moves the contents of the screen to the left by default.

Last of all **lcd.noAutoscroll()** is executed and after **lcd.clear()** clears the screen the cycle starts again.

14. EXAMPLE 8: Custom characters

To finish up this section of examples that show us how to use the most useful functions in the "**LiquidCrystal.h**" library, you're going to create and use your own graphic characters.

To be more specific, you're going to create five characters: a heart, a happy face, a serious face, a figure with its arms up and another one with its arms down.

Each one of them is defined by using an array. These are the five figures: **heart[8] ; happy[8]; serious[8]; down[8] and up[8].** Each of the characters contains a binary description of what points or pixels should be enabled in each case.

The **lcd.createChar()** functions transfer the content of the arrays to the screen CGRAM memory during setup(). The functions transfer a total of 40 bytes, eight per character. In addition to this, each of the characters is assigned a number between 0 and 5:

```
lcd.createChar(0, heart);        //Create new character nº 0
lcd.createChar(1, happy);        //Create new character nº 1
lcd.createChar(2, serious);      //Create new character nº 2
lcd.createChar(3, down);         //Create new character nº 3
lcd.createChar(4, up);           //Create new character nº 4
```

Last of all, you can display each graphic character on the screen wherever the cursor is as if it were a normal ASCII character. All you have to do is use the **lcd.write(n)** function, “n” being the number assigned to each character as in the example 0 to 5.

15. EXAMPLE 9: Display

Here’s an example that’s really practical. You’re going to create a link between the serial communications channel on your PC and the LCD screen. Naturally the NANO Arduino controller will be in the middle.

The controller functions as a bridge between the two. On one side it’s connected to the PC through the USB port; on the other, it’s connected to the screen in the same way we’ve been connecting it up until now. Using a serial monitor, you’re going to send a number of characters from the PC using the serial communications channel; they’ll be received by the Arduino controller and displayed on the LCD screen as they arrive.

16. EXAMPLE 10: Displaying whole numbers

You’ve already seen how you can display all kinds of texts and characters on the LCD screen and also achieve different kinds of display effects. You can also display numbers as well of course.

This example is going to display the numbers from 1 to 15 in sequence each time you push a button connected to the D2 pin. In addition, the number in question will be displayed as a decimal, hexadecimal and binary number.

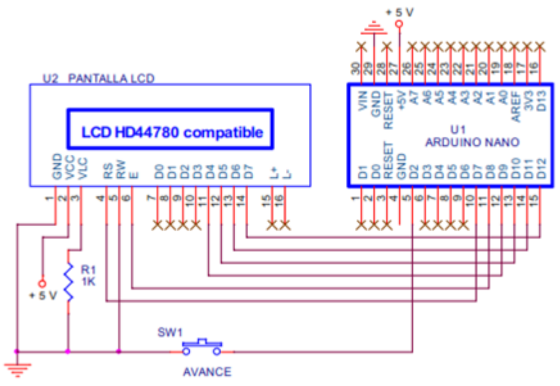

+5 V
U2 PANTALLA LCD
LCD HD44780 compatible
GND VCC VLC RS RW E D0 D1 D2 D3 D4 D5 D6 D7 L+ L-
U1
ARDUINO NANO
VIN GND RESET +5V A7 A6 A5 A4 A3 A2 A1 A0 AREF 3V3 D13
D1 D0 RESET GND D2 D3 D4 D5 D6 D7 D8 D9 D10 D11 D12
R1
1K
+5 V
SW1
AVANCE

Figure 8

The program won't present any difficulties at all given what you know now. Here's a small detail that'll come in handy all the time: as the numbers you'll be displaying have different lengths, it's a good idea to delete the first line of the LCD screen where you're going to display them. You can do this by filling up the line in question with blank spaces (" "); that way there are no "leftovers".

Have a close look at the program. Try and take out the first two **loop()** functions and see what happens: I think you'll get what I mean by "leftovers" now.

17. EXAMPLE 11: Displaying floating-point numbers

Following on from the above example, we're going to talk about displaying floating-point numbers. In this case, we're going to display the square root of the number N to two decimal places and N multiplied by the PI (3.1416) constant to four decimal places. The number N advances sequentially each time you push the D2 button; it goes from 0 to 9.

Both the circuit diagram and the assembly are the same ones you used for the previous example 10.

18. EXAMPLE 12: Menu

An LCD screen in an ideal peripheral for making an intuitive user interface. And in fact there are menus of options that enable you to make a selection from the different tasks to be executed.

In this example the idea is to make a menu of six options that appear on the screen one after the other. By using one button to go forward and another to go back, you'll be able to browse through the menu and have a look at all the available options. Here's what the circuit diagram looks like:

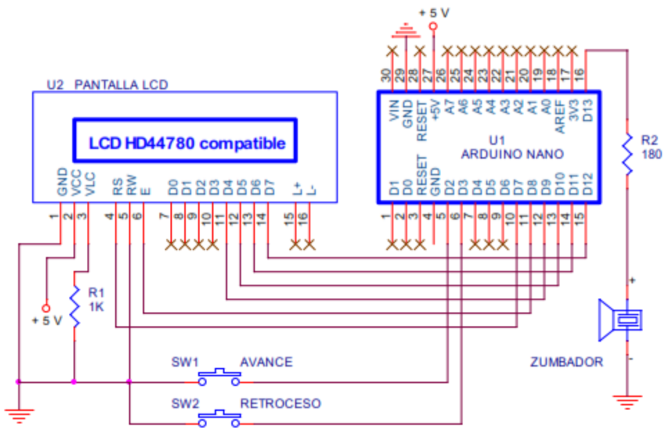

+ 5 V
U2 PANTALLA LCD
LCD HD44780 compatible
GND VCC VLC RS RW E D0 D1 D2 D3 D4 D5 D6 D7 L+ L-
U1
ARDUINO NANO
VIN GND RESET +5V A7 A6 A5 A4 A3 A2 A1 A0 AREF 3V3 D13
D1 D0 RESET GND D2 D3 D4 D5 D6 D7 D8 D9 D10 D11 D12
R2
180
R1
1K
+ 5 V
SW1
AVANCE
SW2
RETROCESO
ZUMBADOR

Figure 9

The connections between the Arduino controller and the screen are the same as the ones you've been using up until now. In this case there's a second button called SW2. The SW1 button is connected to the D2 entry and the options available are shown on the screen from first to last. When we connect the SW2 button to the D3 pin we can see the options in reverse order: from last to first. Both pins are configured as pull-up resistors. This way you save yourself having to fit the two corresponding external resistors.

The screen does a vertical scroll to display all the options available on the menu one after the other. We're also going to use a piezo electric buzzer connected to the D13 output pin. When either the SW1 or the SW2 button is pushed they make a noise.

Now you're going to have a look at the program. I'll leave it up to you to study and examine it and try and understand how it works. All I'll say is that I've tried to find the easiest and most instructive solution possible; I'm sure it's not the only one around or even the best; I hope you can make it a lot better using your own initiative.

Start:

First of all you include the "LiquidCrystal.h" library and connect up the screen; nothing new there. Next you declare the Option and N option variables. The first one contains the option that's in use at the time and the second one the number of options available on the menu. You also have to declare an arrowhead array that defines the graphic characters.

Functions:

Two functions have been created for this project: visualize() and beep(). The first displays the message on the screen that corresponds to the option in use at the time based on the value of the "Option" variable. And you've already seen the beep() function: it makes a beeping sound.

Setup():

Configure the D2 and D3 pins as inputs with internal pull-up resistors, the D13 pin as an output, the screen with two lines of 16 characters and finally generate the graphic character nº 0 (an arrowhead). There's not really anything important here.

loop():

- begins by clearing the screen and displaying the graphic character nº 0 (the arrowhead).
- locates the cursor on the first row and displays the option in use at the time based on the "option" variable. It uses the visualize() function.

- locates the cursor in the second row and displays the following option (Opcion+1). Once again it uses the visualize() function.
- waits until someone pushes the SW1 or SW2 pushbuttons connected to D2 and D3.
- if D2 is pressed it eliminates debouncing and waits until the button is released. It makes a whistling sound and increases the "Option" variable. It goes on to the next option on the menu.
- if D3 is pressed it eliminates debouncing and waits until the button is released. It makes a whistling sound and increases the "Option" variable. It goes back to the previous option on the menu.

19. EXAMPLE 13: Option selection

Think of it as an extension of the previous example. Have a look at the circuit diagram

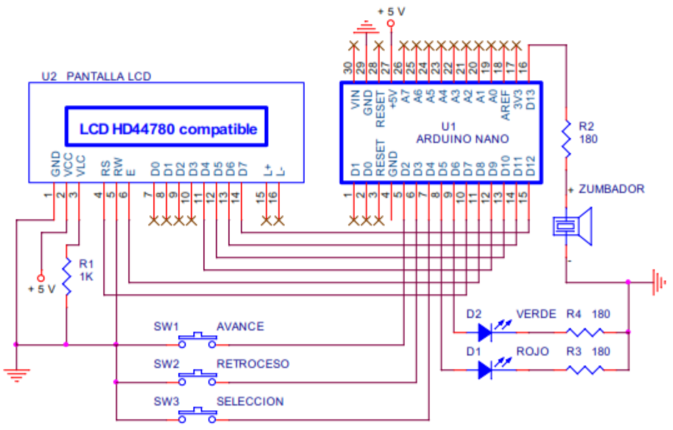


Figure 10

We add another button SW3 connected to D4; it executes the option selected at the time. We add two more LED screens as well: a red one and a green one connected to D5 and D6. They show the result of executing the option.

This program is very similar to the previous example. The only difference is the Execute() function which runs every time the SW3 pushbutton is pressed. This new function assesses the value of the "Option" variable. Depending on what option has been selected, the appropriate task is executed and enables or disenables the red and green LED screens connected to the D5 and D6 outputs.

20. EXAMPLE 14: One last improvement

Imagine an application that doesn’t have just six options on its menu like the ones up until now: it’s got tens of them. Every time you turn the system on or restart it by pressing RESET, it always begins from the first option; you’ll have to scroll down until you find the one you want.

Maybe it’d be a good idea if the LCD screen showed the last option you’d selected so that you didn’t have to go and look for it every time you restarted the system: what do you reckon? Once again you could use the EEPROM data memory. That’s right: every time you choose a new option, it’s recorded in the EEPROM memory. At the end of the day, each option is represented with nothing more than a number saved in the “Option” variable.

Every time the system is restarted, this variable is assigned the last value recorded in the EEPROM memory. You’ll be using position 1 of the 1024 available.

Record the program and make sure it works. Select and execute the various options. Then switch the system off and then turn it on again. See how the last option you selected always appears.

REFERENCES

BOOKS

[1]. **EXPLORING ARDUINO**, Jeremy Blum, Ed.Willey
[2]. Practical ARDUINO, Jonathan Oxer & Hugh Blemings, Ed.Technology in action
[3]. Programing Arduino, Next Steps, Simon Monk
[4]. Sensores y actuadores, Aplicaciones con ARDUINO, Leonel G.Corona Ramirez, Griselda S. Abarca Jiménez, Jesús Mares Carreño, Ed.Patria
[5]. ARDUINO: Curso práctico de formación, Oscar Torrente Artero, Ed.RC libros
[6]. 30 ARDUINO PROJECTS FOR THE EVIL GENIUS, Simon Monk, Ed. TAB
[7]. Beginning C for ARDUINO, Jack Purdum, Ph.D., Ed.Technology in action
[8]. ARDUINO programing notebook, Brian W.Evans

WEB SITES

[1]. https://www.arduino.cc/
[2]. https://www.prometec.net/
[3]. http://blog.bricogeek.com/
[4]. https://aprendiendoarduino.wordpress.com/
[5]. https://www.sparkfun.com