HD44780U(LCD-II)

(Dot Matrix Liquid Crystal Display Controller/Display)

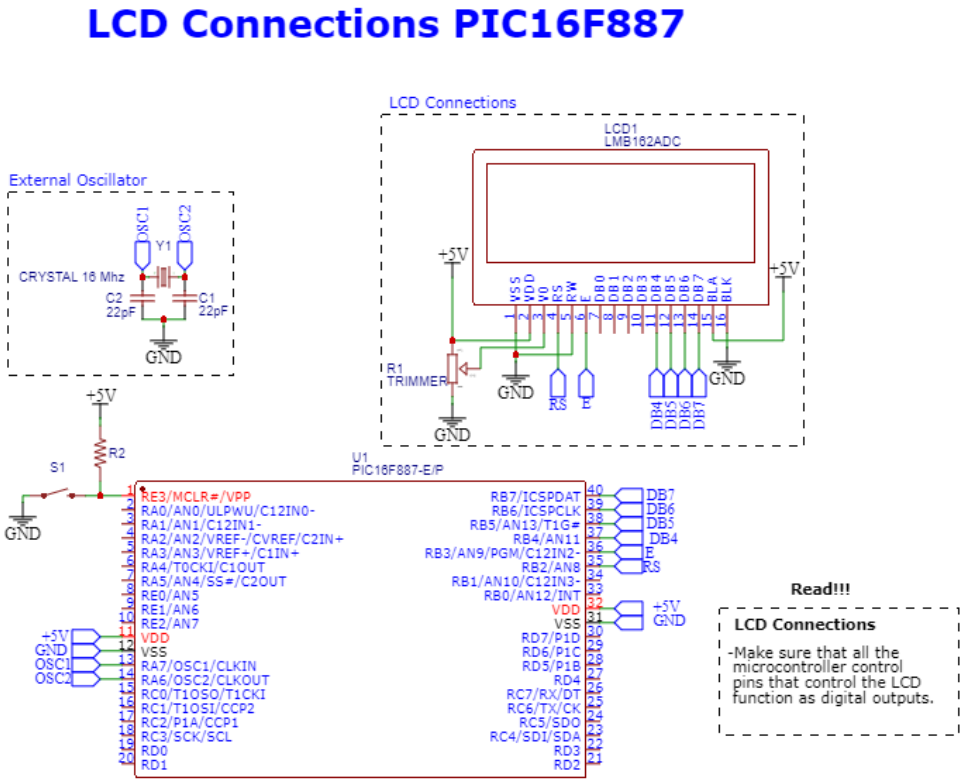
Library for LCD control compiler XC8 8-bits microcontroller

By. Marcos Becerra.

The information presented here will help you understand the way the controller works in this case the HD44780, this library works in 4-bits mode, which means that it uses the high nibble of the data bus (D4..D7), to transmit the data in two part, first send the top 4 bits of the data byte, and finally the last lower 4 bits.

The HD44780 can send data in either two 4-bit operations or one 8-bit operation, thus allowing interface with 4- or 8- bit MPUs.

The connections between the display and the microcontroller are:



Connections

LCD		PIC16F887
VSS	----------	GND
VDD	----------	+5V
Vo	----------	(-) potentiometer
RS	----------	RB2
RW	----------	GND
E	----------	RB3
D0..D3	----------	(-) no connections
D4	----------	RB4
D5	----------	RB5
D6	----------	RB6
D7	----------	RB7

What to do first?

The first step we need to do is read and extract the relevant information from the device datasheet, the link is the following:

https://www.sparkfun.com/datasheets/LCD/HD44780.pdf

The relevant information is this:

- 5 x 8 dot matrix possible.
- Power wide range 3.0 to 11 V.
- 4-bit or 8-bit MPU interface enabled.
- 80 x 8-bit display RAM (80 characters max, 40 characters by line.)
- 208-character fonts (5 x 8 dot)
- Wide range of instruction functions:
  - Display clear, cursor home, display on/off, cursor on/off, etc.
- Pin function compatibility with HD44780S.
- Automatic reset circuit that initializes the controller/ drive after power on.

The following table is shown with the pin functions with a imagen of the pins LCD:

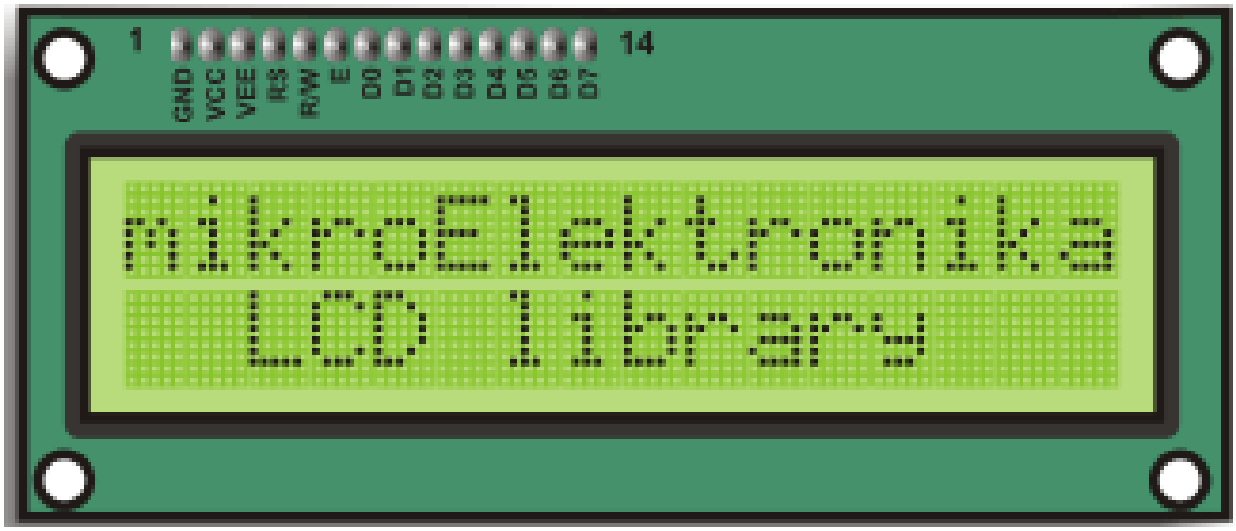


Figure 1. LCD pins model LM016L.

Signal	Lines	I/O	Interfaced with	Function
RS	1	I	MPU	Selects registers. 0: Instruction register (for write) Busy flag: address counter (for read) 1: Data register (for write and read)
R/$\overline{W}$	1	I	MPU	Selects read or write. 0: Write 1: Read
E	1	I	MPU	Starts data read/write.
DB4 to DB7	4	I/O	MPU	Four high order bidirectional tristate data bus pins. Used for data transfer and receive between the MPU and the HD44780U. DB7 can be used as a busy flag.
DB0 to DB3	4	I/O	MPU	Four low order bidirectional tristate data bus pins. Used for data transfer and receive between the MPU and the HD44780U. These pins are not used during 4-bit operation.
CL1	1	O	Extension driver	Clock to latch serial data D sent to the extension driver
CL2	1	O	Extension driver	Clock to shift serial data D
M	1	O	Extension driver	Switch signal for converting the liquid crystal drive waveform to AC
D	1	O	Extension driver	Character pattern data corresponding to each segment signal
COM1 to COM16	16	O	LCD	Common signals that are not used are changed to non-selection waveforms. COM9 to COM16 are non-selection waveforms at 1/8 duty factor and COM12 to COM16 are non-selection waveforms at 1/11 duty factor.
SEG1 to SEG40	40	O	LCD	Segment signals
V1 to V5	5	—	Power supply	Power supply for LCD drive V_{CC} –V5 = 11 V (max)
V_{CC}, GND	2	—	Power supply	V_{CC}: 2.7V to 5.5V, GND: 0V
OSC1, OSC2	2	—	Oscillation resistor clock	When crystal oscillation is performed, a resistor must be connected externally. When the pin input is an external clock, it must be input to OSC1.

Figure 2. Pin functions

The HD44780U has two 8-bit registers, and instruction register (IR) and a data register (DR). The IR stores instruction codes, such as display clear and cursor shift, the following table shows the following commands o instructions:

Instruction	Code										Description	Execution Time (max) (when f_{cp} or f_{osc} is 270 kHz)
	RS	R/$\overline{W}$	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0		
Clear display	0	0	0	0	0	0	0	0	0	1	Clears entire display and sets DDRAM address 0 in address counter.	
Return home	0	0	0	0	0	0	0	0	1	—	Sets DDRAM address 0 in address counter. Also returns display from being shifted to original position. DDRAM contents remain unchanged.	1.52 ms
Entry mode set	0	0	0	0	0	0	0	1	I/D	S	Sets cursor move direction and specifies display shift. These operations are performed during data write and read.	37 µs
Display on/off control	0	0	0	0	0	0	1	D	C	B	Sets entire display (D) on/off, cursor on/off (C), and blinking of cursor position character (B).	37 µs
Cursor or display shift	0	0	0	0	0	1	S/C	R/L	—	—	Moves cursor and shifts display without changing DDRAM contents.	37 µs
Function set	0	0	0	0	1	DL	N	F	—	—	Sets interface data length (DL), number of display lines (N), and character font (F).	37 µs
Set CGRAM address	0	0	0	1	ACG	ACG	ACG	ACG	ACG	ACG	Sets CGRAM address. CGRAM data is sent and received after this setting.	37 µs
Set DDRAM address	0	0	1	ADD	ADD	ADD	ADD	ADD	ADD	ADD	Sets DDRAM address. DDRAM data is sent and received after this setting.	37 µs
Read busy flag & address	0	1	BF	AC	AC	AC	AC	AC	AC	AC	Reads busy flag (BF) indicating internal operation is being performed and reads address counter contents.	0 µs

The DR temporarily stores data to be written into DDRAM or CGRAM and temporarily stores data to be read from DDRAM or CGRAM. Data written into the DR from the MPU is automatically written into DDRAM or CGRAM by an internal operation. The DR is also used for data storage when reading data from DDRAM or CGRAM. When address information is written into the IR, data is read and then stored into the DR from DDRAM or CGRAM by an internal operation. Data transfer between the MPU is then completed when the MPU reads the DR. After the read, data in DDRAM or CGRAM at the next address is sent to the DR for the next read from the MPU. By the register selector (RS) signal, these two registers can be selected.

Instruction	Code RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Description	Execution Time (max) (when f_{cp} or f_{OSC} is 270 kHz)
Write data to CG or DDRAM	1	0	Write data								Writes data into DDRAM or CGRAM.	37 µs t_{ADD} = 4 µs*
Read data from CG or DDRAM	1	1	Read data								Reads data from DDRAM or CGRAM.	37 µs t_{ADD} = 4 µs*
	I/D = 1: Increment I/D = 0: Decrement S = 1: Accompanies display shift S/C = 1: Display shift S/C = 0: Cursor move R/L = 1: Shift to the right R/L = 0: Shift to the left DL = 1: 8 bits, DL = 0: 4 bits N = 1: 2 lines, N = 0: 1 line F = 1: 5 × 10 dots, F = 0: 5 × 8 dots BF = 1: Internally operating BF = 0: Instructions acceptable										DDRAM: Display data RAM CGRAM: Character generator RAM ACG: CGRAM address ADD: DDRAM address (corresponds to cursor address) AC: Address counter used for both DD and CGRAM addresses	Execution time changes when frequency changes Example: When f_{cp} or f_{OSC} is 250 kHz, $37\ \mu s \times \frac{270}{250} = 40\ \mu s$

Note: — indicates no effect.

* After execution of the CGRAM/DDRAM data write or read instruction, the RAM address counter is incremented or decremented by 1. The RAM address counter is updated after the busy flag turns off. In Figure 10, t_{ADD} is the time elapsed after the busy flag turns off until the address counter is updated.

Figure 3. Instructions LCD

Display Data Ram

Display data RAM (DDRAM) stores display data represented in 8-bit character codes, which have a capacity of 80 x 8 bits, or 80 characters, in this case the LCD has 2 lines with 16 columns, so it's possible to show 32 characters in the LCD, The DDRAM stores the character ASCII code that appear on the screen, and there is a correspondence between the rows on the LCD and the consecutive positions.

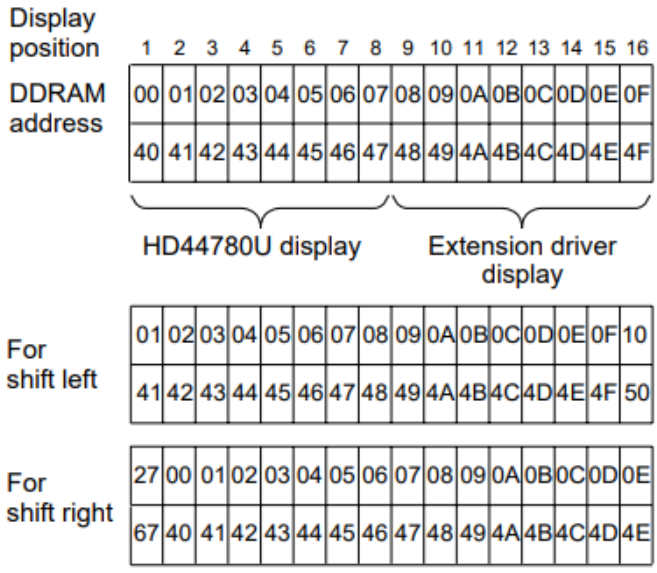


Figure 4, 2 line by 16-Character Display Example

Character Generator ROM (CGROM)

The next table shows the correspondence between Character Codes and Character Patterns (ROM code: A00), the Character generator ROM generates 5x8 dot or 5x10 dot character patterns from 8-bit character codes. It can generate 208 5 x 8 dot character pattern and 32 5 x 10 dot character patterns. User defined character patterns are also available by mask-programmed ROM.

Figure 5. 2 Characters define within the CGROM.

Interfacing to the MPU:

The HD44780U can send data in either 4-bit operations or one 8-bit operations, thus allowing interfacing with 4- 8-bit MPUs.

- For 4-bit interface data, only four bus lines (DB4 to DB7) are used for transfer. Bus lines DB0 to DB3 are disabled. The data transfer between the HD44780U and the MPU is completed after the 4-bit data has been transferred twice. As for the order of data transfer, the four high order bits (for 8-bit operation, DB4 to DB7) are transferred before the four low order bits (for 8-bit operation, DB0 to DB3). The busy flag must be checked (one instruction) after the 4-bit data has been transferred twice. Two more 4-bit operations then transfer the busy flag and address counter data.

- For 8-bit interface data, all eight bus lines (DB0 to DB7) are used.

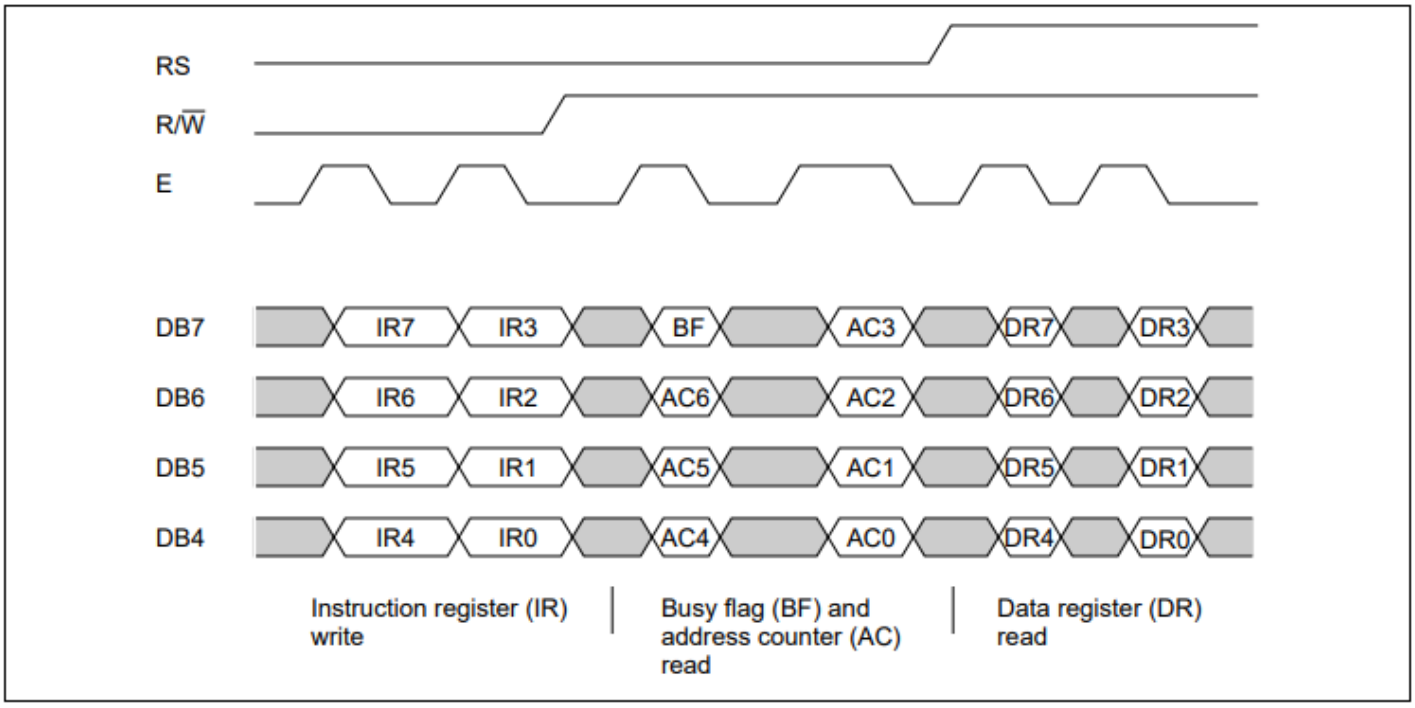


Figure 9 4-Bit Transfer Example

Figure 6. 4-bit transfer Example

Finally, at the moment when the lcd is connected to the power after Vcc rises to 2.7 V, it's necessary do the following steps that the manufacturer recommends to do to initialize the screen, the flow chart is shown below:

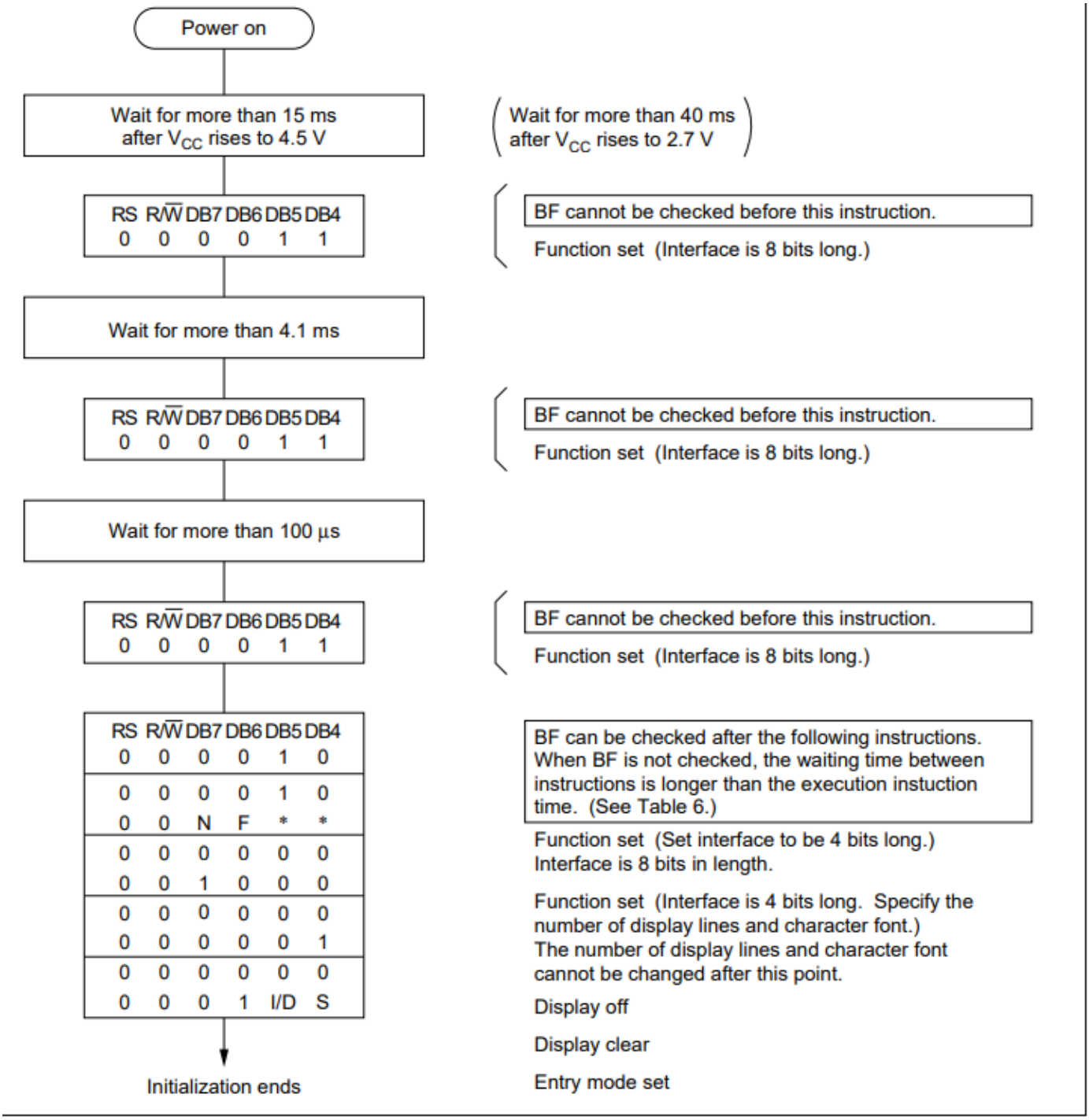


Figure 24 4-Bit Interface

Figure 7. 4-bit interface steps to initialize the LCD

This flow chart is very important, because we need to convert these steps to code to initialize the LCD, so all this information will go inside a function. The next image shows the conversion process based on the connections already proposed (PIC16F887-LCD).

	DATA BUS 4 bits mode				0-Enable 1-Disabled	0-IR 1-DR		
HEX CODE	D7-RB7	D6-RB6	D5-RB5	D4-RB4	E-RB3	RS-RB2	X-RB1	X-RB0
0x00	0	0	0	0	0	0	0	0
	delay 20 ms							
0x38	0	0	1	1	1	0	0	0
	delay 5 ms							
0x30	0	0	1	1	0	0	0	0
0x38	0	0	1	1	1	0	0	0
	delay 150 us							
0x30	0	0	1	1	0	0	0	0
0x38	0	0	1	1	1	0	0	0
	delay 5ms							
0x30	0	0	1	1	0	0	0	0
0x28	0	0	1	0	1	0	0	0
	delay 5ms							
0x20	0	0	1	0	0	0	0	0

	HIGH NIBBLE				Enable	LOW NIBBLE				Enable
0	0	0	1	0	0	1	0	0	0	0
1	0	0	1	0	1	1	0	0	0	1
	delay 5 ms					delay 5ms				
0	0	0	0	0	0	1	1	0	0	0
1	0	0	0	0	1	1	1	0	0	1
2	delay 5 ms					delay 5ms				
0	0	0	0	0	0	0	0	0	1	0
1	0	0	0	0	1	0	0	0	1	1
2	delay 5 ms					delay 5ms				
0	0	0	0	0	0	0	1	1	0	0
1	0	0	0	0	1	0	1	1	0	1
2	delay 5 ms					delay 5ms				

0x28	Enviar comando 00101000 para 4 bits, 2 lineas, 5x7
0x0C	Enviar comando 00001100 para: Display ON, cursor OFF
0x01	clear display
0x06	Enviar comando 00000110 para incrementar y parpadeo apagado

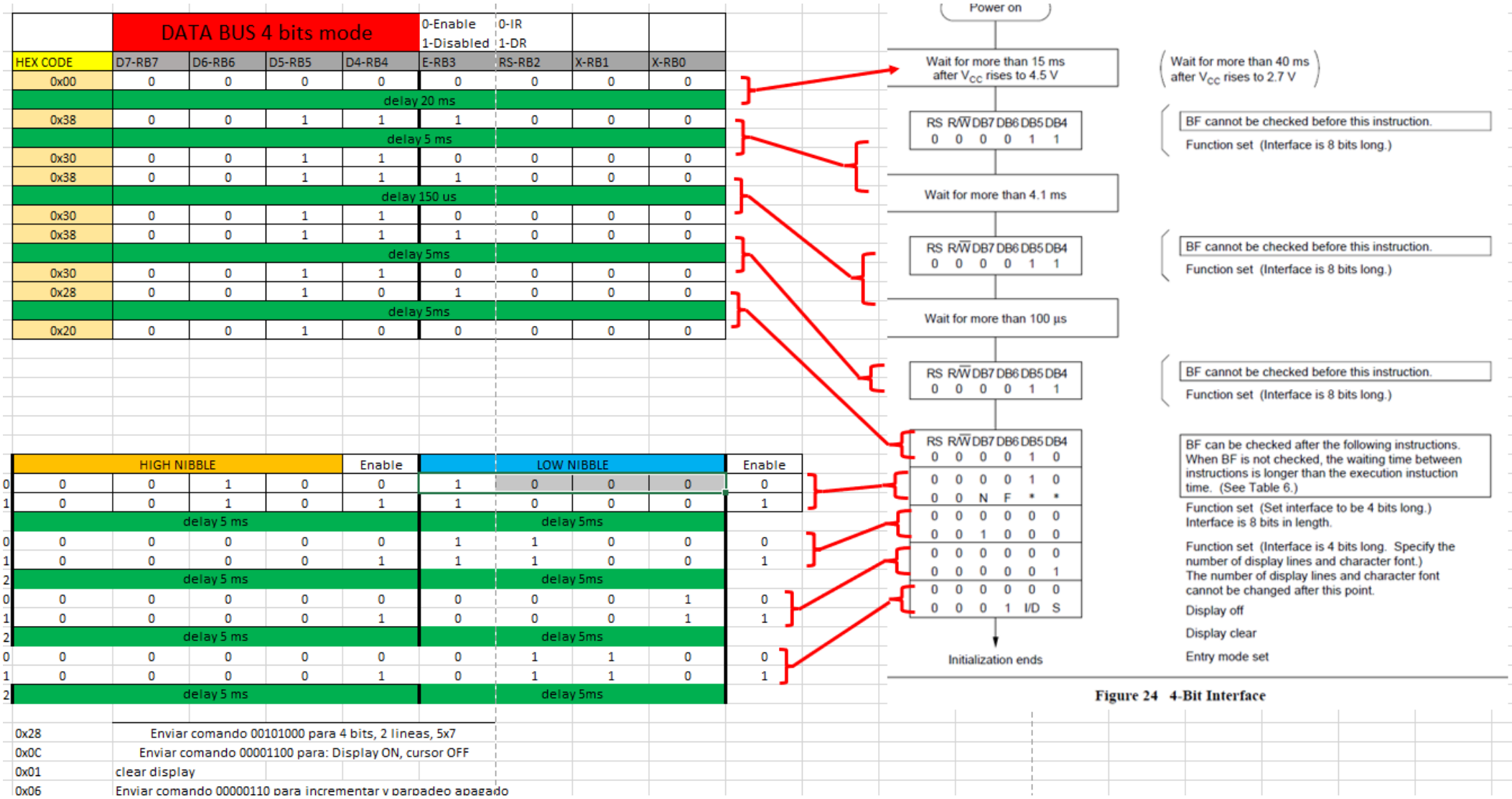


Figure 24 4-Bit Interface

Figure 7. LCD initialization 4-bit mode.

NIBBLE HIGH	Hex code	BIT7	BIT6	BIT5	BIT4		BIT3	BIT2	BIT1	BIT0
	0x28	0	0	1	0		1	0	0	0
	0xF0	1	1	1	1		0	0	0	0
nibble high	&	0	0	1	0		0	0	0	0
	0x08	0	0	0	0		1	0	0	0
nibble high enable	|	0	0	1	0		1	0	0	0
NIBBLE LOW	Hex code	BIT7	BIT6	BIT5	BIT4		BIT3	BIT2	BIT1	BIT0
	0x28	0	0	1	0		1	0	0	0
	<<	1	0	0	0		0	0	0	0
	0xF0	1	1	1	1		0	0	0	0
nibble low	&	1	0	0	0		0	0	0	0
	0x08	0	0	0	0		1	0	0	0
nibble low enable	|	1	0	0	0		1	0	0	0

Figure 8. The operations to send the NIBBLE_HIGH and NIBBLE_LOW

The header file code is this: **LCD_4bits.h**

```
/*
 * File:   LCD_4bits.h
 * Author: Ing. Marcos Becerra.
 * Date: January 11, 2021
 * Copyright (c) 2021 Marcos Becerra
 * see the file LICENSE for copying permission.
 *
 * Version 1.0 developed in 2020.
 * -Development of the source code and its control function.
 * Version 2.0 developed in 2021.
 * -Library creation with its own functions.
 * Version 3.0 developed in 2022.
 * -Integration of more functions.
 *
 * INFO:
 * This library is open source, so you can modify it as you like.
 * The library was developed for the XC8 V2.32 compiler specifically for the
 * PIC16F887 microcontroller, this code is flexible to implement for any 8-bit
 * microcontroller, clearly making certain modifications to be 100% compatible
 * with the device you use, also the library is prepared to work in 4 bit mode.
 *
 * This library is compatible for 16x2 LCD model "LM016L" which uses the chip
 * HITACHI HD44780U or equivalent.
 *
 * For more information contact me:
 * My Account:
 * marcos.bzarate@alumnos.udg.mx
 *
 * My YouTube Channel:
 * https://www.youtube.com/channel/UCaW-5pa18N2Yxjkc4ejOPdQ
 * LCD library header file
 *
 */

/*****************************************************************************/
/*      Pin Connections
 *      |---LCD---|     |--PIC16F887---|
 *          VSS ------------- GND
 *          VDD ------------- +5V.
 *          Vo  ------------- (-)  to the potentiometer.
 *          RS  ------------- RB2
```

```
 *          RW  ------------ GND
 *          E   ------------ RB3
 *        D0..D3------------ (-) No connection, the lcd works in mode 4 bits.
 *          D4  ------------ RB4
 *          D5  ------------  RB5
 *          D6  ------------  RB6
 *          D7  ------------  RB7
 *
 * For the library to work well, it's necessary to include these two functions
 * in the main code.
 *
 * void delay_5ms_LCD(void) { __delay_ms(5); }
 * void delay_2ms_LCD(void) { __delay_ms(2); }
 * void delay_40us_LCD(void) { __delay_us(40); }
 */




#ifndef LCD_4BITS_H
#define LCD_4BITS_H

#include <xc.h>

#ifdef __cplusplus
extern "C" {
#endif
//Macros:
//----------------------------Control commands---------------------------------
#define LCD_CLEAR 0x01                                  //Clear LCD.
#define LCD_CURSOR_HOME 0x02                            //Return Cursor (0,0).

//Entry mode set
#define LCD_DECREASE_DDRAM_DISPLAY_SHIFT_OFF   0x04           //Decrease DDRAM pointer, no shift.
#define LCD_DECREASE_DDRAM_DISPLAY_SHIFT_ON    0x05           //Decrease DDRAM pointer, shift.
#define LCD_INCREASE_DDRAM_DISPLAY_SHIFT_OFF   0x06           //Increase DDRAM pointer, no shift.
#define LCD_INCREASE_DDRAM_DISPLAY_SHIFT_ON    0x07           //Increase DDRAM pointer, shift.

//Display on/off control
#define LCD_DISPLAY_OFF_CURSOR_OFF_CHARACTER_BLINK_OFF  0x08   //Display off, cursor off, and character no blink.
#define LCD_DISPLAY_OFF_CURSOR_OFF_CHARACTER_BLINK_ON   0x09   //Display off, cursor off, and character blink.
#define LCD_DISPLAY_OFF_CURSOR_ON_CHARACTER_BLINK_OFF   0x0A   //Display off, cursor on, and character no blink.
#define LCD_DISPLAY_OFF_CURSOR_ON_CHARACTER_BLINK_ON    0x0B   //Display off, cursor on, and character blink.
#define LCD_DISPLAY_ON_CURSOR_OFF_CHARACTER_BLINK_OFF   0x0C   //Display on, cursor off, and character no blink.
```

```
#define LCD_DISPLAY_ON_CURSOR_OFF_CHARACTER_BLINK_ON    0x0D   //Display on, cursor off, and character blink.
#define LCD_DISPLAY_ON_CURSOR_ON_CHARACTER_BLINK_OFF    0x0E   //Display on, cursor on, and character no blink.
#define LCD_DISPLAY_ON_CURSON_ON_CHARACTER_BLINK_ON     0x0F   //Display on, cursor on, and character blink.

//Cursor or display shift
#define LCD_CURSOR_SHIFT_LEFT   0x10    //Move cursor and shift to the left.
#define LCD_CURSOR_SHIFT_RIGHT  0x14    //Move cursor and shift to the right.
#define LCD_DISPLAY_SHIFT_LEFT  0x18    //Move display to the left.
#define LCD_DISPLAY_SHIFT_RIGHT 0x1C    //Move display to the right.

//Function set
#define LCD_4BITS_1LINE_5X8     0x20    //4 bits, 1 line and 5x8 dots.
#define LCD_4BITS_1LINES_5X10   0x24    //4 bits, 1 line and 5x10 dots.
#define LCD_4BITS_2LINES_5X8    0x28    //4 bits, 2 lines and 5x8 dots.
#define LCD_4BITS_2LINES_5X10   0x2C    //4 bits, 2 lines, and 5x10 dots.
#define LCD_8BITS_1LINE_5X8     0x30    //8 bits, 1 line, and 5x8 dots.
#define LCD_8BITS_1LINE_5X10    0x34    //8 bits, 1 line, and 5x10 dots.
#define LCD_8BITS_2LINES_5X8    0x38    //8 bits, 2 lines, and 5x8 dots.
#define LCD_8BITS_2LINES_5X10   0x3C    //8 bits, 2 lines, and 5x10 dots.

//Select port for lcd.
#define PORT_LCD PORTB
#define TRIS_LCD TRISB

void Lcd_Init(void);
void Lcd_Cmd(char command);
void Lcd_SetCursor(char row, char column);
void Lcd_Chr(char row, char column, char out_char);
void Lcd_Chr_Cp(char out_char);
void Lcd_String(char row, char column, char *text);
void Lcd_String_Cp(char *text);
extern void delay_5ms_LCD(void);
extern void delay_2ms_LCD(void);
extern void delay_40us_LCD(void);

#ifdef  __cplusplus
}
#endif

#endif  /* LCD_4BITS_H */
```

The file that contains the source code of the functions is here:

```
#include "LCD_4bits.h"

void Lcd_Init(void) {
    char SR = TRIS_LCD & 0x03;
    char data = PORT_LCD & 0x03;
    TRIS_LCD = (0x00 | SR);
    PORT_LCD = (0x00 | data);
    for(char a = 0; a < 4; a++) delay_5ms_LCD();
    PORT_LCD = (0x38 | data);
    delay_5ms_LCD();;
    PORT_LCD = (0x30 | data);
    PORT_LCD = (0x38 | data);
    for(char a = 0; a < 3; a++) delay_40us_LCD();
    PORT_LCD = (0x30 | data);
    PORT_LCD = (0x38 | data);
    delay_5ms_LCD();;
    PORT_LCD = (0x30 | data);
    PORT_LCD = (0x28 | data);
    delay_5ms_LCD();;
    PORT_LCD = (0x20 | data);

    Lcd_Cmd(LCD_4BITS_2LINES_5X8);
    Lcd_Cmd(LCD_DISPLAY_ON_CURSOR_OFF_CHARACTER_BLINK_OFF);
    Lcd_Cmd(LCD_CLEAR);
    Lcd_Cmd(LCD_INCREASE_DDRAM_DISPLAY_SHIFT_OFF);
}
void Lcd_Cmd(char cmd)  {
    char nibble_high, nibble_high_enable, nibble_low, nibble_low_enable;
    char data;
    data = PORT_LCD & 0x03;
    nibble_high = (cmd & 0xF0) | data;
    nibble_high_enable = nibble_high | 0x08;
    nibble_low = ((cmd << 4) & 0xF0) | data;
    nibble_low_enable = nibble_low | 0x08;

    PORT_LCD = nibble_high_enable,
    delay_2ms_LCD();
    PORT_LCD = nibble_high;
    PORT_LCD = nibble_low_enable;
    delay_2ms_LCD();
    PORT_LCD = nibble_low;
```

```
}
void Lcd_SetCursor(char row, char column)   {
    char address;
    address = 0x80+((row-1)*0x40)+(column-1);
    Lcd_Cmd(address);
}
void Lcd_Chr_Cp(char out_char)  {
    char nibble_high, nibble_high_enable, nibble_low, nibble_low_enable;
    char data;
    data = PORT_LCD & 0x03;
    nibble_high = (out_char & 0xF0) | 0x04 | data;;
    nibble_high_enable = nibble_high | 0x08;
    nibble_low = ((out_char << 4) & 0xF0) | 0x04 | data;
    nibble_low_enable = nibble_low | 0x08;

    PORT_LCD = nibble_high_enable;
    delay_40us_LCD();
    PORT_LCD = nibble_high;
    PORT_LCD = nibble_low_enable;
    delay_40us_LCD();
    PORT_LCD = nibble_low;
}
void Lcd_Chr(char row, char column, char out_char)  {
    char address;
    address = 0x80+((row-1)*0x40)+(column-1);
    Lcd_Cmd(address);
    Lcd_Chr_Cp(out_char);
}
void Lcd_String_Cp(char *txt)  {
    for(char a = 0; txt[a] != '\0'; a++)    {
        Lcd_Chr_Cp(txt[a]);
    }
}
void Lcd_String(char row, char column, char *txt)   {
    char address;
    address = 0x80+((row-1)*0x40)+(column-1);
    Lcd_Cmd(address);
    for(char a = 0; txt[a] != '\0'; a++)    {
        Lcd_Chr_Cp(txt[a]);
    }

}
```

Downloads from my GitHub repository:

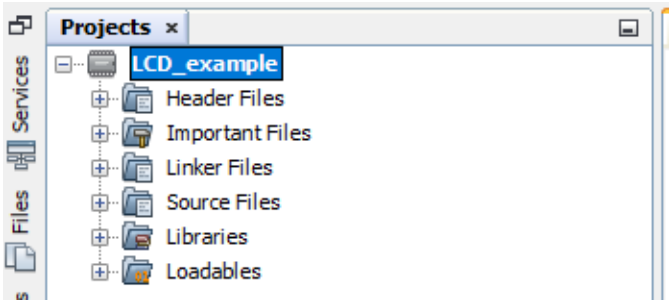
https://github.com/MarcosBeZa/LCD_4bits.git

My YouTube Channel, you can see all the videos about this Library:

https://youtu.be/70_wHiakESY

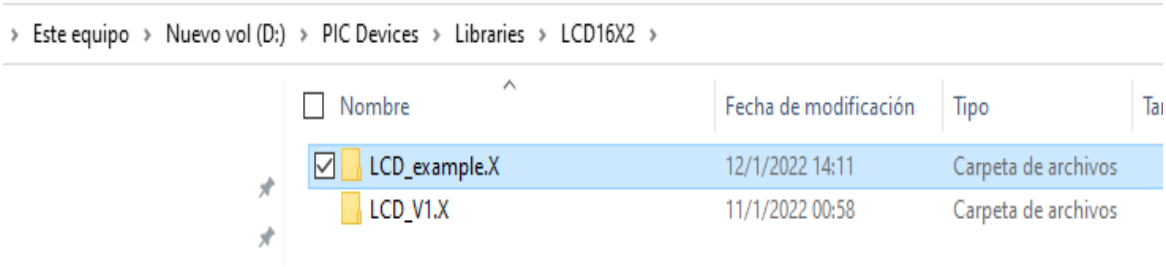
Steps to incorporate the library in your project.

- Create a new project in MPLAB X:
  Once the project is created, find the folder where the project is located, example:



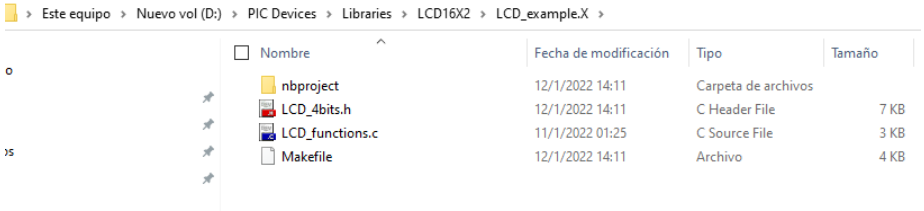
My project created

Find the folder where the project is located:



My project is located

- Page the header file (LCD_4bits.h) and the source code of the library (LCD_functions.c) into the carpet, example:



Page the two files LCD_4bits.h and LCD_functions.c

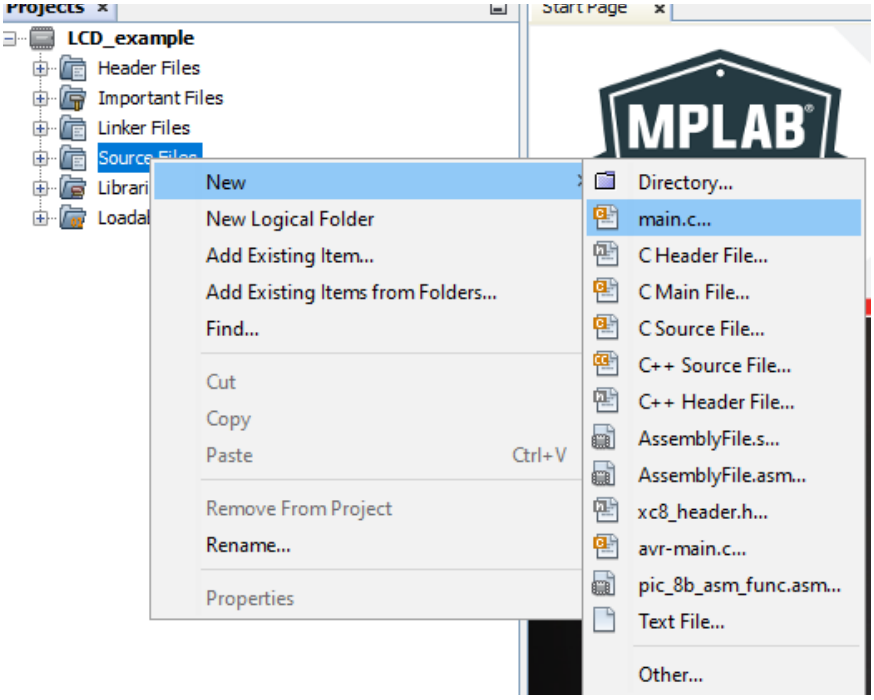
- Now we create a new file **newmain.c** to sabe the source code of our project and paste the next code:

```
#pragma config FOSC = HS
#pragma config WDTE = OFF
```

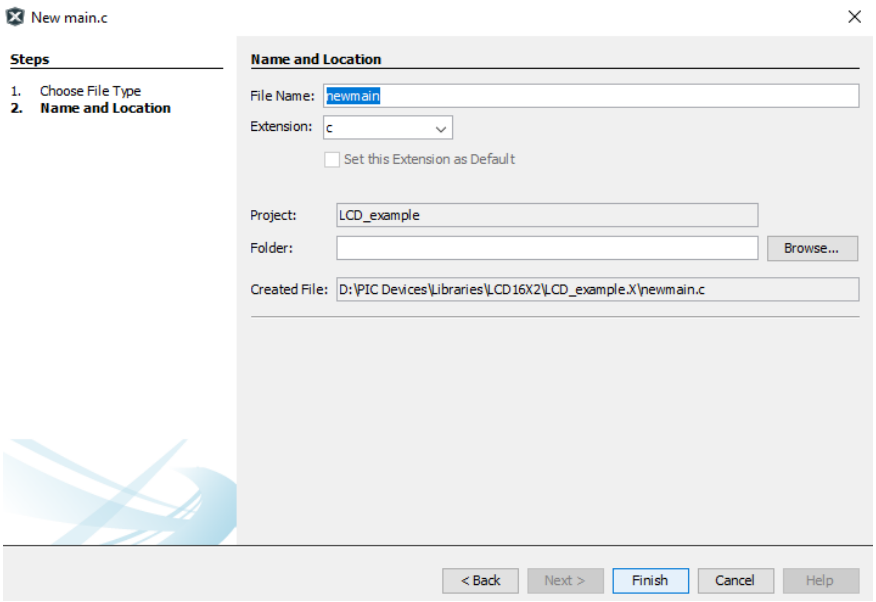
```
#pragma config LVP = OFF
#pragma config FCMEN = OFF

#include <xc.h>
#include "LCD_4bits.h"
#define _XTAL_FREQ 16000000

void setup(void);
void delay_5ms_LCD(void);
void delay_2ms_LCD(void);
void delay_40us_LCD(void);

void main(int argc, char** argv) {
    setup();
    Lcd_Init();
    Lcd_Cmd(LCD_CLEAR);
    Lcd_SetCursor(1,1);
    Lcd_String_Cp("Mi Libreria LCD");
    Lcd_SetCursor(2,1);
    Lcd_String_Cp("Hola a todos. ");
    __delay_ms(1000);

    while(1)    {}
    return;
}

void setup()    {
    OSCCONbits.SCS = 0;             //Clock source defined by FOSC<2:0> of the
CONFIG1 register
    ANSELH = 0x00;
    TRISB = 0xFC;
    PORTB = 0x00;
}

void delay_5ms_LCD(void) {
    __delay_ms(5);
}
void delay_2ms_LCD(void) {
    __delay_ms(2);
}
void delay_40us_LCD(void) {
    __delay_us(40);
}
```

The source code of the project.

First step, create a newmain.c:

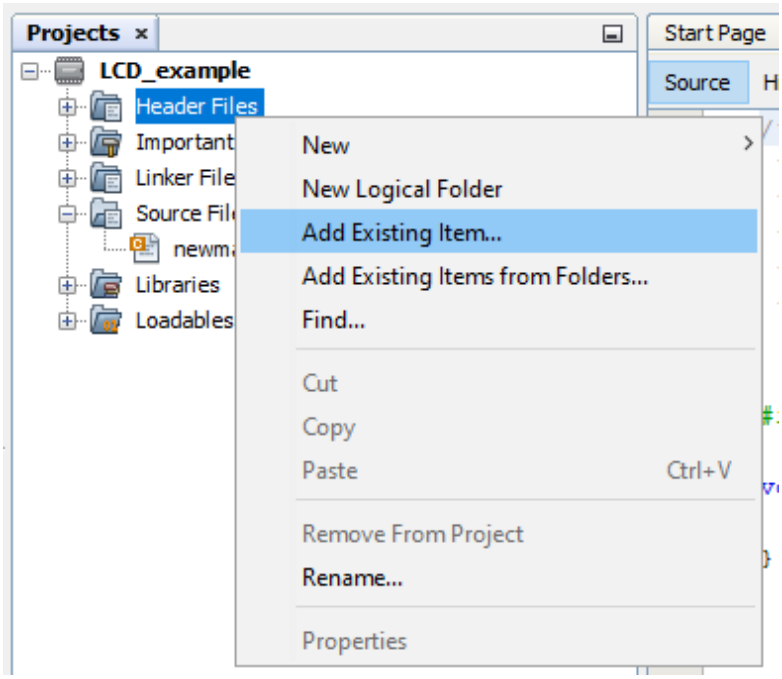


First step create a main.c

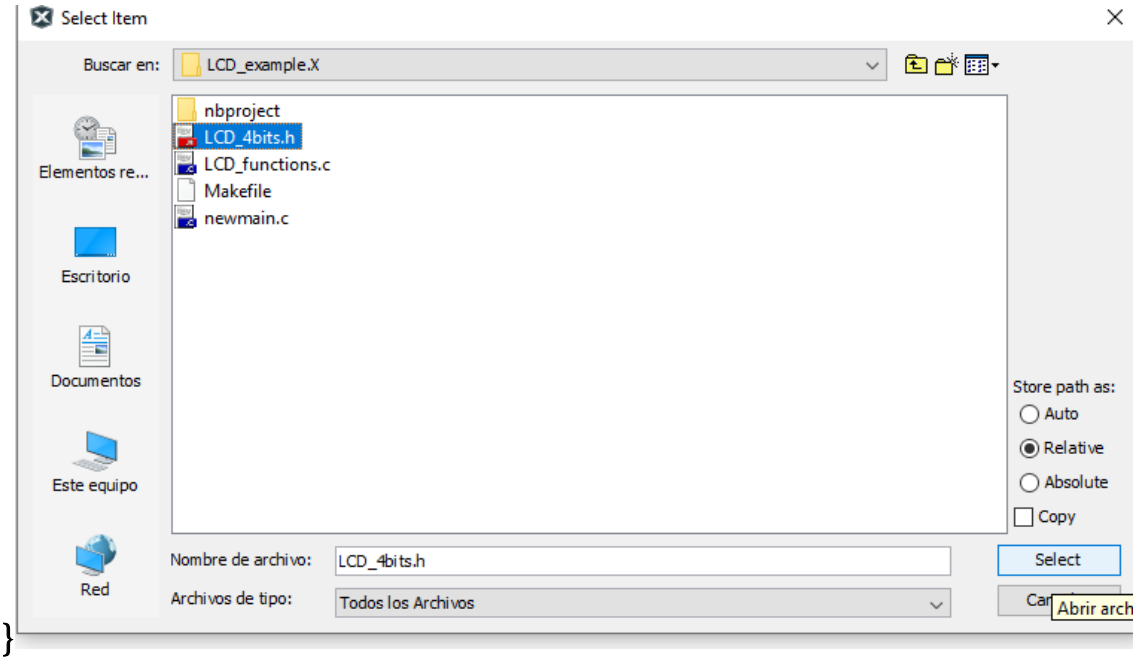


The default name is newmain.c and click on finish

Later, you must do the next steps to add the header file and source code .c:

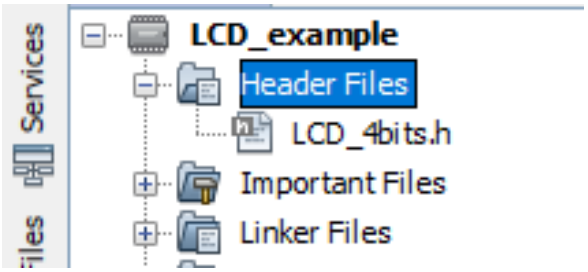


Add the existing header file

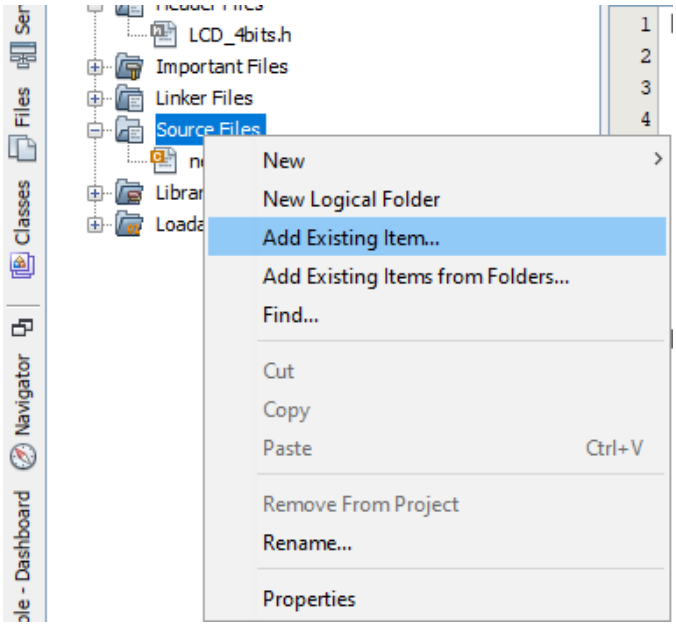


}

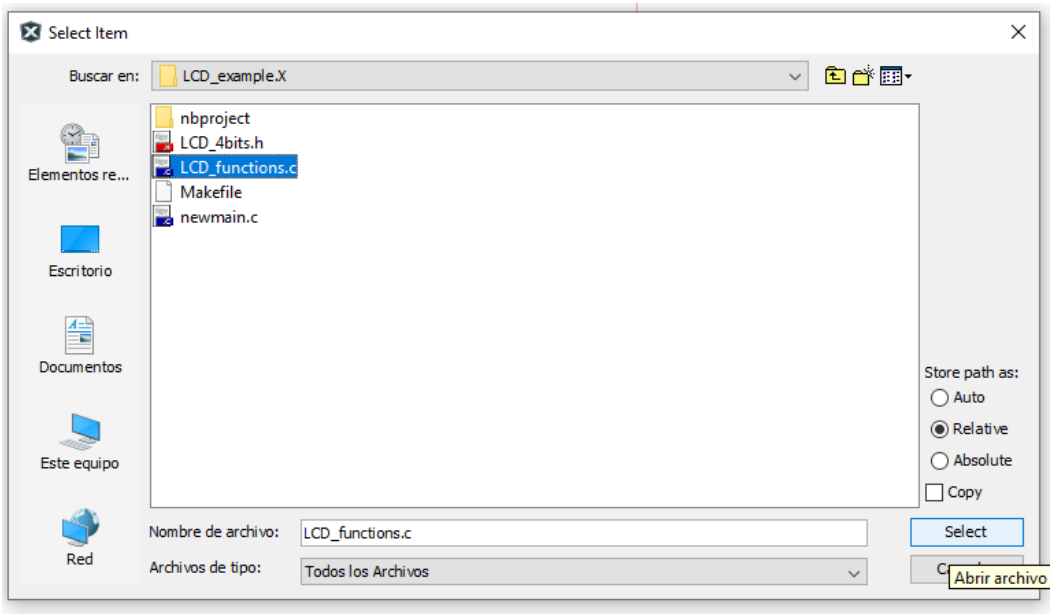
Select the LCD_4bits.h file.



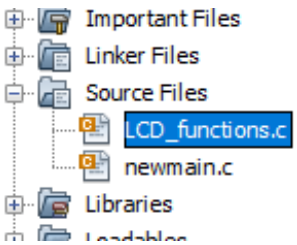
The file was added



Add the existing LCD_function.c file



Click on select

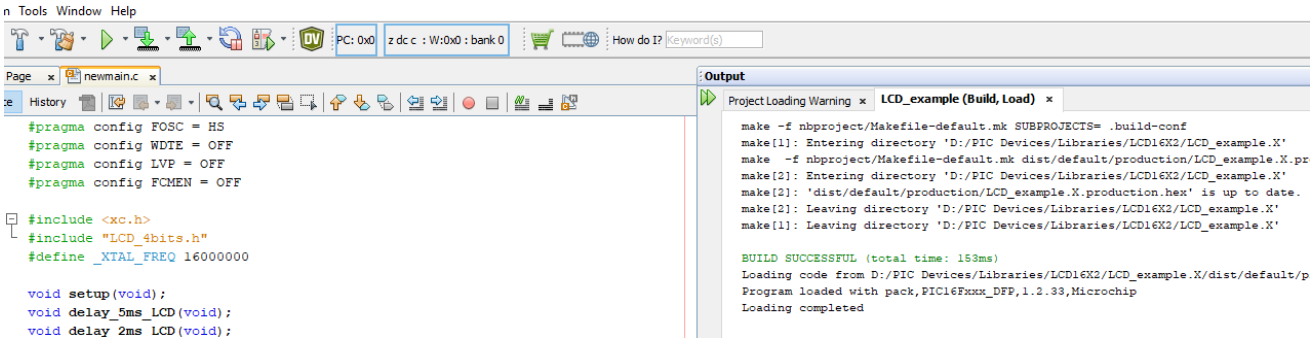


The file was added

Finally paste the example code:

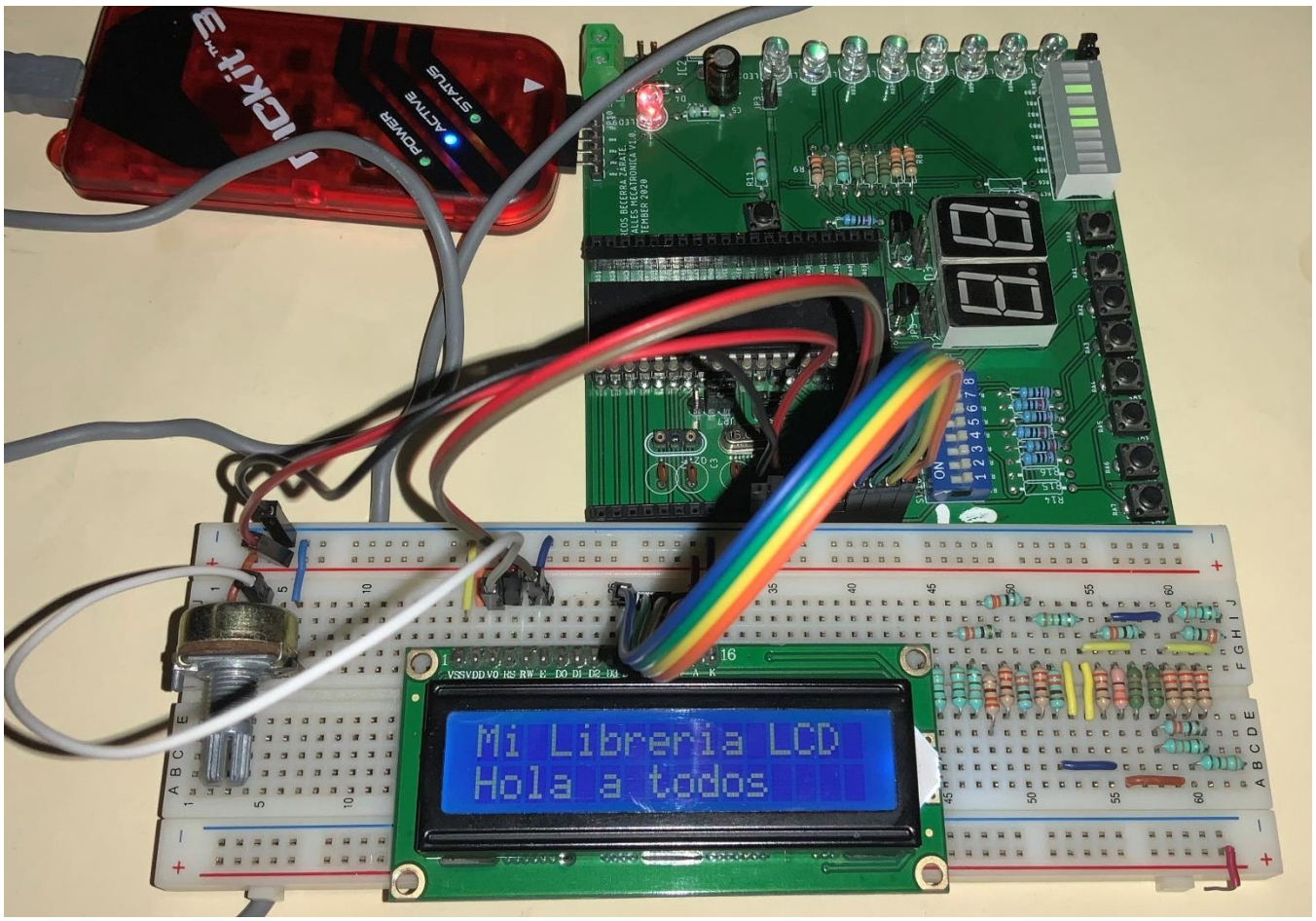
Example code

Now compile the code:



Compilation successful

The code works:



Finally, we can create a project using proteus, as shown: