3 Lab 3 – Liquid Crystal Display (LCD)

3.1 Aim

This laboratory practical will introduce you to an industry-standard 2 x 16 liquid crystal display (LCD) which is interfaced to the MCU using a HD44780 driver IC. Both the LCD and driver IC integrated together on the LCD PCB that is mounted on your tutorial board. This lab will explore the development of code to display characters and numerals on the LCD using functions provided by a custom LCD library.

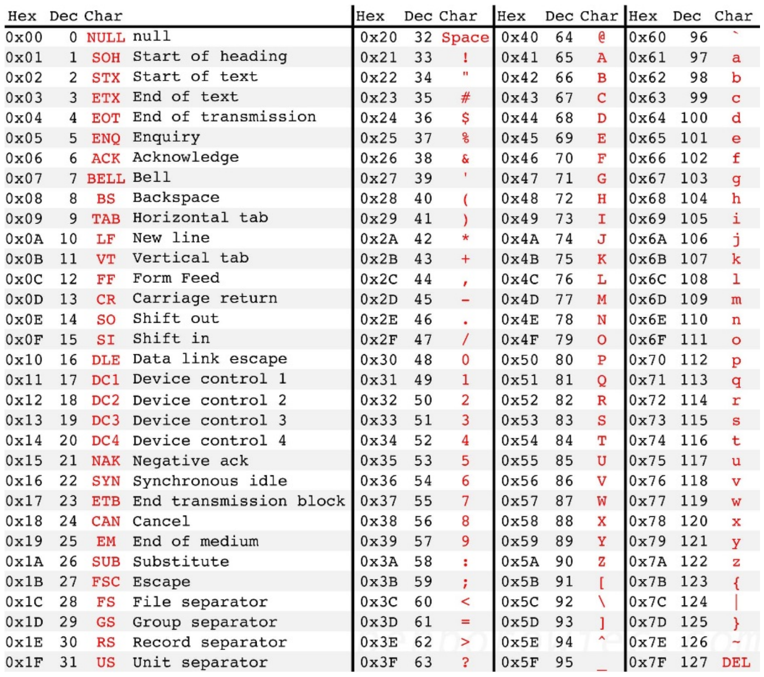
3.2 Learning outcomes

- Understand how an industry-standard generic LCD display can be interfaced with a microcontroller using the HD44780 driver IC.
- Understand how to add a custom library to your projects to provide useful functionality.
- Write code for the PIC16F882 to utilise an LCD display for a variety of display purposes.

3.3 Background

3.3.1 Storing alphanumeric characters in C

ASCII stands for American Standard Code for Information Interchange. Because computers only store binary, a representation of other characters e.g. the alphabet needs to be provided. ASCII code is a numerical representation of common characters e.g. 'a' or '#'. Figure 3-1 shows the standard ASCII character table. The first 32 characters are 'non-printable' characters because ASCII was originally designed for use with, now archaic, equipment such as teletypes. Most of these non-printable characters rarely find use in modern computer systems with some notable exceptions e.g. tab (TAB), carriage return (CR), and linefeed (LF). The printable characters go from 32 to 127 (decimal).



Hex	Dec	Char		Hex	Dec	Char	Hex	Dec	Char	Hex	Dec	Char
0x00	0	NULL	null	0x20	32	Space	0x40	64	@	0x60	96	`
0x01	1	SOH	Start of heading	0x21	33	!	0x41	65	A	0x61	97	a
0x02	2	STX	Start of text	0x22	34	"	0x42	66	B	0x62	98	b
0x03	3	ETX	End of text	0x23	35	#	0x43	67	C	0x63	99	c
0x04	4	EOT	End of transmission	0x24	36	$	0x44	68	D	0x64	100	d
0x05	5	ENQ	Enquiry	0x25	37	%	0x45	69	E	0x65	101	e
0x06	6	ACK	Acknowledge	0x26	38	&	0x46	70	F	0x66	102	f
0x07	7	BELL	Bell	0x27	39	'	0x47	71	G	0x67	103	g
0x08	8	BS	Backspace	0x28	40	(	0x48	72	H	0x68	104	h
0x09	9	TAB	Horizontal tab	0x29	41	)	0x49	73	I	0x69	105	i
0x0A	10	LF	New line	0x2A	42	*	0x4A	74	J	0x6A	106	j
0x0B	11	VT	Vertical tab	0x2B	43	+	0x4B	75	K	0x6B	107	k
0x0C	12	FF	Form Feed	0x2C	44	,	0x4C	76	L	0x6C	108	l
0x0D	13	CR	Carriage return	0x2D	45	-	0x4D	77	M	0x6D	109	m
0x0E	14	SO	Shift out	0x2E	46	.	0x4E	78	N	0x6E	110	n
0x0F	15	SI	Shift in	0x2F	47	/	0x4F	79	O	0x6F	111	o
0x10	16	DLE	Data link escape	0x30	48	0	0x50	80	P	0x70	112	p
0x11	17	DC1	Device control 1	0x31	49	1	0x51	81	Q	0x71	113	q
0x12	18	DC2	Device control 2	0x32	50	2	0x52	82	R	0x72	114	r
0x13	19	DC3	Device control 3	0x33	51	3	0x53	83	S	0x73	115	s
0x14	20	DC4	Device control 4	0x34	52	4	0x54	84	T	0x74	116	t
0x15	21	NAK	Negative ack	0x35	53	5	0x55	85	U	0x75	117	u
0x16	22	SYN	Synchronous idle	0x36	54	6	0x56	86	V	0x76	118	v
0x17	23	ETB	End transmission block	0x37	55	7	0x57	87	W	0x77	119	w
0x18	24	CAN	Cancel	0x38	56	8	0x58	88	X	0x78	120	x
0x19	25	EM	End of medium	0x39	57	9	0x59	89	Y	0x79	121	y
0x1A	26	SUB	Substitute	0x3A	58	:	0x5A	90	Z	0x7A	122	z
0x1B	27	FSC	Escape	0x3B	59	;	0x5B	91	[	0x7B	123	{
0x1C	28	FS	File separator	0x3C	60	<	0x5C	92	\	0x7C	124	|
0x1D	29	GS	Group separator	0x3D	61	=	0x5D	93	]	0x7D	125	}
0x1E	30	RS	Record separator	0x3E	62	>	0x5E	94	^	0x7E	126	~
0x1F	31	US	Unit separator	0x3F	63	?	0x5F	95	_	0x7F	127	DEL

Figure 3-1 The ASCII character set (adapted from http://benborowiec.com/2011/07/23/better-ascii-table)

The standard ASCII character set runs from 0-127 allowing it to be stored as a single byte. C allows for this by providing the *char* type which informs the compiler we wish to store and manipulate data using ASCII codes. We can store an ASCII character in memory using a variable by using the following syntax:

```
char myCharacter = ‘B’;
```

This stores the numeric ASCII code for an uppercase B character (0x42) (see figure 3-1). A sequence of characters may be stored in a *string,* which is a one-dimensional array of characters (ASCII codes) appended with a NULL character (ASCII code 0x00). An example of the syntax is as follows, note the use of double quotes here:

```
char myString[] = “Hello”;
```

Note that in C the NULL character is automatically appended when the array is initialised, so the array in this case is 6 bytes in length. A character sequence between double quotes, in this case “Hello”, is called a string literal.

Table 3-1 Stored data for the string literal "Hello"

Array Index	0	1	2	3	4	5
ASCII Character	H	e	l	l	o	NULL
ASCII code (hex)	0x48	0x65	0x6C	0x6C	0x6F	0x00

The PIC16F882 MCU has very limited amount of RAM with only 128 bytes available. If a large number of strings need storing, for instance to implement a complex LCD menu system, it is easy to envisage running out of RAM quickly. Embedded compilers such as the XC8 compiler in MPLAB X generally interpret that *const* datatypes are to be stored in program memory (ROM), because they are constant and thus cannot be modified. It is sensible practice to declare strings as const if they do not need to be modified during runtime because the PIC16F882 has 2K program words of ROM available which is much larger than the available storage in RAM. The following example shows the syntax for declaring a const string:

```
const char myConstString[] = “Hello”;
```

In this case the compiler assigns the 6 bytes of storage required to the program memory, thus saving valuable RAM. This practive should be implemented whenever possible.

3.3.2 The Liquid Crystal Display (LCD)

This tutorial board LCD contains a 16 character x 2 line alphanumeric LCD which is driven by an industry-standard driver IC, the Hitachi HD44780. Each LCD character is made up of a 5 x 8 dot matrix which is able to display any character from the standard ASCII character set as well as user-defined characters. The 16 character columns on the LCD are denoted by numbers starting from the left (0) and extending to the right (15). The top line of the LCD is row 0, the bottom line is row 1.

The HD44780 driver IC takes incoming instructions and data from a 6 wire parallel bus and directly addresses the dot matrix of pixel that the LCD comprises. The LCD is connected to PORTC of the PIC16F882. Pins RC0, RC3 & RC4-7 are used to communicate with the HD44780, and thus control the LCD.

Data is transferred on pins RC4-7, with each byte being transferred as two consecutive nibbles. A high-to-low transition on the E pin of the HD44780 (connected to RC3) latches the nibble value on lines D4 to D7. The *Register Select* (RS) pin (connected to RC0) tells the HD44780 whether the data being latched is an instruction (RS=0) or data (RS=1).

The PORTC pins assigned to the LCD should not be used for any other GPIO purposes and should be set up as outputs at all times.

3.3.3 Custom LCD library for MPLAB X

Interacting directly with the HD44780 in order to drive the LCD using the PORTC pins directly is a complex process, so a custom library of useful functions has been provided to you to enable a more simple interaction.

The library consists of a header file, *LCDdrive882.h* which contains the essential information to use the library functions e.g. number and types of parameters. There is also a C source file, *LCDdrive882.c* containing the actual function definitions. Both files may on Brightspace in the laboratory work area, a copy of these files should be downloaded for later use. If you wish to deploy the custom LCD library, you should take a copy of both the *LCDdrive882.h* and *LCDdrive882.c* files and place them in the project directory so they will be easily linked during the build process.

The *LCDdrive882.c* file must be added to your projects prior to use by using the *Add Existing Item…* context menu option from the project navigator window.

You must also include the header file in any source file within the project that makes use of the custom LCD library functions by adding the following line near the top of the file:

```
#include “LCDdrive882.h”
```

3.3.4 Summary of custom LCD library functions

Function Name	Function Purpose	Usage example
`void LCD_initialise (void)`	Send initialisation commands to LCD. You MUST run this function once before prior to using any of the LCD display functions contained in this library.	`LCD_initialise();`
`void LCD_clear (void)`	Clears the display and homes the cursor to position (0,0).	`LCD_clear();`
`void LCD_putch (unsigned char)`	Writes a single character to the LCD at the current cursor position.	`LCD_putch('f');`

`void LCD_puts ( unsigned char *)`	Write string to LCD. NB for const strings use the LCD_putsc() function.	`char myString[] = "Hello";`
		`LCD_puts(myString);`

`void LCD_putsc ( const unsigned char *)`	Write const string s to LCD. Use this function to display strings that you have declared as const so they are stored in program memory.	`const char myString[] =`
		`"Goodbye";`
		`LCD_putsc (myString);`

`void LCD_cursor (unsigned char, unsigned char)`	Move cursor to position (column, row). The column parameter may assume any value between 0 (leftmost) and 15 (rightmost).). The row parameter may assume a value of 0 or 1 only, representing the top and bottom line respectively.	`LCD_cursor(13,1);`
`void LCD_cursor_on (void)`	Turn cursor flash on	`LCD_cursor_on();`
`void LCD_cursor_off (void)`	Turn cursor flash off	`LCD_cursor_off();`

`void LCD_display_value (unsigned int)`	Displays an unsigned numerical value on the LCD. The function will calculate and display the correct number of digits up to a maximum of 4. Thus it will correctly display values ranging from 0 to 9999.	`unsigned char myNumber = 64;`
		`LCD_display_value(myNumber);`

`void LCD_display_float (float, unsigned int)`	Displays a floating point numerical value on the LCD. Takes the parameter 'value' and displays it to the number of decimal places specified by the second parameter. NOTE: The function is limited to displaying 4 significant figures due to headroom limitations of the 24 bit float type in MPLAB X. Attempting to display more than 4 sig figs will result in string "ERR" displayed.	`Float myNumber = 12.35;`
		`LCD_display_float(myNumber, 2);`

Note: The LCD module **must** be initialised before sending any data/instructions. The *LCD_initialise()* function must be executed once prior to using any of the other LCD library functions. The *LCD_initialise()* function configures the display to 16x2 characters, clears it, and places the cursor at the home position (0,0).

3.4 Procedure

3.4.1 Exercise 1 – Demonstrating the custom LCD library

1) Create a new project using in the MPLAB X IDE called *Lab3Ex1* or similar on your K: drive.

2) Download a copy the files *LCDdrive882.h* and *LCDdrive882.c* from Brightspace and place a copy in your project directory, this will be named *<Project_name>.X.* If you followed the suggested naming convention above the directory will be called *Lab3Ex1.X*.

   You now have the required library files in your project directory, but you still need to add them to your project.

3) In the project navigator window (see figure 3-2), right click on the *Source Files* icon and select *Add Existing Item…* from the context menu. Select the file *LCDdrive882.c* stored in the current project directory and click *Select.* You should see the file represented under the *Source Files* icon in the project directory.

4) You have now successfully added the custom LCD library file to your project. Of course we also need to create a source file for our code, as normal. Click Add New Item… and add a suitably named empty source file e.g. *Lab3Ex1.c*

   By the time you have finished the Projects Navigator window should look something like figure 3-2.

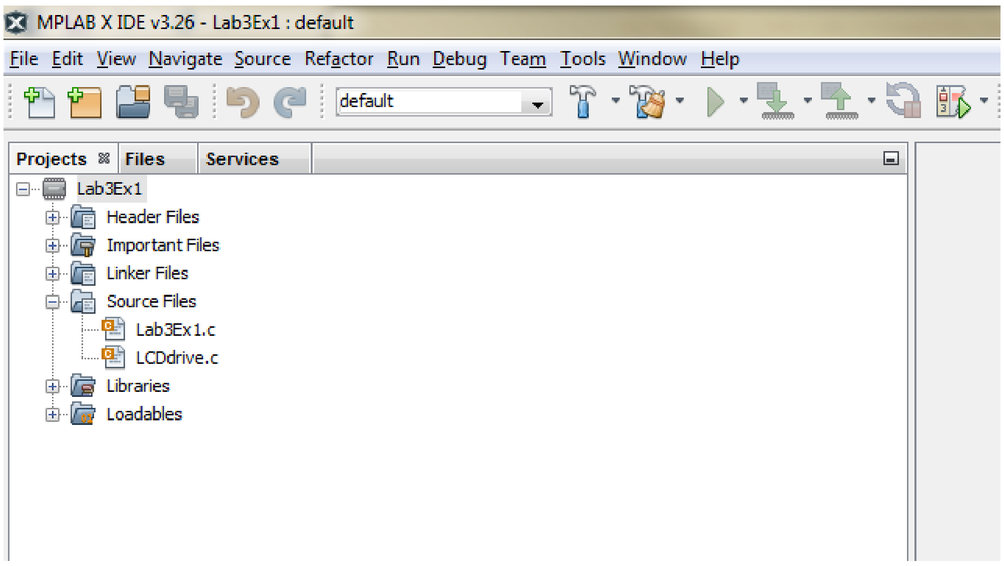


Figure 3-2 The projects navigator window in the MPLAB X IDE

5) We are now ready to utilise the functions provided by the LCD library so populate the empty source files you created with the following code listing:

```
// Filename:     Lab3Ex1.c
// Version:      1.0
// Date:         <Insert current date>
// Author:       <Insert your name>
//
// Description: Simple demonstration of the LCD custom library functions
// PIC16F882 Configuration Bit Settings
#pragma config FOSC = INTRC_NOCLKOUT // Oscillator Selection bits
#pragma config WDTE = OFF       // Watchdog Timer Enable bit
#pragma config PWRTE = OFF      // Power-up Timer Enable bit
#pragma config MCLRE = ON       // RE3/MCLR pin function select bit
#pragma config CP = OFF         // Code Protection bit
#pragma config CPD = OFF        // Data Code Protection bit
#pragma config BOREN = OFF      // Brown Out Reset Selection bits
#pragma config IESO = OFF       // Internal External Switchover bit
#pragma config FCMEN = OFF      // Fail-Safe Clock Monitor Enabled bit
#pragma config LVP = OFF        // Low Voltage Programming Enable bit
#pragma config BOR4V = BOR40V   // Brown-out Reset Selection bit
#pragma config WRT = OFF        // Flash Program Memory Self Write Enable bits

#include <xc.h>             // Required by compiler, PIC specific definitions
#include "LCDdrive882.h"    // Header file needed to access to LCD custom library
#define _XTAL_FREQ 4000000  // MCU clock speed - required for delay macros

void main(void)
{
    char myString[] = {"Embedded Systems"}; // Initialise an array with a string
    unsigned short i;                       // Indexing variable

    LCD_initialise();          // Initialise the LCD ready for use
    LCD_puts(myString);        // Display the myString string
    LCD_cursor(0,1);           // Move the cursor to the 2nd line
    LCD_cursor_on();           // Turn flashing cursor ON

   // Illustration of the putch() function
    __delay_ms(2000);          // Wait 2 seconds
    LCD_putch('A');            // Print character A
    __delay_ms(2000);          // Wait 2 seconds
    LCD_putch('B');            // Print character B
    __delay_ms(2000);          // Wait 2 seconds
    LCD_putch('C');            // Print character C
    __delay_ms(2000);          // Wait 2 seconds
    LCD_cursor_off();          // Turn flashing cursor OFF

    for (i=0; i<1000; i++)      // Set up loop to display counting numerals
    {
        LCD_cursor(4,1);        // Move the cursor to the column 4 on the 2nd line
        LCD_display_value(i);   // Display the value of x at the cursor position
        __delay_ms(100);        // Short delay
    }
}
```

6) Study the code listing carefully, ensuring you read all the comments. Build the project and upload the HEX file to the PIC16F882 and observe what happens.

7) The code demonstrates the following operations on the LCD:

   - Displaying a numerical value
   - A flashing cursor
   - Printing of individual characters
   - Printing of a string

   Determine which sections of the code listing relate to each of operations listed above.

8) Write a high-level (plain English) flow chart to illustrate the execution of the code.

9) Look at the dashboard sidebar and make a note of the program and data memory used for the code above.

   Now change the definition of the string *mystring* from type *char* to *const char.* Rebuild the project and note any differences in memory usage. What is the reason for the change?

TIP!

The cursor automatically moves to the next column on the right once a character or string has been printed on the LCD, just like it would in a text editor. This is the case whether the cursor has been made visible or not.

3.4.2 Exercise 2 – Displaying strings on the LCD

1) Start a new project and link the custom LCD library files as you did in section 3.4.1. Populate a new source file with the following code listing:

```
// Filename:     Lab3Ex2.c
// Version:      1.0
// Date:         <Insert current date>
// Author:       <Insert your name>
//
// Description: Simple demonstration of the LCD custom library functions

// PIC16F882 Configuration Bit Settings
#pragma config FOSC = INTRC_NOCLKOUT // Oscillator Selection bits
#pragma config WDTE = OFF       // Watchdog Timer Enable bit
#pragma config PWRTE = OFF      // Power-up Timer Enable bit
#pragma config MCLRE = ON       // RE3/MCLR pin function select bit
#pragma config CP = OFF         // Code Protection bit
#pragma config CPD = OFF        // Data Code Protection bit
#pragma config BOREN = OFF      // Brown Out Reset Selection bits
#pragma config IESO = OFF       // Internal External Switchover bit
#pragma config FCMEN = OFF      // Fail-Safe Clock Monitor Enabled bit
#pragma config LVP = OFF        // Low Voltage Programming Enable bit
#pragma config BOR4V = BOR40V   // Brown-out Reset Selection bit
#pragma config WRT = OFF        // Flash Program Memory Self Write Enable bits

#include <xc.h>             // Required by compiler, PIC specific definitions
#include "LCDdrive882.h"    // Header file needed to access to LCD custom library
#define _XTAL_FREQ 4000000  // MCU clock speed - required for delay macros

void main(void)
{

}
```

2) Starting out with the template above develop code to perform the following functions:

- Print your first name on the top line of the LCD.

- Print your last name on the bottom line of the LCD. This should be done at a rate of one character per second, displaying from left to right.

This will require storing your name data as a string (array of type char) and indexing this string in either a *for* or *while* loop.

You may view the correct operation for your developed code in this task by uploading the HEX file *DisplayName.hex* (available on Brightspace).

TIP!

When you start a new project that is similar to the previous one you can save a lot of setup time by using the *Copy...* function in MPLAB X. This function copies the whole project directory including all files and linkages and the project projecties.

Simply right-click on the relevant project title on the *Projects* sidebar and select *Copy...* from the context menu. You will be prompted to enter a name for the project to be copied.

3.4.3 Exercise 3 - Displaying numerical information

1) Develop a new project to display on the top line of the LCD all the odd numbers between 1 and 99. There should be a 0.5 second delay between each update of the displayed number.

   If you use the *LCD_clear()* function between display updates you will notice that the screen flickers. Try and implement your code in such a way that the display update is flicker-free. One way of doing this is to overwrite parts the of the display to be updated with a whitespace characters (ASCII coder 0x20 = ‘ ‘). Such a statement might look like this:

   ```
   LCD_puts("    ");
   ```

   You may view the correct operation for your developed code in this task by uploading the HEX file *CountingOdd.hex* (available on Brightspace).

3.5 Further reading

More information on the Hitachi HD44780 controller can be found on the datasheet available here:

http://www.matrixtsl.com/resources/files/datasheets/HD44780_Datasheet.pdf

More useful information on utilising characters and strings can be found in the MPLAB XC8 C Compiler User Guide available on Unilearn or at:

http://ww1.microchip.com/downloads/en/DeviceDoc/50002053F.pdf.

Relevant sections are as follows:

- Section 5.4.6.3 - Character and string constants
- Section 6.4.5.2 - Character constants and strings

For more information on arrays, character constants and strings more generally you should review the Introduction to C/C++ text supplied with this module (see Brightspace).

In addition, a useful tutorial on arrays in C can be found here:

https://www.tutorialspoint.com/cprogramming/c_arrays.htm

While a follow-on tutorial for strings may be found here:

https://www.tutorialspoint.com/cprogramming/c_strings.htm