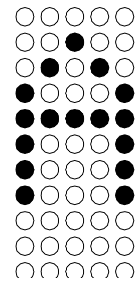
LCD Display Interfacing in C

Background:

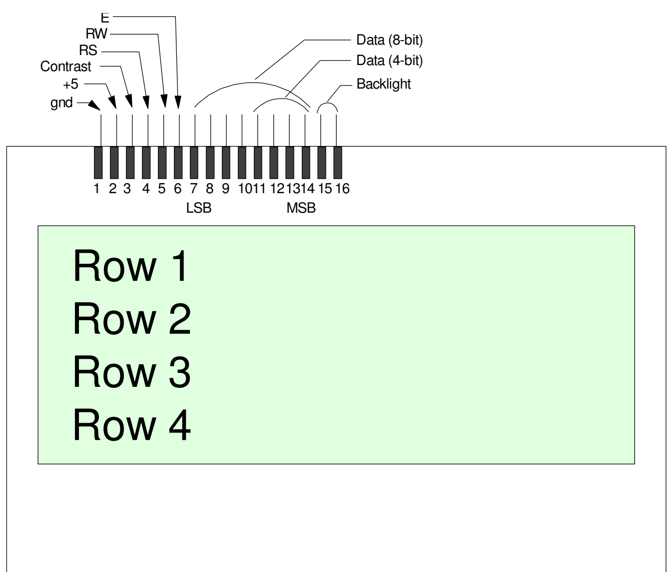
A liquid crystal display (LCD) is an electronic device designed to display data. Liquid crystals are designed so that they polarize light - with the orientation being shifted by 90 degrees when voltage is applied. By using a second polarizing filter, this can create a transparent or black display by applying a voltage.



An alphanumeric LCD display has an array of pixels, typically with 55 pixels grouped together into 5x11 rectangles. A driver chip on the display determines which of these pixels is turned on (dark) to create letters and numbers. For example, an 'A' may look like the figure to the right.

Several rows and columns allow you to display information. The designation of an LCD (such as 16x4) tells you the number of columns and number of rows. Common sizes range from 4x1 up to 32x4. LCD's are also fairly inexpensive, costing between $15 each (16x4 from Jameco) to $130 (20x4 from Digikey).

The driver chips for the LCD are fairly standard - resulting in most (if not all) alphanumeric displays having the same pins in the same order with the same procedures for programming them. Most (all?) alphanumeric displays use 14 or 16 pins. Pins 15 & 16 are only used on LCD with a backlight display.



Pin Connections

Pin	Description
Ground, +5	Power for the LCD. Note: connecting these backwards will destroy the LCD.
Contrast:	0 to 5V signal for the 'brightness' of the display.
RS	Register Select. 1 = an instruction (such as blink the cursor) 0 = data (such as display 'A')
RW	Read / Write 0 = write to the LCD 1 = read data from the LCD
E	Clock. Data or instructions are read in when E is pulsed.
Data 0:7	in 8-bit mode, each byte is read in 8-bits at a time
Data 4:7	in 4-bit mode, each byte is read in two nibbles: left nibble first (MSB), right nibble last (LSB)
Backlight:	0 to 5V (sometimes 12V) to turn on the backlight (if available)

Options:

Instruction	RS	R/W	Data msb ...... lsb	Description
Clear Display	0	0	0000 0001	Clears display and returns cursor to home position (address 0). Execution time: 1.64ms
Home Cursor	0	0	0000 001x	Returns cursor to home position, returns a shifted display to original position. Display data RAM (DD RAM) is unaffected. Execution time: 40us to 1.64ms
Entry Mode Set	0	0	0000 01is	Sets cursor move direction and specifies whether or not to shift display. Execution time: 40us i=1: increment, i=0: decrement DD RAM address by 1 after each DD RAM write or read. s=1: display scrolls in the direction specified by the "i" bit when the cursor is at the edge of the display window
On / Off Control	0	0	0000 1dcb	Turn display on or off, turn cursor on or off, blink character at cursor on or off. Execution time: 40us d=1: display on. c=1: cursor on b=1: blink character at cursor position
Cursor Shift	0	0	0001 srxx	Move cursor or scroll display without changing display data RAM. Execution time: 40us s=1: scroll display, s=0: move cursor. r=1: to the right, r=0: to the left. x= don't care

Function Set	0	0	001d nfxx	Set interface data length, mode, font. Execution time: 40us d=1: 8-bit interface, d=0: 4-bit interface. n=1: 1/16 duty, n=0: 1/8 or 1/11 duty (multiplex ratio). For 2-line displays, this can be thought of as controlling the number of lines displayed (n=0: 1-line, n=1: 2-line) except for 1x16 displays which are addressed as if they were 2x8 displays--two 8-character lines side by side. f=1: 5x11 dots, f=0: 5x8 dots.
Character RAM Address Set	0	0	01aa aaaa	To read or write custom characters. Character generator (CG) RAM occupies a separate address space from the DD RAM. Data written to, or read from the LCD after this command will be to/from the CG RAM. Execution time: 40us. aaaaaa: 6-bit CG RAM address to point to.
Display RAM Address Set	0	0	1aaa aaaa	Reposition cursor. Display Data (DD) RAM occupies a separate address space from the CG RAM. Data written to, or read from the LCD after this command will be to/from the DD RAM. Execution time: 40us aaaaaaa: 7-bit DD RAM address to point to.
Write Data to CD or DD RAM	1	0	dddd dddd	Data is written to current cursor position and (DD/CG) RAM address (which RAM space depends on the most recent CG_RAM_ Address_Set or DD_RAM_Address_Set command). The (DD/CG) RAM address is incremented/decremented by 1 as determined by the "entry mode set" command. Execution time: 40us for120us for character generator ram write. dddddddd: 8-bit character code

Programming: (pulse E after each step)

Initialize to 4-bit mode. Clear the screen. Move the cursor to the top left position. Place each subsequent character one spot to the right.

Step	RS	R/W	D7:D4	then wait...	Comment
				15ms	Power On
1	0	0	0011	4.1 ms	
2	0	0	0011	100us	
3	0	0	0011	4.1ms	
4	0	0	0010	40us	4-bit mode
5	0	0	0010		
6	0	0	1Fxx	40us	F = font. 1 = 5x11, 0 for 5x8
7	0	0	0000		
8	0	0	1000	40us	Display off, cursor off, blink off
9	0	0	0000		
10	0	0	0001	1.6ms	Clear screen, cursor home
11	0	0	0000		
12	0	0	0110	40us	Increment cursor to the right when writing. Don't shift the screen.
Initialization Complete					

Place an 'AB' at location (2,6) (address 0x96)

Address = Row + Column

Row	Address of Col #0 (16xN LCD)	Address of Col #0 (20xN LCD)
0	0x80	0x80
1	0xC0	0xC0
2	0x90	0x94
3	0xD0	0xD4

Procedure:

Step	RS	R/W	D7:D4	then wait...	Comment
1	0	0	9		
2	0	0	6	40us	Move the cursor to row 2 column 6
3	1	0	4		
4	1	0	1	40us	Write 'A' (ascii 65 or 0x41) to the current position. Move one column to the right.
5	1	0	4		
6	1	0	2	40us	Write 'B' (ascii 66 of 0x42) to the current position. Move one column to the right.

LCD Routines:

Assume PORTB is used with

PORTD	7	6	5	4	3	2	1	0
LCD	D7	D6	D5	D4	E	R/S	-	-

void Wait_ms(unsigned int X)

Pause Xms then return.

void LCD_Init(void)

Initialize the LCD display, set the cursor to go from left to right, set the cursor to blink, move to top left corner.

void LCD_Move(unsigned char R, C)

Move the cursor to row R column C.

void LCD_Write(unsigned char DATA)

write DATA to the present position on the LCD display. Move the cursor one to the right.

void LCD_Out(unsigned int DATA, unsigned char D, unisgned char N)

Write

- DATA: The data to display
- D: The number of digits to display
- N: The number of digits to the right of the decimal point to display

For example,

```
LCD_Out(12345, 6, 3)
```

outputs

```
012.345
```

void LCD_Inst(0x01);

clear the LCD display

Example 1: Write a routine to send

- 0..19 to the LCD display starting at row 0, column 0.
- 48..67 to the LCD display starting at row 1, column 0

Program

```
// Global Variables

// Subroutine Declarations
#include <pic18.h>

// Subroutines
#include        "lcd_portd.c"

void main(void)
{
   unsigned char i;

   LCD_Init();

   LCD_Move(0,0);
   for (i=0; i<20; i++) LCD_Write(i);
   LCD_Move(1,0);
   for (i=48; i<68; i++) LCD_Write(i);
   while(1);
   }
```

Result:

LCD Display: The first row displays ASCII 0 to 19, the second row displays ASCII 48 to 67

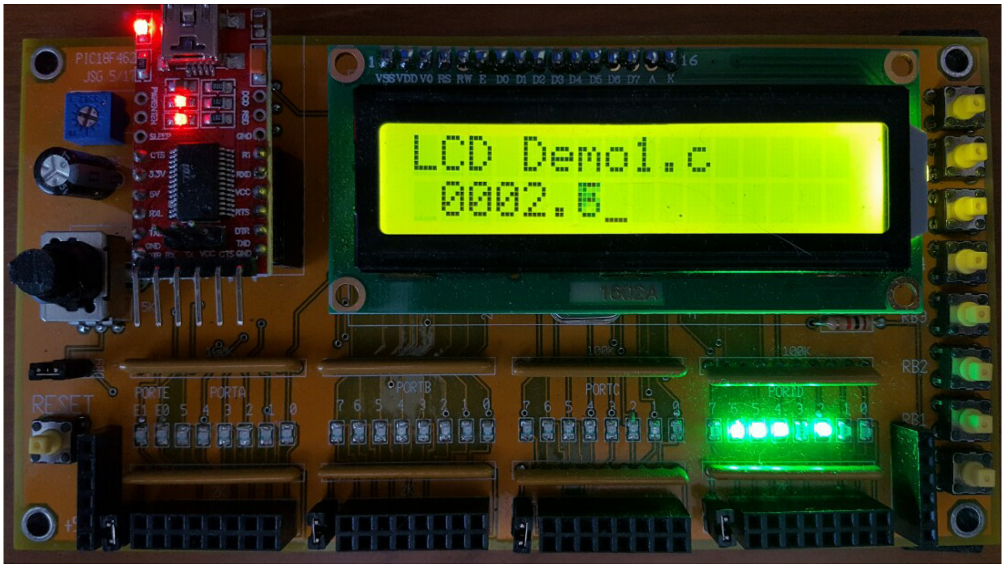
Comment: LCD displays present your data in ASCII format. 0x00 does not display as '0': 0x32 displays as '0'. It's kind of a pain, but that's the standard. The function *ascii* in routine *function.c* converts binary to ascii for display purposes.

Example 2: Write a routine to count and display the count on the LCD display starting at row 1 column 0. Use a subroutine to display the data.

Program

```
// Global Variables

const unsigned char MSG0[20] = "LCD Demo1.c        ";

// Subroutine Declarations
#include <pic18.h>

// Subroutines
#include        "lcd_portd.c"


// Main Routine

void main(void)
{
   unsigned int i, j;
   unsigned int TIME;

   TRISA = 0;
   TRISB = 0;
   TRISC = 0;
   TRISD = 0;
   TRISE = 0;
   ADCON1 = 0x0F;

   LCD_Init();                  // initialize the LCD

   LCD_Move(0,0);  for (i=0; i<20; i++) LCD_Write(MSG0[i]);

   TIME = 0;

   while(1) {
      TIME = TIME + 1;

      LCD_Move(1,0);       // Move to row #1, column #1

      LCD_Out(TIME, 5, 1);  // 1 decimal place, 5 digits

      Wait_ms(100);        // Wait 100ms (approx)

      }
}
```

Result: A count appears on the LCD display. The timing is somewhat random - the LCD updates as fast as it can.

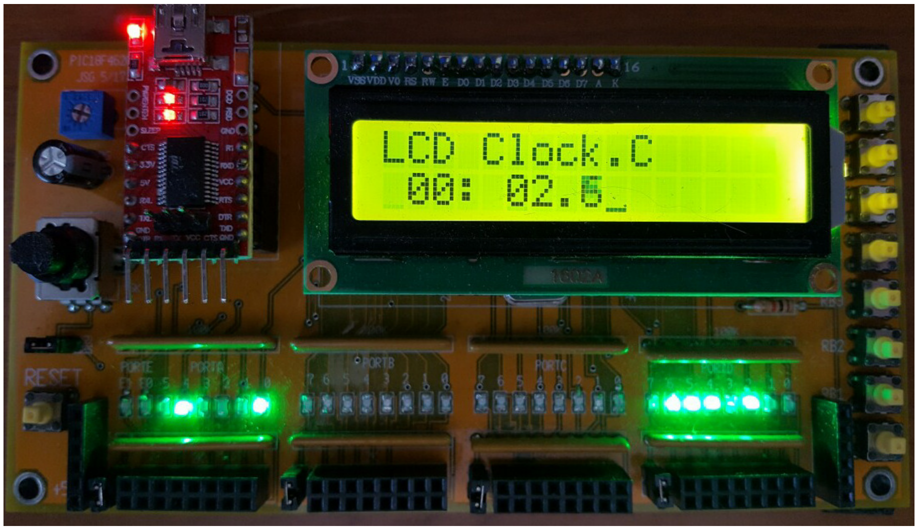


Counting as fast as you can on the LCD display. Note that the count is in decimal - making it easier to read.

Example 3: (LCD Clock.C) Write a routine which displays the time in seconds on the LCD display. Display time to 100ms.

```
// Global Variables

const unsigned char MSG0[20] = "LCD Clock.C         ";

// Subroutine Declarations
#include <pic18.h>

// Subroutines
#include        "lcd_portd.c"

// Main Routine

void main(void)
{
   unsigned int MIN, SEC;
   unsigned int i;

   TRISA = 0;
   TRISB = 0;
   TRISC = 0;
   TRISD = 0;
   TRISE = 0;
   ADCON1 = 0x0F;

   LCD_Init();                   // initialize the LCD

   MIN = 0;
   SEC = 0;

   LCD_Move(0,0);  for (i=0; i<20; i++) LCD_Write(MSG0[i]);
   Wait_ms(100);


   while(1) {
      LCD_Move(1,0);
      LCD_Out(MIN,2,0);
      LCD_Write(':');
      LCD_Out(SEC,3,1);

      SEC = SEC + 1;
      RA0 = 1;
      Wait_ms(100);
      RA0 = 0;
      }
}
```

Result:



What your PIC should look like when running LCD_Clock.C

Note several things:

The first row displays the message "LCD_Demo.C". The section of the code in red does this.

The second row display the time in minutes : seconds

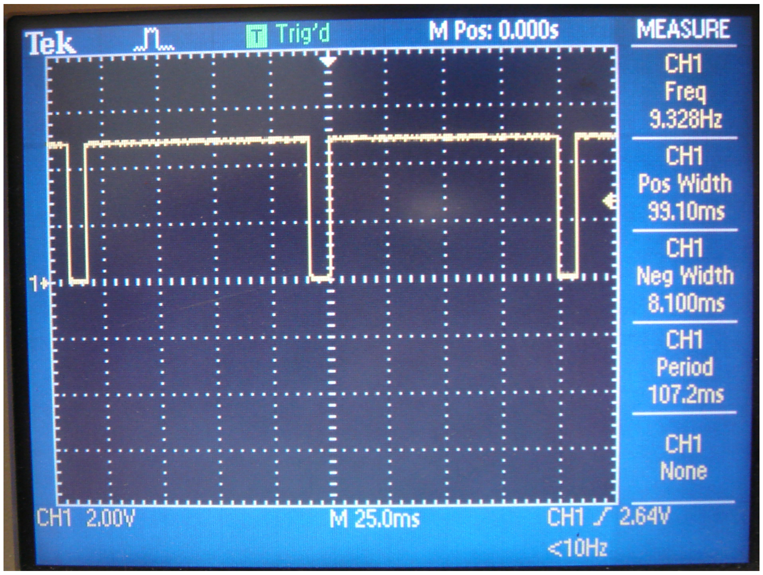
- The LCD_Out() routine will display the next message to the right of the previous message. You don't need to use LCD_Move() in this case.
- The colon is output using the command LCD_Write(':'); This sends the ascii character for a colon to the LCD.
- Seconds are displayed with one desimal point. This results from the command

```
LCD_Out(SEC, 5, 1);
```

Also in this code, RA0 is used so you can measure the loop timing as:

```
RA0 = 1;
Wait(100);
RA0 = 0;
```

If you look at RA0 on the oscilloscope, the signal looks like the following:



Signal on RA0

What this shows is:

RA0 is high for 99.1ms. Ideally, it should be high for 100ms due the instruction

```
Wait_ms(100);
```

The timing in the suboutine Wait() is set by the line

```
for (j=0; j<617; j++);
```

The number 617 is chosen so that it waits 100ms when you tell it to wait 100 units. If you adjust this number, you might be able to get it closer.

RA0 is low for 8.1ms. This is how long everything else in the main loop takes to execute (mostly the LCD routines).

The period of RA0 is 107.2ms. For the timing to be accurate, this should be 100ms.

To fix the timing

- The LCD routines take about 8ms, meaning
- The wait routine should burn the remaining 92ms.

You can do this by changing the number you pass to Wait() as

```
Wait_ms(92);
```