

Lab 7: LCD Screen

UCR EE/CS120B

Pre-Lab

Before using the LCD display, you will have to solder the display to the connector pins. Watch this youtube video before attempting to solder any connections. [How and WHY to Solder Correctly](#). Here is the LCD [datasheet](#).

Introduction



LCD PIN #	Symbol	Function	Connection
1	VSS	Power supply (For LCD contrast)	GND
2	VDD		5V
3	V0		Potentiometer
4	R/S	H: Data, L: Instr	PD6
5	R/W	H: Read, L: Write	GND
6	E	Enable	PD7
7-14	DBn	Data bus	PCn
15	A	Blacklight	5V
16	K		GND
Summarized from datasheet			

Figure 1: Functional LCD

We have provided a header ([io.h](#)) and source ([io.c](#)) file that contain code providing a number of simple functions to be used to write characters on the LCD display. The source file goes in your “source/” directory and the header goes in your “header/” directory.

If the LCD display has been connected according to the pinout given above, and the “io” files have been added to the proper directories, then the sample code below should print “Hello World” to the LCD display.

Sample LCD code: Program this code into the micro-controller to test the LCD.

```

#include <avr/io.h>
#include "io.h"

int main(void)
{
    DDRC = 0xFF; PORTC = 0x00; // LCD data lines
    DDRD = 0xFF; PORTD = 0x00; // LCD control lines

    // Initializes the LCD display
    LCD_init();

    // Starting at position 1 on the LCD screen, writes Hello World
    LCD_DisplayString(1, "Hello World");

    while(1) {continue;}
}

```

The following four functions from “io.c” are the only functions that will be needed to operate the LCD display

LCD_ClearScreen(void): clears the LCD display

LCD_Cursor(unsigned char column): positions the cursor on the LCD display

LCD_WriteData(unsigned char Data): Writes a char at the position the cursor is currently in

LCD_DisplayString(unsigned char column, const unsigned char* string):
Writes a char* to the LCD display starting at position “column”

Knowing the current state of your program when testing is a critical debugging tool. A useful way to debug your program is to write the current state of your synchSM onto LEDs attached to an unused port. This approach gives you a visual representation of where your synchSM is at any given point.



Exercises

- Buttons are connected to PA0 and PA1. Output PORTC and PORTD drive the LCD display, initially displaying 0. Pressing PA0 increments the display (stopping at 9). Pressing PA1 decrements the display (stopping at 0). If both buttons are depressed (even if not initially simultaneously), the display resets to 0. *If a button is held, then the display continues to increment (or decrement) at a rate of once per second.* Use a synchronous state machine captured in C.

Note: The LCD display displays ASCII values so 9 is not the same as ‘9’. For displaying numbers 0 thru 9, a quick conversion technique is to add the character ‘0’ to the number:

```
LCD_WriteData( 9 ); // will display nothing on the LCD  
LCD_WriteData( 9 + '\0' ); // will display 9 on the LCD
```

Video Demonstration: <http://youtu.be/cRNJc6lhKs>

2. **(Challenge)** Extend the earlier light game to maintain a score on the LCD display. The initial score is 5. Each time the user presses the button at the right time, the score increments. Each time the user fails, the score decrements. When reaching 9, show victory somehow.

Video Demonstration: <http://youtu.be/r8yzRMPD3IE>

Submission

Each student must submit their source files (.c) and test files (.gdb) according to instructions in the [lab submission guidelines](#).

```
$ tar -czvf [cslogin]_lab2.tgz turnin/
```

Don't forget to commit and push to Github before you logout!