ENGR 323L: Microprocessor Systems
Engineering Department, Trinity College
Instructor: Professor T. Ning

LCD Introduction

The LCD consists of two registers - Instruction Register (IR) and Data Register (DR). A command is sent (written) to the IR register to control the action. Display data are sent (written) to the DR. The IR stores instruction codes such as display clear and cursor shift, and address information for display data RAM (DD RAM) and character generator RAM (CG RAM). The IR can be written from the microprocessor.

Table 1: Instruction and their corresponding codes

	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	RS	R/W	E
Clear Display	0	0	0	0	0	0	0	1	0	0	1->0
Function Set	0	0	1	1	1	1	0	0	0	0	1->0
Display Off	0	0	0	0	1	0	0	0	0	0	1->0
Entry Mode	0	0	0	0	0	1	1	0	0	0	1->0
Display On	0	0	0	0	1	1	1	1	0	0	1->0
Return Home	0	0	0	0	0	0	1	0	0	0	1->0

Table 2: Register selection

RS	R/W	Enable	Operation
0	0	H,H→L	IR-write as internal operation (Display clear, etc.)
0	1	H	Read-busy flag (DB7) and address counter (DB0-DB6)
1	0	H,H→L	DR-write as internal operation DR to DD RAM or CG RAM)
1	1	H	DR-read as internal operation (DD RAM or CG RAM to DR)

The IR is chosen by setting Register Select (RS) to “0” and DR with “1`”.

Busy Flag

The LCD also has a busy flag. When the busy flag is "1", the LCD is in the internal operation mode, and the next instruction will not be accepted. As the **Register selection** table above shows, the busy flag is output to DB7 when RS = 0 and R/W = 1. The next instruction must be written after ensuring that the busy flag is "0", i.e., a low on this line.

LCD interfacing circuit

The 16x2 LCD Display (CFAH1602A-YYH-JPE) is commonly used for many design applications. The description of the other 14 pins is given in the table below.

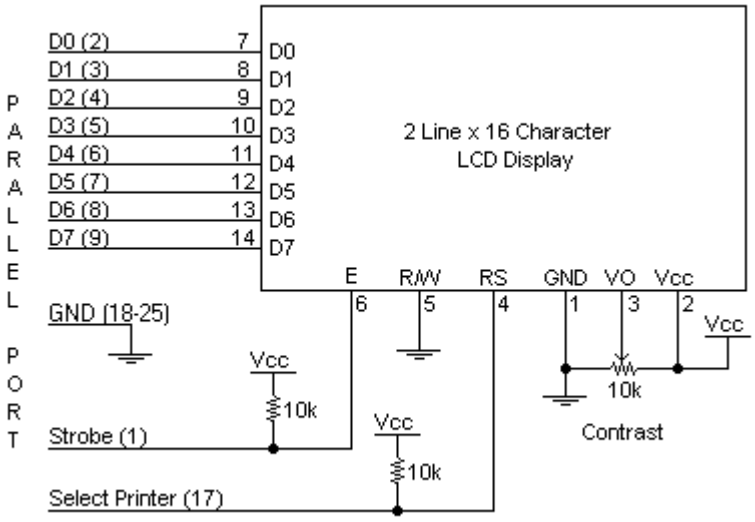
1	V_{SS}	Ground
2	V_{DD}	Supply Voltage (5V)
3	VO	Operating Voltage For LCD (Contrast)
4	RS	*High*: DATA *Low*: Instruction Code
5	R/W	*High*: Read *Low:* Write
6	E	Chip Enable Signal
7	DB0 - DB7	Data Bit 0 - Data Bit 7

The operating voltage (V_O) for the LCD contrast was connected to a potentiometer circuit as the ones described above in order to easily vary the contrast. It should be noted that VO should not be set too high or otherwise the display may be damaged. The seven data bits (DB0 – DB7) were connected to Port_5 on the SiliconLab C8051F120DK MCU to allow data transmission. The RS, R/W, and the E pins were connected to P6.0, P6.1, and P62 respectively. A discussion about the problems encountered with the connections is included in the Implementation section.

Hardware Configuration (connected to an 8-pin parallel port)

The LCD is powered at pins 1 and 3. The VSS at pin-1 is connected to the ground at 0V while VCC at pin-2 is connected to +5V. VCC and VSS should be connected to a 10K potentiometer to adjust the brightness of the LCD display. The output from the

potentiometer was connected to V_{EE} at pin-3. The voltage at this line changes the brightness of the pixels of LCD. Grounds of the cadet board and the target board must be connected.



The data line DB7 is also the busy flag.

Port Configuration for LCD

```
//-------------------------------------------
//Port Configuration for LCD
//-------------------------------------------
SFRPAGE = 0x0F;
P5MDOUT = 0xFF;      //Port 5 as output
P6MDOUT = 0x07;      //Port 6.7 is input, 6.0-6.2 are configured as output -push pull
DB_7 = 1;
```

Software Configuration

In order for the LCD to work it must be initialized in a particular way. A function, e.g., LCD_Init() can be written. In this function, the busy flag is checked constantly to ensure that the commands are being accepted by the LCD. So a function called LCD_busy_flag() was created. This function simply selects the IR and sets the R/W and enable to high to read the busy flag from the IR. The busy flag at line DB7 is read from

Port 6.7 and returned as an integer. It was also made sure that the port was configured as an input for this purpose.

Busy Flag Reading Function

```
int LCD_busy_flag(void){
int busy_temp;
char SFRPAGE_save;
SFRPAGE_save = SFRPAGE;
SFRPAGE = 0x0F;
P5MDOUT = 0x00;                 //Port 5 as digital input
P5 = 0xFF;                      //Port 5 as digital input
RS = 0;                         //Select Instruction Register
R_W    = 1;                     //Read from IR
E =    1;                       //Enable set to high to read the busy flag
busy_temp = DB_7;
        if(busy_temp == 0)
        {
        LED = 1;
        }
        E = 0;                          //Set enable to 0 to stop reading the busy
        flag
        P5MDOUT = 0xFF;                 //Changing back to output mode
        SFRPAGE = SFRPAGE_save;
        return (busy_temp);
        }
```

Inside the LCD_Init () function, a sequence of six steps are executed one after the other. At each step, the enable is set to high and then to low because the internal operations work only when the enable is high and there is a high to low transition. The six steps of initialization and the pattern of data bus lines and select bits are described in the Table 1.

Before each of the instructions being executed, the status of the busy flag must be checked for clear. The instruction is carried out only when the busy flag is low. The code for initialization is shown below.

LCD Initialization Function

```
void LCD_Init(void)
{
int init_step = 0;
char SFRPAGE_save;
```

```
SFRPAGE_save = SFRPAGE;
SFRPAGE = 0x0F;
while (init_step<6){
        E = 0;
        if((init_step == 0) && (LCD_busy_flag() == 0)){
        E = 1;
        init_step ++;
        RS = 0;
        R_W = 0;
        P5 = 0x01;                      //Clear Display
        E = 0;
        }

        if((init_step == 1) && (LCD_busy_flag() == 0)){
        E = 1;
        init_step ++;
        RS = 0;
        R_W = 0;
        P5 = 0x3C;                      //Function Set
        E = 0;
        }

        if((init_step == 2) && (LCD_busy_flag() == 0)){
        E=1;
        init_step ++;
        RS = 0;
        R_W = 0;
        P5 = 0x08;                      //Display off, will be turned on later
        E = 0;
        }

        if((init_step == 3) && (LCD_busy_flag() == 0)){
        E=1;
        init_step ++;
        RS = 0;
        R_W = 0;
        P5 = 0x06;                      //Entry mode
        E = 0;
        }

        if((init_step == 4) && (LCD_busy_flag() == 0)){
        E=1;
        init_step ++;
        RS = 0;
        R_W = 0;
        P5 = 0x0F;                      //Display on
```

```
        E = 0;
        }

        if((init_step == 5) && (LCD_busy_flag() == 0)){
        E=1;
        init_step ++;
        RS = 0;
        R_W = 0;
        P5 = 0x02;                          //Return home
        E = 0;
        }

        E=1;
        }
SFRPAGE = SFRPAGE_save;
}
```

Once the initialization is complete, the LCD needs to be set up for writing. In our application, the LCD is used only for writing to data lines. The LCD needs to be refreshed constantly to update the display. This is achieved by using Timer 1 overflow interrupt. The LCD is set to refresh itself at the rate of 50Hz. It also needs to be instructed when to switch to the next line or start from line 1. Since the LCD consists of two 16 character lines, the LCD must be told to go to line 2 from line 1 once the 16^{th} character has been displayed. Similarly when the 32^{nd} character has been displayed, the LCD must be told to return home. This process is made simpler by defining a variable called LCD_pointer that keeps track of which character the display is working with.

LCD Refresh Function

```
void TF1_ISR(void)interrupt 3{
        SFRPAGE = 0x0F;
if (LCD_busy_flag() == 0)
{
        if ((LCD_pointer == 16)&&(indicator == 1))              //Go to line 2
        {
        E = 1;
        RS = 0;
        R_W = 0;
        P5 = 0xC0;
        E = 0;
        indicator = 0;
```

```
        }

        else if ((LCD_pointer == 32)&&(indicator == 1))                    //Return home
        {
        E = 1;
        RS = 0;
        R_W = 0;
        P5 = 0x02;
        E = 0;
        LCD_pointer = 0;
        indicator = 0;
        Numeric_Character(C_RPM,RPM_change);                //
        }

        else
        {
        E = 1;
        RS = 1;
        R_W = 0;
        P5 = LCD_text[LCD_pointer];
        E = 0;
        LCD_pointer++;
                if ((LCD_pointer == 16)||(LCD_pointer == 32))
                {
                indicator = 1;
                }
}}
```

If the LCD cursor is anywhere else, then the 'else' condition is executed. This condition simply writes to the LCD the character stored a 32 element array. The array is initialized by a blank space (' ') in the main function. The element to be displayed is chosen by the pointer. The program below shows how current RPM and the change in RPM were displayed in the LCD.

Numeric to Character Data Type Function

```
void Numeric_Character(int number1,int number2)
{
int number;
number = abs(number1);
```

```
LCD_text[4] = (unsigned char)(number%10)+(unsigned char)48;
//least significant bit by modulus
LCD_text[3] = (unsigned char)((number/10)%10)+(unsigned char)48;
LCD_text[2] = (unsigned char)((number/100)%10)+(unsigned char)48;
LCD_text[1] = (unsigned char)((number/1000)%10)+(unsigned char)48;
LCD_text[0]='+';
if (number1<0) LCD_text[0]='-';

number=abs(number2);
LCD_text[11] = (unsigned char)(number%10)+(unsigned char)48;
        //least significant bit by modulus+modulus
LCD_text[10] = (unsigned char)((number/10)%10)+(unsigned char)48;
LCD_text[9] = (unsigned char)((number/100)%10)+(unsigned char)48;
LCD_text[8] = (unsigned char)((number/1000)%10)+(unsigned char)48;
LCD_text[7]='+';
if (number2<0) LCD_text[7]='-';
}
//Function calling example
Numeric_Character(C_RPM,RPM_change);
```

Assuming that both the current RPM and the change in RPM are four bit signed decimal numbers, they are both used as arguments for the function shown above that coverts numeric data type to character data type. The function simply uses the modulus operator to separate each of the four bits. The least significant number is the modulus of the given number when divided by 10. This number when added to 48 gives the ASCII code for that number. It is then stored in the 5^{th} element of the array. Similarly, the number in tens place is calculated by taking the modulus of the number divided by 100 and so on. The sign is stored in the array as character and it is determined by comparison with 0.

Conclusion

By following the proper procedure for hardware integration, initialization and sending data, it was possible to successfully implement the LCD for the motor. The testing procedure was relatively simple. First a character was sent to see if it works. Once this was achieved, the problem of changing to the second line was tackled. However, by sheer luck major problems did not happen and saved us a lot of time in terms of testing and debugging. Once the LCD was working, it proved to be of immense help in testing the controller code.