SINGAPORE POLYTECHNIC
SCHOOL OF ELECTRICAL AND ELECTRONICS ENGINEERING

ET0104 Embedded Computer Systems Laboratory

Laboratory 4 Interfacing to a Liquid Crystal Display

1. Introduction

In this lab, you will use the CM3 I/O Board to scan a keypad and interact with a LCD module.

2. Objectives

- To interface to a Liquid Crystal Display module (LCD)
- To perform keypad scanning on a 4x3 device
- How to use combined LCD, LED, Keypad using GPIO.

3. Interfacing to an LCD module

1) Liquid Crystal Displays (LCD) are a popular display device. More commonly LCDs come in the form of modules. These incorporate controller chips with driver functions on board. Some of its advantages are low power, ease of programming and low overhead in terms of refreshing such a device. To make more efficient use of these modules, some points must be noted.

2) Firstly, a 4 bit data transfer protocol is preferred, as this uses up less pins.

3) Secondly, the ENABLE pin cannot be driven by the hardware generated control signals of the SBC such as the R/-W or E pins. A software generated signal has to be used. In our case this means that to toggle a bit, we set the bit to low, write to a latch, set the bit to high, write again, and write one more time with the bit low.

4) Third, certain operations take up more time than others. It is possible to read the status of the LCD, but that may involve extra hardware and decoding. A standard approach is to assume a common time whereby all operations are complete. Note that since different LCD modules may exhibit different execution times, it may be worthwhile to adjust the time delay accordingly.

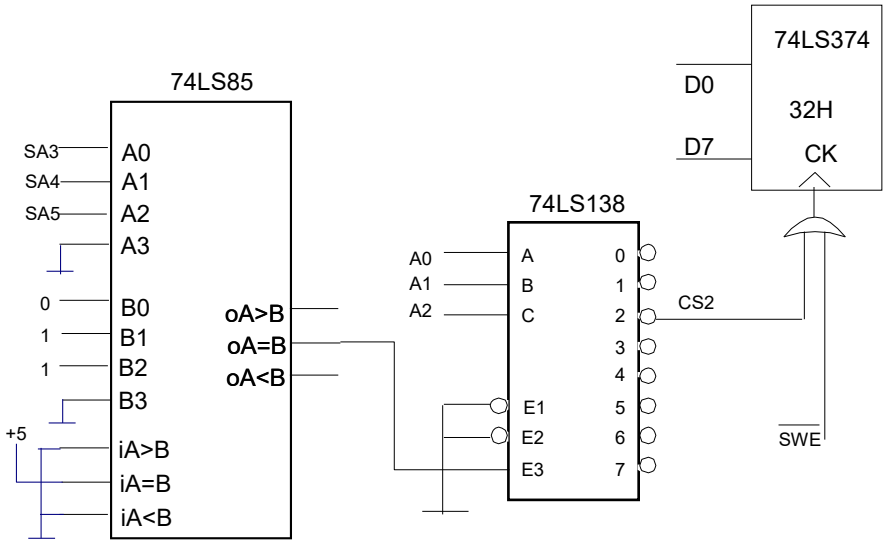
5) We will use the same keypad from the previous laboratory, but this time, we will output to the LCD module. The following gives the layout of the pins.

Pin layout of LCD latch:	7	6	5	4	3	2	1	0
LCD Control/Data pins	D7	D6	D5	D4	X	E	R/W	RS

6) I/O Addresses

The SMI bus of the CM3 allows a maximum of 64 I/O devices to be accessed. It is useful to have the flexibility of setting these addresses to fall within a certain range. The advantage of the 74LS85 is that it allows the fixing of this range by just setting a few DIP switches.

In this lab we want the LED which is set up as Device 2, to be at address 2AH. The hardware circuit is shown below.



Fill in the corresponding values for the LCD and KeyPad, which are Devices 3 and 4 respectively. You can find the required information from earlier labs about the I/O map for the board. Set the DIP switches accordingly. The following truth table is given to help you:

Device/Addr	A5	A4	A3	A2	A1	A0
LED						
LCD						
KeyPad						

4. Instructions

1) The program lab4.c is given in the appendix. Fill in the blanks, using the LCD instruction sheet if necessary. Load it into the development system and test it to see if a message appears and what is typed into the keypad is reflected in the display.

2) Using `sprintf` for more versatile displays.

i) We will modify `lab4.c` now to try out some useful printing functions. The array `LCDStr` accepts a pointer to a C-formatted text string. That is, the end of the string is indicated by a 0x00. We can use the C function `sprintf` to generate message strings for us.The advantages are many; for example, data conversions from all kinds of number formats to text strings are done for us. However, we have to use the C formatting commands as found in the `printf` command. We also have great flexibility in composing strings using functions like `strcat` (concatenate strings) and so on.

ii) **Modify** lab4.c as follows:

1. Declare new variables as follows:

```
char LCDStr[17];     /* 16 char LCD + 1 for end of string */
char test;           /* to test */
```

2. In the main program, *replace* the line `LCDprint("12345678");` by:

```
test=104;
sprintf(LCDStr,"Subject:ET%d-OK”,test); /* don’t exceed num chars!*/
LCDprint(LCDStr);
```

What do you expect to see?

3. Another way of doing the above, illustrating the use of `strcat`.

```
test=104;sprintf(LCDStr,"Subject:ET%d",test); /*generate 1st string */
strcat(LCDStr,"-OK");                         /* append using strcat */
LCDprint(LCDStr);
```

4. Modify point 3. above by displaying a floating point conversion with the use of a message string kept elsewhere – that is *not* part of the `sprintf` statement. For example, display the price of an item ($104) with GST computed. (Hint: do an online search for “format specifiers in C”).

Appendix

Program template for lab 4:

```
#include <unistd.h>
#include <stdio.h>
#include <string.h>
#include "library.h"

#define LEDPort ____
#define KbdPort ____
#define LCDPort ____

unsigned char ProcKey();
unsigned char ScanKey();

const unsigned char ScanTable [12] =
{
/* 0     1     2     3 */
0x__, 0x__, 0x__, 0x__,
/* 4     5     6     7 */
0x__, 0x__, 0x__, 0x__,
/* 8     9     *     # */
0x__, 0x__, 0x__, 0x__
};

const unsigned char Bin2LED[] =

      /*  0     1     2    3 */
      {0x__, 0x__, 0x__, 0x__,

      /*  4     5     6    7*/
      0x__, 0x__, 0x__, 0x__,

      /*  8     9     A    B*/
      0x__, 0x__, 0x__, 0x__};

unsigned char ScanCode;

#define Col7Lo ____            // column 7 scan
#define Col6Lo ____            // column 6 scan
#define Col5Lo ____            // column 5 scan
#define Col4Lo ____            // column 4 scan

static void initlcd();
static void lcd_writecmd(char cmd);
static void LCDprint(char *sptr);
static void lcddata(unsigned char cmd);
```

```
int main(int argc, char *argv[])
{
CM3DeviceInit();

initlcd();
lcd_writecmd(0x80);
LCDprint("LCD Lab");
lcd_writecmd(0xC0);
LCDprint("12345678");

while(1)
{
      unsigned char i,ii;
      i = ScanKey();
      if (i != 0xFF)
      {
            if (i > 0x39) {
                  ii = i - 0x37;
            } else {
                  ii = i - 0x30;
            }
            lcddata(i);
            CM3_outport(LEDPort, Bin2LED[ii]);
            usleep(300000);
      }
}
CM3DeviceDeInit();
}

static void initlcd(void)
{
       usleep(20000);
       lcd_writecmd(0x30);
       usleep(20000);
       lcd_writecmd(0x30);
       usleep(20000);
       lcd_writecmd(0x30);

       lcd_writecmd(0x__);  // 4 bit mode
       lcd_writecmd(0x__);  // 2 line  5*7 dots
       lcd_writecmd(0x__);  //clear screen
       lcd_writecmd(0x__);  //dis on cur off
       lcd_writecmd(0x__);  //inc cur
lcd_writecmd(0x80);
}
static void lcd_writecmd(char cmd)
{
```

```
char data;

data = (cmd & 0xf0);
CM3_outport(LCDPort, data | 0x04);
usleep(10);
CM3_outport(LCDPort, data);

usleep(200);

data = (cmd & 0x0f) << 4;
CM3_outport(LCDPort, data | 0x04);
usleep(10);
CM3_outport(LCDPort, data);

usleep(2000);
}
static void LCDprint(char *sptr)
{
while (*sptr != 0)
{
      int i=1;
      lcddata(*sptr);
      ++sptr;
}
}
static void lcddata(unsigned char cmd)
{
char data;

data = (cmd & 0xf0);
CM3_outport(LCDPort, data | 0x05);
usleep(10);
CM3_outport(LCDPort, data);
usleep(200);

data = (cmd & 0x0f) << 4;
CM3_outport(LCDPort, data | 0x05);
usleep(10);
CM3_outport(LCDPort, data);

usleep(2000);
}
/*----------- Keypad Functions --------------*/

unsigned char ScanKey()
{
CM3_outport(KbdPort, Col7Lo);
```

```
    ScanCode = CM3_inport(KbdPort);
    ScanCode |= 0x0F;
    ScanCode &= Col7Lo;
    if (ScanCode != Col7Lo)
    {
        return ProcKey();
    }
    CM3_outport(KbdPort, Col6Lo);
    ScanCode = CM3_inport(KbdPort);
    ScanCode |= 0x0F;
    ScanCode &= Col6Lo;
    if (ScanCode != Col6Lo)
    {
        return ProcKey();
    }
    CM3_outport(KbdPort, Col5Lo);
    ScanCode = CM3_inport(KbdPort);
    ScanCode |= 0x0F;
    ScanCode &= Col5Lo;
    if (ScanCode != Col5Lo)
    {
        return ProcKey();
    }
    CM3_outport(KbdPort, Col4Lo);
    ScanCode = CM3_inport(KbdPort);
    ScanCode |= 0x0F;
    ScanCode &= Col4Lo;
    if (ScanCode != Col4Lo)
        return ProcKey();
    }
    return 0xFF;
    }
unsigned char ProcKey()
{
    unsigned char j;
    for (j = 0 ; j <= 12 ; j++)
    if (ScanCode == ScanTable [j])
    {
       if(j > 9) {
             j = j + 0x37;
       } else {
             j = j + 0x30;
       }
       return j;
    }
    if (j == 12)
    {
```

```
            return 0xFF;
        }
        return (0);
    }
```

Instruction	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Description	Execution Time
Clear Display	0	0	0	0	0	0	0	0	0	1	Clears Display and returns cursor to the Home Position (Address 00)	80uS = 1.64mS
Return Home	0	0	0	0	0	0	0	0	1	*	Returns cursor to Home Position. Returns shifted display to original position. Does not clear display	40uS = 1.6mS
Entry Mode Set	0	0	0	0	0	0	0	1	I/D	S	Sets DD RAM counter to increment or decrement (I/D) Specifies cursor or display shift during to Data Read or Write (S)	40uS
Display ON/OFF Control	0	0	0	0	0	0	1	D	C	B	Sets Display ON/OFF (D), cursor ON/OFF (C), and blink character at cursor position	40uS
Cursor or Display Shift	0	0	0	0	0	1	S/C	R/L	*	*	Moves cursor or shifts the display w/o changing DD RAM contents	40uS
Function Set	0	0	0	0	1	DL	N	F	*	*	Sets data bus length (DL), # of display lines (N), and character font (F)	40uS
Set CG RAM Address	0	0	0	1	A_{CG}						Sets CG RAM address. CG RAM data is sent and received after this instruction	40uS
Set DD RAM Address	0	0	1	A_{DD}							Sets DD RAM address. DD RAM data is sent and received after this instruction	40uS
Read Busy Flag & Address	0	1	BF	AC							Reads Busy Flag (BF) and address counter contents	1uS
SIZE=2>Write Data from DD or CG RAM	1	0	Write Data								Writes data to DD or CG RAM and increments or decrements address counter (AC)	40uS
Read Data from DD or CGRAM	1	1	Read Data								Reads data from DD or CG RAM and increments or decrements address counter (AC)	40uS
I/D=1: Increment S=1: Display Shift on data entry S/C=1: Display Shift (RAM unchanged) R/L=1: Shift to the Right DL=1: 8 bits N=1: 2 Lines F=1: 5x10 Dot Font D=1: Display ON C=1: Cursor ON B=1: Blink ON BF=1: Cannot accept instruction				I/D=0: Decrements S=0: Cursor Shift on data entry S/C=0: Cursor Shift (RAM unchanged) R/L=0: Shift to the Left DL=0: 4 bits N=0: 1 Line F=0: 5x7 Dot Font D=0: Display OFF C=0: Cursor OFF B=0: Blink OFF BF=0: Can accept instruction							Definitions: DD RAM: Display data RAM CG RAM: Character generator RAM A_{CG}: CG RAM Address A_{DD}: DD RAM Address(Cursor Address) AC: Address Counter used for both DD and CG RAM Address	Execution Time changes when Frequency changes per the following example: If F_{CP} or f_{osc} is 27 KHz 40uS x 250/270 = 37uS