SINGAPORE POLYTECHNIC
SCHOOL OF ELECTRICAL AND ELECTRONICS ENGINEERING

ET0104 Embedded Computer Systems Laboratory

Laboratory 4 Interfacing to a Liquid Crystal Display

Objectives

- To interface to a Liquid Crystal Display module (LCD)
- To perform keypad scanning on a 4x3 device
- How to set different base addresses for the I/O board.

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

Interfacing to an LCD module

Liquid Crystal Displays (LCD) are a popular display device. They consume very little power. However, interfacing to a bare display requires an amount of refresh logic to maintain the display, not to mention the drive signals to the LCD electrodes. That is why the more popular form of 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 have to be noted.

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

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.

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

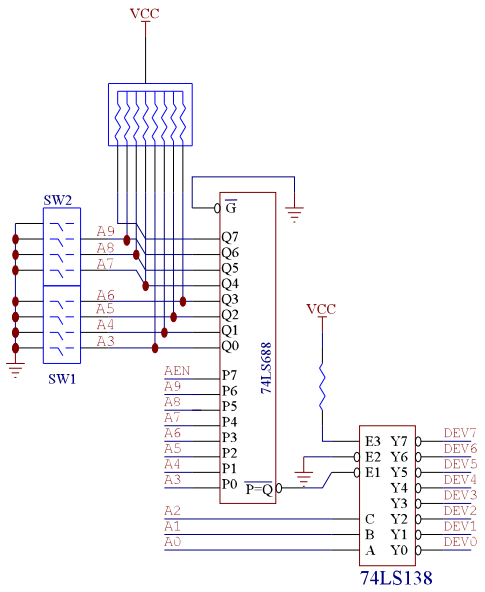
We will use the same keypad from the previous laboratory, but this time, we will output to the LCD module. The following gives the layout.

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

I/O Addresses

In this lab, we will also examine how to set different base addresses for the I/O board. We have seen that several of the I/O addresses for the PC/104 bus are used up by the system. The advantage of the 74LS688 is that it allows changing of the base address 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 24AH. The hardware circuit is shown below.



Fill in the corresponding values for the LCD and KeyPad, which are Devices 3 and 4 . 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	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
LED										
LCD										
KeyPad										

Some points to note:

1. Although we are using a 4 bit mode, the routine LCDOut will take in a byte of data and output it one nybble at a time.

2 The control pins are set in internal memory location LCDCmd. They are then moved to the accumulator before outputting to the port.

3. The telephone keypad is quite 'sticky' and requires quite a large time to let it settle down. Calculate the approximate time values.

The program lab6.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.

Using sprintf

We will modify lab4.c now to try out some useful printing functions. The routine LCDPrint 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.

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 17 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);
```

Appendix

```
/* Keypad and LCD interface
* The LCD interface uses lcdcmd/lcddata to output 8 bit command/data
* to the module
* The physical interface however is four bit and is handled
* internally by these routines. This is of concern only when
* initializing the LCD module to 4 bits */

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

#define Ubyte unsigned char
#define Ushort unsigned short int
#define Uword unsinged int
#define byte char
#define DELAY 20
#define LEDPort ____
#define LCDPort ____
#define KbdPort ____
#define Col7Lo    ____          /* column 7 scan */
#define Col6Lo    ____          /* column 6 scan */
#define Col5Lo    ____          /* column 5 scan */
#define Col4Lo    ____          /* column 4 scan */
#define TRUE      1             /* neater to use! */


const unsigned char ScanTable[12] =
/*    0    1     2     3   */
{____, ____, ____, ____,
/*    4    5     6     7  */
  ____, ____, ____, ____,
/*  8    9     A     B  */
   ____, ____, ____, ____
};
/* store this table in code space (ROM) */
const unsigned char Bin2LED[] =
/*0-3 */
{____, ____, ____, ____,
/*4-7 */
 ____, ____, ____, ____,
/*8-B */
 ____, ____, ____, ____,
/*C-F */
 ____, ____, ____, ____
};

unsigned char   ScanCode;       /* hold scan code returned     */


void  lcddata(Ubyte dval);
void  lcdcmd(Ubyte d);
void  LCDOut(Ubyte data);
void  LCDprint(byte * sptr);
void  initlcd();
void  DispNum(Ubyte n);

void  main()
{
    initlcd();
    lcdcmd(0x80);
    LCDprint("LCD Lab");
    lcdcmd(0xC0);
    LCDprint("12345678");

while (TRUE)
      {
      unsigned char   i;
      i = ScanKey();
      if (i != 0xFF)          /* if key is pressed * */
            _outp(LEDPort ,Bin2LED[i]);   /* output to LED */
           LCDprint(ProcKey());
      }
}

void  lcddata(Ubyte dval)
{
    Ubyte  c;
      c = dval & 0xf0;
      c = c | 0x01;
```

```
        LCDOut(c);
        c = c | 0x04;
        LCDOut(c);
        c = c & 0xfb;
        LCDOut(c);

        c = dval;
        c <<= 4;
        c = c | 0x01;
        LCDOut(c);
        c = c | 0x04;
        LCDOut(c);
        c = c & 0xfb;
        LCDOut(c);
        Sleep(1);
}
void      lcdcmd(Ubyte d)
{
    Ubyte           c;
        c = d & 0xf0;
        LCDOut(c);
        c = c | 0x04;
        LCDOut(c);
        c = c & 0xfb;
        LCDOut(c);
        d <<= 4;
        c = d & 0xf0;
        LCDOut(c);
        c = c | 0x04;
        LCDOut(c);
        c = c & 0xfb;
        LCDOut(c);
        Sleep(1);
}
void  LCDOut(byte data)
{
    _outp(LCDPort, data);
}

/* output a string at address sptr */
void  LCDprint(byte * sptr)
{
    while (*sptr!=0)
    {
        int i=1;
        lcddata(*sptr);
        ++sptr;
    }
}

void  initlcd()
{
    lcdcmd(0x33);        /* Initialise */
    lcdcmd(0x33);        /* Twice! */
    lcdcmd(0x32);        /* 4 bit mode */
    lcdcmd(____);        /* 4 bit, 2 lines, 5x7 font */
    lcdcmd(____);        /* display off */
    lcdcmd(____);        /* display on */
    lcdcmd(____);        /* entry mode set, increment, cursor move */
    lcdcmd(____);        /* cursor at 00H */
    lcdcmd(____);        /* blink */
}
void  DispNum(Ubyte n)
{
    if (n > 9)
        n = n + 55;
    else
        n = n + 48;
    lcddata(n);
}
/* Check for key press: if none, return 0xFF */
unsigned char   ScanKey()
{
    _outp(KbdPort, Col7Lo);           /* bit 7 low */
    ScanCode = _inp(KbdPort);         /* Read */
    ScanCode |= 0xF0;           /* high nybble to 1 */
    ScanCode &= Col7Lo;         /* AND back scan value */
    if (ScanCode != Col7Lo)     /* in <> out get key */
        return ProcKey();
```

```
      _outp(KbdPort, Col6Lo);           /* bit 6 low  */
      ScanCode = _inp(KbdPort);         /* Read */
      ScanCode |= 0xF0;           /* high nybble to 1 */
      ScanCode &= Col6Lo;         /* AND back scan value */
      if (ScanCode != Col6Lo)     /* in <> out get key and display */
          return ProcKey();

      _outp(KbdPort, Col5Lo);           /* bit 5 low  */
      ScanCode = _inp(KbdPort);         /* Read */
      ScanCode |= 0xF0;           /* high nybble to 1 */
      ScanCode &= Col5Lo;         /* AND back scan value */
      if (ScanCode != Col5Lo)     /* in <> out get key and display */
          return ProcKey();

      _outp(KbdPort,Col4Lo);           /* bit 4 low  */
      ScanCode = _inp(KbdPort);         /* Read */
      ScanCode |= 0xF0;           /* high nybble to 1 */
      ScanCode &= Col4Lo;         /* AND back scan value */
      if (ScanCode != Col4Lo)     /* in <> out get key and display */
          return ProcKey();

      return 0xFF;
}                                 /* while */
 /* main */

/* Procedure here */
unsigned char   ProcKey()
{
      unsigned char   i;          /* index of scan code returned */
      for (i = 0; i <= 12; i++)
      if (ScanCode == ScanTable[i]) /* search in table */
      {
            if(i > 9)
            i = i + 0x37;
      else
            i = i + 0x30;
      return i;           /* exit loop if found */
      }
      if (i == 12)
            return 0xFF;       /* if not found, return 0xFF */
      return (0);
}
```

LCD INSTRUCTION TABLE

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

* Don't Care **(when F_{cp} or f_{osc} is 250KHz)