SINGAPORE POLYTECHNIC
SCHOOL OF ELECTRICAL AND ELECTRONICS ENGINEERING

ET0104 Embedded Computer Systems Laboratory

Laboratory 3 - Keypad Interfacing

Objectives

- To perform keypad scanning on a 4x3 key device

In this lab, you will use Device 4 of the I/O board to scan a keypad, and output the key pressed to an LED. You will also have to map the scanned key result to the actual value on the key pad.

Interfacing to a keypad

Telephone keypads are an economical, commonly available input device. They are made up of a wires arranged as a matrix. A pair of wires will make contact when a key is pressed. For convenience, these wires end up as connector pins on the keypad. The sample which we are using has 8 pins and 12 keys.

Very often, there is no documentation for the device at all. In order to find out how it works, we have to physically press the keys and see which wires make contact, coming up with a so called “key connection table”.

To interface a keypad in practice, we need to do the following:

1) Determine which pins make contact when a key is pressed
2) Construct a connector to join the keypad to the processor board
3) Specify which bits in the processor match the connector pins.

This lab will go through these design steps.

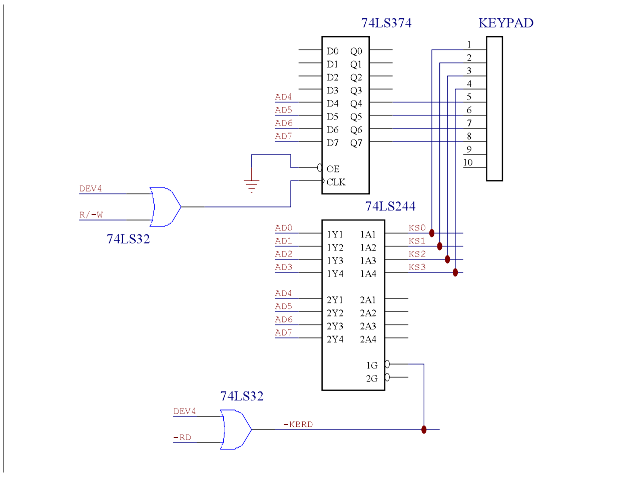
To save some effort, the key connection table is given below. Looking at the keypad face up, and taking the leftmost pin as 1, the following pins short when the corresponding keys are pressed. For example, when you press the '3' key, pins 3 and 4 will make contact.

Keypad	pin 4	pin 5	pin 6	pin 7
pin 3	3	6	9	#
pin 2	2	5	8	0
pin 1	1	4	7	*

How do we detect a keypress? We may easily do so by using an 8 bit bidirectional port, or at least one with 3 output and 4 input pins. A common way will be to pull up pins 1 to 3 of the keypad with resistors, and output a '0' from pins 4 to 7 one at a time. Then pins 1 to 3 are checked for the presence of a '0'.

Let's say we have output a '0' to pin 4. If the '3' key was pressed, we would see a '0' at pin 3 when we read in pins 1 to 3.

In the I/O Board, we will be using Device 4 to do keypad detection. This device is configured so that the upper nybble is configured as an output and the lower nybble is configured as an input. This is shown in the following figure.



Schematic-semi-bidirectional port for keypad I/O

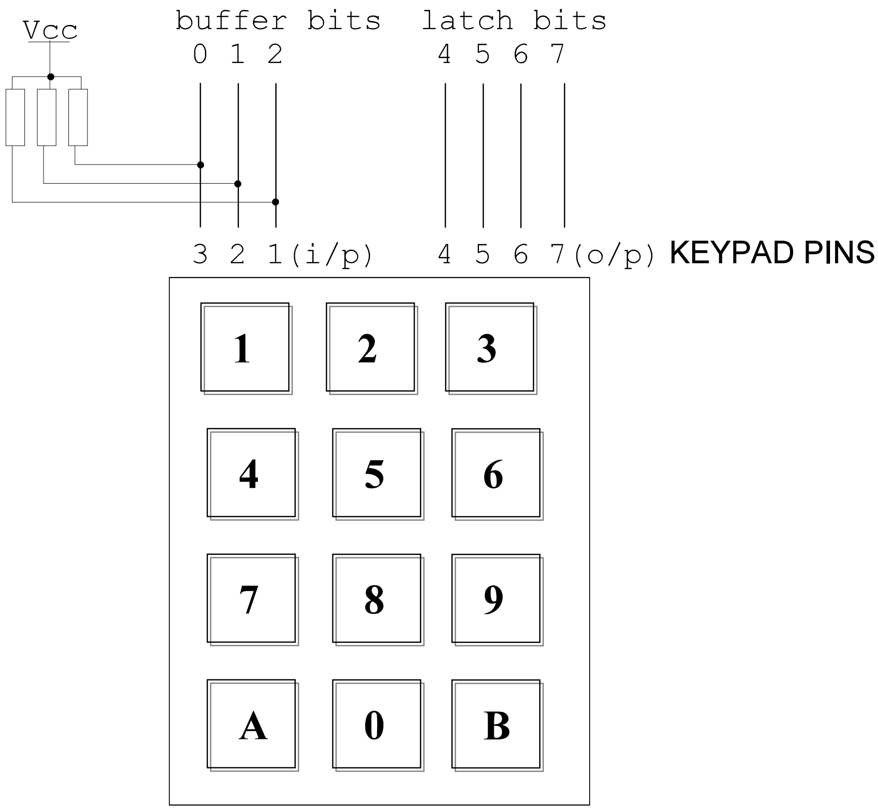
Connector pins 1 to 3 are *input* from pins 3 to 1 on the keypad, and connector pins 5 to 8 are *output* to pins 4 to 7 on the keypad. Remember, if NO keys were pressed, pins 1 to 3 on the keypad will return all 1's.

You may experience some keyboard bounce problems, which may be overcome by detecting a key press, and reading the same key again after a delay to confirm that it is a valid keypress.

The following diagram shows the keypad pins and their numbering. This key connection table describes which of these pins make contact when a key is pressed.

The connector joins these pins to the processor board. It has its own numbering. But it will map to the processor port and this is the portion that is of interest.

Thus, from the key connection table which is given, we should be able to determine the corresponding port bits of interest.



Schematic of Keypad connection

INSTRUCTIONS
As in the previous labs, create a *lab3* project. The file LAB3.C with the blanks as shown below, is given to you. Complete the blanks, with reference to the values generated by the various key presses. If you wish, you may create a function called ProcKey.c. Use the schematic of the key pad as shown above. Compile the program and load it into the SBC.

Program listing notes
The main program runs as a continuous loop, calling the function Scankey() which does the scanning. It returns 0xFF if no key is pressed. Scankey can thus be easily used in other routines. If a key is pressed, it is converted to the equivalent 7 segment code by means of an array.

The program assumes the existence of a port that has the higher nybble as an output and the lower nybble as an input. In many microcontrollers the ports, are bidirectional. Here, we use a latch and buffer respectively, where some bits are not used.
A “walking zero” bit pattern as described in the lecture, will be output to the latch. This bit pattern will be the higher nybble of a byte. If a key is pressed, the buffer will have a ‘0' voltage level read into one of its four bit positions. Combining the input pattern with the output pattern, we will have a unique combination of bits that will identify which key is pressed. This is called the *scan code*. The routine ProcKey will search the table KeyTab sequentially to find the scan code. That is, KeyTab is arranged so that the first entry will correspond to the ‘0' key, the second to the ‘1' key and so on. So by counting at which position in KeyTab the *detected scancode* is, we know what key is pressed.

Listing of lab3.C

```
/***************************/
/*       Keypad lab        */
/***************************/

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

#define LEDPort  *((char *) 0x132          /* LED Port  */
#define KbdPort  *((volatile char *)0x134) /* Key pad port */

#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 */
const unsigned char Bin2LED[] =

{ ----    ,----    ,----    ,----,

  ----    ,----    ,----    ,----,

  ----    ,----    ,----    ,----,

  ----    ,----    ,----    , ----
};

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

/***************************************************************
**********/
/***************************       MAIN PROGRAM
*********************/
/***************************************************************
**********/
void main (void)
{                           /* main entry for program */
     unsigned char i;
```

```
     while (TRUE)
     {
          i = ScanKey();
          if (i != 0xFF)                 /* if key is pressed **/
               LEDPort = Bin2LED[i];    /* output to LED */
     }
}                                        /* loop forever */

/* Check for key press: if none, return 0xFF */
unsigned char ScanKey()
{
     KbdPort = Col7Lo;                  /* bit 7 low  */
      ScanCode = KbdPort;                /* Read */
      ScanCode |= 0xF0;                  /* high nybble to 1 */
      ScanCode &= Col7Lo;                /* AND back scan value */
      if (ScanCode != Col7Lo)            /* in <> out, get key */
            return ProcKey();

     KbdPort = Col6Lo;                  /* bit 6 low  */
     ScanCode = 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();

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

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

     return 0xFF;

    }   /* while */
}   /* main */

/* function 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 */
          return i;                /* exit loop if found */
     if (i == 12)
```

```
            return  0xFF;              /* if not found, return 0xFF
*/

      return(0);
}
```