Laboratory 3 Keypad Interfacing

1. Introduction

In this lab, you will use the buffers and latches 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 keypad.

2. Objectives

- To perform keypad scanning on a 4x3 key device
- Learn about interfacing to a keypad

3. Interfacing to a keypad

1) Telephone keypads are an economical, commonly available input device. They are made up of 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.

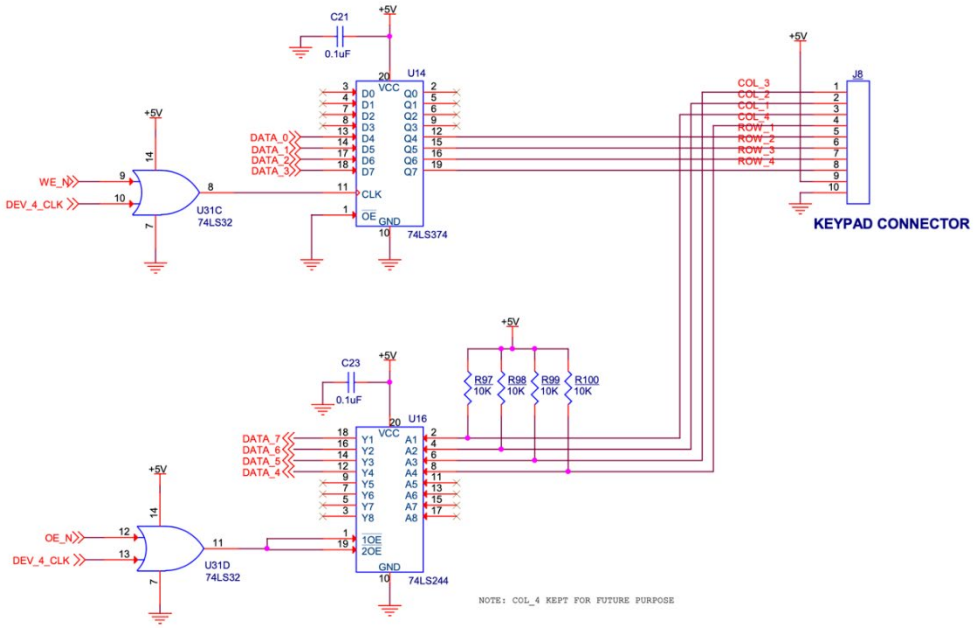
2) To interface a keypad in practice, we need to do the following:

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

   This lab will go through these design steps.

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 input and the lower nybble is configured as an output. This is shown in the following figure.

4) 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	*

Figure 1: Key connection table

5) 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'.

6) 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.

7) 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.

8) 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.

9) 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.

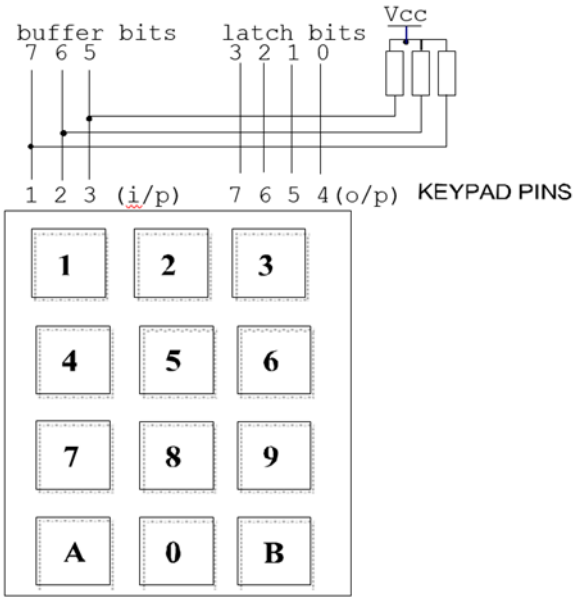


Figure 2: Schematic of Keypad connection

4. Program listing notes

1) The main program runs as a continuous loop, calling the function `ScanKey()` which performs one round of scanning the four columns. It returns `0xFF` if no key is pressed and thus `ScanKey()` can 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, so it can be displayed.

2) 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.

3) 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*.

4) The routine `ProcKey()` will search the `ScanTable` sequentially to find the scan code. That is, `ScanTable` 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 `ScanTable` the *detected scancode* is, we know what key is pressed.

5. Instructions

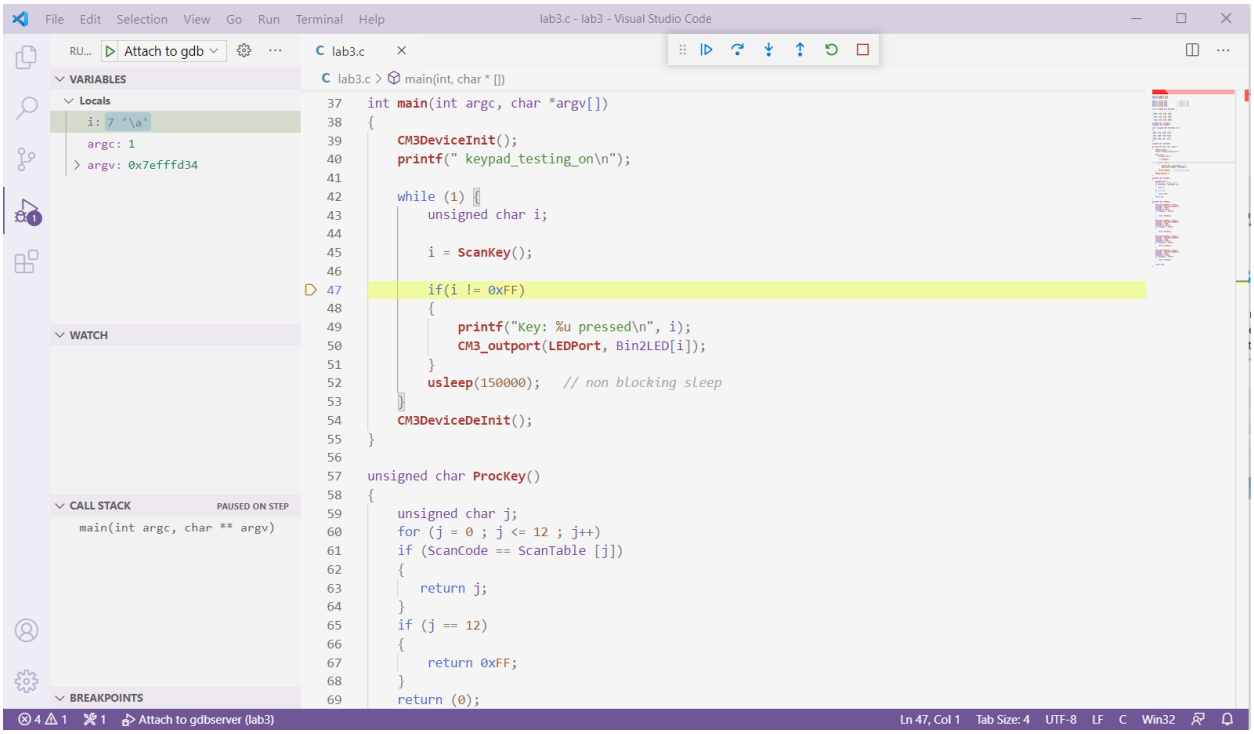
1) As in the previous labs, the `ECSLAB\lab3` directory has been created. The file `lab3.c` with the blanks as shown below, is given to you.

2) Refer to `lab3.c` in the appendix and fill in the blanks, with reference to the values generated by the various key presses. Use the schematic of the key pad as shown above. Compile the program and load it into the CM3.

Appendix – Hints to debug a keypress and program template for lab3

Using VS Code to debug a keypress

In VS Code, the return value of the `ScanKey` function corresponds to key pressed when executing the function. If the key '7' is pressed, you will notice the value `'\a'` appear in the variable `i` in the debug pane. This is actually the ASCII representation of the value 0x07. You may inspect other keypad return values during the lab.



Program template for lab 3

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

#define LEDPort 0x32
#define KbdPort 0x34

#define Col7Lo 0x__            // column 7 scan
#define Col6Lo 0x__            // column 6 scan
#define Col5Lo 0x__            // column 5 scan
#define Col4Lo 0x__            // column 4 scan

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 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__
};

unsigned char ScanCode;

int main(int argc, char *argv[])
{
      CM3DeviceInit();
      printf(" keypad_testing_on\n");
      while (1) {
            unsigned char i;
            i = ScanKey();

            if(i != 0xFF){
                  printf("Key: %u pressed\n", i);
                  CM3_outport(LEDPort, Bin2LED[i]);
            }
            usleep(150000);    // non blocking sleep
      }
      CM3DeviceDeInit();
}
unsigned char ProcKey()
{
      unsigned char j;
      for (j = 0 ; j <= 12 ; j++)
      if (ScanCode == ScanTable [j])
      {
         return j;
      }
      if (j == 12)
```

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