4x4 Matrix Keypad on the ECET Development Board

Objective:
Students will write a program that continuously reads keypad keys as they are pressed, and display each number in binary form on the Port C LEDs. Students are introduced to the 4x4 Matrix keypad hardware. They will learn how the keypad is wired to read keypresses, how it is interfaced and memory mapped on our MC9S12 microcontroller and then apply their understanding of the keypad operation and interface to write a program that continuously reads and displays keypad keypresses.

Keypad Hardware Configuration:
The Hex keypad on the development board is a 4 x 4 array of switches in the lower right corner of the training board. Each of the four rows of switches is connected to an output bit of Port A. Each of the four columns of switches is connected to an input bit of Port A. To use the keypad, initialize Port A with the lower four bits as outputs, and the upper four bits as inputs.

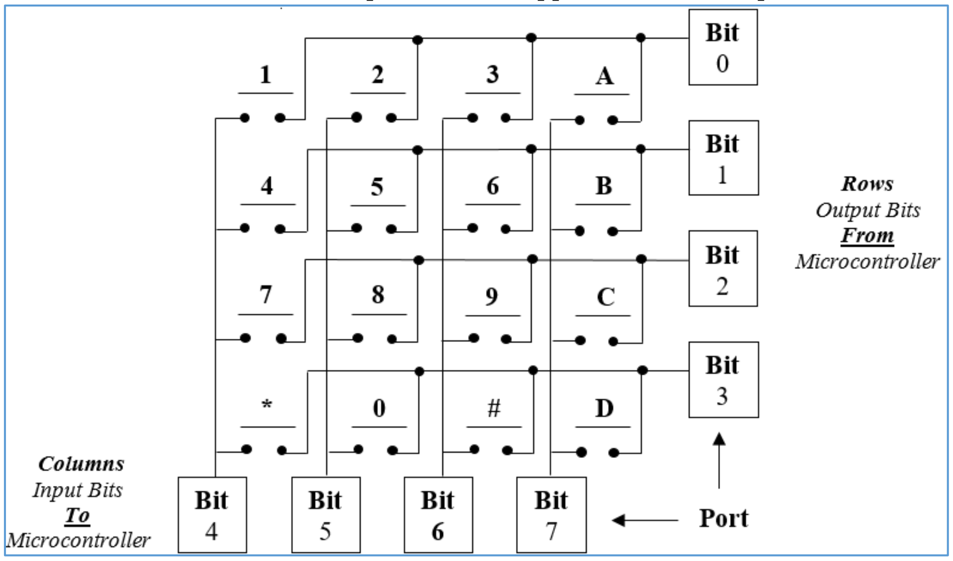

Figure 1
Keypad Switches

+5 VDC is connected to the switches through pull-up resistors as shown in the example switch detail of Figure 2. The default state when no switches are pressed is +5V (or logic 1) on the associated input bit of a column.

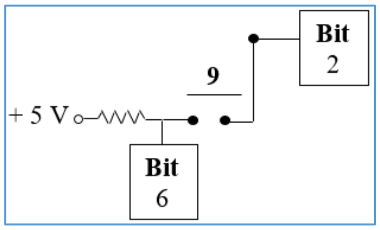

Figure 2
Switch Detail

Theory of Operation:
To detect a key press on a row, put a ground or logic 0 on the associated output bit for that row. If any one of the keys on that row is pressed, the ground on the output bit is connected to the input bit of the key column.

For example, if we put a logic 0 on Bit 0, and then find a logic 0 on Bit 6, the “3” key is pressed. If we instead find a logic 0 on Bit 4, the “1” key is pressed. If all input bits (bit 4 – bit 7) remain at logic 1, no key was pressed on that row.

The key rows are polled one at a time by placing a ground on one row (and only one row) and reading the input bits to find if one of the keys in that row is pressed. If a key press is not found, the next row is checked. If a microprocessor is waiting for a key press, the four rows are checked in a continuous loop until a key press is detected.

The table below shows the Port A value corresponding to each active key. It was derived by tracing the Port A values created by pressing each key, starting at the top row, proceeding left to right:

Port A (hex) Hardware value	Key Label
0xee	1
0xde	2
0xbe	3
0x7e	A
0xed	4
0xdd	5
0xbd	6
0x7d	B
0xeb	7
0xdb	8
0xbb	9
0x7b	C
0xe7	*
0xd7	0
0xb7	#
0x77	D

Now we need to consider the mechanics of reading key presses.

Waiting For Release:
Humans press keys. Microcontrollers read key presses. There is a huge difference between the time it takes a human to press and release a key, and the time it takes a microcontroller to detect and analyze a key press…. One of them is much faster. If we do not plan and program for the time difference, the human and the microcontroller will not communicate successfully. The microcontroller can detect a key press, analyze it, and run a few laps around the program loop before the human ever gets their finger off the key.

To prevent multiple “false” readings of what should be only one key press, the microcontroller ***must be programmed to wait for the key to be released*** before reading another key press.

The logic is pretty simple; just keep looking at the input bits until they show no key presses. The logic is simple. The reality is a bit messier… Keep Reading!

Switch Bounce and De-bouncing:
At the speed of microprocessors, all simple switches "bounce". This is a real physical phenomenon. If we watch the electrical signal as one of the switches is pressed, it might look rather like Figure 3. A microprocessor might read the switch as pressed just after time 1, read the switch as released just before time 2, and then record another press at time 3.

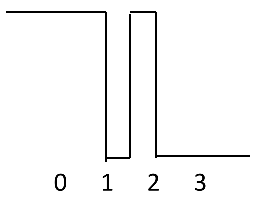

Figure 3
Switch Bounce
(Transition from open to closed)

Oh dear. Not what we had in mind at all! The solution is called "de-bouncing". That is a fancy name for waiting just a bit for the switch to settle, both during closing and opening, before reading it again. For the switches on our development board, a simple 1-millisecond delay will do the trick.

Laboratory Application:
The function of the hex keypad exercise is to continuously read keys as they are pressed, and display each number in binary form on the Port C LEDs.

Program Structure:
The program will include the following functions: Initialize, GetKeyPress, KeyWrite, Delay and KeyRelease. You will also include your two parallel lookup functions from Lab02. All these functions should be placed in a file named Keypad.c. The main.c file will call the Initialize and GetKeyPress functions.

The **Initialization** function will handle initialization of port data direction registers. Port C and Port A.

The Key **Write** function shall shift the number at Port C left four times, and write the four bit digit of the key pressed in the lower four bits of Port C.

- Example code:

```
void KeyWrite(unsigned char
val) {
        PORTC <<= 4;
        PORTC |= (val & 0x0f);
}
```

Waring: Before putting this code in your program Make sure you understand and can add relevant comments about what it is doing.

The **Delay** function shall be a simple 1-millisecond delay. But make it have an integer input that is roughly the number of microseconds to delay.

- Approximately, Delay(1000) will delay about one millisecond.

The Key **Release** function should wait for a pressed key to be released by reading Port A and waiting for the number 0xf0 (waiting for a hex f to appear on the upper nibble).

- One way is to wait for a specific number on **only** the upper nibble (the upper four bits) and to AND the value read from the port with the number 0xf0 must do this line two times. Then compare the result with the desired number, in this case again 0xf0.

- Another way is to wait while the value read from port A is equal to port A. Meaning the finger is still on the button and the reading is the same.

Note: the second option is the easiest and most successful.

- The **GetKeyPress** function: keypress detection loop explained below.

Additional information:

- Port A is named PORTA
- The Port A data directional register is named DDRA
- Pull-up resistors for Port A must be turned on by setting bit PUCR_PUPAE=1.
- Port C is named PORTC
- The Port C data directional register is named DDRC
- The hex numbers to poll each row (send out on Port A): 0x0e, 0x0d, 0x0b, and 0x07.
- The **GetKeyPress** function detection loop will do the following:
  - Store a hex number to Port A (will poll each row, one row at a time).
  - Read Port A. (must do this line two times)
  - Call your lookup function passing value read from port A
    - test both numerical version and the ASCII version lookups. But, only have one in the code at a time.
    - Note: the lookups are your parallel array functions from Lab 02.
  - If your lookup functions returns a 0xff, no key was pressed. Continue the detection loop.
  - If your lookup function returns a value other than 0xff, a key has been pressed. In that case:
    - Call the Write function.
    - Call the Delay function to de-bounce the key press.
    - Call the Release function to wait for the key to be released.
    - Call the Delay function (again) to de-bounce the key release.
    - Continue the detection loop.

Before you start programming you must use the Program Development Cycle worksheet to outline your program completely. You must have this outline checked off before you proceed to write code. You must comment your code and include commented headers on all your files. Name your project LastName_Lab03

Be sure to demonstrate your working lab to either the lab Assistant or Instructor.

Submission: You are to submit your Program files: your main.c, Keypad.c, protos.h files and entire zipped project to the Lab 03 Assignments and your completed Program Development Cycle worksheet worksheet.

Note the closing date and time; as late submissions are not accepted, nor any files sent as messages in Canvas email or attachments.