# A Virtual Peripheral Keyboard Scanner

## Introduction

This application note presents programming techniques for scanning a 4x4 keyboard usually found in both consumer and industrial applications for simple numeric data entry. This implementation uses the SX's internal interrupt feature to allow background operation of the code as a virtual peripheral, uses the Parallax demo board and the internal pull-up capability of SX ports to eliminate the need of any external components.

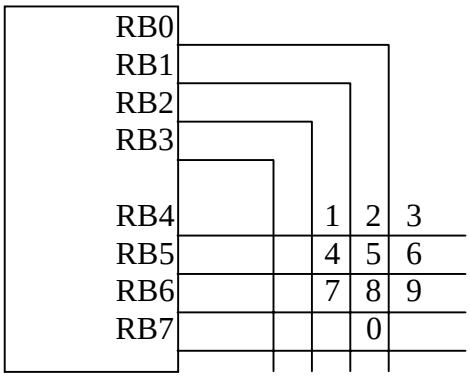


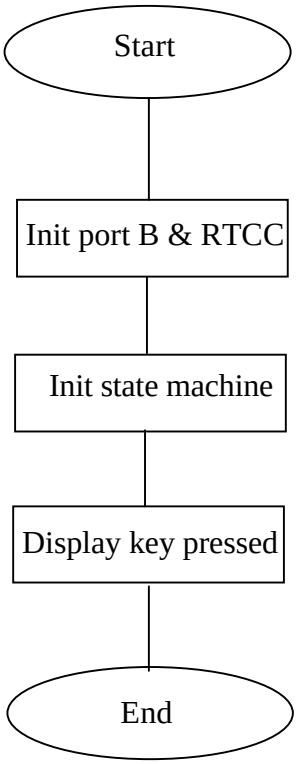

Figure 1 - Keyboard scanner circuit diagram

## How the circuit and program work

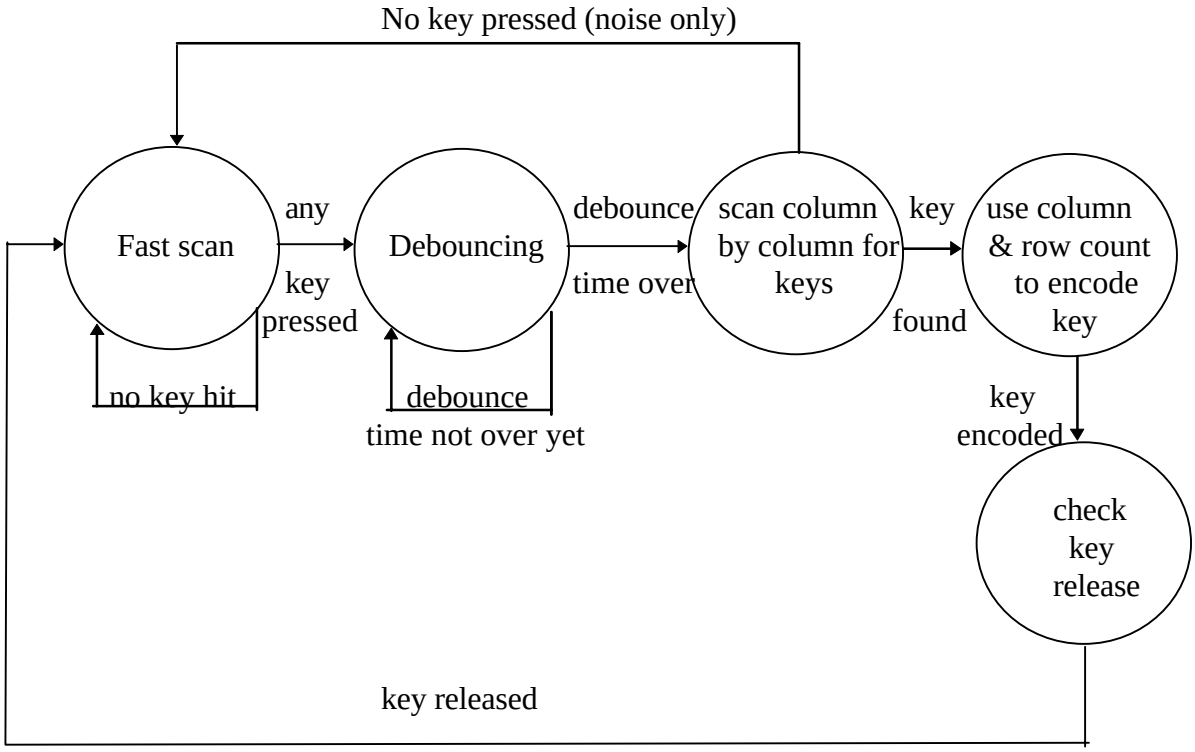
The circuit is just a simple connection to a keyboard matrix as indicated. Note that only 10 out of 16 positions are used. All 16 positions can be used by simply altering the virtual keyboard matrix, which assigns value to the key pressed according to their row and column positions. No external resistors are used due to the capabilities of SX to have internal weak pull up in the range of 20K. (Note: for Parallax demo board users, the connection to R13 for LED drive must be clipped off or disconnected since it will create a pull down effect.)

The interrupt code segment uses a state machine approach to scan the keyboard so as not to tie up the CPU for a long time during keyboard scan.

Flowchart for main loop




State diagram of interrupt routine



In the main program, bit 4-7 of port B are initialized as inputs with Schmitt-trigger and pull-ups while bit 3-0 are configure as outputs. Scanning is accomplished by outputting a zero on a column. If any key on that column is pressed, a zero will be read in the row where that key is located. Due to the internal pull-ups, all inactive inputs will be read as '1's.

The RTCC is initialized to be incremented on internal clock cycles. With a prescaler ratio of 8, coupled with a value of 250 in the RETIW interrupt routine which will be loaded into RTCC, it will give us approximately 1 mS per interrupt.

The state machine is initialized to start with fast scan which output zeroes to all columns and detect all keys at the same time. Once any key is detected, debouncing will be started. Normally this is dependent on the mechanical characteristics of the keys. In this case, we are using a 20 mS debouncing time.

After debouncing is done, a detailed scanning will be done. Each column will have a zero output at a time to detect if any key in that certain column is pressed. If that is the case, a value other than 11110000 (binary) will be read from the inputs (RB4-7) and we will know that a key is pressed. Otherwise, the detection that we made from the fast scan stage may just have been noise and we transition back to fast scan state.

When a value other than 11110000 (binary) is read, then we use that value to find out on which row the key is. That row count and the column count that we keep during column scan will give us the exact location of the key hit by the formula:

index to virtual matrix = 4*row count + column count

| Column count (cnt1) | 3 | 2 | 1 | 0 | | |
|---|---|---|---|---|---|---|
| | RB3 | RB2 | RB1 | RB0 | | |
| scan pattern (RB3-0) | 0111 | 1011 | 1101 | 1110 | | |
| Row count (cnt2) | | | | | input value (RB7-4) | |
| 3 | F | E | D | C | 1110 | RB4 |
| | B | A | 9 | 8 | 1101 | RB5 |
| 2 | | | | | | |
| 1 | 7 | 6 | 5 | 4 | 1011 | RB6 |
| 0 | 3 | 2 | 1 | 0 | 0111 | RB7 |

Note: Index to virtual matrix is indicated at each intersection.

Encoding is done by simply loading the value in the virtual matrix table using the IREAD instruction. This enables us to easily change the encoded value and adapt to different keyboard layouts.

After encoding, the interrupt routine will transition to fast scan state when the key is released. This is to ensure that single key press will not be detected as multiple key strokes.

## Modifications and further options

To accommodate a different 4 x 4 keyboard layout, it is very easy to just change the virtual matrix. The first entry is for index 0 and the last entry is for the index value of 0F (hex).

Notice that all values are negated to enable display on LED on port B. A zero causes the LED to be lit and an one will turn it off. If desired, the ~ sign can be removed to obtain the true value.

For example, the following matrix can be used for a 4 x 4 hexadecimal keyboard with real value (not display value):

```
virtual matrix  dw      $F              ; bottom right
                dw      $E
                dw      $D
                dw      $C
                dw      $B
                dw      $A
                dw      9
                dw      8
                dw      7
                dw      6
                dw      5
                dw      4
                dw      3
                dw      2
                dw      1
                dw      0               ; upper left
```

Finally, to integrate this code with other virtual peripherals, keep in mind that it has a varying execution rate depending on the state and therefore should be placed after code with uniform execution rate.