



Z i L O G

*Technical Note*

# 4 x 4 Keyscan

TN000501-0603

---

## General Overview

The ability to scan the keys of a small keypad is fundamental to many microcontroller-based applications, including telephony, security, instrumentation, and remote control. This Technical Note provides a straight-forward and code-efficient method for key-scanning. When the concepts are understood, the user can modify the code to reduce it even further.

As written, this key-scanning application operates on any Z8 Microcontroller.

## Discussion

Port 2 is used for the scanning function. The upper nibble of Port 2 is used for columns, while the lower nibble is used for rows. Columns are driven Low, sequentially, while each row is tested, in turn, to determine whether it is connected to the active column by the action of a key press. If no active key is found for one column, the next column in the sequence is brought Low and each row is again tested for an active key. For code efficiency, the actual key-test function is a subroutine, called GET\_KEY. If an active key is found, GET\_KEY returns with the Zero flag set.

Key debouncing is a method of detecting the same key four consecutive times via the KEY\_SCAN function. The value of the KEY\_SCAN function can easily be changed if more or less debouncing is required, or to improve the rate at which key presses are detected. When a new key is debounced, KEY\_SCAN returns with the key\_flag set and the active key in the key\_cnt register.

This module always returns the active key, regardless whether the active key is held down without releasing since the last call of the KEY\_SCAN function. A higher level of code in the body of the program must make the decision to either act on continuously-held keys or ignore them. If the active key is to be ignored, the software waits until the KEY\_SCAN function returns the null key flag or a different key than previously pressed.

Port 2 is set up using the Port 2 Mode Register (P2M) such that P27 through P24 are outputs, while P23 through P20 are inputs. Port 2 is also programmed for open-drain outputs by clearing bit 0 of the Port 3 Mode Register (P3M). Because Port 2 cannot drive any output High, as programmed, eight resistor pull-ups are required. The advantage to this requirement is that accidentally pressing multiple keys cannot cause a port to be over-driven (and possibly cause damage to the device). Standard resistor packs are available for this purpose that save on assembly cost over individual discreet resistors.

## Sample Code

```

; 4 X 4 Keyscan routine.
; Columns are driven Low, sequentially, on Ports 27 to 24. Rows are input on
; Ports 23 to 20. Upon return, key_flag bit 0 is set if a new key is found and
; debounced. It is reset, otherwise. As implemented here, this is a called
; subroutine. The comments are included if the keyscan is inside an interrupt
; service routine. One keyscan method is to
; use a timer interrupt to determine the rate at which the keys are scanned.
; The same timer interrupt could also be used for a delay or clock function.

;*****
;(The following code should be included in the equate table area.)

WORK_REG_0 equ 00h ;Working register group 0. (00h to 0Fh,
;register space)

key_cnt equ r4
key_tmp equ r5
row_cnt equ r6
row_tmp equ r7
bounce equ r8
key_flag equ r9

;*****
;(This would be included with initialization code.)

LD P2M,#00001111b ;Set Port 2 for upper nibble outputs,
;lower nibble inputs.
LD P3M,#xxxxxxx0b ;Set Port 2 for open-drain outputs.
;External pull-up resistor
;should be used on all (eight) port 2 pins.
;Set-up port 3 as required.

;*****

KEY_SCAN:
PUSH FLAGS ;Not required if inside an interrupt service rou-
tine.
PUSH RP ;Save old RP.
SRP #WORK_REG_0 ;Point to working reg. group 0.
AND key_flag,#11111110b ;Reset new key flag, bit 0.
COL_0:
LD p2,#11101111b ;Column 0 driven low.
CLR key_cnt ;Key 0-3.
CALL GET_KEY ;This finds active key if any.
JR Z,KEY_FOUND ;Jump to debounce key.

COL_1:
LD p2,#11011111b ;Column 1 driven low.
LD key_cnt,#4 ;Key 4-7
CALL GET_KEY
JR Z,KEY_FOUND

COL_2:

```

```

LD      p2,#10111111b      ;Column 2 driven low.
LD      key_cnt,#8         ;Key 8-11
CALL    GET_KEY
JR      Z,KEY_FOUND

COL_3:
LD      p2,#01111111b      ;Column 3 driven low.
LD      key_cnt,#12        ;Key 12-15
CALL    GET_KEY
JR      NZ,SCAN_EXIT       ;Jump for no active key found.

KEY_FOUND:
                ;Active key found.
ADD     key_cnt,row_cnt    ;Add row value from GET_KEY to key base value.
CP      key_cnt,key_tmp    ;If key is same,
JR      Z,KEY_SAME        ;go debounce it.
LD      bounce,#4         ;Not same, so set debounce counter
LD      key_tmp,key_cnt    ;Make them the same for the next time around.
JR      SCAN_EXIT         ;Exit with no action taken.

KEY_SAME:
                ;Debounce counter. The same key for 4 reads?
DJNZ   bounce,SCAN_EXIT   ;If not, exit with no action taken.

DEBOUNCED:
LD      key_tmp,#0FFh      ;Else debounced so set key_tmp = non-key value.
OR      key_flag,#00000001b ;Debounced new key is in the "key_cnt" register.
                ;Set new key flag, bit 0.

SCAN_EXIT:
LD      p2,#11111111b      ;All columns inactive.
POP     RP                 ;Restore old RP.
POP     FLAGS              ;Not required if inside an interrupt service rou-
tine.
RET                                ;Use IRET if inside an interrupt service routine.

;*****

; GET_KEY is called to return the active KEY, if any. The Zero Flag is set if
an active
; key is found.

GET_KEY:
LD      row_tmp,p2         ;Input port 2 and save in temp_rows.
AND     row_tmp,#0Fh       ;Clear for Row data only.

ROW_0:
CLR     row_cnt            ;Set for Row 0.
CP      row_tmp,#00001110b ;Row 0 key?
JR      Z,KEY_RET         ;Yes, return.

ROW_1:
INC     row_cnt            ;No. Set for Row 1.
CP      row_tmp,#00001101b ;Row 1 key?
JR      Z,KEY_RET         ;Yes, return.

ROW_2:
INC     row_cnt            ;No. Set for Row 2.
CP      row_tmp,#00001011b ;Row 2 key?

```



```
JR      Z,KEY_RET      ;Yes, return.

ROW_3:
  INC   row_cnt      ;No. Set for Row 3.
  CP    row_tmp,#00000111b ;Row 3 key?

KEY_RET:
  RET   ;Return.

;*****
```



This publication is subject to replacement by a later edition. To determine whether a later edition exists, or to request copies of publications, contact:

**ZiLOG Worldwide Headquarters**

532 Race Street  
San Jose, CA 95126  
Telephone: 408.558.8500  
Fax: 408.558.8300  
[www.zilog.com](http://www.zilog.com)

ZiLOG is a registered trademark of ZiLOG Inc. in the United States and in other countries. All other products and/or service names mentioned herein may be trademarks of the companies with which they are associated.

**Information Integrity**

The information contained within this document has been verified according to the general principles of electrical and mechanical engineering. Any applicable source code illustrated in the document was either written by an authorized ZiLOG employee or licensed consultant. Permission to use these codes in any form, besides the intended application, must be approved through a license agreement between both parties. ZiLOG will not be responsible for any code(s) used beyond the intended application. Contact the local ZiLOG Sales Office to obtain necessary license agreements.

**Document Disclaimer**

©2003 by ZiLOG, Inc. All rights reserved. Information in this publication concerning the devices, applications, or technology described is intended to suggest possible uses and may be superseded. ZiLOG, INC. DOES NOT ASSUME LIABILITY FOR OR PROVIDE A REPRESENTATION OF ACCURACY OF THE INFORMATION, DEVICES, OR TECHNOLOGY DESCRIBED IN THIS DOCUMENT. ZiLOG ALSO DOES NOT ASSUME LIABILITY FOR INTELLECTUAL PROPERTY INFRINGEMENT RELATED IN ANY MANNER TO USE OF INFORMATION, DEVICES, OR TECHNOLOGY DESCRIBED HEREIN OR OTHERWISE. Except with the express written approval ZiLOG, use of information, devices, or technology as critical components of life support systems is not authorized. No licenses or other rights are conveyed, implicitly or otherwise, by this document under any intellectual property rights.