ZBasic

Application Note

AN-205 Interfacing a Matrix Keyboard

Introduction

Some applications require a means for a user to enter data for used by the system. Often, this requirement can be met by the addition of a keyboard or keypad. In most cases, inexpensive keypads will be of the keyswitch matrix type, meaning that the keys are arranged in an X-Y matrix of rows and columns. In this configuration, pressing one of the keyswitches makes a connection between one row line and one column line. This application note shows how a matrix keyboard can be connected to a ZX processor using only a few I/O lines and gives some sample software for scanning the keyboard.

Although the examples given in this application note are based on the ZX-24, the principles can be equally well applied to the other ZX processors and to microcontrollers in general.

Connecting the Keypad

There are many different ways that you could connect a matrix keypad to a ZX processor. Which you choose depends on the size of the matrix and the number of I/O lines that you have available for that purpose. The example connection shown in Figure 1, illustrates connecting a 4x3 TouchTone keypad using just 3 I/O lines. For clarity, other required connections to the ZX-24 are not shown. Note, particularly, that the row drivers (74HC05) are open drain devices. These are used because when multiple keys in a column are pressed simultaneously the outputs of the row drivers are shorted together.

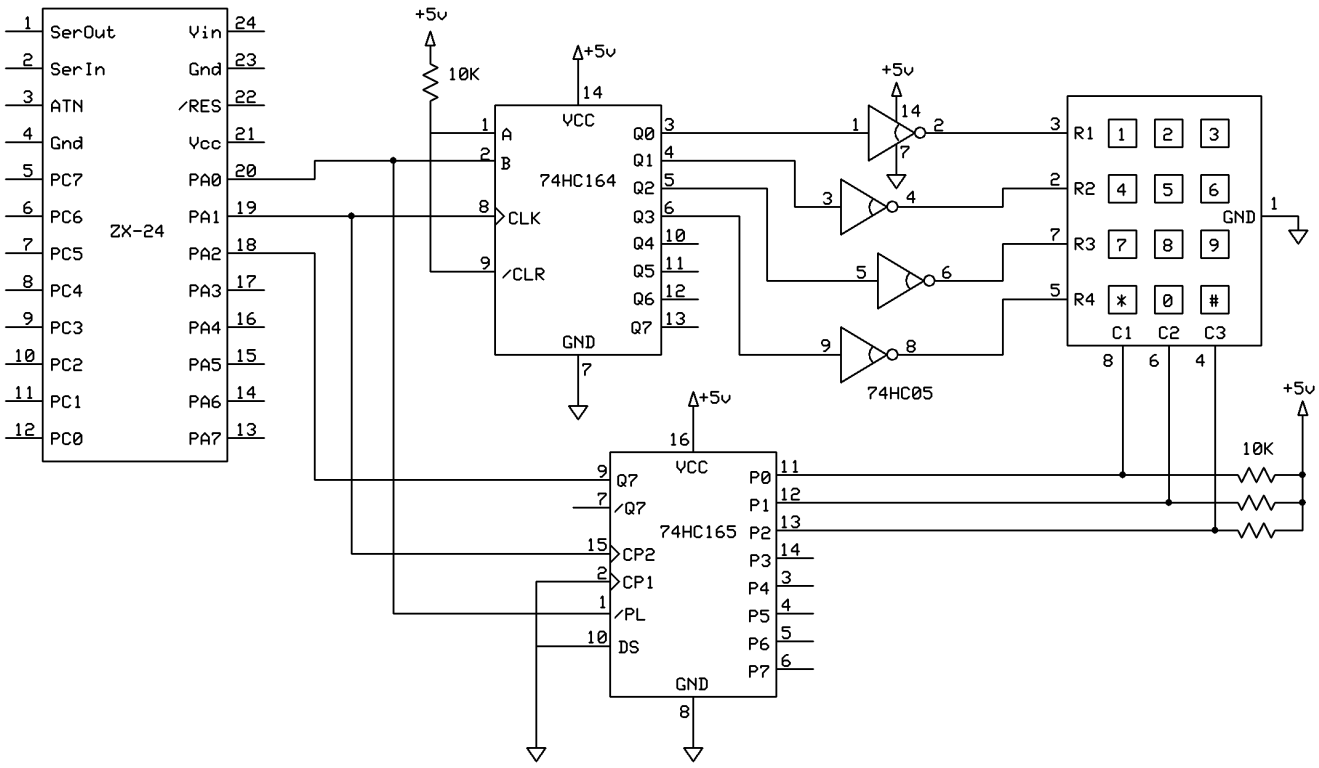

Figure 1 Matrix Keypad Interface

This example uses two shift registers: one to shift a single bit across the rows of the keypad and one to read in the resulting column signals. The keypad scanning process, used to determine if a key is pressed, proceeds as follows. The '164 shift register is loaded with a value so that only one of the outputs Q3-Q0 is high. For this first step, assume that the Q0 output, corresponding to row 1 of the keypad, is high. Next, the column outputs are latched into the '165 shift register. Since the inputs to the '165 shift register have pull-up resistors, if no key in row 1 is pressed all of the inputs to the '165 will be high. If a key is pressed, the input to the '165 corresponding to the column of the pressed key will be low. Note that if multiple keys are pressed simultaneously in a row, multiple inputs to the '165 will be low.

If no keypress is detected in the row, the output of the '164 corresponding to the next row is made high, again with the remaining outputs low. The column outputs are again sampled as before. This process is repeated until all rows have been scanned or until a keypress is detected.

Note that this example economizes on I/O line use by sharing the clock line between the two shift registers and by having the "data in" line to the '164 also serve as the "load" signal to the '165. If more I/O lines are available, the driver software could be simplified slightly. Also, instead of using a shift register to capture the column data it could be fed directly into the ZX. That change would require a substantial, but straightforward, change to the driver software.

The Scanning Software

The code excerpts below illustrate the matrix scanning and key decoding processes. Note that pin number assignments are specified using the port.bit constants instead of physical pin numbers.

```
' define the port/pin numbers used
Private Const SR_Clock as Byte = A.1
Private Const SR_DataOut as Byte = A.0

Private Const SR_DataIn as Byte = A.2
Private Const SR_DataLatch as Byte = A.0

' define the keyboard layout
Private Const kbdRows as Byte = 4    ' number of keypad rows
Private Const kbdCols as Byte = 3    ' number of keypad columns

'-------------------------------------------------------------------------------
'
'' KeyIsDown
'
' Determine if a key is pressed.  If so, return its key code, otherwise, 0.
' Note that this function returns the code of the first key detected.  No
' effort is made to determine if multiple keys are down.
'
Public Function KeyIsDown() as Byte
   Dim row as Byte
   Dim col as Byte
   Dim rowBits as Byte
   Dim colBits as Byte

   KeyIsDown = 0

   ' scan by rows, looking for a key that is down
   rowBits = &H10
   For row = 1 to kbdRows
       ' initialize the shift register with one row line high
       Call ShiftOut(SR_DataOut, SR_Clock, 4, rowBits)

       ' latch the column data, shift it in
```

```
        Call PutPin(SR_DataLatch, zxOutputLow)
        Call PutPin(SR_DataLatch, zxOutputHigh)
        colBits = ShiftIn(SR_DataIn, SR_Clock, 8)

        ' scan the column bits looking for a zero
        For col = 1 to kbdCols
            If (Not CBool(colBits And 1)) Then
                ' determine the key code of the key pressed
                KeyIsDown = keyCode(row, col)
                Exit Function
            End If
            colBits = Shr(colBits, 1)
        Next

        ' prepare to scan the next row
        rowBits = Shl(rowBits, 1)
    Next
End Function

'--------------------------------------------------------------------------------

' This data table maps row/column pairs to ASCII codes.
' Note that it must be indexed as keyTable(col, row).
Private keyTable as ByteTableData({
    "123"
    "456"
    "789"
    "*0#"
})

'--------------------------------------------------------------------------------
'
'' keyCode
'
' Given row and column values, return the code for the corresponding
' key or zero if the row/column combination is invalid.
'
Private Function keyCode(ByVal row as Byte, ByVal col as Byte) as Byte
    If ((row >= 1) And (row <= CByte(UBound(keyTable, 2))) And _
            (col >= 1) And (col <= CByte(UBound(keyTable, 1)))) Then
        keyCode = keyTable(col, row)
    Else
        keyCode = 0
    End if
End Function
```

This code provides a foundation upon which additional functions may be based. One function that may be useful is to await a keypress. An example of such a routine is shown below. This code implements a “debouncing” delay and also implements “auto-repeat”. This example code only supports single key presses – it does not implement “N-key rollover”. If you need this capability, you should be able to find example code on the Internet for doing so.

```
' delay times, in RTC ticks
Private Const DebounceDelay as UnsignedInteger = 15
Private Const AutoRepeatDelay As Long = 125
Private Const KeyWait As UnsignedInteger = 25
```

```
'-------------------------------------------------------------------------------
'
'' WaitKey
'
' Wait for a key to be pressed and return its code.  If a key is already down
' upon entry, wait for it to be released first.  However, after waiting a
' certain period, auto-repeat is invoked and the key that is down is returned.
'
Public Function WaitKey() as Byte
   Dim key as Byte

   ' wait for all keys to be released
   key = KeyIsDown()
   If (key <> 0) Then
       Dim repeatTime as Long

       repeatTime = Register.RTCTick + AutoRepeatDelay
       Do
           ' wait a bit
           Call Sleep(KeyWait)

           ' see if a key is still down
           If (KeyIsDown() = 0) Then
               ' debounce the key release
               Call Sleep(DebounceDelay)
               If (KeyIsDown() = 0) Then
                   key = 0
                   Exit Do
               End If
           ElseIf (Register.RTCTick > repeatTime) Then
               ' effect auto-repeat
               Exit Do
           End If
       Loop
   End If

   ' wait for a key to be pressed
   Do While (key = 0)
       key = KeyIsDown()
       If (key <> 0) Then
           ' debounce the key down
           Call Sleep(DebounceDelay)
       Else
           Call Sleep(KeyWait)
       End If
   Loop

   WaitKey = key
End Function
```

Once this building block is in place, additional functions can be created. The example project keyboard.pjt in the accompanying .zip file contains a function in keyboard.bas that reads keypresses from the keyboard and composes a numeric value corresponding to the sequence. Both floating point and integral value forms are provided.

Adding More Keys

You can add more keys to your application by using some individual keyswitches in addition to a standard keypad. The circuit of Figure 2 shows how an additional “column” can be added alongside the TouchTone keypad. (Connections to the ZX are the same as in the preceding figure; they are omitted for clarity.) Depending on your needs, the actual physical

position may or may not reflect the logical positioning of the keys as a column. For example, the four additional switches could be arranged horizontally beneath an LCD display. To accommodate the additional column the only change required in the driver software is to modify the value of the constant `kbdCols`. Additional columns and/or rows can be added by extending the concepts illustrated here.

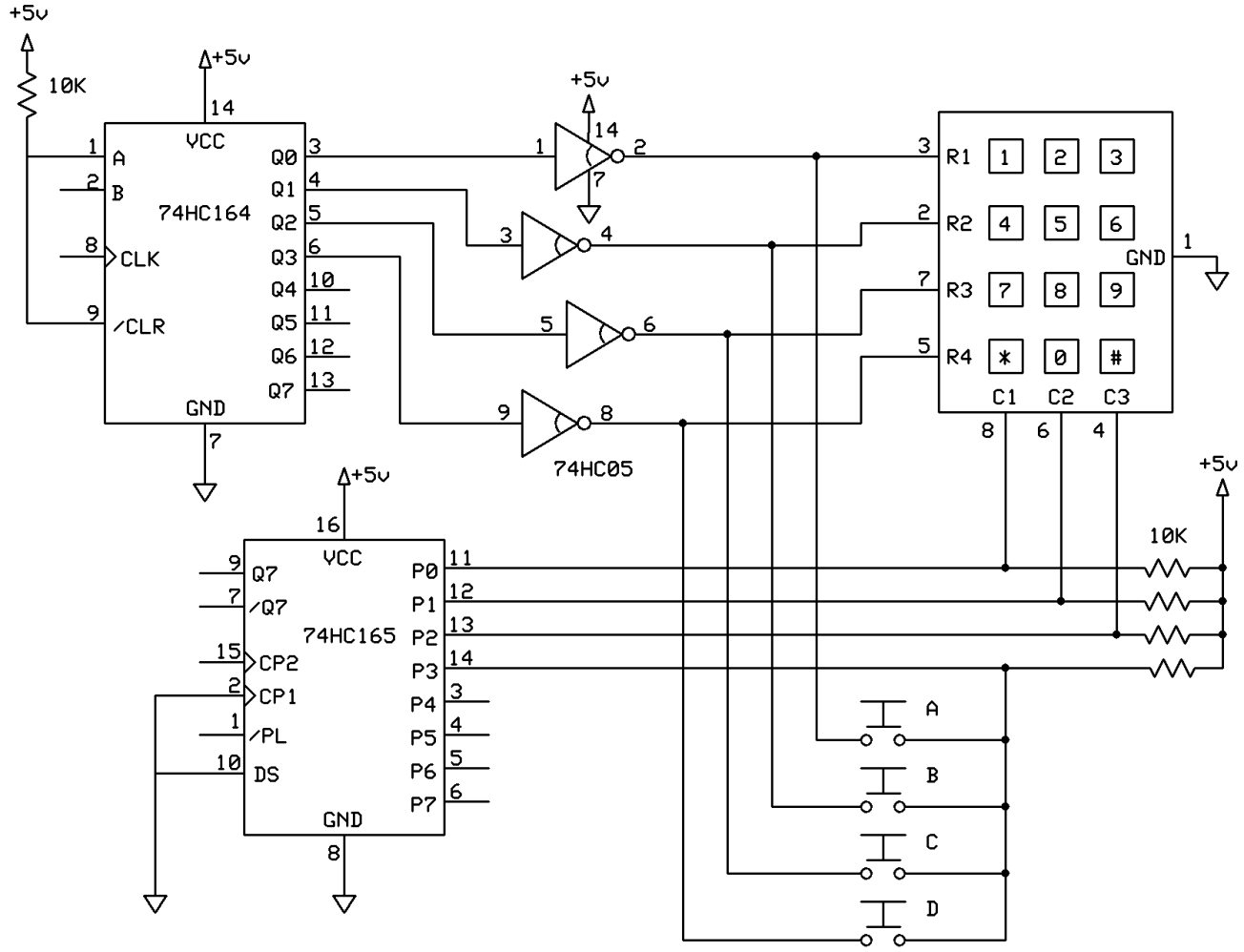

Figure 2: Adding More Keys to a Keypad

An Alternate Interface Method

An alternate method of interfacing a matrix keypad up to 4x4 in size is to use an I2C I/O expander like the PCF8574. Connecting the I/O expander to a ZX only requires two I/O lines and if you're already using other I2C devices in your application, connecting the keyboard this way requires no additional I/O lines.

The schematic in Figure 3 illustrates the simplicity of this interfacing method. The I/O lines of the PCF8574 are bi-directional, using weak pull-ups for lines used as inputs. This allows a simpler circuit with no additional pull-up resistors required (although they can be added if desired). As with the circuits described earlier, a fourth column of switches can be added easily using the same technique.

The code for scanning the keypad via the I2C I/O expander is nearly identical to that shown earlier. The only difference is that the code for driving row lines and reading column lines is performed using I2C routines instead of ShiftOut() and ShiftIn(). The full source code for the I2C version is contained in the accompanying .zip file as i2c_keyboard.pjt

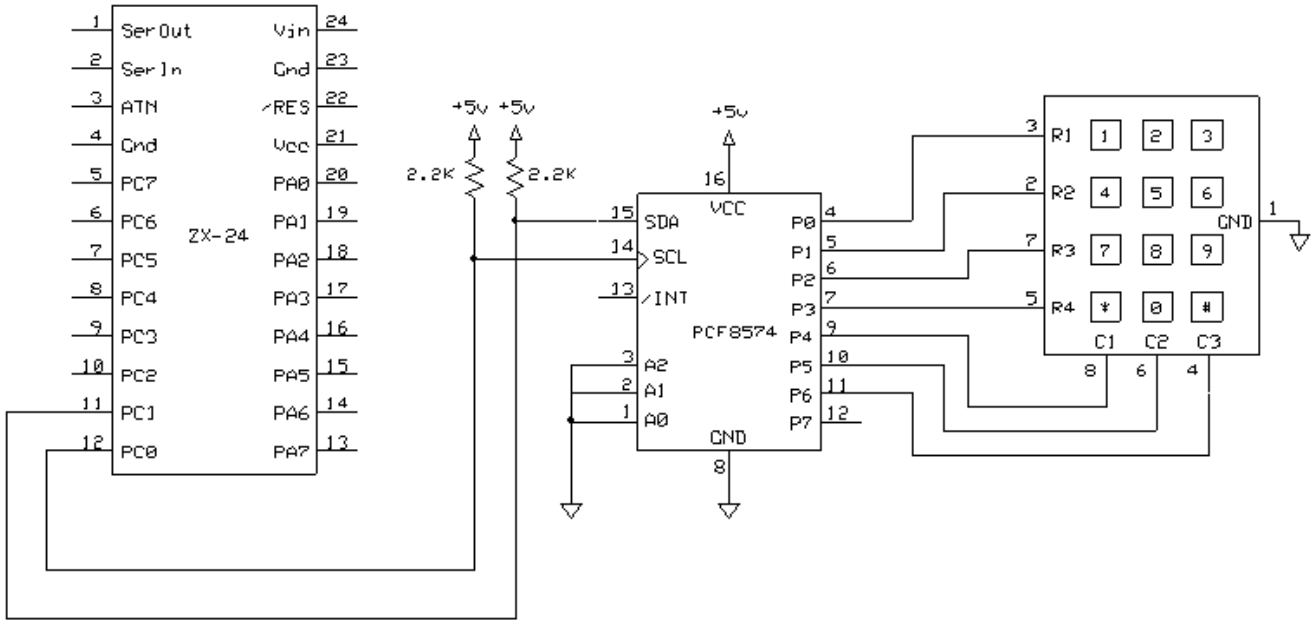

Figure 3: I2C Keypad Interface

Interrupt Driven Keyboard

Both of the examples described earlier utilize a technique called “polling” in which the keyboard is checked from time to time for a key being pressed. Any keys pressed and released between these times are not recognized. One way to avoid missing key presses is to design the keyboard interface so that it generates an interrupt to the processor when a key is pressed.

The I2C I/O expander provides an open-drain output that goes low whenever one of the I/O lines changes state. This capability can be used notify the ZX of keyboard activity by setting all of the row lines low when not actively scanning the keypad. Then, when any key is pressed the corresponding column line will go low causing a high-to-low transition on the /INT output of the I/O expander. If this output is connected to one of the external interrupt inputs of the ZX (or any I/O line that can generate a pin change interrupt) you can then use the ZBasic System Library routine WaitForInterrupt() to be notified when keyboard activity occurs. Note that an external pull-up resistor on the /INT output is needed – a 10K resistor should be suitable. An example of using this method is given in the project i2c_int_keyboard.pjt in the accompanying .zip file.

Author

Don Kinzer is the founder and CEO of Elba Corporation. He has many years experience working with microprocessors, microcontrollers and general purpose computers. Don can be contacted via email at *dkinzer@zbasic.net*.