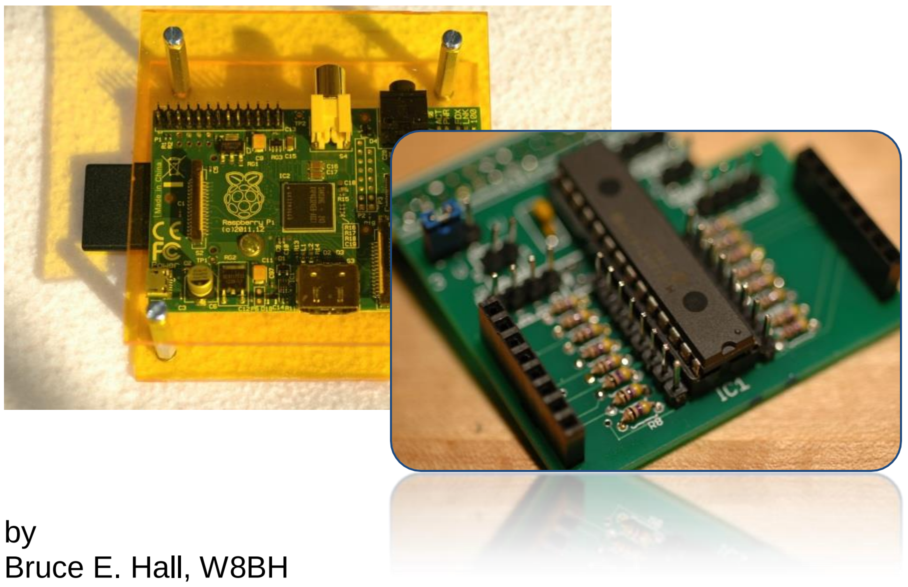


Beginner's Guide to the PI MATRIX

Part 4 :
Text Scrolling

by
Bruce E. Hall, W8BH

1) INTRODUCTION

In the first three parts of this series, we looked at how to build the Pi Matrix and how to drive its 16 pins. But 16 pins cannot completely address all 64 LEDs at the same time. In this tutorial we'll talk about multiplexing and learn how to scroll text characters across the display.

2) MULTIPLEXING

So far, all of our display routines involve turning on a number of LEDs in a row, and repeating the *same* pattern over any number of rows. But for some applications, this is not enough. Suppose we need to turn on 3 LEDs in row 1, a different number of LEDs in row 2, and yet another pattern in row 3. We will need this capability if we want to display complex symbols on the display, like text characters. We need multiplexing.

With multiplexing, we don't display all the rows at the same time; we display them sequentially:

For each row:
- Display pattern #n on row #n
- Wait a few milliseconds (at most)

If we do this fast enough, our eyes are tricked into thinking that all of the rows are being displayed at the same time. We must refresh the entire display at least 30 times per second, which means that we cannot spend more than about 4 mS on each row. Python does not run too quickly on the Pi, so the wait between rows may be omitted. The following simple routine will do all the multiplexing we need:

```
def MultiplexDisplay (z,count):
```

```
for count in range(0,count):
    for row in range(0,8):
        SetPattern(1<<row,z[row]))
```

The variable Z is a list with 8 elements, each element holding a row pattern: z[0] holds the pattern for row 0, etc. The whole display is refreshed (count) times, giving the user enough time to view and interpret the display. Easy!

3) PUPPIES AND FONTS

I thought it would be interesting to draw some characters on the matrix, but my 9 year old daughter had a different idea: puppies. She pulled out some graph paper, asked me to draw the right-sized box, and then proceeded to shade in the squares of her puppy design.

	C0	C1	C2	C3	C4	C5	C6	C7
R0			X					
R1		X	X					
R2	X	X	X			X		
R3			X				X	
R4			X	X	X	X		
R5			X	X	X	X		
R6			X			X		
R7			X			X		

She handed the paper back to me, and waited for me to put her design on the matrix. Let's do it! Starting with the top row, we see that only bit 2 is lit. For this we need binary 00000100, which is 0x04. In the next row, we have bits 1 and 2, which is 0x06. Here are the values that we will need for all eight rows: 0x04, 0x06, 0x27, 0x44, 0x3C, 0x3C, 0x24, 0x24. That's our list input for the routine:

```
def DisplayPuppy():
        z = [0x04, 0x06, 0x27, 0x44, 0x3C, 0x3C, 0x24, 0x24]
        MultiplexDisplay(z,100)
```

My daughter was pleased, and I was too! This gave me enough encouragement to tackle the alphabet. But surely someone has done this before me. A quick internet search for 8x8 fonts gave me exactly what I needed: 128 characters in ASCII order, encoded as 8 rows of 8 pixels (bits). All I needed to do was 'pythonize' the data into one big list of lists, like this:

```
data = [ …

        [ 0x0C, 0x1E, 0x33, 0x33, 0x3F, 0x33, 0x33, 0x00],   # U+0041 (A)
        [ 0x3F, 0x66, 0x66, 0x3E, 0x66, 0x66, 0x3F, 0x00],   # U+0042 (B)
        [ 0x3C, 0x66, 0x03, 0x03, 0x03, 0x66, 0x3C, 0x00],   # U+0043 (C)

        … ]
```

This looks useful, so I put it into its own file, font0.py. To add it to your program, all you have to do is import it. The big list is referred to by the module name, dotted with list name:

font0.data. The index to any character in the list is the ascii number of the character, which we can get by using the python function ord. Try this:

```
import font0
char = raw_input("Enter a character to display: ")
z = font0.data[ord(char)]
#print char, z
MultiplexDisplay(z,100)
```

With just a few lines of code we can display any character on our Pi Matrix. I added the '#print z' statement for debugging purposes. It is commented out, so it doesn't do anything. Remove the hash and it will show you the character's data.

4) TEXT

Now let's try displaying words and sentences. Enter a string, and display each character in the string:

```
def DisplayString():
    message = raw_input("Enter a message to display: ")
    for char in message:
        z = font0.data[ord(char)]
        #print char, z
        MultiplexDisplay(z,100)
```

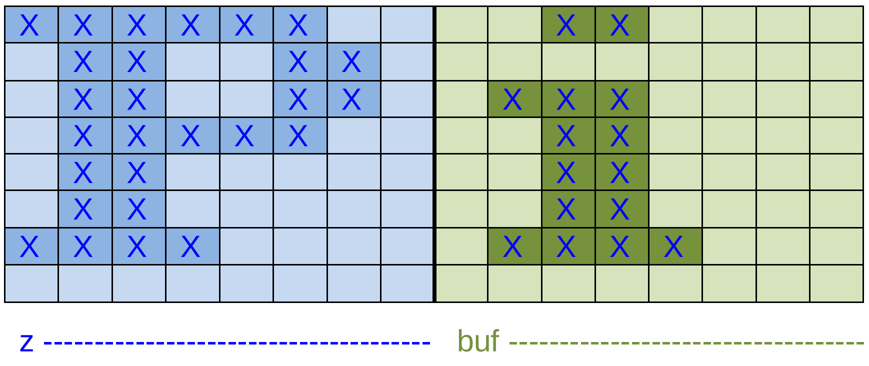
Python is pretty cool here, because it is able to iterate over all of the characters in the message without having to grab each character or explicitly set boundaries.

5) SCROLLING

Its confession time: I have no idea how one is supposed to implement a text scroll function. So what follows may be the worst text scrolling routine ever made. But I made it, and it works.

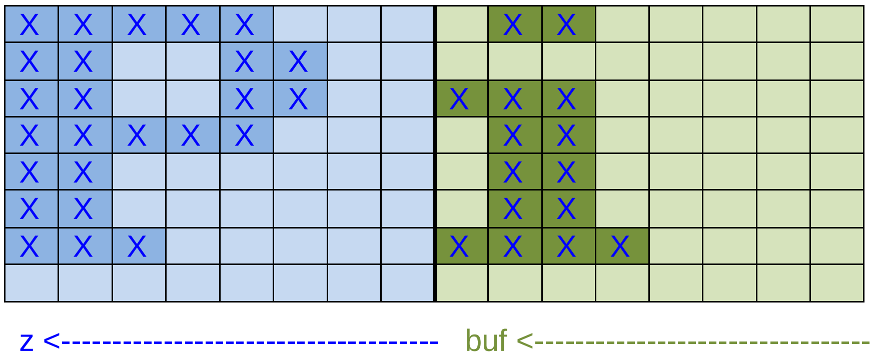
To scroll you need two pieces of data: the data that is currently being displayed and the data that is about to be displayed. On our matrix this means we will keep track of data for two characters at a time. I call our current character 'z', since this is how I started (above), and call the next (buffered) character 'buf'. You can name them better! To scroll, we shift our current display one pixel to the left, and move our buffered data onto the display by the same amount. English is written left-to-right, so leftward scrolling works the best.

Let's try an example, the word 'Pi':



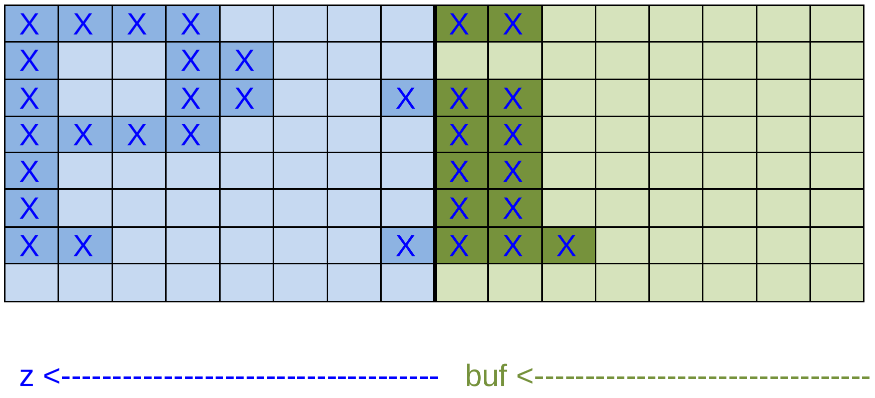
Here is our data: the blue boxes are 'Z', representing what is being displayed on the matrix. The green boxes are 'buf', the buffered data waiting to be written to the display.

In this example we will scroll the word 'Pi'. The 'P' is being displayed, and the 'i' waiting its turn. If we scroll to the left by one pixel, the leftmost blue column disappears. Although only the 'P' is visible, both characters have shifted slightly to the left:



To do this in code, take one row at a time. Shift the bits in the row to the left. We just need to make sure that the leftmost bit in the green box (buf) gets transferred to the rightmost bit in blue (z) .

Here is what things look like on the next scroll. The leading part of the 'I' is now appearing on the display. But wait a second – what if we were doing a longer word, like 'pie'. We need to add the 'e', don't we? There is too much space behind the 'i' already:



Don't worry. Remember that the green box is just a buffer, and isn't being displayed. We'll fill it with a new character as soon as the previous character has been completely shifted 'into the blue'.

It is time to code the shift routine, using my friends z & buf:

```
def ScrollLeft (z,buf):
    for row in range(0,8):
        z[row] >>= 1                #shift current image row
        if buf[row] & 0x01:         #is lsb of buffer high?
            z[row] |= 0x80          #rotate lsb of buf into msb of Z
        buf[row] >>= 1              #shift buffer row, too.
```

The '>>= 1' operation shifts the operand one bit: b7 becomes b6, and so on. We do this for each z[row] and buf[row]. The if statement shifts data from buf to z.

Now we have all the parts we need to scroll. Start with a blank display, and load the first character into buf. Scroll one bit at a time, and display the data. Every 8th scroll, load a new character into buf. Done!

6) MAKING IT USEFUL

My first python program grabbed user input from a prompt like this: st = raw_input("Enter something: "). There are other ways to get input, though. One method is to take it from a command line parameter, like this:

```
./matrix4.py "Go Away!"
```

It is easy to get the command line parameters in Python. First, import the sys module. Then, sys.argv will return the list of command line parameters:

```
import sys
print sys.argv
```

The program name 'matrix4.py' is contained in sys.argv[0], and our text is in sys.argv[1]. We can also get text from something called 'standard input'. Stdin is the source for command-line programs in all Unix-like operating systems. By default, this is the keyboard. If we want to look for other sources from Python, we can read from sys.stdin instead. Combining the two gives us lots of input choices:

```
import sys

if len(sys.argv)>1:
    st = sys.argv[1]
else:
    st = sys.stdin.read()
```

Now we can get do all sorts of fancy linux stuff, like pipes and redirects:

```
./matrix4.py "The yellow brown dog"
echo "once I pondered weak and weary" | ./matrix4.py
cat poem.txt | ./matrix4.py
./matrix4.py <poem.txt
```

7) CONCLUSION

Visit mypishop.com and pick up a Pi Matrix display. Build it. Learn some Python and have fun!

Pi Matrix tutorial #1, construction: http://w8bh.net/pi/PiMatrix1.pdf

Pi Matrix tutorial #2, test routines: http://w8bh.net/pi/PiMatrix2.pdf

Pi Matrix tutorial #3, toolkit: http://w8bh.net/pi/PiMatrix3.pdf

Pi Matrix tutorial #4, text scrolling: http://w8bh.net/pi/PiMatrix4.pdf

YouTube video: http://youtu.be/VbPBNmlGy34

8) PYTHON SCRIPT for PART4:

```
#!/usr/bin/python

########################################################################
#
#    Matrix4:  text scrolling on the Pi Matrix
#
#    This application will let you scroll text
#    across the Pi Matrix.
#
#    To learn more, visit w8bh.net
#
#    Author:  Bruce E. Hall
#    Date  :  21 Feb 2013 <bhall66@gmail.com>
#
########################################################################

import smbus                #gives us a connection to the I2C bus
import time                 #for timing delays
import random               #random number generator
import font0                #basic bitmapped 8x8 VGA font
import sys                  #for access to command-line parameters

#Definitions for the MCP23017 chip
ADDR   = 0x20               #The I2C address of our chip
DIRA   = 0x00               #PortA I/O direction, by pin. 0=output, 1=input
DIRB   = 0x01               #PortB I/O direction, by pin. 0=output, 1=input
PORTA  = 0x12               #Register address for PortA
PORTB  = 0x13               #Register address for PortB


ORIENTATION = 180           #default viewing angle for the pi & matrix
                            #set this value according to your viewing angle
                            #values are  0 (power jack on left)
                            #           90 (power jack on bottom)
                            #          180 (power jack on right)
                            #          270 (power jack on top)

rows   = 0x00               #starting pattern is (0,0) = all LEDS off
columns= 0x00
delay  = 0.08               #time delay between LED display transitions
                            #smaller values = faster animation
```

```
########################################################################
#
#   Lower level routines for bit-manipulation.
#   Nothing here relates to our snazzy Pi Matrix.
#

def ReverseBits (byte):
    #reverse the order of bits in the byte: bit0 <->bit 7, bit1 <-> bit6, etc.
    value = 0
    currentBit = 7
    for i in range(0,8):
        if byte & (1<<i):
            value |= (0x80>>i)
            currentBit -= 1
    return value

def ROR (byte):
    #perform a 'rotate right' command on byte
    #bit 1 is rotated into bit 7; everything else shifted right
    bit1 = byte & 0x01          #get right-most bit
    byte >>= 1                  #shift right 1 bit
    if bit1:                    #was right-most bit a 1?
        byte |= 0x80            #if so, rotate it into bit 7
    return byte

def ROL (byte):
    #perform a 'rotate left' command on byte
    #bit 7 is rotated into bit 1; everything else shifted left
    bit7 = byte & 0x080         #get bit7
    byte <<= 1                  #shift left 1 bit
    if bit7:                    #was bit7 a 1?
        byte |= 0x01            #if so, rotate it into bit 1
    return byte


########################################################################
#
#   Lower Level LED Display routines.
#   These write data directly to the Pi Matrix board
#

def Write (register, value):
    #Abstraction of I2C bus and the MCP23017 chip
    #Call with the chip data register & value pair
    #The chip address is constant ADDR (0x20).
    bus.write_byte_data(ADDR,register,value)

def EnableLEDS ():
    #Set up the 23017 for 16 output pins
    Write(DIRA, 0x00)           #all zeros = all outputs on PortA
    Write(DIRB, 0x00)           #all zeros = all outputs on PortB

def DisableLEDS ():
    #Set all outputs to high-impedance by making them inputs
    Write(DIRA, 0xFF);          #all ones = all inputs on PortA
    Write(DIRB, 0xFF);          #all ones = all inputs on PortB

def TurnOffLEDS ():
    #Clear the matrix display
    Write(PORTA, 0x00)          #set all columns low
    Write(PORTB, 0x00)          #set all rows low

def TurnOnAllLEDS ():
    #Turn on all 64 LEDs
    Write(PORTA, 0xFF)          #set all columns high
    Write(PORTB, 0x00)          #set all rows low

def WriteToLED (rowPins,colPins):
    #set logic state of LED matrix pins
    Write(PORTA, 0x00)          #turn off all columns; prevent ghosting
    Write(PORTB, rowPins)       #set rows first
    Write(PORTA, colPins)       #now turn on columns


########################################################################
#
#   Intermediate-level routines for
#   LED Pixel, Row, Column, and Pattern display
#

def SetPattern (rowPattern,colPattern,orientation=ORIENTATION):
    #Applies given row & column patterns to the matrix.
    #For columns, bit 0 is left-most and bit 7 is at far right.
    #For rows, bit 0 is at the top and bit 7 is at the bottom.
    #Example: (0x07,0x03) will set 3 row bits & 2 columns bits,
    #formaing a rectagle of  6 lit LEDS in upper left corner of
    #the matrix, three rows tall and two columns wide.
    #Why?  0x07 = 0b00000111 (three lower row bits set).
    #      0x03 = 0b00000011 (two lower column bits set).

    global rows, columns                    #save current row/column
    rows = rowPattern
    columns = colPattern

    if orientation==0:
        WriteToLED(~rows,ReverseBits(columns))
    elif orientation==90:
        WriteToLED(~columns,rows)
    elif orientation==180:
        WriteToLED(~ReverseBits(rows),columns)
    elif orientation==270:
        WriteToLED(~ReverseBits(columns),ReverseBits(rows))

def MultiplexDisplay (z,count):
    # call this routine with Z, the image to display on the matrix
    # Z is a list containing 8 bytes (8x8=64 bits for 64 LEDs)
    # z[0] is data for the top row of the display; z[7] is bottom row.
    # count is number of times to cycle the display
    for count in range(0,count):
        for row in range(0,8):
            SetPattern(1<<row,z[row])       #quickly display each row


########################################################################
#
#   Application Routines
#   Main Program & supporting routines go here.
#

def GetSomeText():
    # Get some text for scrolling on the LEDs.
    #
    # The easiest way is just to prompt for it, like this:
    # st = raw_input("Enter a string to display: ")
    #
    # Instead, let's check the command line.
    # If nothing there, then get from standard input.
    # This allows us to do fancy stuff like pipes & redirects like:
    #
    # ./myscript.py time
    # ./myscript.py "The yellow brown dog"
    # echo "once I pondered weak and weary" | ./myscript.py
    # cat poem.txt | ./myscript.py
    # ./myscript.py <poem.txt

    if len(sys.argv)>1:         #are there any command line args?
        st = sys.argv[1]        #yes, so take first one as text
    else:
        print "Enter text to display, followed by EOF (Ctrl-D)."
        st = sys.stdin.read()   #no, so get standard input instead

    # See if the input was either 'time' or 'date'
    # Display time and date if they were requested.
    if st=="time":
        st = 'The time is ' + time.strftime("%I:%M %p")
    elif st=="date":
        st = 'The date is ' + time.strftime("%A, %b %d, %Y")

    # Add a space to the end of the message - looks better for repeats
    st += ' '
    return st

def CharImage (ch):
    # returns the 8x8 raster image for given character.
    # the result is a list of 8 bytes, 1 byte for each raster row.
    index = ord(ch)
    image = list(font0.data[index])
    return image

def ScrollLeft (z,buf):
    # scrolls image in Z leftward, rotating in data from buf
    # data in Z & buf are both changed by this operation!

    for row in range(0,8):              #for each row in image:
        z[row] >>= 1                    #shift current image row
        if buf[row] & 0x01:             #is lsb of buffer high?
            z[row] |= 0x80              #rotate lsb of buf into msb z
        buf[row] >>= 1                  #shift buffer row, too.

def ScrollText (st):
    # This is the all-important routine!
    # It will scroll the text on the Pi Matrix, left to right.
    # st holds the string that is being scrolled.
    # ch is the character to be scrolled onto the display.
    # buf holds the character image waiting to be scrolled onto display
    # Z holds the character image now being displayed
    # Each character is 8 bits (columns wide), so we will need to scroll
    # left 8 times for each character.  Between each scroll operation,
    # we also need to quickly display (multilplex) each of the 8 rows.

    SPEED = 10                          #scroll speed: lower=faster
    z = [0,0,0,0,0,0,0,0]               #start with a blank display
    for character in st:                #for each character of text
        buf = CharImage(character)      #get next character image
        for shiftCount in range(0,8):   #For each column in image
            MultiplexDisplay(z,SPEED)   #display current image
            ScrollLeft(z,buf)           #scroll display leftward
```

```
#
#   Main Program here.
#

print "Pi Matrix text scroller starting"
bus = smbus.SMBus(1)            #initialize the I2 bus; use '0' for
                                #older Pi boards, '1' for newer ones.
EnableLEDS()                    #initialize the Pi Matrix board
text = GetSomeText()            #get the text
print "displaying '%s'" %text   #echo text to console
ScrollText(text)                #and scroll it!
DisableLEDS()                   #dont leave any LEDs on.
print "Done."
```