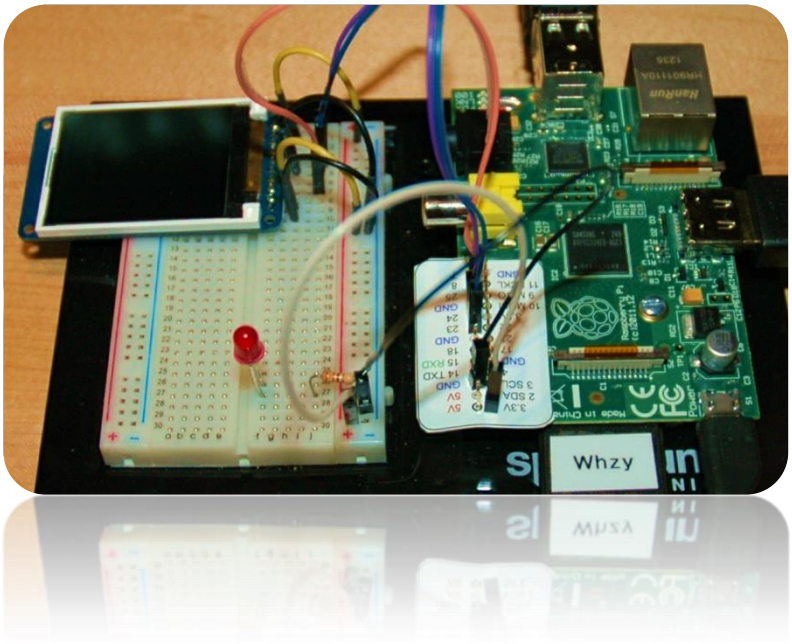


Add a TFT Display to the Raspberry Pi

Part 4: Text

Bruce E. Hall, W8BH

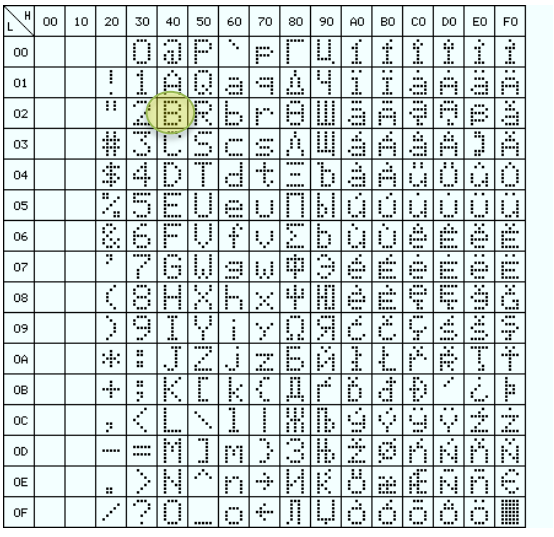
Objective: Send text to a 160x128 pixel TFT LCD module using Python.

1) INTRODUCTION

In Part 1, we connected a 1.8" TFT module from Adafruit to the Raspberry Pi. Part 2 added speed using the hardware SPI interface, and Part 3 added a simple graphics library. Now let's add the ability to send text.

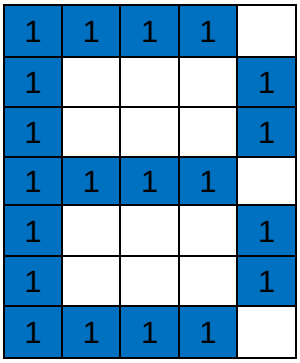
2) TEXT

The TFT controller does not have any command for 'Draw the letter B'. Unlike the 16x2 LCD character display modules, we cannot send it ASCII code 0x42 and expect to see the corresponding 'B' appear on the display. Instead, we must send each pixel dot that forms the B character.



Figuring out each character dot would be tedious work. Fortunately it was done many years ago! Tables like the one shown map out each character on pixel grids of various sizes. The granddaddy of them all is the 5x7 ASCII font, in which each character is (at most) 5 pixels wide and 7 pixels tall.

So how do we make a 'B'?



Here is a 5x7 matrix for an uppercase B. Each matrix pixel will be represented by a bit. We can consider the whole character as a series of 35 individual bits. For simplicity, these bits are usually packaged in 8-bit bytes, with each row (or column) represented by 1 or more bytes.

There are two basic formats for font data. Row Major order means that the bits for the first row come first, then the second row, etc. Using Row Major order, each 5x7 character is a 7 element list of row bytes, with 5 active bits in each byte and 3 bits unused. Each character is requires 7 bytes.

The second format is Column Major order. The bits of the first column are represented first, then the second column, etc. Using column-major order, a 5x7 character is a 5 element list of column bytes, with 7 active bits in each byte and 1 bit unused. Most representations of 5x7 characters use column major order, since it requires only 5 bytes to represent each character.

Our B character is represented by 5 columns, left-most column first. Consider the third column, shown here horizontally, with the top pixel on the left:

There are 3 active pixels in blue. Substituting 1 for blue and 0 for white, the binary value is 0x1001001 or hexadecimal 0x49. The entire character is represented by an list (array) of the five columns, or [0x7F, 0x49, 0x49, 0x49, 0x36]. I 'pythonized' 96 characters from the standard 5x7 ASCII character set above and saved them in the file font5x7.py.

3) DRAW A CHARACTER

Here is the pseudocode for drawing a character:

- Get the 5 bytes of character data from the 5x7 font file
- Set the display window to a 5x7 pixel region
- For each row & column bit,
  - If it's a 0, set the pixel color to the background color
  - If it's a 1, set the pixel color to the foreground color
- Send all 35 pixels to the display

That is fairly straightforward. Here is the code:

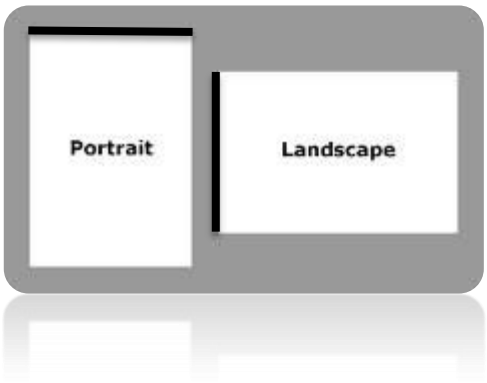
```
def PutCh (ch,xPos,yPos,color=fColor):
  charData = font0.data[ord(ch)-32]          #get the character data
  SetAddrWindow(xPos,yPos,xPos+4,yPos+6)     #set display window to 5x7 rect
  SetPin(DC,0)
  spi.writebytes([RAMWR])                    #pixel data follows
  SetPin(DC,1)
  buf = []                                   #setup buffer for pixel data
  mask = 0x01                                #start with lsb in each column
  for row in range(7):                      #iterate over all 35 pixels
```

```
    for col in range(5):                    #do all columns for each row
      bit = charData[col] & mask            #get the next bit
      if (bit==0):                          #is it a 0 or 1?
        pixel = bColor                      #0: use background color
      else:
        pixel = color                       #1: use foreground color
      buf.append(pixel>>8)                  #add pixel to buffer
      buf.append(pixel&0xFF)
    mask <<= 1                              #go to next bit in column
  spi.writebytes(buf)                       #write all 35 pixels to TFT
```

The font data is stored as a list of 96 characters in variable font0.data. Since the characters are stored in ASCII order, starting with the space character (ASCII 32), the index to each character in the list is: ord(ch)-32. The character data returned into charData is a list of 5 bytes, each byte corresponding to a pixel column for the character.

In my font table, the columns are encoded with the least significant bit *at the top*, so we must lay down row pixel data starting with bit0, then row data for bit1, etc. This can be accomplished by creating a mask byte with bit0 initially set (mask = 0x01), then moving the bit leftward for each successive row (mask <<= 1). Other tables may put the msb at the top instead. In such tables you would start with a mask on bit 7 (mask = 0x80) and move the bit rightward for each successive row (mask >>= 1).

3) PORTRAIT MODE



Finally, it is time to fill our screen with text! How many characters will fit on the display?

The default layout is portrait, with the display connectors along the top of the module. The display width is 128 pixels. Each character is 5 pixels wide. Allowing for one pixel space between characters, we can fit 128/6 = 21 characters per row.

The display height is 160 pixels. Each character is 7 pixels tall. Allowing for one pixel space between rows, we can fit 160/8 = 20 rows of characters.

The total number of characters per screen is therefore 21 chars/row x 20 rows = 420 characters. Here is a routine to display 420 characters on the screen:

```
def PortaitChars():
  ClearScreen()
  charsPerRow = 21
  for i in range(420):
    col = i % charsPerRow
    row = i / charsPerRow
    ascii = (i % 95)+32
    PutCh(chr(ascii),col*6,row*8)
```

3) LANDSCAPE MODE

In the default portrait mode (0°), the display connectors are on the top and the long edges are vertical. In landscape mode (90°), the display connectors are on the left, and the long edges are horizontal. In fact, using the MADCTL command, it is equally easy to have upright text with the display connectors up, down, left or right. Additional "mirror image" displays are also possible, but not quite as useful.

Orientation	Mode Byte
0°	0x00
90°	0x60
180°	0xC0
270°	0xA0

```
def SetOrientation(degrees):
  if   degrees==90:  arg=0x60
  elif degrees==180: arg=0xC0
  elif degrees==270: arg=0xA0
  else:              arg=0x00
  Command (MADCTL,arg)
```

In landscape mode, the display width and height are switched. With 160 pixels per row, the number of characters per row increases from 21 to (160/8) = 26. The number of rows deceases from 20 to (128/8) = 16. The total number of characters per screen is 26x16 = 416.

4) PROPORTIONAL FONTS

You almost need a magnifying glass to see those itty-bitty 5x7 characters! After a while you might want to use larger and/or more interesting fonts. There are some nifty and free utilities online that will convert fonts on your system to their bitmap equivalents. One such utility is "The Dot Factory" by Eran Duchan. I used this utility to convert one of my fonts, **Cooper Black 14 pt**, and saved it to file cooper14.py.

Proportional fonts have varying character widths: a 'W' is much wider than, say, an exclamation point '!'. We cannot assume a fixed size of 5x7. Larger font characters may have widths (and heights) larger than one byte. So the PutCh routine must accommodate these changes.

Here is the pseudocode for drawing a character:

- Get the character height, width, and pixel data
- Set the display window to size: (char width) x (char height)
- For each row:
    - For each bit in the row:
        - If it's a 0, set the pixel color to the background color
        - If it's a 1, set the pixel color to the foreground color
- Send all pixels to the display

Here is the corresponding code. It's a bit long for a single routine. You could break up the nested for loops into separate routines, and improve readability, at the cost of execution speed.

```
def PutChar (ch,xPos,yPos,color=fColor):
  charData = GetCharData(ch)
  xLen = GetCharWidth(ch)                    #char width, in pixels
  numRows = GetCharHeight(ch)
  bytesPerRow = 1+((xLen-1)/8)               #char width, in bytes
  numBytes = numRows*bytesPerRow
  SetAddrWindow(xPos,yPos,xPos+xLen-1,yPos+numRows-1)
  SetPin(DC,0)
  spi.writebytes([RAMWR])                    #pixel data to follow
  SetPin(DC,1)
  index = 0
  buf = []
  for a in range(numRows):                   #row major order
    bitNum = 0                               #count bits in each row
    for b in range(bytesPerRow):             #do whole row
      mask = 0x80                            #look at msb first
      for c in range(8):                     #do all bits in this byte
        if (bitNum<xLen):                    #still within char width?
          bit = charData[index] & mask
          if (bit==0):                       #check the bit
            pixel = bColor                   #0: background color
          else:
            pixel = color                    #1: foreground color
          buf.append(pixel>>8)               #add pixel to buffer
          buf.append(pixel&0xFF)
          mask >>= 1                         #goto next bit in byte
        bitNum += 1                          #goto next bit in row
     index += 1                              #goto next byte of data
  spi.writebytes(buf)                        #send char data to display
```

5) TESTING IT ALL

The code to test these routines is longer than the text routines themselves. The test suite exercises code for both the fixed 5x7 and the cooper14 fonts, and does screen fills in all four orientations. The final test does something a little more useful, displaying the IP address, CPU temperature, and current time.

6) PYTHON SCRIPT for TFT DISPLAY, PART 4:

```
#!/usr/bin/python

#############################################################################
#
#  A Python script for controlling the Adafruit 1.8" TFT LCD module
#  from a Raspbery Pi.
#
#   Author :   Bruce E. Hall, W8BH    <bhall66@gmail.com>
#   Date   :   25 Feb 2014
#
#   This module uses the ST7735 controller and SPI data interface
#   PART 4 --- TEXT
#
#   For more information, see w8bh.net
#
#############################################################################

import cooper14 as font
import font5x7 as font0
import RPi.GPIO as GPIO
import time
import os                       #for popen
import spidev                   #hardware SPI
from random import randint
from math   import sqrt


#TFT to RPi connections
# PIN TFT           RPi
# 1 backlight       3v3
# 2 MISO            <none>
# 3 CLK             SCLK (GPIO 11)
# 4 MOSI            MOSI (GPIO 10)
# 5 CS-TFT          GND
# 6 CS-CARD         <none>
# 7 D/C             GPIO 25
# 8 RESET           <none>
# 9 VCC             3V3
# 10 GND            GND

DC      = 25
XSIZE   = 128
YSIZE   = 160
XMAX    = XSIZE-1
YMAX    = YSIZE-1
X0      = XSIZE/2
Y0      = YSIZE/2

#Color constants
BLACK   = 0x0000
BLUE    = 0x001F
RED     = 0xF800
GREEN   = 0x0400
LIME    = 0x07E0
CYAN    = 0x07FF
MAGENTA = 0xF81F
YELLOW  = 0xFFE0
```

```
WHITE   = 0xFFFF
PURPLE  = 0x8010
NAVY    = 0x0010
TEAL    = 0x0410
OLIVE   = 0x8400
MAROON  = 0x8000
SILVER  = 0xC618
GRAY    = 0x8410

bColor  = BLACK
fColor  = YELLOW

COLORSET  = [BLACK,BLUE,RED,GREEN,LIME,CYAN,MAGENTA,YELLOW,
             WHITE,PURPLE,NAVY,TEAL,OLIVE,MAROON,SILVER,GRAY]

#TFT display constants
SWRESET = 0x01
SLPIN   = 0x10
SLPOUT  = 0x11
PTLON   = 0x12
NORON   = 0x13
INVOFF  = 0x20
INVON   = 0x21
DISPOFF = 0x28
DISPON  = 0x29
CASET   = 0x2A
RASET   = 0x2B
RAMWR   = 0x2C
RAMRD   = 0x2E
PTLAR   = 0x30
MADCTL  = 0x36
COLMOD  = 0x3A


#############################################################################
#
#   Low-level routines
#
#

def SetPin(pinNumber,value):
    #sets the GPIO pin to desired value (1=on,0=off)
    GPIO.output(pinNumber,value)

def InitGPIO():
    GPIO.setmode(GPIO.BCM)
    GPIO.setwarnings(False)
    GPIO.setup(DC,GPIO.OUT)

def InitSPI():
       "returns an opened spi connection to device(0,0) in mode 0"
       spiObject = spidev.SpiDev()
       spiObject.open(0,0)
       spiObject.mode = 0
       return spiObject
```

```
#############################################################################
#
#   ST7735 TFT controller routines:
#
#

def WriteByte(value):
       "sends an 8-bit value to the display as data"
       SetPin(DC,1)
       spi.writebytes([value])

def WriteWord (value):
       "sends a 16-bit value to the display as data"
       SetPin(DC,1)
       spi.writebytes([value>>8, value&0xFF])

def Command(cmd, *bytes):
       "Sends a command followed by any data it requires"
       SetPin(DC,0)                    #command follows
       spi.writebytes([cmd])           #send the command byte
       if len(bytes)>0:                #is there data to follow command?
              SetPin(DC,1)                #data follows
              spi.writebytes(list(bytes)) #send the data bytes

def InitDisplay():
       "Resets & prepares display for active use."
       Command (SWRESET)               #reset TFT controller
       time.sleep(0.2)                 #wait 200mS for controller init
       Command (SLPOUT)                #wake from sleep
       Command (COLMOD, 0x05)          #set color mode to 16 bit
       Command (DISPON)                #turn on display

def SetOrientation(degrees):
       "Set the display orientation to 0,90,180,or 270 degrees"
       if   degrees==90:  arg=0x60
       elif degrees==180: arg=0xC0
       elif degrees==270: arg=0xA0
       else:              arg=0x00
       Command (MADCTL,arg)

def SetAddrWindow(x0,y0,x1,y1):
       "sets a rectangular display window into which pixel data is placed"
       Command (CASET,0,x0,0,x1)       #set column range (x0,x1)
       Command (RASET,0,y0,0,y1)       #set row range (y0,y1)

def WriteBulk (value,reps,count=1):
       "sends a 16-bit pixel word many, many times using hardware SPI"
       "number of writes = reps * count. Value of reps must be <= 2048"
       SetPin(DC,0)                    #command follows
       spi.writebytes([RAMWR])         #issue RAM write command
       SetPin(DC,1)                    #data follows
       valHi = value >> 8              #separate color into two bytes
       valLo = value & 0xFF
       byteArray = [valHi,valLo]*reps  #create buffer of multiple pixels
       for a in range(count):
              spi.writebytes(byteArray)   #send this buffer multiple times

def WritePixels (byteArray):
       "sends pixel data to the TFT"
       SetPin(DC,0)                    #command follows
       spi.writebytes([RAMWR])         #issue RAM write command
       SetPin(DC,1)                    #data follows
       spi.writebytes(byteArray)       #send data to the TFT
```

```
#############################################################################
#
#   Graphics routines:
#
#


def FillRect(x0,y0,x1,y1,color):
       "fills rectangle with given color"
       width = x1-x0+1
       height = y1-y0+1
       SetAddrWindow(x0,y0,x1,y1)
       WriteBulk(color,width,height)

def FillScreen(color):
       "Fills entire screen with given color"
       FillRect(0,0,XMAX,YMAX,color)

def ClearScreen():
       "Fills entire screen with black"
       FillScreen(BLACK)


#############################################################################
#
#   Text routines:
#
#

def GetCharData (ch):
       "Returns array of raster data for a given ASCII character"
       pIndex = ord(ch)-ord(' ')
       lastDescriptor = len(font.descriptor)-1
       charIndex = font.descriptor[pIndex][1]
       if (pIndex >= lastDescriptor):
              return font.rasterData[charIndex:]
       else:
              nextIndex = font.descriptor[pIndex+1][1]
              return font.rasterData[charIndex:nextIndex]

def GetCharWidth(ch):
       "returns the width of a character, in pixels"
       pIndex = ord(ch)-ord(' ')
       return font.descriptor[pIndex][0]

def GetCharHeight(ch):
       "returns the height of a character, in pixels"
       return font.fontInfo[0]

def GetStringWidth(st):
       "returns the width of the text, in pixels"
       width = 0
       for ch in st:
              width += GetCharWidth(ch) + 1
       return width

def PutCh (ch,xPos,yPos,color=fColor):
       "write ch to X,Y coordinates using ASCII 5x7 font"
       charData = font0.data[ord(ch)-32]
       SetAddrWindow(xPos,yPos,xPos+4,yPos+6)
       SetPin(DC,0)
       spi.writebytes([RAMWR])
       SetPin(DC,1)
```

```
        buf = []
        mask = 0x01
        for row in range(7):
                for col in range(5):
                        bit = charData[col] & mask
                        if (bit==0):
                                pixel = bColor
                        else:
                                pixel = color
                        buf.append(pixel>>8)
                        buf.append(pixel&0xFF)
                mask <<= 1
        spi.writebytes(buf)

def PutCharV2 (ch,xPos,yPos,color=fColor):
        "Writes Ch to X,Y coordinates in current foreground color"
        charData = GetCharData(ch)
        xLen = GetCharWidth(ch)
        numRows = GetCharHeight(ch)
        bytesPerRow = 1+((xLen-1)/8)
        numBytes = numRows*bytesPerRow
        SetAddrWindow(xPos,yPos,xPos+xLen-1,yPos+numRows-1)
        SetPin(DC,0)
        spi.writebytes([RAMWR])
        SetPin(DC,1)
        i= 0
        while (i< numBytes):
                mask = 0x01
                rowBits = 0
                for b in range(bytesPerRow):
                        rowBits <<= 8
                        mask    <<= 8
                        rowBits +=  charData[i]
                        i +=  1
                mask >>= 1
                #at this point, all bits for current row should
                #be in rowBits, regardless of number of bytes
                #it takes to represent the row

                rowBuf = []
                for a in range(xLen):
                        bit = rowBits & mask
                        mask >>= 1
                        if (bit==0):
                                pixel = bColor
                        else:
                                pixel = color
                        rowBuf.append(pixel>>8)
                        rowBuf.append(pixel&0xFF)
                spi.writebytes(rowBuf)

def PutChar (ch,xPos,yPos,color=fColor):
        "Writes Ch to X,Y coordinates in current foreground color"
        charData = GetCharData(ch)
        xLen = GetCharWidth(ch)                     #char width, in pixels
        numRows = GetCharHeight(ch)
        bytesPerRow = 1+((xLen-1)/8)                #char width, in bytes
        numBytes = numRows*bytesPerRow
        SetAddrWindow(xPos,yPos,xPos+xLen-1,yPos+numRows-1)
        SetPin(DC,0)
        spi.writebytes([RAMWR])                     #pixel data to follow
        SetPin(DC,1)
        index = 0
```

```
        buf = []
        for a in range(numRows):                    #row major
            bitNum = 0
            for b in range(bytesPerRow):            #do whole row
                mask = 0x80                         #msb first
                for c in range(8):                  #do all bits in this byte
                    if (bitNum<xLen):               #still within char width?
                        bit = charData[index] & mask
                        if (bit==0):                #check the bit
                            pixel = bColor          #0: background color
                        else:
                            pixel = color           #1: foreground color
                        buf.append(pixel>>8)        #add pixel to buffer
                        buf.append(pixel&0xFF)
                        mask >>= 1                  #goto next bit in byte
                    bitNum += 1                     #goto next bit in row
                index += 1                          #goto next byte of data
        spi.writebytes(buf)                         #send char data to TFT

def PutString(xPos,yPos,st,color=fColor):
        "Draws string on display at xPos,yPos."
        "Does NOT check to see if it fits!"
        for ch in st:
                width = GetCharWidth(ch)+1
                PutChar(ch,xPos,yPos,color)
                xPos += width


#########################################################################
#
#   Testing routines:
#
#

def PrintElapsedTime(function,startTime):
        "Formats an output string showing elapsed time since function start"
        elapsedTime=time.time()-startTime
        print "%15s: %8.3f seconds" % (function,elapsedTime)
        time.sleep(1)

def ColorBars():
        "Fill Screen with 8 color bars"
        for a in range(8):
                FillRect(0,a*20,XMAX,a*20+19,COLORSET[a+1])

def ScreenTest():
        "Measures time required to fill display twice"
        startTime=time.time()
        FillScreen(LIME)
        FillScreen(MAGENTA)
        ColorBars()
        PrintElapsedTime('ScreenTest',startTime)


def PortaitChars():
        "Writes 420 characters (5x7) to screen in Portrait Mode"
        #font is 5x7 with 1 pixel spacing
        #so character width = 6 pixels, height = 8 pixels
        #display width = 128 pixels, so 21 char per row (21x6 = 126)
        #display ht = 160 pixels, so 20 rows (20x8 = 160)
        #total number of characters = 21 x 20 = 420

        CHARS_PER_ROW = 21
```

```
        FillRect(0,0,XMAX,YMAX,bColor)  #clear screen
        for i in range(420):
                x= i % CHARS_PER_ROW
                y= i / CHARS_PER_ROW
                ascii = (i % 96)+32
                PutCh(chr(ascii),x*6,y*8)
        time.sleep(1)

def LandscapeChars():
        "Writes 416 characters (5x7) to screen, landscape mode"
        #font is 5x7 with 1 pixel spacing
        #so character width = 6 pixels, height = 8 pixels
        #display width = 160 pixels, so 26 char per row (26x6 = 156)
        #display ht = 128 pixels, so 16 rows (16x8 = 128)
        #total number of characters = 26 x 16 = 416

        CHARS_PER_ROW = 26
        FillRect(0,0,YMAX,XMAX,bColor)  #clear screen
        for i in range(416):
                x= i % CHARS_PER_ROW
                y= i / CHARS_PER_ROW
                ascii = (i % 96)+32
                PutCh(chr(ascii),x*6,y*8,CYAN)
        time.sleep(1)

def LargeFontTest():
        "Writes 90 characters (11x17) to the screen"
        title = 'Large Font'
        startTime = time.time()
        for i in range(90):
                x= i % 10
                y= i / 10
                ascii = (i % 96)+32
                PutChar(chr(ascii),x*12,y*18,LIME)
        PrintElapsedTime(title,startTime)

def RandColor():
        "Returns a random color from BGR565 Colorspace"
        index = randint(0,len(COLORSET)-1)
        return COLORSET[index]

def SmallFontTest():
        "Writes 2000 random 5x7 characters to the screen"
        title = 'Small Font'
        startTime = time.time()
        for i in range(2000):
                x= randint(0,20)
                y= randint(0,19)
                color = RandColor()
                ascii = (i % 96)+32
                PutCh(chr(ascii),x*6,y*8,color)
        PrintElapsedTime(title,startTime)

def OrientationTest():
        "Write 5x7 characters at 0,90,180,270 deg orientations"
        title = 'Orientation'
        startTime = time.time()
        PortaitChars()
        SetOrientation(90)              #display-top on right
        LandscapeChars()
        SetOrientation(180)             #upside-down
        PortaitChars()
        SetOrientation(270)             #display-top on left
```

```
        LandscapeChars()
        SetOrientation(0)               #return to 0 deg.
        PrintElapsedTime(title,startTime)

def GetTempCPU():
        "Returns CPU temp in degrees F"
        tPath = '/sys/class/thermal/thermal_zone0/temp'
        tFile = open(tPath)
        temp = tFile.read()
        tFile.close()
        return (float(temp)*0.0018 + 32)

def Run(cmd):
        "Runs a system (bash) command"
        return os.popen(cmd).read()

def GetIPAddr():
        "Returns IP address as a string"
        cmd = "ifconfig | awk '/192/ {print $2}'"
        res = Run(cmd).replace('\n','') #remove end of line char
        return res.replace('addr:','')  #remove 'addr:' prefix

def InfoTest():
        "Show IP address, CPU temp, and Time"
        title = 'Info'
        startTime = time.time()
        PutString(0,0,'IP addr',WHITE)
        PutString(0,20,GetIPAddr())
        PutString(0,60,'CPU temp',WHITE)
        temp = GetTempCPU()
        PutString(0,80,'{:5.1f} deg F'.format(temp))
        tStr = time.strftime("%I:%M:%S  ")
        PutString(0,120,'Time',WHITE)
        PutString(0,140,tStr)
        PrintElapsedTime(title,startTime)

def RunTextTests():
        startTime = time.time()         #keep track of test duration
        ScreenTest()
        LargeFontTest()
        SmallFontTest()
        OrientationTest()
        InfoTest()
        PrintElapsedTime('Full Suite',startTime)



#########################################################################
#
#   Main Program
#

print "Adafruit 1.8 TFT display demo with hardware SPI"
spi = InitSPI()                     #initialize SPI interface
InitGPIO()                          #initialize GPIO interface
InitDisplay()                       #initialize TFT controller
RunTextTests()                      #run suite of text tests
spi.close()                         #close down SPI interface
print "Done."


#   END  ###############################################################
```