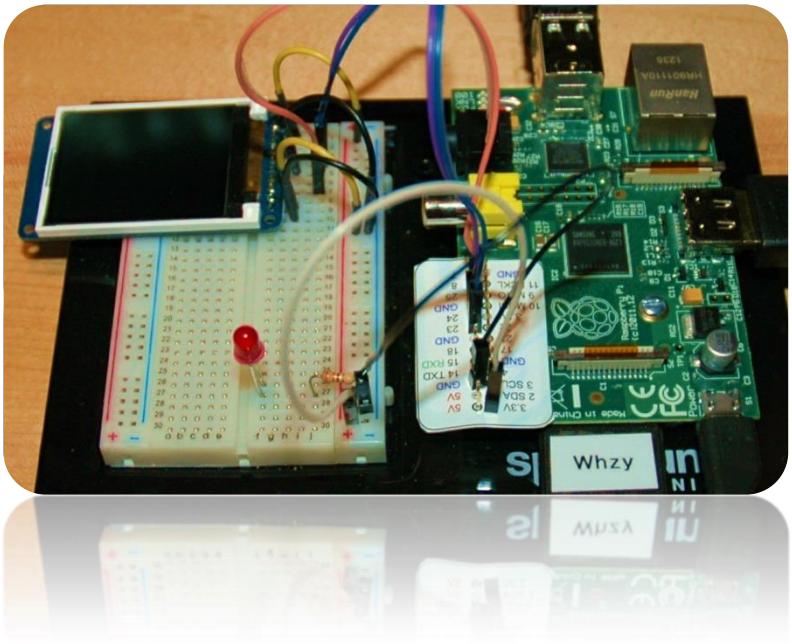


Add a TFT Display to the Raspberry Pi

Part 3: Graphics

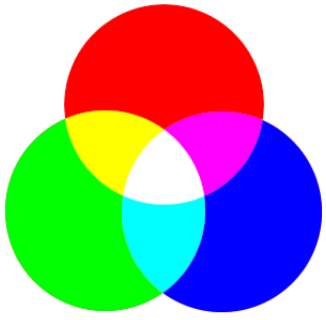
Bruce E. Hall, W8BH

Objective: Draw simple graphics on 160x128 pixel TFT LCD module using Python.

1) INTRODUCTION

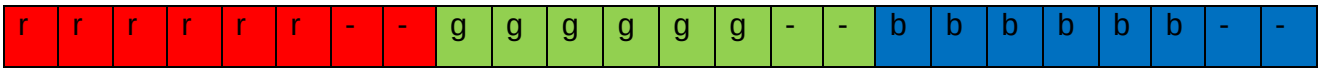
In Part 1, we connected a 1.8" TFT module from Adafruit to the Raspberry Pi. Part 2 added speed using the hardware SPI interface. Now it's time to build a simple graphics library in Python. Please refer to Parts 1 and 2 for instructions on how to connect your TFT device to the Raspberry Pi.

2) DISPLAY COLOR MODES



The default color mode for this controller is RGB666. Pixel colors are a combination of red, green, and blue color values. Each subcolor has 6 bits (64 different levels) of intensity. Equal amounts of red, green, and blue light produce white. Equal amounts of blue and green produce cyan. Red and green make yellow. Since each color component is specified by 6 bits, the final color value is 18 bits in length. The number of possible color combinations in RGB666 colorspace is 2^18 = 262,144 or 256K.

We represent these color combinations as an 18 bit binary number. The 6 red bits are first, followed by 6 green bits, followed by 6 blue bits. Our controller wants to see data in byte-sized chucks, however. For every pixel we must send 24 bits (3 bytes), arranged as follows:

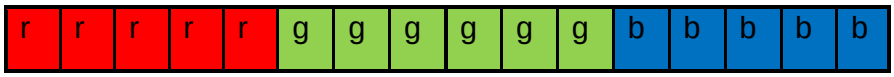


The lowest two bits of each red, green, and blue byte are ignored; only the 6 upper bits of each byte are used. That's a bit wasteful, isn't it? It takes time to send those empty bits.

The TFT controller supports three different color depths. Here they are:

COLOR MODE	RGB SPACE	Bits per Pixel	Unique Colors
3	RGB444	12	16x16x16 = 4K
5	RGB565	16	32x64x32 = 64K
6 (default)	RGB666	18	64x64x64 = 256K

Mode 6 is the default, requiring three bytes per pixel. Notice that Mode 5 is 16 bits in size, which is exactly 2 bytes in length. No waste! And it still provides for plenty of colors. If we use mode 5 instead of mode 6, each pixel can be sent in 2 bytes instead of 3. Data transmission will be faster, at the cost of less color depth. If 65,536 different colors are enough for you, this is a good tradeoff. Let's use it. Here is how the red, green, and blue bits are packed into a 2-byte word:



To use a 16-bit color depth instead of the old 18-bit one, the pixel writing routine must be modified. Compare the old Write888 routine with the new WriteBulk routine:

```
def Write888(value,reps,count=1):
      "sends a 24-bit RGB pixel data to display, with optional repeat"
      red = value>>16                  #red = upper 8 bits
      green = (value>>8) & 0xFF        #green = middle 8 bits
      blue = value & 0xFF              #blue = lower 8 bits
      RGB  = [red,green,blue]          #assemble RGB as 3 byte list
      SetPin(DC,1)                     #data follows
      for a in range(count):
            spi.writebytes(RGB*reps)

def WriteBulk (value,reps,count=1):
      "sends a 16-bit pixel word to display, with optional repeat"
      valHi = value >> 8               #each pixel is two bytes
      valLo = value & 0xFF
      RGB = [valHi,valLo]              #assemble RGB as 2 byte list
      SetPin(DC,1)                     #data follows
      for a in range(count):
            spi.writebytes(RGB*reps)
```

There are 128x160 = 20,480 pixels on the TFT screen. Clearing the screen using the old routine would require 20,480 x 3 = 61,440 bytes to be sent to the TFT controller. The new routine requires only 40,960 bytes. Each screen refresh is 33% faster.

3) PIXELS

In Part 1 the DrawPixel routine was introduced. Drawing single pixels is the basis for other graphics routines. This routine should be as efficient as possible. Here is an updated version of the original code:

```
def DrawPixel(x,y,color):
  "draws a pixel on the TFT display"
  SetAddrWindow(x,y,x,y)                  #set display window to x,y
  Command (RAMWR, color>>8, color&0xFF)   #send pixel color (2 bytes)
```

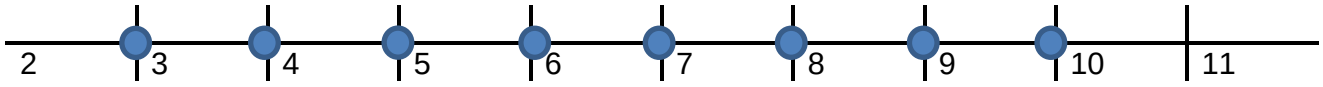
There isn't much to simplify: set the drawing window to a single x,y location, and send the pixel color as two bytes. We can make it a little more efficient, though, by inlining the code. Python, as an interpreted language, uses processing time for each procedure call. Inlining removes as many procedure calls as possible. Compare the above code with the following:

```
def FastDrawPixel(x,y,color):
  "draws a pixel on the TFT display; increases speed by inlining"
  GPIO.output(DC,0)                       #set display window to x,y
  spi.writebytes([CASET])
  GPIO.output(DC,1)
  spi.writebytes([0,x,0,x])
  GPIO.output(DC,0)
  spi.writebytes([RASET])
  GPIO.output(DC,1)
  spi.writebytes([0,y,0,y])
  GPIO.output(DC,0)                       #send pixel color (2 bytes)
  spi.writebytes([RAMWR])
  GPIO.output(DC,1)
  spi.writebytes([color>>8, color&0xFF])
```

Notice how the routine has been reduced to a series of low-level IO calls. In testing, this routine is 10-20% faster – despite increasing the procedure length from 2 to 12 statements. The tradeoff is reduced code readability.

4) LINES

The simplest line to draw is a horizontal line along the x axis. For example, from x=3 to x=10. The length of this line is the difference, 10-3 = 7. Add one pixel so that the start-pixel and end-pixel are both included. The total length is 8 pixels.



To code in python, set up a display window between the two positions on the x axis:

```
def HLine (x0,x1,y,color):
  "draws a horizontal line in given color"
  length = x1-x0+1
  SetAddrWindow(x0,y,x1,y)
  WriteBulk(color,length)
```

Vertical lines are exactly the same, swapping the x and y axes.

```
def VLine (x,y0,y1,color):
  "draws a vertical line in given color"
  height = y1-y0+1
  SetAddrWindow(x,y0,x,y1)
  WriteBulk(color,height)
```

For all other lines we can use the high-school algebra formula, y=mx+b, where m is the slope of the line. And the slope is Δy/Δx, or (y1-y0)/(x1-x0). The value of b, the y-intercept, equals y0 if we substitute x-x0 for x. (Graphically, this is equivalent to moving the line leftward by x0 pixels.) The following python code should work:

```
slope = float(y1-y0)/(x1-x0)
for x in range(x0,x1+1):
  y = slope*(x-x0) + y0
  DrawPixel(x,int(y),color)
```

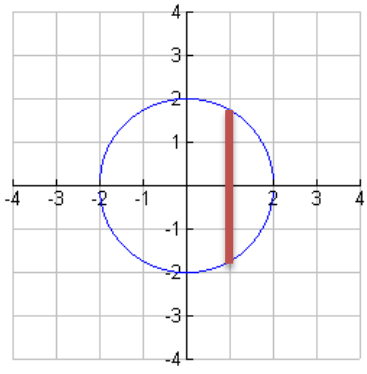
In the FOR loop, y is calculated for each x, and the pixel is drawn at x,y. Try it! You will find that it works well for lines with a shallow slope (m<1). But for steeper lines, y changes more rapidly than x, and pixels will get 'dropped' if we are counting by x units. One solution for steeper lines is to count by y units instead of x, like this:

```
if (slope>1):
  for y in range(y0,y1+1):
    x = (y-y0)/slope + x0
    DrawPixel(int(x),y,color)
```

In the source code I combined these into a single Line() routine.

5) CIRCLES

The algebraic equation for a circle is $x^2 + y^2 = r^2$, where r is the radius. It's easiest to think of the center of circle at the (0,0) origin. For example, here is the graph of $x^2 + y^2 = 4$. The radius, the distance from the origin, is 2.



To create a filled circle, let's do the right half of the circle first, starting from x=0 and ending with x=2. For each x value we will draw a vertical line from +y to –y, shown as a red line on the graph, where y is calculated from the circle equation. The following Python code will work:

```
r2 = radius * radius
for x in range(radius):
  y = int(sqrt(r2-x*x))
  VLine(x,y,-y,color)
```

This code creates the right half of the circle. Fortunately the left half of the circle is a mirror image. All we need to add is a second vertical line at –x for each +x:

```
r2 = radius * radius
for x in range(radius):
  y = int(sqrt(r2-x*x))
  VLine(x,y,-y,color)
  VLine(-x,y,-y,color)
```

Finally, we must translate this circle from (0,0) to any arbitrary xPos,yPos position. To do this, substute (xPos+x) for x and (yPos+y) for y in the VLine calls. See the source code for the completed FillCircle routine.

To create non-filled circles, use DrawPixel instead of VLine, drawing only the 'ends' of the line:

```
r2 = radius * radius
for x in range(radius):
  y = int(sqrt(r2-x*x))
  DrawPixel(x,y,color)        #Vline(x,y,-y,color)
  DrawPixel(x,-y,color)
  DrawPixel(-x,y,color)       #Vline(-x,y,-y,color)
  DrawPixel(-x,-y,color)
```

You can half the number of square root calculations, exploiting the symmetry between the top and bottom halves of the circle. But it doubles the number of DrawPixel calls. Which is faster? See the source code for the completed Circle() routine.

6) TESTING IT ALL

The code to graphically test these routines is longer than the routines themselves. The test suite contains 9 different tests for screen fills, pixels, lines, and circles. Each test takes 0-10 seconds, for a total runtime of about 60 seconds. Each test will vary slightly in length from run to run, because we are running in a multitasking environment. Linux preempts our code from time to time. Nevertheless, you can modify the code and immediately see if it becomes more or less efficient. You can find ways to improve the simple code in this graphics library. Have fun!

In Part 4 of this series we'll add text to our display.

7) PYTHON SCRIPT for TFT DISPLAY, PART 3:

```
#!/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   :   21 Feb 2014
#
#   This module uses the ST7735 controller and SPI data interface
#   PART 3 --- GRAPHICS
#
#   For more information, see w8bh.net
#
#############################################################################


import RPi.GPIO as GPIO
import time
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

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
FRMCT1  = 0xB1
FRMCT2  = 0xB2
FRMCT3  = 0xB3
INVCTR  = 0xB4
DISSET  = 0xB6
PWRCT1  = 0xC0
PWRCT2  = 0xC1
PWRCT3  = 0xC2
PWRCT4  = 0xC3
PWRCT5  = 0xC4
VMCTR1  = 0xC5
PWRCT6  = 0xFC
GAMCTP  = 0xE0
GAMCTN  = 0xE1


#############################################################################
#
#   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 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                  #each pixel is 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


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

def DrawPixel(x,y,color):
        "draws a pixel on the TFT display"
        SetAddrWindow(x,y,x,y)
        Command (RAMWR, color>>8, color&0xFF)
```

```
def FastDrawPixel(x,y,color):
	"draws a pixel on the TFT display; increases speed by inlining"
	GPIO.output(DC,0)
	spi.writebytes([CASET])
	GPIO.output(DC,1)
	spi.writebytes([0,x,0,x])
	GPIO.output(DC,0)
	spi.writebytes([RASET])
	GPIO.output(DC,1)
	spi.writebytes([0,y,0,y])
	GPIO.output(DC,0)
	spi.writebytes([RAMWR])
	GPIO.output(DC,1)
	spi.writebytes([color>>8, color&0xFF])

def HLine (x0,x1,y,color):
	"draws a horizontal line in given color"
	width = x1-x0+1
	SetAddrWindow(x0,y,x1,y)
	WriteBulk(color,width)

def VLine (x,y0,y1,color):
	"draws a verticle line in given color"
	height = y1-y0+1
	SetAddrWindow(x,y0,x,y1)
	WriteBulk(color,height)

def Line (x0,y0,x1,y1,color):
	"draws a line in given color"
	if (x0==x1):
		VLine(x0,y0,y1,color)
	elif (y0==y1):
		HLine(x0,x1,y0,color)
	else:
		slope = float(y1-y0)/(x1-x0)
		if (abs(slope)< 1):
			for x in range(x0,x1+1):
				y = (x-x0)*slope + y0
				FastDrawPixel(x,int(y+0.5),color)
		else:
			for y in range(y0,y1+1):
				x = (y-y0)/slope + x0
				FastDrawPixel(int(x+0.5),y,color)

def DrawRect(x0,y0,x1,y1,color):
	"Draws a rectangle in specified color"
	HLine(x0,x1,y0,color)
	HLine(x0,x1,y1,color)
	VLine(x0,y0,y1,color)
	VLine(x1,y0,y1,color)

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(0,0,127,159,color)
```

```
def ClearScreen():
	"Fills entire screen with black"
	FillRect(0,0,127,159,BLACK)

def Circle(xPos,yPos,radius,color):
	"draws circle at x,y with given radius & color"
	xEnd = int(0.7071*radius)+1
	for x in range(xEnd):
		y = int(sqrt(radius*radius - x*x))
		FastDrawPixel(xPos+x,yPos+y,color)
		FastDrawPixel(xPos+x,yPos-y,color)
		FastDrawPixel(xPos-x,yPos+y,color)
		FastDrawPixel(xPos-x,yPos-y,color)
		FastDrawPixel(xPos+y,yPos+x,color)
		FastDrawPixel(xPos+y,yPos-x,color)
		FastDrawPixel(xPos-y,yPos+x,color)
		FastDrawPixel(xPos-y,yPos-x,color)

def FillCircle(xPos,yPos,radius,color):
	"draws filled circle at x,y with given radius & color"
	r2 = radius * radius
	for x in range(radius):
		y = int(sqrt(r2-x*x))
		y0 = yPos-y
		y1 = yPos+y
		VLine(xPos+x,y0,y1,color)
		VLine(xPos-x,y0,y1,color)

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

def RandRect():
	"Returns four integers x0,y0,x1,y1 as screen rect coordinates"
	x1 = randint(1,100)
	y1 = randint(1,150)
	dx = randint(30,80)
	dy = randint(30,80)
	x2 = x1 + dx
	if x2>126:
		x2 = 126
	y2 = y1 + dy
	if y2>158:
		y2 = 158
	return x1,y1,x2,y2

def RandColor():
	"Returns a random color from BGR565 Colorspace"
```

```
        index = randint(0,len(COLORSET)-1)
        return COLORSET[index]

def RectTest(numCycles=50):
        "Draws a series of random open rectangles"
        ClearScreen()
        startTime = time.time()
        for a in range(numCycles):
                x0,y0,x1,y1 = RandRect()
                DrawRect(x0,y0,x1,y1,RandColor())
        PrintElapsedTime('RectTest',startTime)

def FillRectTest(numCycles=70):
        "draws random filled rectangles on the display"
        startTime=time.time()
        ClearScreen()
        for a in range(numCycles):
                x0,y0,x1,y1=RandRect()
                FillRect(x0,y0,x1,y1,RandColor())
        PrintElapsedTime('FillRect',startTime)

def LineTest(numCycles=50):
        "Draw a series of semi-random lines on display"
        ClearScreen()
        startTime=time.time()
        for a in range(numCycles):
                Line(10,10,randint(20,126),randint(20,158),YELLOW)
                Line(120,10,randint(2,126),randint(10,158),CYAN)
        PrintElapsedTime('LineTest',startTime)

def PixelTest(color=BLACK, numPixels=5000):
        "Writes random pixels to the screen"
        ClearScreen()
        startTime = time.time()
        for i in range(numPixels):
                xPos = randint(1,127)
                yPos = randint(1,159)
                DrawPixel(xPos,yPos,LIME)
        PrintElapsedTime('PixelTest',startTime)

def FastPixelTest(color=BLACK, numPixels=5000):
        "Writes random pixels to the screen"
        ClearScreen()
        startTime = time.time()
        for i in range(numPixels):
                xPos = randint(1,127)
                yPos = randint(1,159)
                FastDrawPixel(xPos,yPos,YELLOW)
        PrintElapsedTime('FastPixelTest',startTime)

def MoireTest():
        "Draws a series of concentric circles"
        ClearScreen()
        startTime = time.time()
        for radius in range(6,60,2):
                Circle(X0,Y0,radius,YELLOW)
        PrintElapsedTime('MoireTest',startTime)

def CircleTest(numCycles=40):
        "draw a series of random circles"
        ClearScreen()
        startTime = time.time()
        for a in range(numCycles):
```

```
                x = randint(30,90)
                y = randint(30,130)
                radius = randint(10,40)
                Circle(x,y,radius,RandColor())
        PrintElapsedTime('CircleTest',startTime)

def FillCircleTest(numCycles=40):
        "draw a series of random filled circles"
        ClearScreen()
        startTime = time.time()
        for a in range(numCycles):
                x = randint(30,90)
                y = randint(30,130)
                radius = randint(10,40)
                FillCircle(x,y,radius,RandColor())
        PrintElapsedTime('FillCircleTest',startTime)

def RunTests():
        "run a series of graphics test routines & time them"
        startTime = time.time()           #keep track of test duration
        ScreenTest()                      #fill entire screen with color
        RectTest()                        #draw rectangles
        FillRectTest()                    #draw filled rectangles
        PixelTest()                       #draw 5000 random pixels
        FastPixelTest()                   #same as above, w/ modified routine
        LineTest()                        #draw straight lines
        MoireTest()                       #draw concentric circles
        CircleTest()                      #draw random circles
        FillCircleTest()                  #draw filled circles
        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
RunTests()                        #run suite of graphics tests
spi.close()                       #close down SPI interface
print "Done."


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