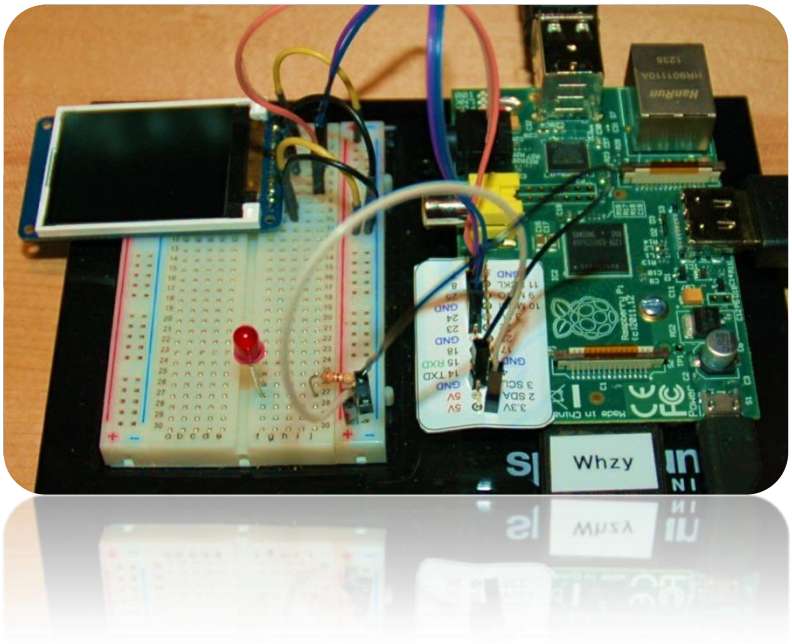

Add a TFT Display to the Raspberry Pi

Part 2: Hardware SPI

Bruce E. Hall, W8BH

Objective: Control a 160x128 pixel TFT LCD module using hardware SPI and Python.

1) INTRODUCTION

In Part 1, we connected a 1.8” TFT module from Adafruit to the Raspberry Pi. We implemented the SPI interface in software, using three of the Pi’s GPIO pins. This bit-ganging method is a great way to learn about SPI, but it is maddeningly slow. It takes about 20 seconds just to clear the screen. There must be a better way.

The Raspberry PI is already equipped with an SPI interface, which is much faster. Let’s use it!

2) ENABLE SPI ON YOUR PI

By default, the hardware SPI is disabled. To enable it, start a recent version of raspi-config and select advanced options -> SPI -> enable.

```
$ sudo raspi-config
```

Reboot, and the spi module will be loaded into the Linux kernel. To confirm that SPI is loaded and enabled, run the ‘listmodules’ command:

```
$ lsmod
```

SPI is loaded if you see an entry for spi_bcm2708. You could also use the following list command:

```
$ls /dev/spidev*
```

When SPI is running this should return two device files: /dev/spidev0.0 and /dev/spidev0.1. You can now use SPI from C. If you want to use it from Python, however, you'll need to install a wrapper module called py-spidev. Enter the following commands to obtain & install spidev:

$sudo apt-get update
$sudo apt-get install python-dev
$git clone git://github.com/doceme/py-spidev
$cd py-spidev
$sudo python setup.py install

Now SPI is enabled, and the python spidev wrapper is installed.

3) CONNECT THE HARDWARE

The Adafruit module has 10 pins. On the bottom of the module each pin is labeled, from pin 1 'Lite' to pin 10 'Gnd'. Mount the display module on a breadboard and connect the pins to your Raspberry Pi GPIO ports. Male to female prototyping wires are very handy for making these point-to-point connections. Alternatively, you can bring out all of the GPIO lines to the breadboard with various third-party cables.

Here are the required connections. First, apply 3.3V power to the backlight, pin 1 and ground to pin 10. You should see the glow of the backlight when you do this. If not, check your power connections before proceeding further.

Next, connect Vcc to 3v3 and the TFT select line to Gnd.

Finally, hook up the three data lines: SCLK, MOSI, and DC. The only change from our software SPI setup in Part 1 is the use of the hardware SPI port (GPIO 10 & 11).

TFT pin	Function	RPi GPIO
1	Backlight	3v3
2	MISO	
3	SCK	SLCK (GPIO 11)
4	MOSI	MOSI (GPIO 10)
5	TFT select	Gnd
6	SD select	
7	D/C	GPIO 25
8	Reset	
9	Vcc	3v3
10	Gnd	Gnd

4) USING SPIDEV IN PYTHON

Just a few lines of code are needed to access the SPI interface from Python. The following lines open, use, and close an SPI object:

```
import spidev                       #include the spidev module
spi = spidev.SpiDev()               #instantiate a spidev object
spi.mode = 0                        #transfer modes 0-3, determined by device
spi.max_speed_hz = 20000000         #set maximum data transfer speed to 20 MHz
spi.open(0,0)                       #connect object to SPI (bus#,device#)

spi.writebytes([byteList])          #output a list of bytes to SPI device

spi.close()                         #disconnect object from SPI interface
```

Setting mode and transfer speed is not always necessary. In this application, data transfer is unidirectional over the MOSI line, from the Pi to the TFT. The routines for unidirectional data transfer are writebytes and readbytes. If your device needs bidirectional data transfer, use spi.xfer2[data] instead. Up to 4K bytes per command can be transferred.

5) MODIFYING THE CODE FOR SPIDEV

Make a copy of the code from Part 1. The first modification is to add spidev. At the top of the program, add the line:

```
import spidev
```

And at the end, change the main program to the following:

```
print "Adafruit 1.8 TFT display demo with Hardware SPI"
spi = spidev.SpiDev()
spi.open(0,0)
spi.mode = 0
InitIO()
InitDisplay()
spi.close()
print "Done."
```

Finally, modify the WriteByte routine to call the hardware SPI instead of the software SPI:

```
def WriteByte(value, data=True):
  SetPin(DC,data)
  spi.writebyte([value])
```

Run the modified program. The TFT screen should fill with green color and then clear, just like before, but faster. On my system, the total time decreased from 46 seconds to 17 seconds. Not bad! But we can do better.

6) OPTIMIZE

When sending data to the TFT, most of the time we are calling the Write888 routine, which sends data for a single RGB pixel as list of three bytes. Each byte gets sent, one at a time, via our new WriteByte routine. We can reduce the number of procedure calls by bypassing WriteByte and directly calling spi.writebytes with our RGB list. Modify the Write888 routine with the following lines:

```
def Write888(value, reps=1):
  …                                    #no change to first few lines
  RGB = [red,green,blue]
  SetPin(DC,1)                         #data follows
  for a in range(reps):
    spi.writebytes(RGB)                #original was "WriteList(RGB)"
```

Run the program again. The screen action should be much faster now. On my system, the total time has decreased to 6 seconds. That's about 7x faster than software SPI!

6) OPTIMIZE AGAIN

The speed of the data transfer is significantly improved by a) reducing the number of procedure calls; and b) putting more data into the transfer buffer before evoking the SPI transfer. Instead of sending individual pixels, let's try sending a whole line of pixels at a time.

```
def Write888(value, width, reps):       #add width = # pixels per transfer
  …                                     #no change to first few lines
  RGB = [red,green,blue]
  SetPin(DC,1)                          #data follows
  for a in range(reps):
    spi.writebytes(RGB*width)           #transfer multiple pixels!


def FillRect(x0,y0,x1,y1,color):        #need small modification to this routine
  …                                     #no change to first few lines
  Write888(color,width,height)          #call modified '888 routine above
```

Run the program with the above modifications. Finally we have some snappy screen action. The runtime has decreased to 2.3 seconds, for a 20-fold improvement overall.

In Part 3 of this series we'll speed up even more, and introduce some graphics routines.

8) PYTHON SCRIPT for TFT DISPLAY, PART 2:

```
#!/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   :   19 Feb 2014
#
#   This module uses the ST7735 controller and SPI data interface
#   PART 2 --- HARDWARE SPI
#
#   For more information, see w8bh.net
#
#######################################################################


import RPi.GPIO as GPIO
import time
import spidev                   #hardware SPI


#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

#RGB888 Color constants
BLACK   = 0x000000
RED     = 0xFF0000
GREEN   = 0x00FF00
BLUE    = 0x0000FF
WHITE   = 0xFFFFFF
COLORSET = [RED,GREEN,BLUE,WHITE]


#ST7735 commands
SWRESET = 0x01                  #software reset
SLPOUT  = 0x11                  #sleep out
DISPON  = 0x29                  #display on
CASET   = 0x2A                  #column address set
RASET   = 0x2B                  #row address set
RAMWR   = 0x2C                  #RAM write
MADCTL  = 0x36                  #axis control
COLMOD  = 0x3A                  #color mode


#######################################################################
#
```

```
#   Low-level routines
#   These routines access GPIO directly
#

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

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


#########################################################################
#
#   Hardware SPI routines:
#
#

def WriteByte(value, data=True):
       SetPin(DC,data)
       spi.writebytes([value])

def WriteCmd(value):
       "Send command byte to display"
       WriteByte(value,False)                   #set D/C line to 0 = command

def WriteWord (value):
       "sends a 16-bit word to the display as data"
       WriteByte(value >> 8)                    #write upper 8 bits
       WriteByte(value & 0xFF)                  #write lower 8 bits

def WriteList (byteList):
       "Send list of bytes to display, as data"
       for byte in byteList:                    #grab each byte in list
              WriteByte(byte)                   #and send it

def Write888(value,width,count):
       "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)
       for a in range(count):
              spi.writebytes(RGB*width)


#########################################################################
#
#   ST7735 driver routines:
#
#

def InitDisplay():
       "Resets & prepares display for active use."
       WriteCmd  (SWRESET)        #software reset, puts display into sleep
       time.sleep(0.2)            #wait 200mS for controller register init
       WriteCmd  (SLPOUT)         #sleep out
       time.sleep(0.2)            #wait 200mS for TFT driver circuits
       WriteCmd  (DISPON)         #display on!
```

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

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)
       WriteCmd(RAMWR)
       Write888(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)


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

def TimeDisplay():
       "Measures time required to fill display twice"
       startTime=time.time()
       print "  Now painting screen GREEN"
       FillScreen(GREEN)
       print "  Now clearing screen"
       ClearScreen()
       elapsedTime=time.time()-startTime
       print "  Elapsed time %0.1f seconds" % (elapsedTime)


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

print "Adafruit 1.8 TFT display demo with hardware SPI"
spi = spidev.SpiDev()
spi.open(0,0)
spi.mode = 0
InitIO()
InitDisplay()
TimeDisplay()
spi.close()
print "Done."


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