Driving 2.8” SPI Display v1.2 Boards With a Pi Zero and Python

Topics Covered:

- Connecting to a Pi Zero via usb ethernet gadget
- Python, CircuitPython, and Micropython
- Running a script
- Using CircuitPython to set up GPIO pins
- SPI – 10 mile overview
- Ili9341 – 10 mile overview
- Ili9341 SPI protocol
- Initializing the Display
- RGB565 Pixel Format and worksheet
- MADCTL, CASET, PASET and worksheet
- Creating and sending data to the screen
- Pi SPI gotchas
- Animation Overview, tearing, lack of vblank signal
- Xpt2046 – Overview
- Reading the touch controller in 8 bit resolution
- Converting values to screen location and Worksheet
- Combining the two
- Using Circuitpython libraries

Bonus Topic:

Bitmapped fonts and how to use them

Things to know in advance

Hexadecimal, Decimal, and Binary

It's impossible to write your own embedded code without understanding how numbers are represented in a computer. I will show how to use the windows calculator to do conversion from normal (base 10) to binary and hexadecimal. I recommend the BBC's lessons if you would like to be more prepared:

Units of information - Fundamentals of data representation - AQA - GCSE Computer Science Revision - AQA - BBC Bitesize

Specialty embedded chips like the ili9341 "understand" numbers that encode commands. The reason we can talk to our desktops and microcontrollers with words (for, if, etc) is that they have the power and time to make translations. When we talk to small chips and devices, they don't have the power to understand our language. Typically our hexadecimal values get turned into high and low values on electrical wires, and it needs to happen almost instantly. The reason why different display controllers have different numbers for the same command is because of the ordering of those lines.

I provide pre-encoded numeric values/commands. While it isn't necessary to ever understand where they came from, understanding how commands are encoded from binary to hex is necessary for understanding the data sheets, and other people's code. We will look at this when using the MADCTL command to set screen orientation. Tutorials on bitfields in python will be helpful to understand

RGB565 Worksheet

Don't panic – windows calculator does the conversions!

RGB565 has the maximums for each color in decimal- (31,63,31)

RGB565 values consist of 5 bits of red, 6 bits of green, and 5 bits of blue. Hexadecimal encodes four bits at a time. We can easily encode multiples of four in hexadecimal – 4, 8, 12, 16:

Decimal: 27,759

Decimal: 0110 1100 0110 1111

Hex: 6 C 6 F

We can convert an unencoded numerical value very quickly – decimal to binary to hex. With RGB565, we need to encode each value to binary, then pack them together.

Form:

Decimal (r, g, b)

Binary rrrrr gggggg bbbbb

Pre-Hex rrrr rggg gggb bbbb

Example:

Decimal (22, 59, 5)

Binary 10110, 111011, 00100

Pre-Hex 1011 0111 0110 0100

Hex B764

Problems:

(10, 5, 30) in RGB565

(33,33,10) in RGB565

(21, 24, 0) in RGB565

(30, 0,0) in RGB565

0xC6FF in Decimal (r, g, b)

0xEFF0 in Decimal

MADCTL, CASET, PASET Worksheet

Writing to the ili9341 involves sending three commands and data. MADCTL sets our orientation

CASET – Page 110 of datasheet,

PASET – Page 112 of datasheet

MADCTL – Page 127 of datasheet

Setting orientation via MADCTL

Landscape – MADCTL 0x28, CASET is X, PASET is Y

Portrait – PASET is X, CASET is Y

We convert X and Y to hex and send them as data after CASET and PASET

General Flow given X, Y, and width, height:

X1 = X

Y1 = Y

X2 = x+width - 1

Y2 = y+height - 1

If Landscape:

CASET X1,X2

PASET Y1,Y2

If Portrait:

CASET Y1, Y2

PASET X1, X2

Example -

With the screen in landscape, we want to write a rectangle of height 30, width 20 to 280,10.

X = 10

Y = 10

Width = 50

Height = 30

X1 = 280 = 0x01A0

Y1 = 10 = 0x00A0

X2 = 280+20-1 = 0x012B

Y2 = 10+30-1 = 0x0027

CASET 0x01A0,0x012B

PASET 0x00A0,0x0027

What values of MADCTL, CASET, and PASET would we write

In landscape, a rectangle at 30,40 with dimensions 22x44

In Landscape, a rectangle at 70, 120 with dimensions 70x70

XPT2046 Converting Readings to Screen Space

Once we've got the X, Y values for touch (don't forget to flip them for landscape), we divide the screen up into boxes designated by row and column – this is tricky because column corresponds to X and row Corresponds to Y

The following formulas and values allow us to map a touch to an area of the screen.

Upper left (UX, UY)

Bottom right (LX, LY)

Display Width

Display Height

Rows

Columns

We calculate the size of a box for both touch and pixels. Since they cover the same area and have the same number of boxes for each row and colum, it allows us to know what pixels to correlate the touch to. This kind of mapping comes up often with low level graphics and UI.

*We do not care about the remainder so we remove it. 3A/2 = 1 +A/2. The remainder can be used to determine more precisely the area inside the box was touched.

Touch_column_size = (UX-LX)/Columns

Touched Column = int(touch_x/touch_column size)

Resolution

The touch controller can read in 8 bit or 12 bit values. We are using 8. 12 bit values allow far more accurate placement. For the purposes of building a simple UI, we just need to place or UI elements at pixel locations corresponding to the X, Y grid.

Mapping the display

Our display maps similarly. We can calculate the x,y of a box with the following formulas

X = (Screen_width/columns)* column

Y = (Screen_height/rows) * row

Which allows you to place UI elements underneath of an expected touch.

Exercise

Upper left is (200,200)

Bottom right is (100,100)

If we've divided our screen into 8 columns and 6 rows:

What row and column would a touch at 170, 120 be in?

Where would we need to place a UI element to show the user where to touch?