WS Pico 1.3" IPS LCD 240x240 Display Workout

by tonygo2

In this tutorial I'm going to try out a new and useful little display from WaveShare with the following features:

- Compatible with Raspberry Pi Pico
- 1.3" IPS LCD
- 240x240 resolution
- 65K RGB colours
- 4x user-programmable buttons
- 1x Joystick - 5 buttons
- Communicates via SPI

Specifications

- Operating Voltage: 2.6-5.5V
- Communication Interface: 4-wire SPI
- Display Panel: IPS
- Driver: ST7789
- Resolution: 240x240 PixelsDisplay
- Size: 23.40mm x 23.40mm
- Pixel Size: 0.0975mm x 0.0975mmDimensions: 52mm x 26.5mm

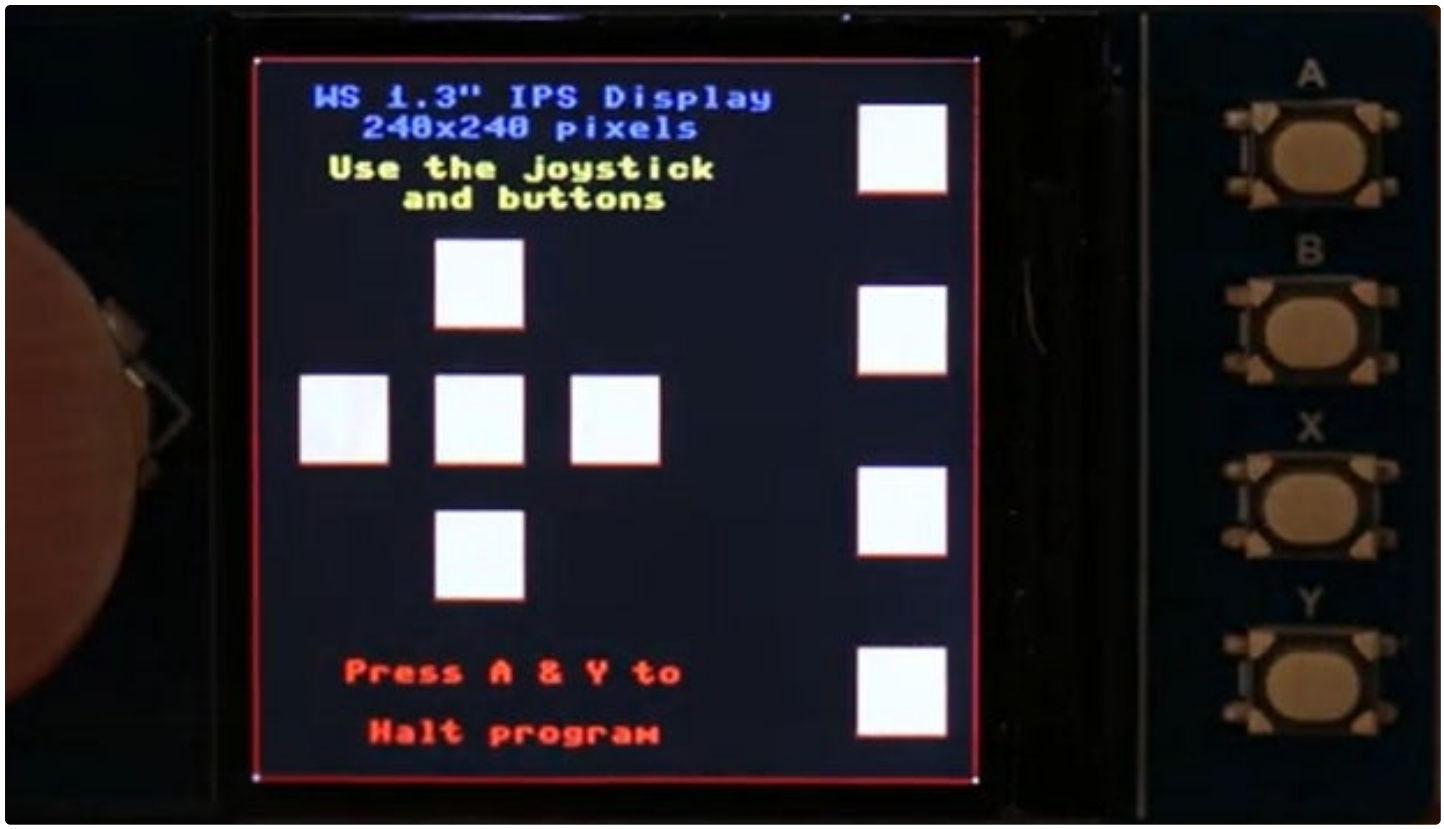
You can plug the pins on the Pico directly into the sockets on the back of the display or use a Pico Decker or similar expander to access the unused Pico GPIO pins.

Supplies:

USB cable

Thonny editor installed on your computer

Raspberry Pi Pico with pins soldered on.



Step 1: Documentation and Driver

WaveShare have provided excellent documentation and a MicroPython driver for the screen.

You can find the documentation here: www.waveshare.com/wiki/Pico-LCD-1.3

Looking at the pinout we find that the display leaves plenty of free GPIO pins for user projects: 0, 1 ,4, 5 ,6, 7, 14, 22, 26, 27, 28. These include I2C pins for connecting sensors/actuators and the 3 ADC pins for voltages/potentiometers.

The resources Tab provides access to a demonstration program in MicroPython. I downloaded it, unzipped it to access the contents and found MicroPython and C examples. I stuck with the MP, copied the code to Thonny, plugged in the display and Pico and tried it out.

I was pleased to find that it worked and allowed the user to press all the buttons and move the joystick to make squares on the screen change from white to red.

A quick look at the code showed that all the switches on the joystick and buttons had pull-ups so would have zero value when pressed. I noticed a few typos and incorrect labels so corrected them.

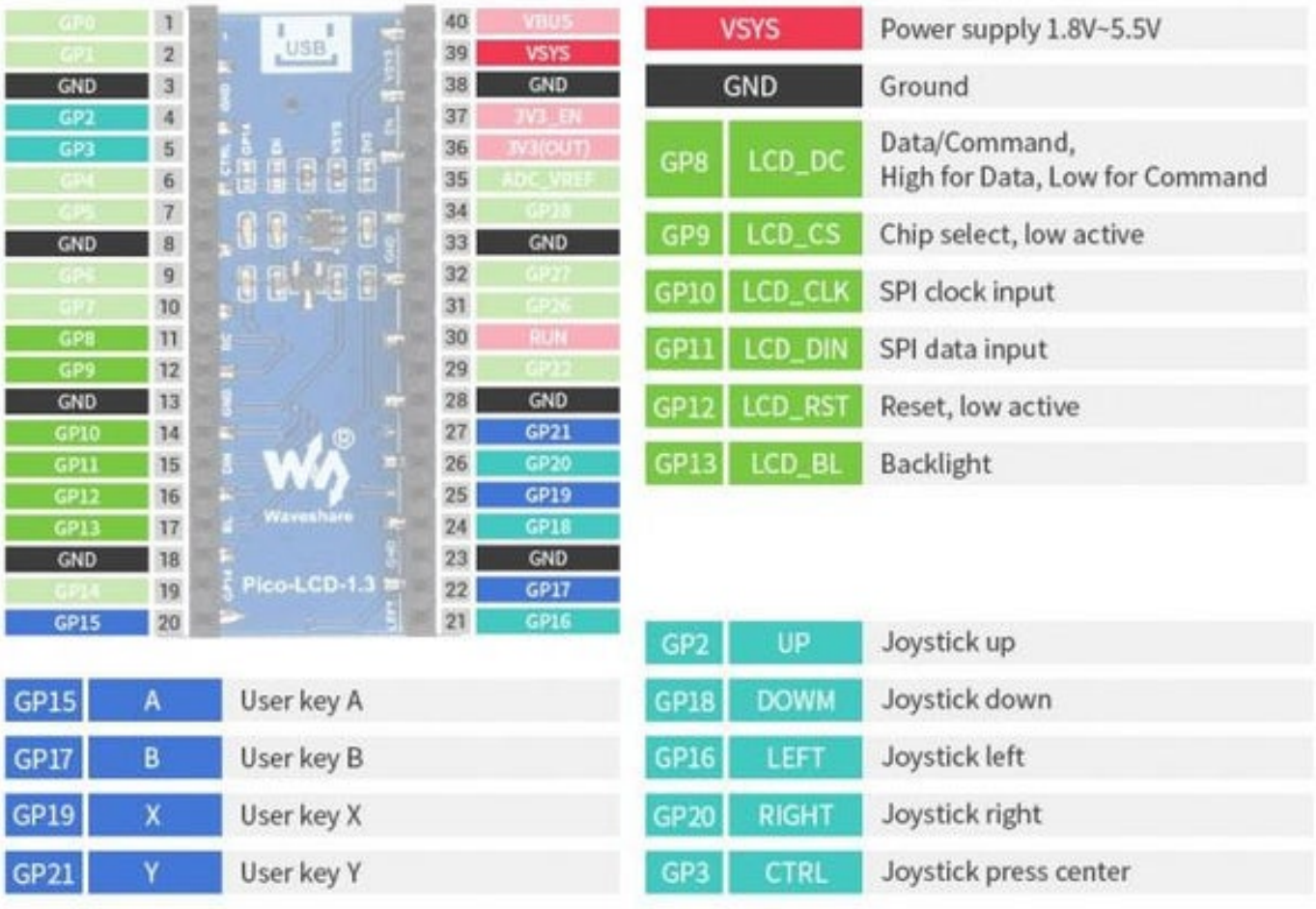
The colour system used is RGB565 meaning 5 bits for Red and Blue and 6 Bits for Green so I added my **colour(r,g,b)** routine to provide translation of RGB 3 byte values to 2 byte values used by the board.

The graphics routines for fills, lines, pixels, rectangles and text are provided by the **framebuf** library. This does not have circle routines, so I included my own.

I always like to test the edges and corner pixels of a new display as these throws up any problems with a driver.

I also moved the button square slightly to the left away from the frame to improve the balance.

The test program is available for download here:



Download

https://www.instructables.com/ORIG/FAK/ZCBC/KSBSGKVV/FAKZCBCKSBSGKVV.py

Step 2: The Code - Part 1

This shows the importation of the necessary library routines. **Math** is needed for the circle routine, **utime** for delays and **framebuf** for the screen.

The GPIO pins used for the display are defined and the start of the WS supplied driver is shown. The driver is pretty long, work perfectly and very difficult to follow. Just copy, paste and use it!

4 colours are defined: red, green blue and white as hexadecimal 2 byte values - difficult to understand. (I've explained how the colour() routine works in a previous Instructable.)

WS 240x240 V3.py

```
# Initial test of WaveShare 1.3 inch 240x240 screen with Joystick and Buttons
# Tony Goodhew - 14th August 2021
# Includes WaveShare driver Demo Code with typos and errors corrected
from machine import Pin,SPI,PWM
import framebuf
import utime
import os
import math
# ============ Start of Drive Code ================
#  == Copy and paste into your code ==
BL = 13  # Pins used for display screen
DC = 8
RST = 12
MOSI = 11
SCK = 10
CS = 9

class LCD_1inch3(framebuf.FrameBuffer):
    def __init__(self):
        self.width = 240
        self.height = 240

        self.cs = Pin(CS,Pin.OUT)
        self.rst = Pin(RST,Pin.OUT)

        self.cs(1)
        self.spi = SPI(1)
        self.spi = SPI(1,1000_000)
        self.spi = SPI(1,100000_000,polarity=0, phase=0,sck=Pin(SCK),mosi=Pin(MOSI),miso=None)
        self.dc = Pin(DC,Pin.OUT)
        self.dc(1)
        self.buffer = bytearray(self.height * self.width * 2)
        super().__init__(self.buffer, self.width, self.height, framebuf.RGB565)
        self.init_display()

        self.red   =   0x07E0 # Pre-defined colours
        self.green =   0x001f # Probably easier to use colour(r,g,b) defined below
        self.blue  =   0xf800
        self.white =   0xffff

    def write_cmd(self, cmd):
        self.cs(1)
        self.dc(0)
        self.cs(0)
        self.spi.write(bytearray([cmd]))
```

https://youtu.be/z1vY4gwZ6jQ

Step 3: The Code - Part 2

This shows the last few lines of the screen driver, the **colour(R,G,B)** and **ring(cx,cy,r,cc)** routines.

The start of the main program sets to brightness level to mid level and clears the display to a dark grey using my colour(0 routine.

```
        self.spi.write(self.buffer)
        self.cs(1)
# ========== End of Driver ============

def colour(R,G,B):
# Get RED value
    rp = int(R*31/255) # range 0 to 31
    if rp < 0: rp = 0
    r = rp *8
# Get Green value - more complicated!
    gp = int(G*63/255) # range 0 - 63
    if gp < 0: gp = 0
    g = 0
    if gp & 1:  g = g + 8192
    if gp & 2:  g = g + 16384
    if gp & 4:  g = g + 32768
    if gp & 8:  g = g + 1
    if gp & 16: g = g + 2
    if gp & 32: g = g + 4
# Get BLUE value
    bp =int(B*31/255) # range 0 - 31
    if bp < 0: bp = 0
    b = bp *256
    colour = r+g+b
    return colour

def ring(cx,cy,r,cc):   # Draws a circle - with centre (x,y), radius, colour
    for angle in range(91):  # 0 to 90 degrees in 2s
        y3=int(r*math.sin(math.radians(angle)))
        x3=int(r*math.cos(math.radians(angle)))
        LCD.pixel(cx-x3,cy+y3,cc)  # 4 quadrants
        LCD.pixel(cx-x3,cy-y3,cc)
        LCD.pixel(cx+x3,cy+y3,cc)
        LCD.pixel(cx+x3,cy-y3,cc)

# ========== Main ============
pwm = PWM(Pin(BL)) # Screen Brightness
pwm.freq(1000)
pwm.duty_u16(32768) # max 65535 - mid value

LCD = LCD_1inch3()
# Background colour - dark grey
LCD.fill(colour(40,40,40))
LCD.show()
```

Step 4: The Code - Part 3

Here we first set up the button and joystick switches with internal pull-ups.

The next section draws a red frame round the edge pixels of the display and puts a white pixel in each corner. The text is then written to the screen just once.

The variable **running** is set to **True** to control the main loop, which will terminate if **running** is re-set to **False**.

The main loop looks at the value of each of the 7 switches in turn. If pressed, showing a value of zero, the corresponding square on the display is turned **RED** and the switch value printed in the REPL.

If the switch is not pressed the corresponding square is set to **WHITE**.

```
# Define pins for buttons and Joystick
keyA = Pin(15,Pin.IN,Pin.PULL_UP) # Normally 1 but 0 if pressed
keyB = Pin(17,Pin.IN,Pin.PULL_UP)
keyX = Pin(19,Pin.IN,Pin.PULL_UP)
keyY= Pin(21,Pin.IN,Pin.PULL_UP)

up = Pin(2,Pin.IN,Pin.PULL_UP)
down = Pin(18,Pin.IN,Pin.PULL_UP)
left = Pin(16,Pin.IN,Pin.PULL_UP)
right = Pin(20,Pin.IN,Pin.PULL_UP)
ctrl = Pin(3,Pin.IN,Pin.PULL_UP)

# Draw background, frame, title and instructions
LCD.rect(0,0,240,240,LCD.red) # Red edge
# White Corners
LCD.pixel(1,1,LCD.white)     # LT
LCD.pixel(0,239,LCD.white)   # LB
LCD.pixel(239,0,LCD.white)   # RT
LCD.pixel(239,239,LCD.white) # RB
LCD.text('WS 1.3" IPS Display', 20, 10, colour(0,0,255))
LCD.text('  240x240 pixels', 20, 20, colour(0,0,255))
LCD.text('Use the joystick', 25, 33, colour(255,255,0))
LCD.text('   and buttons', 25, 43, colour(255,255,0))
LCD.text("Press A & Y to", 30, 200, colour(255,0,0))
LCD.text(" Halt program", 30, 220, colour(255,0,0))
LCD.show()

running = True # Loop control
# ============ Main loop ================
while(running):
    if keyA.value() == 0:
        LCD.fill_rect(200,15,30,30,colour(255,255,0)) # Yellow
        print("A")
    else :
        LCD.fill_rect(200,15,30,30,LCD.white)
        LCD.rect(200,15,30,30,LCD.red)

    if(keyB.value() == 0):
        LCD.fill_rect(200,75,30,30,colour(255,0,255)) # Magenta
        print("B")
    else :
        LCD.fill_rect(200,75,30,30,LCD.white)
        LCD.rect(200,75,30,30,LCD.red)

```

Step 5: The Code - Part 4

This section checks more of the switches.

```
    if(keyX.value() == 0):
        LCD.fill_rect(200,135,30,30,colour(0,255,255)) # Cyan
        print("X")
    else :
        LCD.fill_rect(200,135,30,30,LCD.white)
        LCD.rect(200,135,30,30,LCD.red)

    if(keyY.value() == 0):
        LCD.fill_rect(200,195,30,30,colour(255,180,50)) # Orange
        print("Y")
    else :
        LCD.fill_rect(200,195,30,30,LCD.white)
        LCD.rect(200,195,30,30,LCD.red)

    if(up.value() == 0):
        LCD.fill_rect(60,60,30,30,LCD.red)
        print("UP")
    else :
        LCD.fill_rect(60,60,30,30,LCD.white)
        LCD.rect(60,60,30,30,LCD.red)

    if(down.value() == 0):
        LCD.fill_rect(60,150,30,30,LCD.red)
        print("DOWN")
    else :
        LCD.fill_rect(60,150,30,30,LCD.white)
        LCD.rect(60,150,30,30,LCD.red)

    if(left.value() == 0):
        LCD.fill_rect(15,105,30,30,LCD.red)
        print("LEFT")
    else :
        LCD.fill_rect(15,105,30,30,LCD.white)
        LCD.rect(15,105,30,30,LCD.red)

    if(right.value() == 0):
        LCD.fill_rect(105,105,30,30,LCD.red)
        print("RIGHT")
    else :
        LCD.fill_rect(105,105,30,30,LCD.white)
        LCD.rect(105,105,30,30,LCD.red)

```

Step 6: The Code - Part 5

Here the final switch is tested and then we look to see if switches A and Y are pressed together, the condition to terminate the looping. If so, **running** is set to False and we drop out of the loop.

We then start the 'tidy-up' sequence. A series of concentric rings are drawn in yellow and ' **Halted'** is displayed for 3 seconds before the screen is cleared to black and the execution stops.

The main problem with this screen is the very small size of the text characters provided by the built in font in **framebuf** and the tiny pixel size of only 0.0975mm x 0.0975mm. What we really need is a larger size of text to make things easier to read.

```
    if(ctrl.value() == 0):
        LCD.fill_rect(60,105,30,30,LCD.red)
        print("CTRL")
    else :
        LCD.fill_rect(60,105,30,30,LCD.white)
        LCD.rect(60,105,30,30,LCD.red)

    LCD.show()
    if (keyA.value() == 0) and (keyY.value() == 0): # Halt looping?
        running = False

    utime.sleep(.15) # Debounce delay - reduce multiple button reads

# Finish
LCD.fill(0)
for r in range(10):
    ring(120,120,60+r,colour(255,255,0))
LCD.text("Halted", 95, 115, colour(255,0,0))
LCD.show()
# Tidy up
utime.sleep(3)
LCD.fill(0)
LCD.show()
```

Step 7: Change the Font

Les Wright published a solution to the very small font problem on the Pimoroni Forum for the Pimoroni Display. It was written in pure MicroPython and is easy to follow. I have modified it to work on this WaveShare display and added code for an addition size - 3. You can find a link to the forum at the top of the code. I suggest that you take a look at the discussion on the forum and at the suggestion from **Steve Borg**, which was followed.

As you can see from the video the size 1 version is even smaller than the **framebuf** font but sizes 2 and 3 are very useful.

I've provided the essential code to produce the text in the video and the full code as a download as the font definitions and routines are pretty long.

The **True** and **False** can be replaced with **1** and **0** to speed up typing. I'm not too keen on the slow, terminal like effect and will probably set both the final parameters to zero, to speed things up, and add my own **LCD.show()** where needed.

Now that I can produce easily readable text it is time to move onto a full project the give the display a real workout.

https://youtu.be/5TUY287aMH4

```
359
360 pwm = PWM(Pin(BL)) # Screen Brightness
361 pwm.freq(1000)
362 pwm.duty_u16(32768) # max 65535 - mid value
363
364 LCD = LCD_1inch3()
365 # Background colour
366 LCD.fill(colour(0,0,0)) # BLACK
367 LCD.show()
368
369 c = colour(255,0,0)    # c holds the colour to be used
370 printstring("!! Hello World !!",0,0,2,True,False,c) # Slow
371 c = colour(0,255,0)
372 printstring("Les Wright 2021 - Tiny font",9,30,1,False,True,c) # Fast string
373
374 c = colour(0,0,255)
375 printstring('abcdefghijk',0,60,2,True,False,c) # one char at a time, like an old serial terminal
376 printstring('lmnopqrstuvwxyz',0,80,2,True,False,c) # slow
377 printstring('ABCDEFGHIJKLM',0,100,2,True,False,c)
378 printstring('NOPQRSTUVWXYZ',0,120,2,True,False,c)
379
380 printstring('`~!@#$%^&*()_-+=[',0,140,2,False,False,c) # Fast
381 printstring(']\\{}|;\':"<>?,./',0,160,2,False,False,c)
382 LCD.show()
383
384 c = colour(255,255,0)
385 printstring("Text size 3",17,190,3,0,0,c) # Easier format
386 c = colour(255,255,255) # White
387 printstring("Text size 1",17,220,1,0,0,c) # Tiny Les Wright font
388 LCD.text("Framebuf Text", 110, 220, colour(0,255,255)) # Built in framebuf tiny fount
389 LCD.show()
390 spinner = '-\|/-\|/'
391 for i in range(30):
392     for x in spinner:
393         c=colour(255,255,0)
394         printchar(x,190,50,2,True,c)
395         delchar(190,50,2,True)
396
397 utime.sleep(10)
398 #Tidy up
399 LCD.fill(colour(0,0,0)) # BLACK
400 LCD.show()
401
```

Download

https://www.instructables.com/ORIG/F1S/MM7Z/KSG2RTYA/F1SMM7ZKSG2RTYA.py

Step 8: Menu Project Controlled by the Joystick

Here you can view the finished project and look at the full code.

The program is pretty long, over 600 lines of code, and well commented. It contains the data and routines of the font system and the the 7-segment system, described in a previous Instructable. Each of the menu items have their own sections. Much of it should be pretty easy to follow. I provide images and comments on the more interesting bits.

A choice from the menu is high-lighted in blue by pushing the joystick UP and DOWN and then actioned by pressing the joystick central button straight down. The selection runs and then returns to the menu for further selections until HALT is picked.

I suggest you view the video, download the code and open it in Thonny.

https://youtu.be/0XT_hLpsI0c

Download

https://www.instructables.com/ORIG/F58/AP85/KSIXO124/F58AP85KSIXO124.py

Step 9: The Menu

The top part displays the menu. The current selected item is shown in BLUE and the other items in YELLOW.

Putting this amount of text on the screen takes a considerable amount of processing and provides some 'de-bounce time' when reading the switches. We only look at the UP, DOWN and CTRL switches at this time. the variable **running** is set to **False** which ends the menu loop when **Halt** is selected.

```
555  # ======= Menu ==============
556  m = 0
557  yellow = colour(255,255,0)
558  blue = colour(0,0,255)
559  running = True
560  while running:
561      c = colour(255,0,0)
562      printstring("Menu Project",17,10,3,0,0,c)
563      c = yellow
564      if m == 0:
565          c = blue
566      printstring("Lines & Circles",35,50,2,0,0,c)
567      c = yellow
568      if m == 1:
569          c = blue
570      printstring("7-Segment",35,80,2,0,0,c)
571      c = yellow
572      if m == 2:
573          c = blue
574      printstring("Bar Graph",35,110,2,0,0,c)
575      c = yellow
576      if m == 3:
577          c = blue
578      printstring("Line graph",35,140,2,0,0,c)
579      c = yellow
```

```
580         if m == 4:
581             c = blue
582         printstring("Halt",35,170,2,0,0,c)
583         LCD.show()
584 
585         # Check joystick UP/DOWN/CTRL
586         if(up.value() == 0):
587             m = m - 1
588             if m < 0:
589                 m = 0
590 
591         elif(down.value() == 0):
592             m = m + 1
593             if m > 4:
594                 m = 4
595 
596         elif(ctrl.value() == 0):
597             if(m == 4): # Exit loop and HALT program
598                 running = False
599             if(m == 3):
600                 graphs()
601             if(m == 2):
602                 bar() # Dynamic part in procedure
603             if(m == 1):
604                 sevenSeg()
605             if(m == 0):
606                 lines()
607 
608 # Finish
609 LCD.fill(0)
```

Step 10: Finishing Off

We drop out of the menu loop to this section. It displays the **HALTED** screen in a wide yellow circle and then shuts down the screen.

```
607
608 # Finish
609 LCD.fill(0)
610 for r in range(10):
611     ring(120,120,60+r,colour(255,255,0))
612
613 c = colour(255,0,0)
614 printstring("Halted",80,110,2,0,0,c)
615 LCD.show()
616 # Tidy up
617 utime.sleep(3)
618 LCD.fill(0)
619 LCD.show()
620
```

Step 11: The Bar Graph

The horizonal bar graph is controlled by the LEFT and RIGHT switches on the joystick. Button Y is used to terminate the action and return to the menu. Notice that the colour of the bar changes from BLUE to RED as the value is increased. The loop is controlled by the **active** variable.

```
493
494 def bar(): # Dynamic bar graph under joystick control
495     LCD.fill(0) # BLACK
496     LCD.show()
497     c = colour(0,255,0)
498     printstring("Bar graph",42,20,3,0,0,c)
499     c = colour(160,160,160)
500     printstring("Move Joystick",30,65,2,0,0,c)
501     printstring("Left and Right",25,95,2,0,0,c)
502     printstring("End with button Y",0,125,2,0,0,c)
503     c = colour(150,150,150)
504     LCD.fill_rect(18,160,2,40,c)
505     v = 50
506     c = colour(255,0,0)
507     LCD.fill_rect(20,170,v*2,20,c)
508     LCD.show()
509     active = True
510     while active:
511         if(left.value() == 0):
512             v = v - 1
513             if v < 0:
514                 v = 0
515         if(right.value() == 0):
516             v = v + 1
517             if v > 100:
518                 v = 100
519         c = 0
520         LCD.fill_rect(20,170,200,20,c) # Blanking old bar
521         c = colour(int(255*v/100),0,255-int(255*v/100)) # Calculate bar colour Blue -> Magenta -> Red
522         LCD.fill_rect(20,170,v*2,20,c) # Draw bar
523         c = 0
524         LCD.fill_rect(85,200,120,30,c) # Blanking old percentage
525         c = colour(255,255,0)
526         printstring(str(v) + " %",85,200,3,0,0,c)
527         LCD.show()
528         if(keyY.value() == 0): # Finished?
529             LCD.fill(0) # BLACK
530             LCD.show()
531             active = False
532
```

Step 12: 7 Segment Section

This section calls on a **data table** and **seg()** procedure saved earlier in the program. It demonstrates a simple and compact method of getting large numbers on a graphical display.

```
def sevenSeg():
    LCD.fill(0)
    LCD.show()
    top = 40
    top2 = top+12
    left = 23
    c = colour(255,255,0) # yellow
    seg(left+5,top,7,5,0,c)
    seg(left+30,top,17,5,0,c)
    seg(left+54,top2,5,3,0,c)
    seg(left+72,top2+12,14,2,0,c)
    seg(left+84,top2+12,16,3,0,c)
    utime.sleep(1.5)

    x0 = 30
    y0 = 30
    for f in range(1,10):
        # Adjust Top Left position of number
        x1 = x0 + (9-f) *4
        y1 = y0 + (9-f)
        LCD.fill(0x0)
        c = colour(255,255,0)
        printstring("7-Segment Size",23,140,2,0,0,c)
        printstring(str(f),110,165,3,0,0,c)
        LCD.show()
        for n in range(20):
            nn = n + (f-1)*20 + (f-1)*100
            hnds = int(nn/100)
            nn = nn - (hnds * 100)
            tens = int(nn / 10)
            units = nn - (tens * 10)
            seg(x1,y1,hnds,f,0,0xF)
            seg(x1 + f*6,y1,tens,f,0,0xF)
            seg(x1 + f*12,y1,units,f,0,0xF)
            LCD.show()
    utime.sleep(1.5)
    LCD.fill(0)
    LCD.show()
```

Step 13: Lines and Circles

This quite basic section demonstrates the high resolution of the display. It may be small but there are plenty of pixels.

```
def lines():
    LCD.fill(0)
    LCD.show()
    c = colour(200,200,200)
    printstring("Lines",30,40,2,0,0,c)
    LCD.show()
    c = colour(255,0,0)
    b = colour(0,0,255)
    LCD.vline(0,0,239,c)   # Screen edges
    LCD.hline(0,239,239,c)
    LCD.vline(239,0,239,b)
    LCD.hline(239,1,239,b)
    for i in range(0,240,5): # Sloping lines
        ii = i +1
        LCD.line(0,ii,ii,239,c)
        LCD.line(239,239-ii,239-ii,0,b)
        utime.sleep(0.03)
        LCD.show()

    c = colour(200,200,200)
    printstring("Circles",120,190,2,0,0,c)
    LCD.show()
    ring(120,120,47,colour(170,170,70))
    ring(120,120,41,colour(200,200,200))
    ring(120,120,35,colour(250,250,250))
    LCD.show()
    ring(120,120,30,colour(255,255,0))
    ring(120,120,25,colour(255,0,255))
    ring(120,120,20,colour(0,255,255))
    LCD.show()
    utime.sleep(3)
```

```
452     LCD.fill(0)
453     LCD.show()
454
```

Step 14: Many Calculations

Drawing Sine/Cosine curves and Circles require many calculations. The Pico and WS display managed the task pretty quickly. The Cosine drawing only updates the screen at the end while the Sine is updated after each pixel.

The sharp eyed may have noticed that my display at has 'stuck pixel' in the lower right quadrant of the screen. (It is always lit - even after a black screen fill.) I've not had that happen before but It may be because the pixels on this display much smaller than those on my other screens. (My supplier, ThePiHut, are send a free replacement - thank you.)

I hope you will give this project a try. The resources (Pico, screen and USB cable) are all very reasonably priced and can be used for a variety of other projects.

I hope you have enjoyed this tutorial and have found it useful. I'm happy to receive comments and questions.

```
# == Project specific routines ==============
def ring(cx,cy,r,cc):   # Draws a circle - with centre (x,y), radius, colour
    for angle in range(91):  # 0 to 90 degrees in 2s
        y3=int(r*math.sin(math.radians(angle)))
        x3=int(r*math.cos(math.radians(angle)))
        LCD.pixel(cx-x3,cy+y3,cc)  # 4 quadrants
        LCD.pixel(cx-x3,cy-y3,cc)
        LCD.pixel(cx+x3,cy+y3,cc)
        LCD.pixel(cx+x3,cy-y3,cc)

def graphs(): # Draw Sine and Cosine graphs
    LCD.fill(0)
    LCD.show()
    c = colour(80,80,80)
    factor = 361 /240
    LCD.hline(0,60,239,c)
    LCD.show()
    c = colour(255,0,0)
    for x in range(0,240):
        y = int ((math.sin(math.radians(x * factor)))* -50) + 60
        LCD.pixel(x,y,c)
        LCD.show()
    printstring("Sine", 40, 70, 2, 0,0,c)
    LCD.show()

    c = colour(80,80,80)
    LCD.hline(0,180,239,c)
    LCD.show()
    c = colour(0,255,0)
    for x in range(0,240):
        y = int((math.cos(math.radians(x * factor)))* -50) + 180
        LCD.pixel(x,y,c)
    printstring("Cosine",75,160,2,0,0,c)
    LCD.show()
    utime.sleep(3)
    LCD.fill(0)
    LCD.show()

```