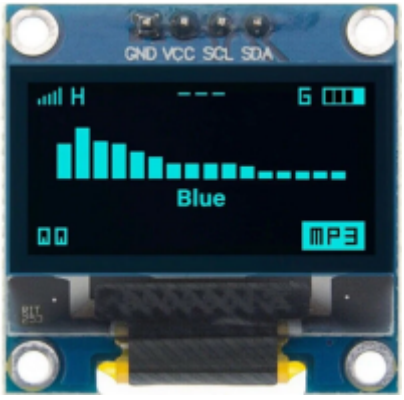
Using Pico with SSD1306 based LCD and I2C

Aug 2026

Introduction



One of the things I have been really looking forward to using with the Pico is an LCD screen. The one I have purchased is quite small (0.96“) but for simple testing will be pretty useful.

About the LCD:

- Its from AliExpress - search 0.96” Inch OLED Display Screen Module I2C IIC 128×64 SS - D - 1306 3.3V-5V
- It operates from 3.3v-5v
- Its based on a 1306 driver chip that sits behind the LCD
- It uses I2C (no SPI support on this one, but you can get them with SPI)

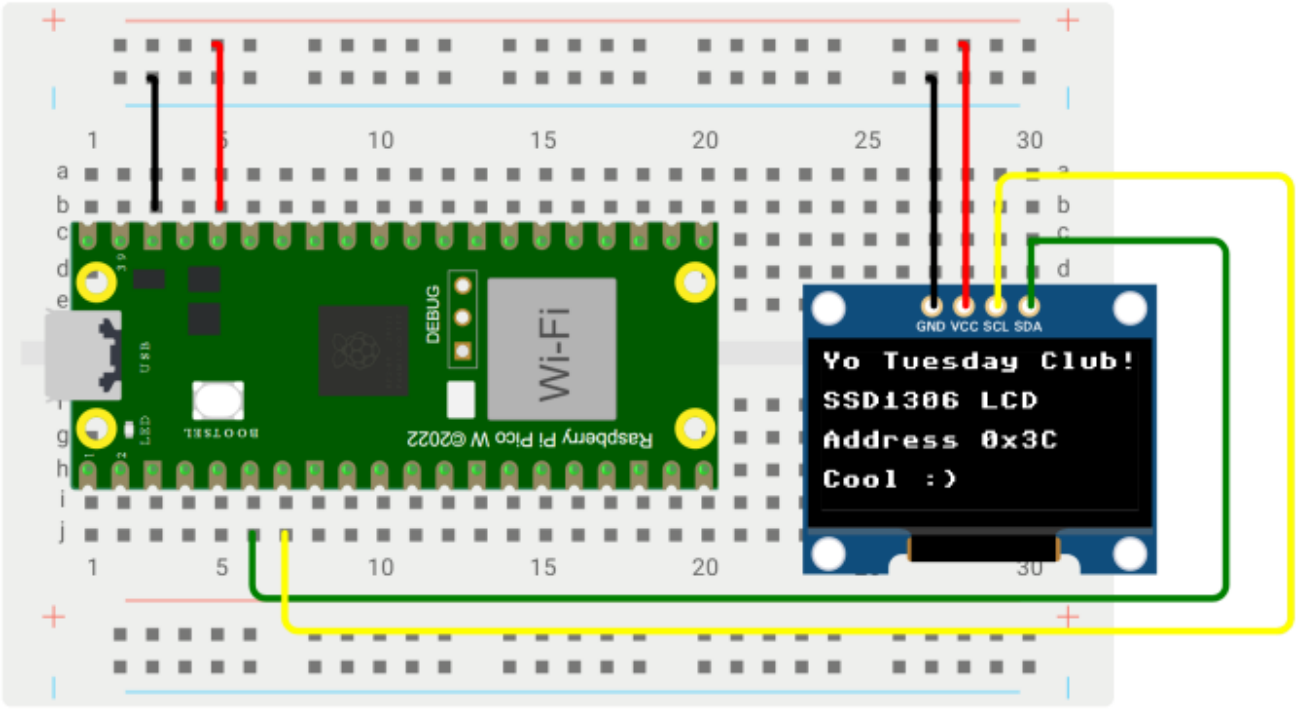
To get the LCD to operate on the Pico is insanely simple (once you have the correct information)

- Wire up the LCD to the Pico I2C pins (we will look at this)
- Connect Power
- Save ssd1306 library to the Pico (provided in this wiki)
- Save a test file to the Pico (provided in this wiki)

Lets get started.

The Wiring

Please see below for the wiring, I have used breadboard to make the connections simple to see, but you could wire this directly to the Pico pins using jumper cables, or use Veroboard etc.



- GND goes to Pin 38 (or any GND pin on the Pico you want)
- VCC goes to Pin 36 on the Pico which is the 3v3 Pin.

These two pins will power the ssd1306 LCD (it won't come on as such, there is no backlight)

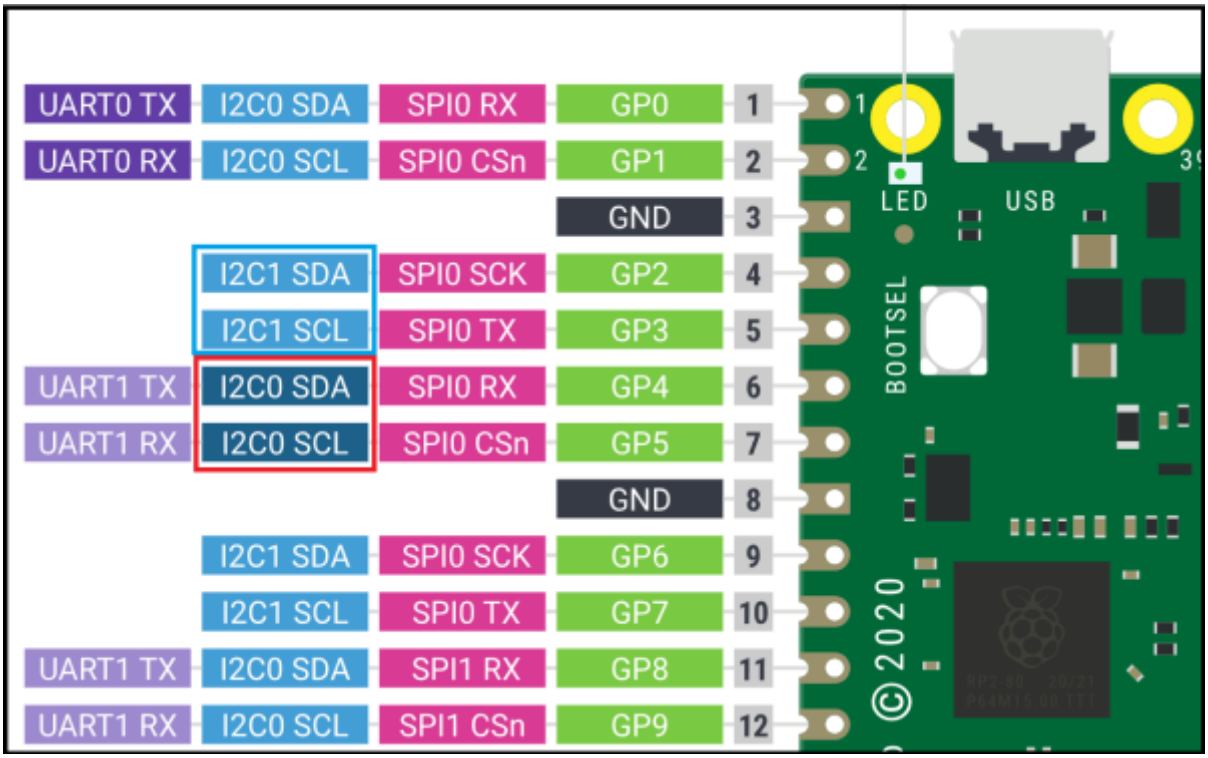
For the data pins.

- SCL goes to Pico Pin 7 (I2C0 SCL)
- SDA goes to Pico Pin 6 (I2C) SDA)

That is the data wired up, so wiring is complete, very simple indeed :)

A Quick Note About the I2C Pins

If we look at this part of the Raspberry Pi Pico Pins (or any RP2040 based device) we can see there are groups of Pins for I2C.



There are a sets of I2C0 pins and I2C1 pins, so you need to remember which Bus you are on (zero for our example) You also need to remember the pin numbers as we will reference both the Pin numbers and Bus number is our code.

To be clear, we are using:

- Bus = 0
- Pins = 6(SDA) and 7(SCL)

Note: Pins 6 and 7 are the physical pins, but for the Python code, we need to supply the GPIO numbers, which will be:

- Pin 6 = GPIO4 (SDA)
- Pin 7 = GPIO5 (SCL)

The Micro Python Code

We need two files on the Pico.

- The library file for the LCD (this interprets our commands from the Pico, and coverts them in to something the LCD understands).
- A test file to write to the LCD.

First the Library, now I am assuming you know how to get files on and off the Pico, else you will have to turn to your favorite Search Engine / AI.

The code below needs to be saved in the lib folder on the Pico and be called ssd1306.py

| download

```
# MicroPython SSD1306 OLED driver, I2C and SPI interfaces

import framebuf

class SSD1306:
    def __init__(self, width, height):
        self.width = width
        self.height = height
        self.pages = height // 8
        self.buffer = bytearray(self.pages * width)
        self.framebuf = framebuf.FrameBuffer(self.buffer, width, height, framebuf.MONO_VLSB)
        self.poweron()
        self.init_display()
```

```
    def init_display(self):
        for cmd in (
            0xae,         # DISPLAYOFF
            0x20, 0x00,   # MEMORYMODE
            0x40,         # SETSTARTLINE
            0xa1,         # SEGREMAP
            0xc8,         # COMSCANDEC
            0xda, 0x12,   # SETCOMPINS
            0x81, 0x7f,   # SETCONTRAST
            0xa4,         # DISPLAYALLON_RESUME
            0xa6,         # NORMALDISPLAY
            0xd5, 0x80,   # SETDISPLAYCLOCKDIV
            0x8d, 0x14,   # CHARGEPUMP
            0xaf          # DISPLAYON
        ):
            self.write_cmd(cmd)

    def poweron(self):
        pass

    def contrast(self, contrast):
        self.write_cmd(0x81)
        self.write_cmd(contrast)

    def invert(self, invert):
        self.write_cmd(0xa7 if invert else 0xa6)

    def show(self):
        for page in range(self.pages):
            self.write_cmd(0xb0 | page)
            self.write_cmd(0x00)
            self.write_cmd(0x10)
            self.write_data(self.buffer[page * self.width:(page + 1) * self.width])

    def fill(self, col):
        self.framebuf.fill(col)

    def pixel(self, x, y, col):
        self.framebuf.pixel(x, y, col)

    def text(self, string, x, y):
        self.framebuf.text(string, x, y)

    def scroll(self, dx, dy):
        self.framebuf.scroll(dx, dy)

    def blit(self, fbuf, x, y):
        self.framebuf.blit(fbuf, x, y)


class SSD1306_I2C(SSD1306):
    def __init__(self, width, height, i2c, addr=0x3c):
        self.i2c = i2c
        self.addr = addr
        super().__init__(width, height)

    def write_cmd(self, cmd):
        self.i2c.writeto(self.addr, bytearray([0x00, cmd]))

    def write_data(self, buf):
        self.i2c.writeto(self.addr, bytearray([0x40]) + buf)
```

We don't need to understand at this point how this code works, we just need to worry about some code to make the LCD do things :)

Below is some simple test code to write four lines of text to the LCD, copy this code and write it to a file on your Pico called lcd-test-text.py

| download

```
from machine import Pin, I2C
import ssd1306
import time

# Your wiring: SDA=GP4, SCL=GP5 → I2C0
i2c = I2C(0, scl=Pin(5), sda=Pin(4), freq=200000)

lcd = ssd1306.SSD1306_I2C(128, 64, i2c)

lcd.fill(0)
lcd.text("Yo Tuesday Club!", 0, 0)
lcd.text("SSD1306 LCD", 0, 16)
lcd.text("Address 0x3C", 0, 32)
lcd.text("Cool :)", 0, 48)
lcd.show()

time.sleep(5)
lcd.fill(0)
lcd.show()
```

This code is very short and pretty simple, lets break it down before we use it:

How the Code Works

This part imports some libraries, including our LCD library, there is a time library here, but we are not using it (I want to make a clock eventually)

| download

```
from machine import Pin, I2C
import ssd1306
import time
```

This next part is crucial. It contains the Pico Pins you used for data. However, we are using the GPIO pin reference, not the actual pin number.

At the start of this line: i2c = I2C(0 the 0 is the Bus number.

| download

```
# Your wiring: SDA=GP4, SCL=GP5 → I2C0
i2c = I2C(0, scl=Pin(5), sda=Pin(4), freq=200000)
```

This next line does the following:

- lcd creates and object called lcd
- ssd1306 refers to the library we created.
- SSD1306_I2C defines we are using I2C (not SPI for example)
- The rest are parameters passed to the library so the library knows how to handle our LCD

| download

```
lcd = ssd1306.SSD1306_I2C(128, 64, i2c)
```

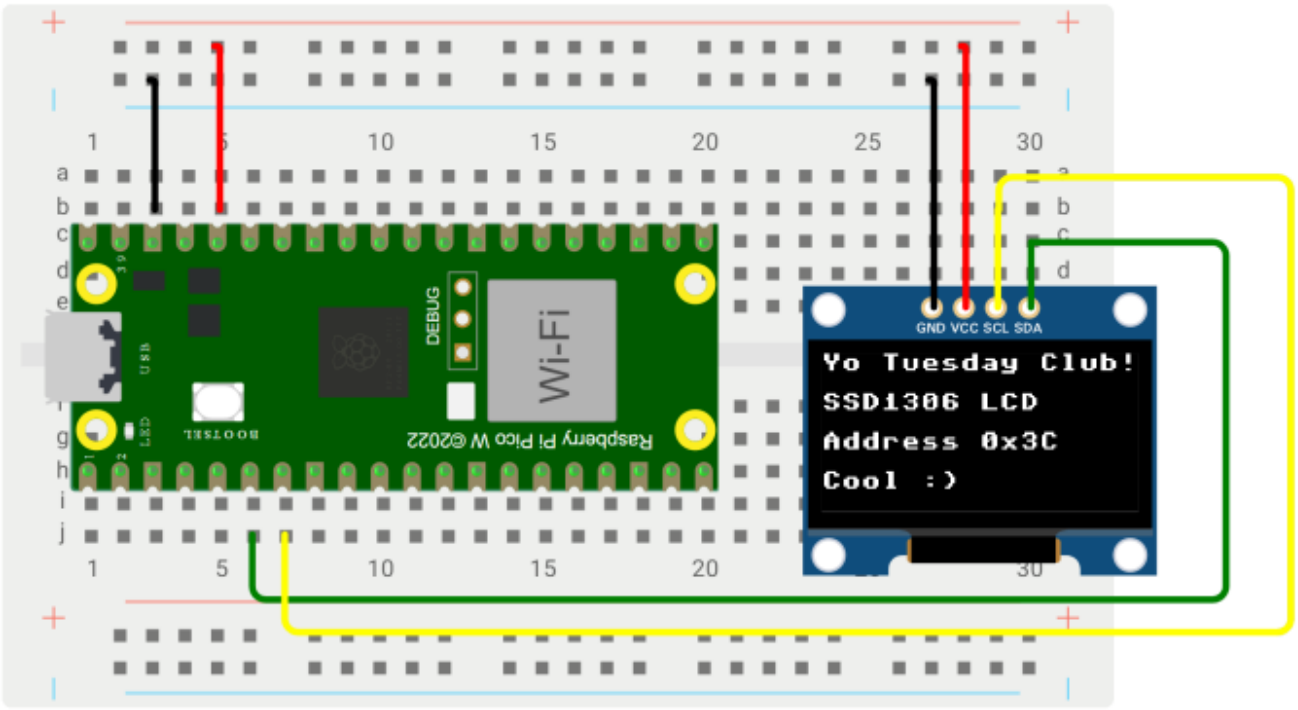
Now for the good part:

- lcd.fill(0) - fils the LCD with zeroes (blank the LCD)
- lcd.text(“Yo Tuesday Club!”, 0, 0) - Set the first line
- lcd.text(“SSD1306 LCD”, 0, 16) - Set the second line
- lcd.text(“Address 0x3C”, 0, 32) - Set the third line

- lcd.text("Cool :)", 0, 48) - Set the last line
- lcd.show() - Write to the LCD.

The numbers (0, 0), (0, 16) are the x,y start point of each line of text, we can see that the height of the text we are using is 16 pixels.

At this point, the LCD will show our text, and you shoud see something very similar to below:



This last part just turns the LCD off after 5 seconds, as I don't know what my LCD is like for burn in.

| download

```
time.sleep(5)
lcd.fill(0)
lcd.show()
```

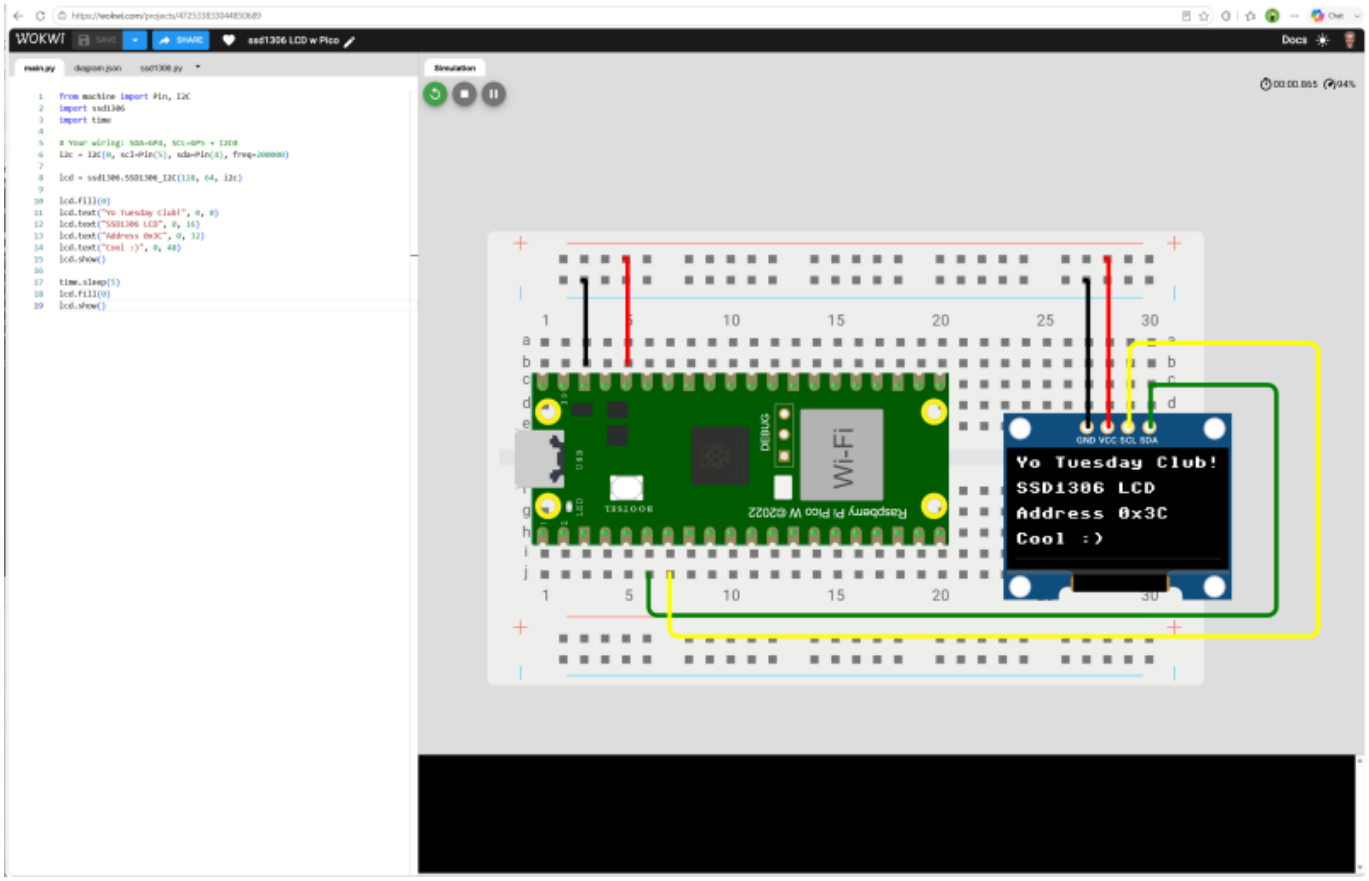
Hopefully, this has worked for you and you can have some fun with the LCD. This example we are only writing text to the LCD, in future examples, we will explore something a little more graphical.

Oh No!!! I don't have an LCD to play with yet!!!

Don't worry, I got you. I have set up this project on Wokwi, a simulator site for Microcontrollers.

Please follow this link: Example Pico ssd1306 LCD test

Here you can see the code, the library and if you press play it will emulate the actual hardware version we have made.

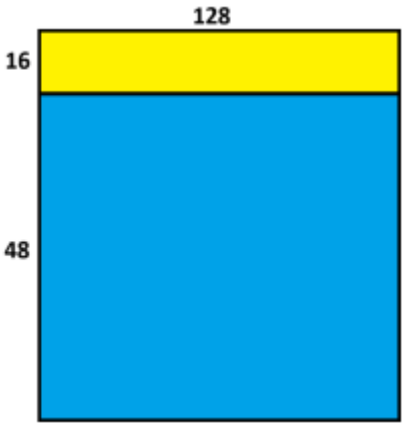


I hope you find this wiki article useful :)

Something Quirky with this OLED Screen

So I have noticed something a little quirky with this LCD Panel. It seems to be in two parts, let me explain.

The Yellow section of the panel is 16 pixels high, then there seems to be a hard barrier between there and the Blue section which is 48 pixels high.



So you can move text vertically up and down the screen, but not through the join in the yellow/blue part, the text will get chopped off. So you will have to think about the screen as two parts.

Below is some code where you can squeeze 8 lines of text on to this little OLED. Each line the text is 6 pixels tall, and there is a two pixel gap between each line. Effectively each line takes 8 pixels, so 8 lines = 8linex8Pixel = 64 pixels, the exact LCD height.

| download

```
from machine import Pin, I2C
import ssd1306
```

```
import time

i2c = I2C(0, scl=Pin(5), sda=Pin(4), freq=200000)
lcd = ssd1306.SSD1306_I2C(128, 64, i2c)

lcd.fill(0)
lcd.text("Yo Tuesday Club!", 0, 0)
lcd.text("SSD1306 LCD",      0, 8)
lcd.text("Address 0x3C",     0, 16)
lcd.text("Cool :)",          0, 24)
lcd.text("Another Line :)",  0, 32)
lcd.text("One More? :)",     0, 40)
lcd.text("Line 7",           0, 48)
lcd.text("Line 8",           0, 56)
lcd.show()

time.sleep(300)
lcd.fill(0)
lcd.show()
```

The LCD can have 16 characters across, so the characters are 8 pixels wide. So 16char x 8pixel =128 Pixels wide.

Using the code above, feel free to change the text and have a play, also play with the line spacing to explore the Yellow/Blue section limitation, each line is moved down by 8 pixels in the script, but have a play with these values to see what fits for you.
