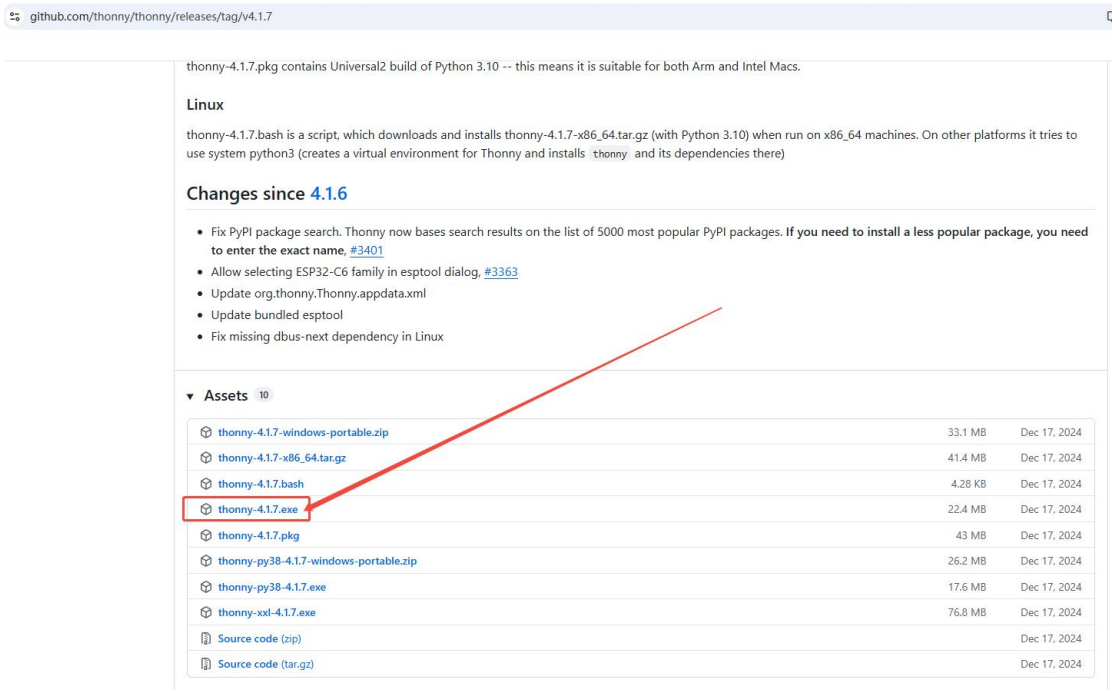
In this tutorial, we will demonstrate how to use Python to display the values read by a temperature and humidity sensor on the screen and control the LED lights by touching the screen.

Note: Because the 5-inch screen only supports 16-bit, and SquareLine Studio 9.0 and above only support 32-bit for now (micropython only), the 5-inch screen cannot currently support SquareLine Studio.

Build projects using Thonny IDE

1. Please visit https://thonny.org/ and download the corresponding software version (the Windows version is used as an example here).

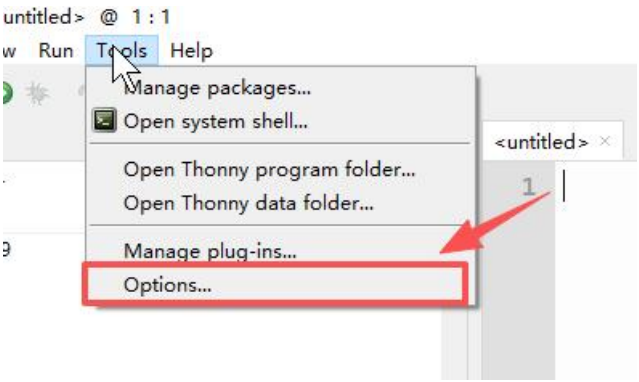
Note: If the latest version of the editor fails to upload firmware or compile successfully, please try the stable version 4.1.7. Download link: https://github.com/thonny/thonny/releases/tag/v4.1.7



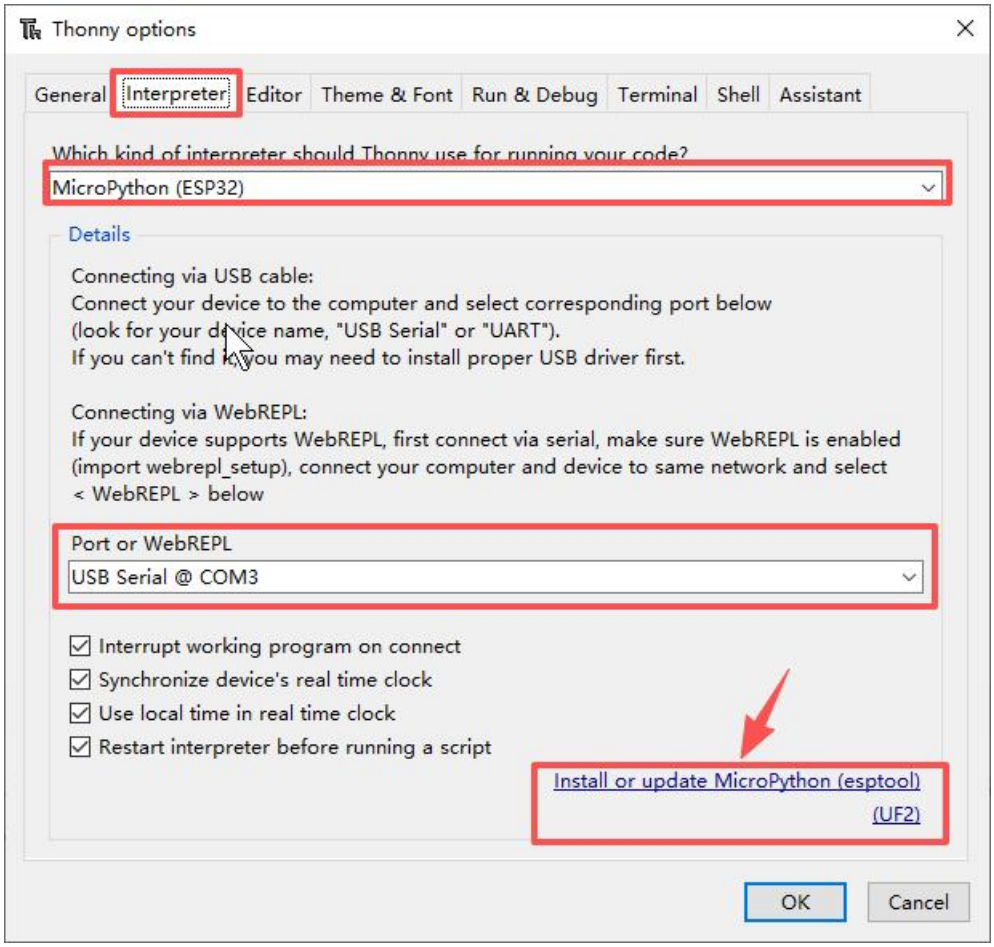
2. Double-click the downloaded .exe file to install the software.

Upload firmware

1. Connect the CrowPanel Advanced 5-inch ESP32-P4 HMI AI Display to your computer.
2. Open Thonny IDE, click " Tools " -> " Options "



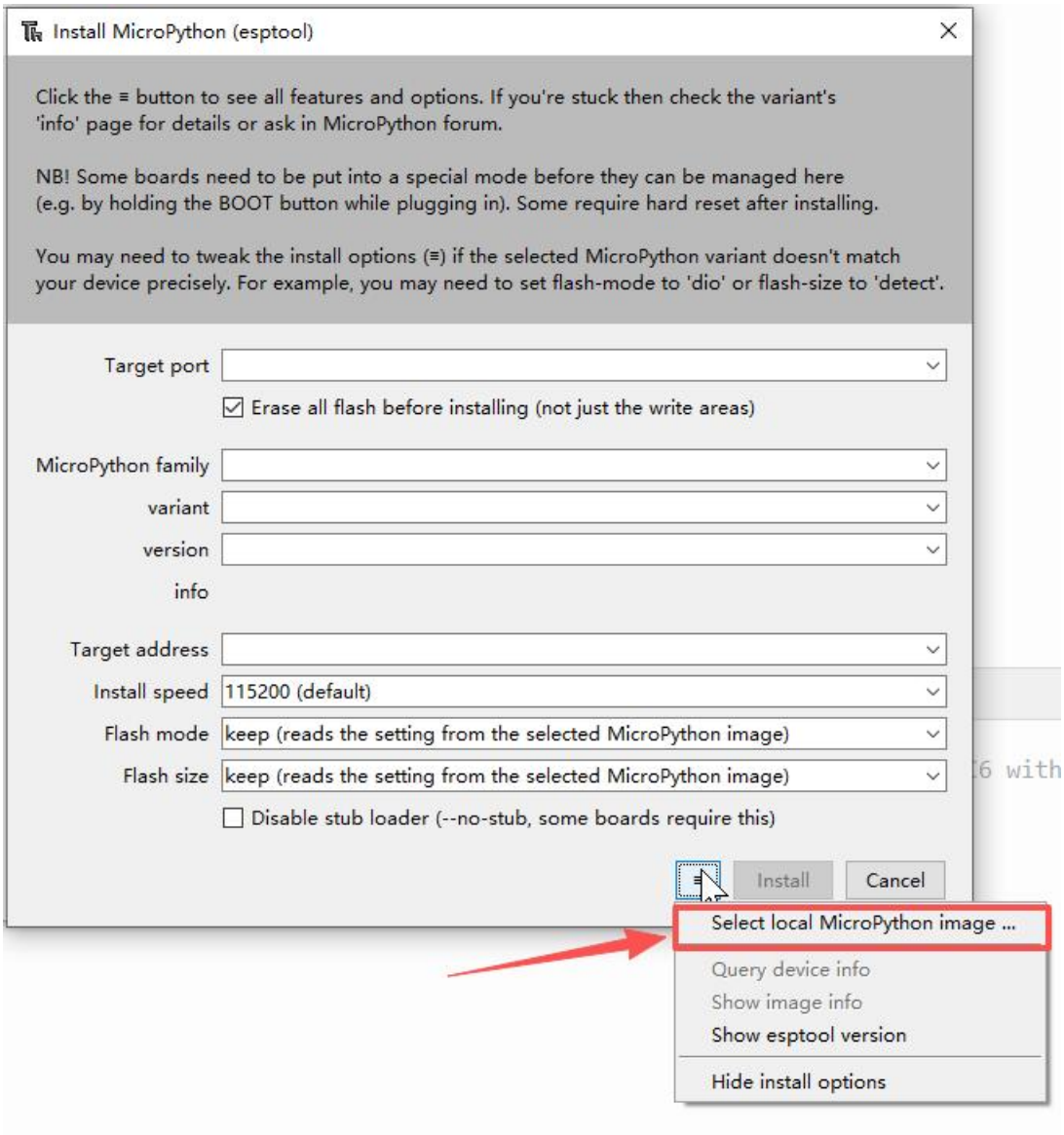
3. Select " MicroPython (ESP32) " as the interpreter, select the corresponding serial port, and then click " Install or update MicroPython (esptool) " .



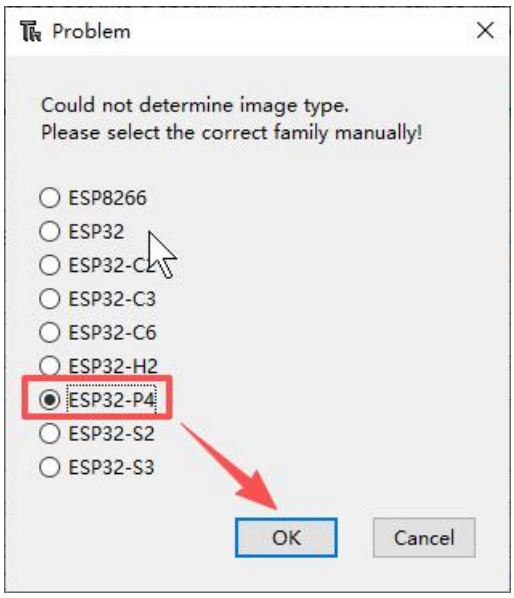
4. Click the icon with three horizontal lines, then click " Select local MicroPython image " , select "lvgl_micropy_ESP32_GENERIC_P4-C6_WIFI-16_ELECROW_INCH5_V1_0" and install it.

Please click the following link to download the bin file:

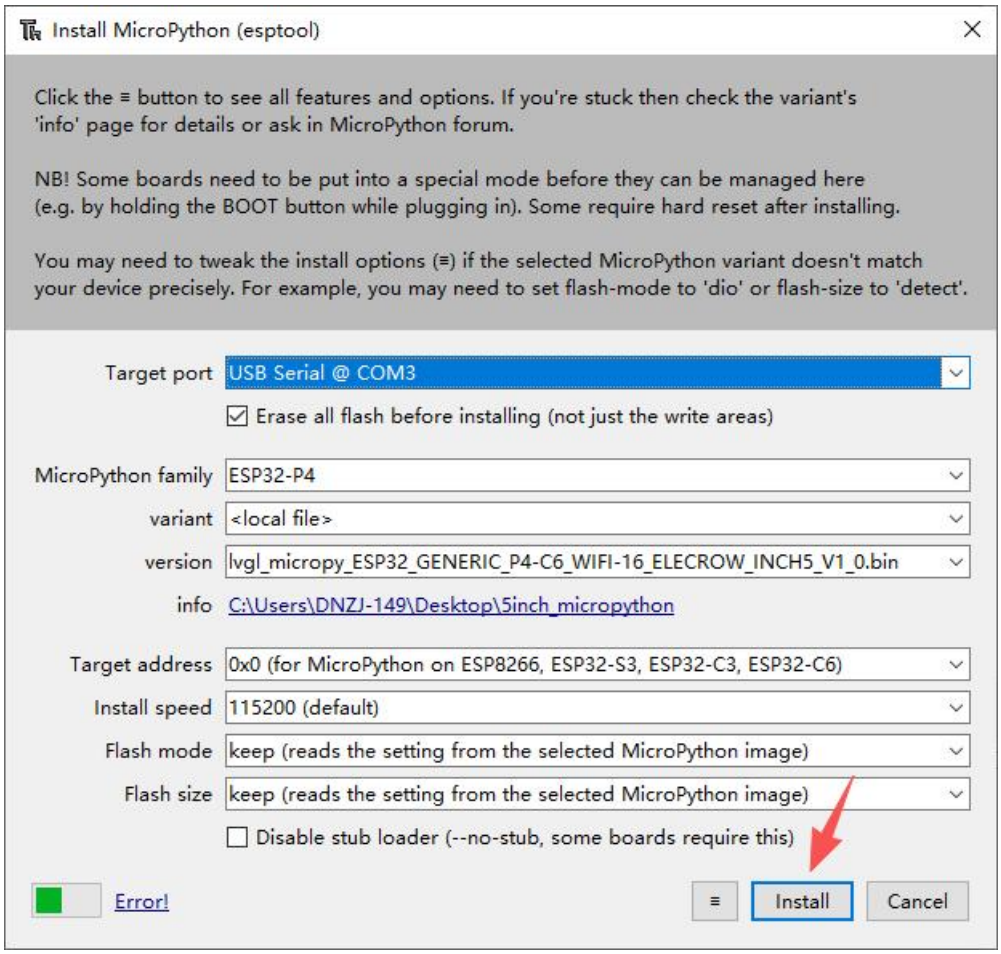
https://github.com/Elecrow-RD/-CrowPanel-Advanced-5inch-ESP32-P4-HMI-AI-Display-800x480-IPS-Touch-Screen/tree/master/example/V1.0/Micropython



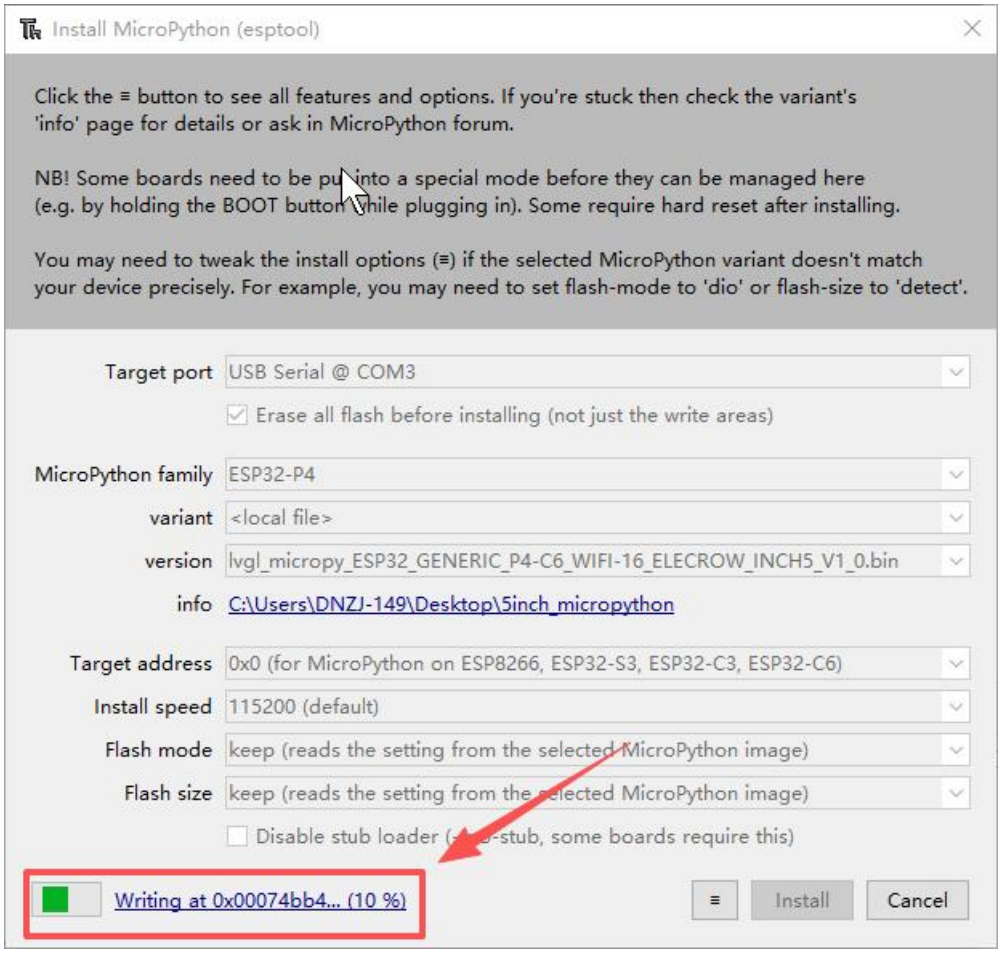
5. Select ESP32-P4

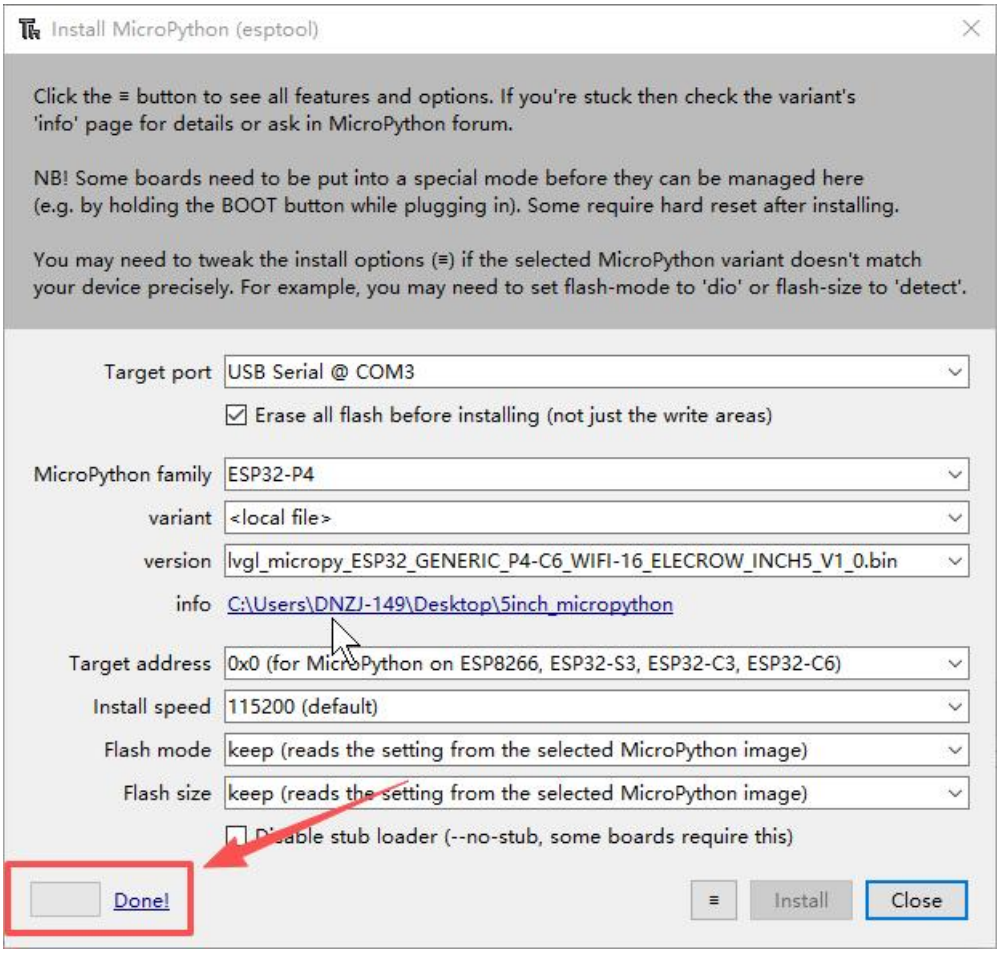


6. After selecting the serial port and other installation parameters, click Install.

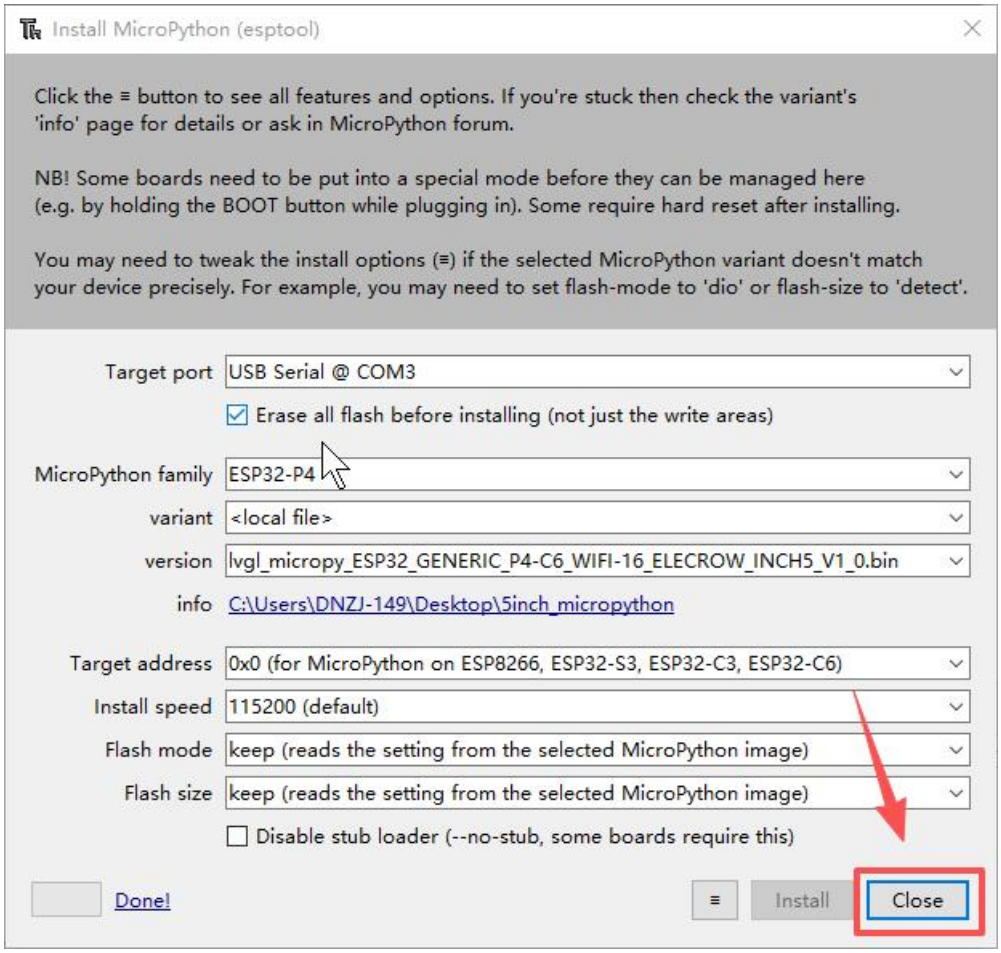


7. Waiting for installation and the installation success indicator.

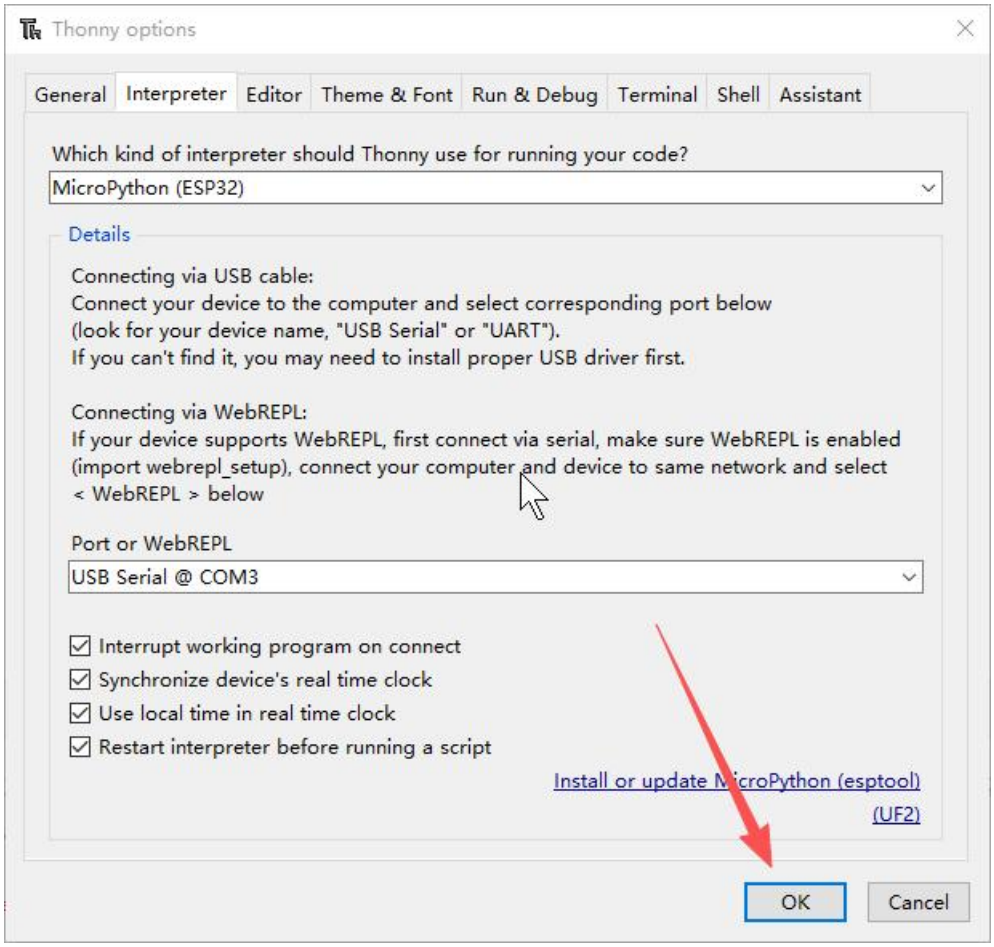

Install MicroPython (esptool)
Click the ≡ button to see all features and options. If you're stuck then check the variant's 'info' page for details or ask in MicroPython forum.
NB! Some boards need to be put into a special mode before they can be managed here (e.g. by holding the BOOT button while plugging in). Some require hard reset after installing.
You may need to tweak the install options (≡) if the selected MicroPython variant doesn't match your device precisely. For example, you may need to set flash-mode to 'dio' or flash-size to 'detect'.
Target port
USB Serial @ COM3
Erase all flash before installing (not just the write areas)
MicroPython family
ESP32-P4
variant
<local file>
version
lvgl_micropy_ESP32_GENERIC_P4-C6_WIFI-16_ELECROW_INCH5_V1_0.bin
info
C:\Users\DNZJ-149\Desktop\5inch_micropython
Target address
0x0 (for MicroPython on ESP8266, ESP32-S3, ESP32-C3, ESP32-C6)
Install speed
115200 (default)
Flash mode
keep (reads the setting from the selected MicroPython image)
Flash size
keep (reads the setting from the selected MicroPython image)
Disable stub loader
Writing at 0x00074bb4... (10 %)
≡
Install
Cancel



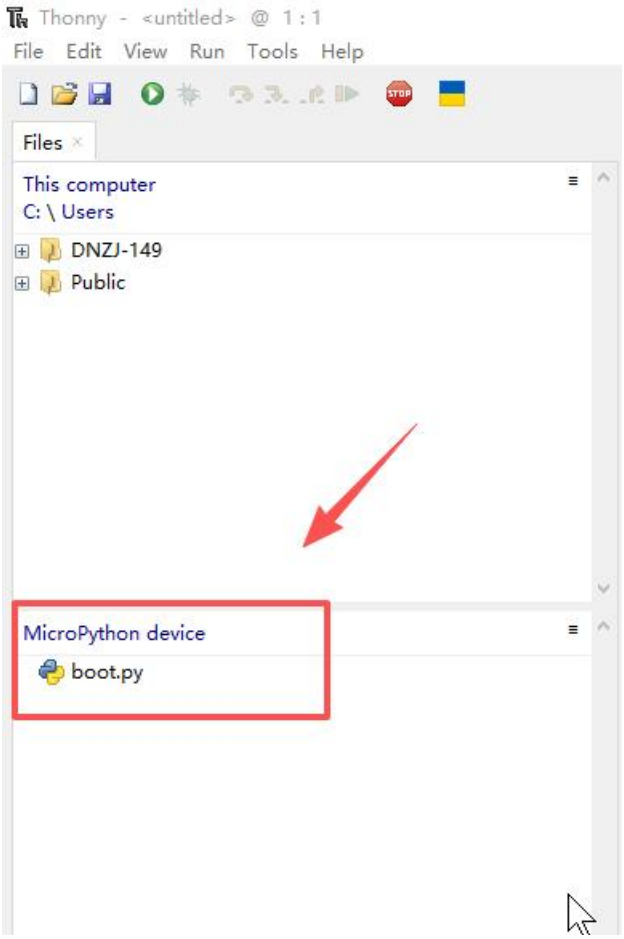
8. Click " Close "


Install MicroPython (esptool)
Click the ≡ button to see all features and options. If you're stuck then check the variant's 'info' page for details or ask in MicroPython forum.
NB! Some boards need to be put into a special mode before they can be managed here (e.g. by holding the BOOT button while plugging in). Some require hard reset after installing.
You may need to tweak the install options (≡) if the selected MicroPython variant doesn't match your device precisely. For example, you may need to set flash-mode to 'dio' or flash-size to 'detect'.
Target port
USB Serial @ COM3
Erase all flash before installing (not just the write areas)
MicroPython family
ESP32-P4
variant
<local file>
version
lvgl_micropy_ESP32_GENERIC_P4-C6_WIFI-16_ELECROW_INCH5_V1_0.bin
info
C:\Users\DNZJ-149\Desktop\5inch_micropython
Target address
0x0 (for MicroPython on ESP8266, ESP32-S3, ESP32-C3, ESP32-C6)
Install speed
115200 (default)
Flash mode
keep (reads the setting from the selected MicroPython image)
Flash size
keep (reads the setting from the selected MicroPython image)
Disable stub loader (--no-stub, some boards require this)
Done!
≡
Install
Close

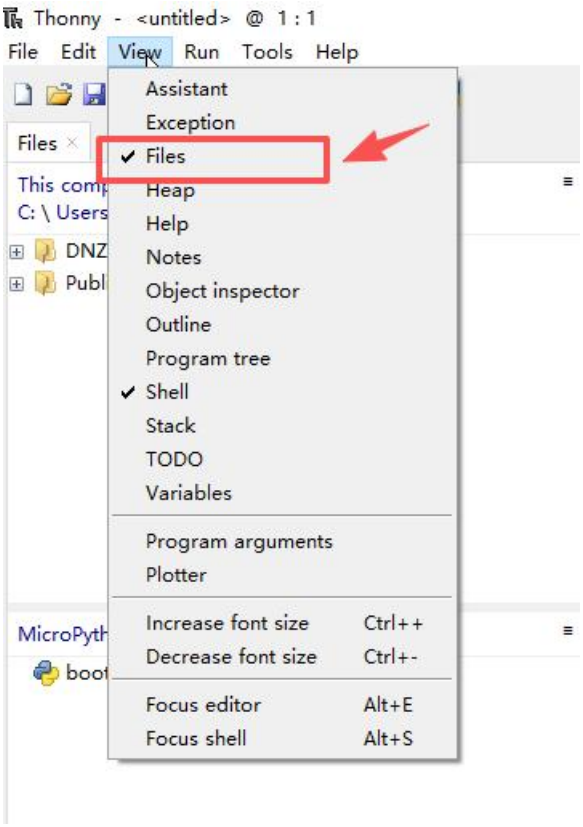
9. Click OK



10. If a " MicroPython device " window and a " boot.py " file appear in the lower left corner, the installation is successful.

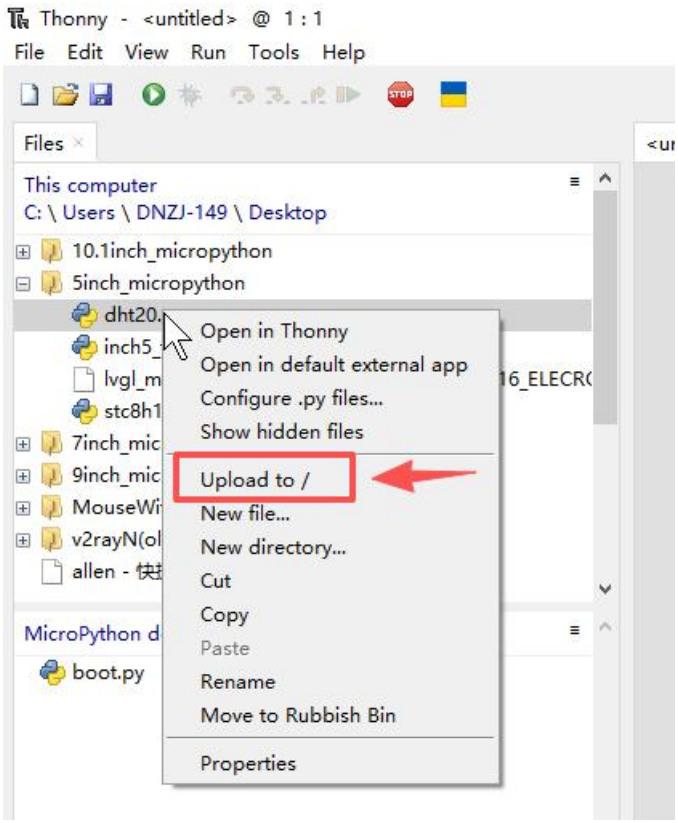


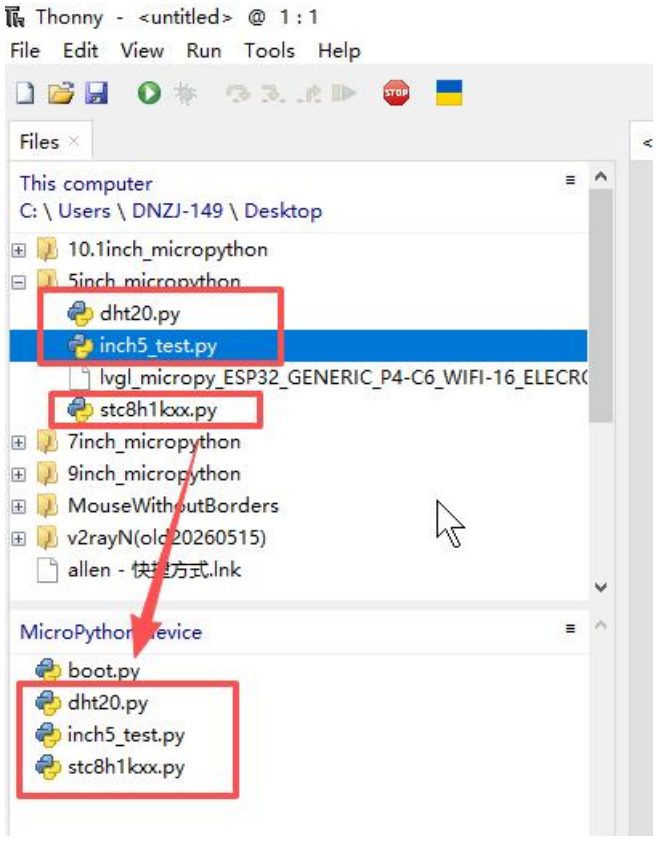
Note: If this window does not appear, you can open it by clicking " View " -> " File " at the top.



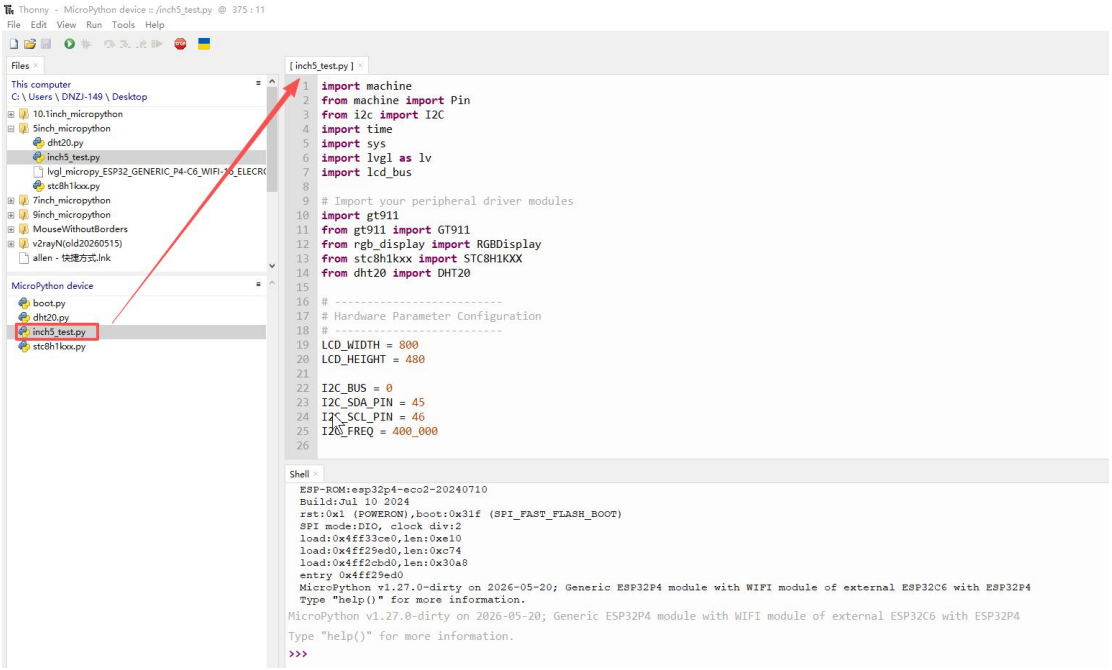
Upload code

1. In the " This computer " window, locate the downloaded code files and upload all four files to the " MicroPython device " in sequence.

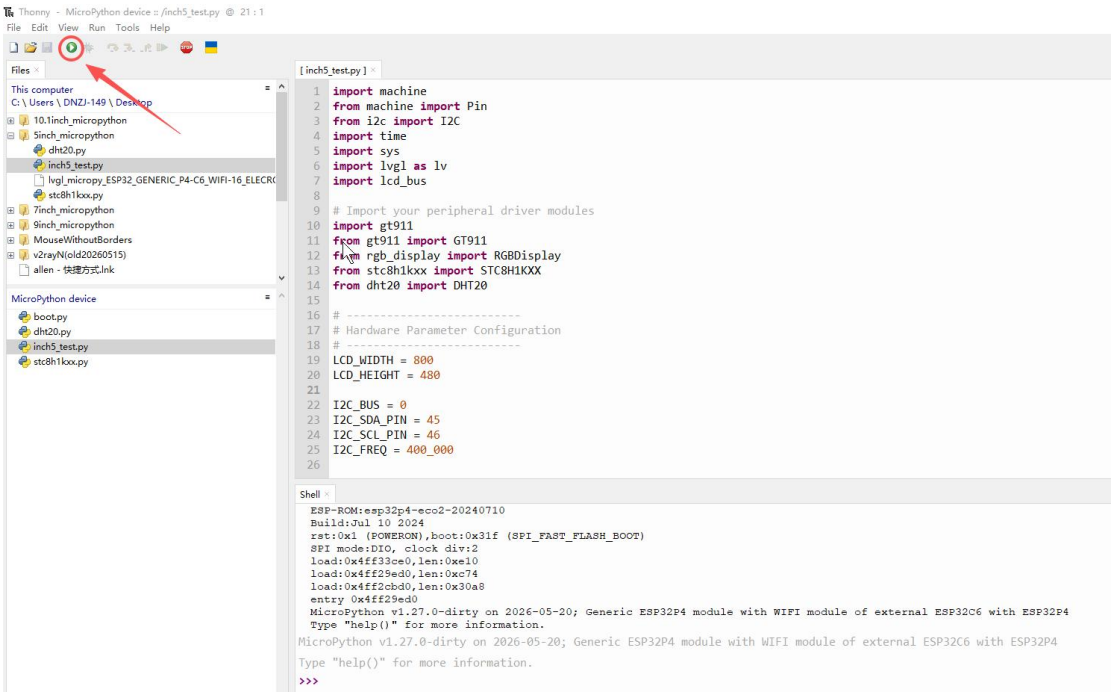




2. Double-click to open the main program " inch 5_test.py " .

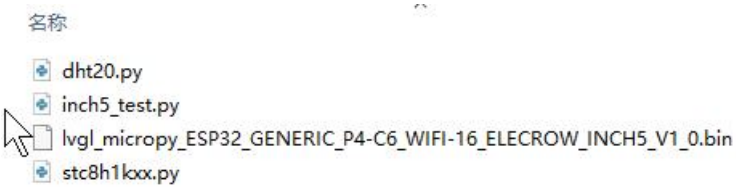


3. Click the Run button, and you will see the UI interface appear on the screen.



Code Explanation

The code files we downloaded contain three "python" files and one " bin" file.



dht20.py : The driver file for the temperature and humidity sensor .
stc8h1kxx.py : Used to communicate with the STC8 chip via the I2C bus to implement functions such as GPIO control and PWM dimming.
lvgl_micropy_ESP32_GENERIC_P4-C6_WIFI-16_ELECROW_INCH5_V1_0.bin :
Elecrow pre-packaged firmware that allows your Python scripts to display DHT20 sensor values on the LCD of the "CrowPanel Advanced 5 inch ESP32-P4 HMI" and control LED light switching via touch.
inch 5_test.py : This is the main program that runs the entire example; we will mainly explain this code file.

Import Module

```
import machine
from machine import Pin
from i2c import I2C
import time
import sys
import lvgl as lv
import lcd_bus
```

```
# Import your peripheral driver modules
import gt911
from gt911 import GT911
from rgb_display import RGBDisplay
from stc8h1kxx import STC8H1KXX
from dht20 import DHT20
```

machine : The MicroPython standard library that provides low-level hardware control (GPIO, I2C, etc.).
Pin : Used to control the input and output of GPIO pins.
I2C : An implementation of the I2C communication protocol used for communication with sensors.
time : Time-related functions (delay, timer, etc.)
sys : System-related functions (error handling, printing, etc.)
lvgl : A lightweight graphics library for creating user interfaces.
lcd_bus : LCD bus driver
GT911 : GT911 Capacitive Touchscreen Driver
RGBDisplay : RGB interface LCD display driver
STC8H1KXX : STC8H microcontroller driver (used to control backlight, etc.)
DHT20 : Driver for DHT20 temperature and humidity sensor

Hardware parameter configuration

```
LCD_WIDTH = 800
LCD_HEIGHT = 480
```

Define the screen resolution as 800x480 pixels.

I2C bus configuration

```
I2C_BUS = 0
I2C_SDA_PIN = 45
I2C_SCL_PIN = 46
I2C_FREQ = 400_000
```

Touch pin configuration

```
TOUCH_RST_PIN = 36
TOUCH_INT_PIN = 42
```

RST : Reset pin, used to restart the touch chip.
INT : Interrupt pin, triggered when a touch event occurs.

RGB screen timing configuration

```
# RGB Screen Timing Configuration
RGB_LCD_PIXEL_CLOCK_HZ     = (18_000_000) # Usually write as 18000000 is fine
RGB_LCD_HSYNC                = 4
RGB_LCD_HBP                  = 8
RGB_LCD_HFP                  = 8
RGB_LCD_VSYNC                = 4
```

```
RGB_LCD_VBP                 = 16
RGB_LCD_VFP                 = 16
```

RGB data pin

```
# RGB Screen Pin Configuration
RGB_PIN_NUM_HSYNC           = 40
RGB_PIN_NUM_VSYNC           = 41
RGB_PIN_NUM_DE              = 2
RGB_PIN_NUM_PCLK            = 3

# 16-bit RGB565 data pins
RGB_PIN_NUM_DATA0           = 8
RGB_PIN_NUM_DATA1           = 7
RGB_PIN_NUM_DATA2           = 6
RGB_PIN_NUM_DATA3           = 5
RGB_PIN_NUM_DATA4           = 4
RGB_PIN_NUM_DATA5           = 14
RGB_PIN_NUM_DATA6           = 13
RGB_PIN_NUM_DATA7           = 12
RGB_PIN_NUM_DATA8           = 11
RGB_PIN_NUM_DATA9           = 10
RGB_PIN_NUM_DATA10          = 9
RGB_PIN_NUM_DATA11          = 19
RGB_PIN_NUM_DATA12          = 18
RGB_PIN_NUM_DATA13          = 17
RGB_PIN_NUM_DATA14          = 16
RGB_PIN_NUM_DATA15          = 15
```

LED pins

```
LED = Pin(48, Pin.OUT)# Set GPIO pin 48 to output mode
```

Set GPIO48 to output mode to control the LED.

The device initialization function, device_init(), is the initialization function for the entire hardware system.

```
def device_init():
    print("Initializing peripherals...")

    # 1. Initialize LVGL
    if not lv.is_initialized():
        lv.init()

    # 2. Initialize I2C bus (for STC8 controller, GT911 touch, DHT20 temperature/humidity sensor)
    i2c_bus = I2C.Bus(
        host = I2C_BUS,
```

```
        sda=I2C_SDA_PIN,
        scl=I2C_SCL_PIN,
        freq=I2C_FREQ
    )
    print(f"\n Scanning I2C bus devices: {[hex(d) for d in i2c_bus.scan()]}")

    i2c_stc8 = I2C.Device(
        bus = i2c_bus,
        dev_id = STC8H1KXX.I2C_ADDR,
        reg_bits=8
    )
    # 3. Initialize STC8 and turn on backlight
    stc8 = STC8H1KXX(i2c_stc8)
    print("Enabling LCD backlight power and PWM...")
    stc8.set_gpio_level(STC8H1KXX.GPIO_OUT_LCD_BL_POWER, 1) # Enable LCD power supply
     stc8.set_pwm_duty(STC8H1KXX.PWM_LCD_BL_EN, 50)              # Set backlight brightness to
50%

    i2c_dht20 = I2C.Device(
        bus = i2c_bus,
        dev_id = DHT20.I2C_ADDR,
        reg_bits=8
    )

    #Initialize DHT20 sensor
    try :
        dht20 = DHT20(i2c_dht20)
    except Exception as e:
        sys.print_exception(e)
        print('Failed to initialize DHT20 sensor!')

    # 4. Initialize underlying RGB bus (lcd_bus.RGBBus)
    # Modify parameter names according to your lcd_bus.pyi file definition
    print("Configuring RGB bus...")
    rgb_bus = lcd_bus.RGBBus(
        hsync=RGB_PIN_NUM_HSYNC,
        vsync=RGB_PIN_NUM_VSYNC,
        de=RGB_PIN_NUM_DE,
        pclk=RGB_PIN_NUM_PCLK,
        data0=RGB_PIN_NUM_DATA0,
        data1=RGB_PIN_NUM_DATA1,
        data2=RGB_PIN_NUM_DATA2,
        data3=RGB_PIN_NUM_DATA3,
        data4=RGB_PIN_NUM_DATA4,
```

```
        data5=RGB_PIN_NUM_DATA5,
        data6=RGB_PIN_NUM_DATA6,
        data7=RGB_PIN_NUM_DATA7,
        data8=RGB_PIN_NUM_DATA8,
        data9=RGB_PIN_NUM_DATA9,
        data10=RGB_PIN_NUM_DATA10,
        data11=RGB_PIN_NUM_DATA11,
        data12=RGB_PIN_NUM_DATA12,
        data13=RGB_PIN_NUM_DATA13,
        data14=RGB_PIN_NUM_DATA14,
        data15=RGB_PIN_NUM_DATA15,
        freq=RGB_LCD_PIXEL_CLOCK_HZ,
        hsync_pulse_width=RGB_LCD_HSYNC,
        hsync_back_porch=RGB_LCD_HBP,
        hsync_front_porch=RGB_LCD_HFP,
        vsync_pulse_width=RGB_LCD_VSYNC,
        vsync_back_porch=RGB_LCD_VBP,
        vsync_front_porch=RGB_LCD_VFP,
        pclk_active_low=False,                  # Default value
    )

    # 5. Instantiate RGBDisplay driver and mount to LVGL
    print("Registering display to LVGL...")
    display = RGBDisplay(
        data_bus=rgb_bus,
        display_width=LCD_WIDTH,
        display_height=LCD_HEIGHT,
        color_space=lv.COLOR_FORMAT.RGB565 # 16 pins correspond to RGB565
    )
    display.init()

    # 6. Initialize GT911 touch screen
    print("Initializing GT911 touch driver...")

    i2c_gt911 = I2C.Device(
        bus = i2c_bus,
        dev_id = gt911.I2C_ADDR,
        reg_bits=16
    )
    # Assuming gt911 module can be instantiated with i2c bus and auto-register to LVGL indev
    touch = GT911(
        device=i2c_gt911,
        reset_pin=TOUCH_RST_PIN,
        interrupt_pin=TOUCH_INT_PIN,
```

```
    )

    # Define a touch callback function that meets LVGL requirements
    def touch_read_cb(indev, data):
        # Assuming touch._get_coords() returns format like (press_state, x, y)
        # You need to adjust according to your actual gt911 library return values
        coords = touch._get_coords()
        if coords:
            # Assuming your library returns non-empty for pressed, or parse state, x, y specifically
            # Here is a common format example:
            data.point.x = coords[0] # Your X coordinate
            data.point.y = coords[1] # Your Y coordinate
            data.state = lv.INDEV_STATE.PRESSED
            #print(data.point.x, data.point.y, data.state)
        else:
            data.state = lv.INDEV_STATE.RELEASED

    indev = lv.indev_create()
    indev.set_type(lv.INDEV_TYPE.POINTER)
    indev.set_read_cb(touch_read_cb)
    #indev.set_display(display) # Only needed for multiple displays, single display default is fine

    return display, touch, stc8, dht20
```

helper function SetFlag()

```
def SetFlag(obj, flag, value):
    if (value):
        obj.add_flag(flag)
    else:
        obj.remove_flag(flag)
    return
```

The unified object flag setting functions are : **'obj.add_flag(flag)'** : adds a flag; **'obj.remove_flag(flag)'** : removes a flag.

Button event handler function - turn on the light

```
def Button1_eventhandler(event_struct):
  event = event_struct.get_code()
  if event == lv.EVENT.CLICKED and True:
     LED.value(1)
  return
```

When the button is clicked, LED.value(1) lights up the LED.

Button event handler function - turn off the lights

```
def Button2_eventhandler(event_struct):
    event = event_struct.get_code()
```

```
    if event == lv.EVENT.CLICKED and True:
        LED.value(0)
    return
```

When the button is clicked, LED.value(1) turns off the LED.

UI creation function — **create_lvgl_ui()**

Create screen

```
ui_Screen1 = lv.obj()
SetFlag(ui_Screen1, lv.obj.FLAG.SCROLLABLE, False)
ui_Screen1.set_style_bg_color(lv.color_hex(0xEFF6DB), lv.PART.MAIN | lv.STATE.DEFAULT)
ui_Screen1.set_style_bg_opa(255, lv.PART.MAIN| lv.STATE.DEFAULT)
```

'lv.obj()' creates a base object (the screen is the top-level object) and sets its background color to light green **(0xEFF6DB). 'bg_opa=255'** sets the background opacity to 100%.

Create Button 1

```
ui_Button1 = lv.button(ui_Screen1)
ui_Button1.set_width(85)
ui_Button1.set_height(85)
ui_Button1.set_x(208)
ui_Button1.set_y(-90)
ui_Button1.set_align(lv.ALIGN.CENTER)
```

lv.button(ui_Screen1) creates a button on the screen with a width and height of 85x85 pixels, offset from the center of the screen by (208, -90).

Gradient color settings

```
ui_Button1.set_style_bg_grad_dir(lv.GRAD_DIR.VER, lv.PART.MAIN | lv.STATE.DEFAULT)
ui_Button1.set_style_bg_grad_color(lv.color_hex(0xFFF9C4), lv.PART.MAIN | lv.STATE.DEFAULT)
ui_Button1.set_style_bg_color(lv.color_hex(0xFFE032), lv.PART.MAIN | lv.STATE.DEFAULT)
```

Gradient from light yellow **0xFFF9C4** to dark yellow **0xFFE032** , creating a 3D button effect.

Pressing status style

```
ui_Button1.set_style_bg_grad_dir(lv.GRAD_DIR.VER, lv.PART.MAIN | lv.STATE.PRESSED)
ui_Button1.set_style_bg_grad_color(lv.color_hex(0xFFE032), lv.PART.MAIN | lv.STATE.PRESSED)
ui_Button1.set_style_bg_color(lv.color_hex(0xFFE032), lv.PART.MAIN | lv.STATE.PRESSED)
```

When pressed, the color turns a uniform dark yellow, indicating a "press" feedback.

Shadow effect

```
ui_Button1.set_style_shadow_width(8, lv.PART.MAIN | lv.STATE.DEFAULT)
ui_Button1.set_style_shadow_color(lv.color_hex(0xFFEB3B), lv.PART.MAIN | lv.STATE.DEFAULT)
```

Add an 8-pixel-wide yellow shadow to enhance the button's three-dimensionality and glow.

Label on the button

```
ui_LabelOn = lv.label(ui_Button1)
ui_LabelOn.set_text("ON")
ui_LabelOn.set_align(lv.ALIGN.CENTER)
ui_LabelOn.set_style_text_color(lv.color_hex(0x000000), lv.PART.MAIN | lv.STATE.DEFAULT)
ui_LabelOn.set_style_text_font(lv.font_montserrat_40, lv.PART.MAIN | lv.STATE.DEFAULT)
```

Create a label inside the button that displays the text "ON" using the Montserrat 40 font.

Button 2 (Lights off button)

```
ui_Button2 = lv.button(ui_Screen1)
    ui_Button2.set_width(85)
    ui_Button2.set_height(85)
    ui_Button2.set_x(205)
    ui_Button2.set_y(40)
    ui_Button2.set_align( lv.ALIGN.CENTER)
    SetFlag(ui_Button2, lv.obj.FLAG.SCROLLABLE, False)
    SetFlag(ui_Button2, lv.obj.FLAG.SCROLL_ON_FOCUS, True)
    # ========== LVGL 9.4 MicroPython Correct gradient settings（Default state） ==========
    # Set gradient direction (top to bottom)
    ui_Button2.set_style_bg_grad_dir(lv.GRAD_DIR.VER, lv.PART.MAIN | lv.STATE.DEFAULT)
    # Set gradient start color (top)
                ui_Button2.set_style_bg_grad_color(lv.color_hex(0x9E9E9E),     lv.PART.MAIN     |
lv.STATE.DEFAULT)
    # Set gradient end color (bottom)
    ui_Button2.set_style_bg_color(lv.color_hex(0x616161), lv.PART.MAIN | lv.STATE.DEFAULT)
    ui_Button2.set_style_bg_opa(255, lv.PART.MAIN| lv.STATE.DEFAULT )
          #    ui_Button2.set_style_bg_img_src(    ui_images.ui_img_off_png,    lv.PART.MAIN    |
lv.STATE.DEFAULT )
    # ========== LVGL 9.4 MicroPython Correct gradient settings（Pressed state） ==========
    ui_Button2.set_style_bg_grad_dir(lv.GRAD_DIR.VER, lv.PART.MAIN | lv.STATE.PRESSED)
                ui_Button2.set_style_bg_grad_color(lv.color_hex(0x616161),     lv.PART.MAIN     |
lv.STATE.PRESSED)
    ui_Button2.set_style_bg_color(lv.color_hex(0x616161), lv.PART.MAIN | lv.STATE.PRESSED)
    ui_Button2.set_style_bg_opa(255, lv.PART.MAIN| lv.STATE.PRESSED )
    ui_Button2.add_event_cb(Button2_eventhandler, lv.EVENT.ALL, None)

    ui_LabelOff = lv.label(ui_Button2)
    ui_LabelOff.set_text("OFF")
    ui_LabelOff.set_align( lv.ALIGN.CENTER)
    ui_LabelOff.set_style_text_color(lv.color_hex(0x000000), lv.PART.MAIN | lv.STATE.DEFAULT )
    ui_LabelOff.set_style_text_opa(255, lv.PART.MAIN| lv.STATE.DEFAULT )
    ui_LabelOff.set_style_text_font( lv.font_montserrat_40, lv.PART.MAIN | lv.STATE.DEFAULT )
```

Button 2 has a similar structure to button 1, but its color is changed to gray . Default

state: Gradient from light gray **0x9E9E9E** to dark gray **0x616161 .** Pressed state: Turns to a consistent dark gray and displays the text "OFF".

Image components

```
ui_Image1 = lv.image(ui_Screen1)
ui_Image1.set_width(lv.SIZE_CONTENT)
ui_Image1.set_height(lv.SIZE_CONTENT)
ui_Image1.set_x(-90)
ui_Image1.set_y(-14)
ui_Image1.set_align(lv.ALIGN.CENTER)
```

Create an image object , where **lv.SIZE_CONTENT** is set to adapt the size to the content size.

Temperature label

```
ui_Label1 = lv.label(ui_Screen1)
ui_Label1.set_text("")
ui_Label1.set_width(450)
ui_Label1.set_height(60)
ui_Label1.set_x(-100)
ui_Label1.set_y(-89)
ui_Label1.set_style_radius(10, lv.PART.MAIN | lv.STATE.DEFAULT)
```

Create a label to display the temperature , **set_style_radius(10)** : corner radius 10 pixels. Set the background to a blue gradient.

Humidity label

```
ui_Label2 = lv.label(ui_Screen1)
    ui_Label2.set_text("")
    ui_Label2.set_width(450)     # 1
    ui_Label2.set_height(60)    # 1
    ui_Label2.set_x(-100)
    ui_Label2.set_y(40)
    ui_Label2.set_style_radius(10, lv.PART.MAIN | lv.STATE.DEFAULT)
    ui_Label2.set_align( lv.ALIGN.CENTER)
    ui_Label2.set_style_bg_grad_dir(lv.GRAD_DIR.VER, lv.PART.MAIN | lv.STATE.DEFAULT)
    ui_Label2.set_style_bg_grad_color(lv.color_hex(0xBBDEFB), lv.PART.MAIN | lv.STATE.DEFAULT)
    ui_Label2.set_style_bg_color(lv.color_hex(0x0000FB), lv.PART.MAIN | lv.STATE.DEFAULT)
    ui_Label2.set_style_bg_opa(255, lv.PART.MAIN| lv.STATE.DEFAULT )
    # ui_Label2.set_style_pad_ver(lv.SIZE_CONTENT, lv.PART.MAIN | lv.STATE.DEFAULT)
    # ui_Label2.set_style_text_align(lv.TEXT_ALIGN.CENTER, lv.PART.MAIN | lv.STATE.DEFAULT)
    ui_Label2.set_style_text_color(lv.color_hex(0x000000), lv.PART.MAIN | lv.STATE.DEFAULT )
    ui_Label2.set_style_text_opa(255, lv.PART.MAIN| lv.STATE.DEFAULT )
    ui_Label2.set_style_text_font( lv.font_montserrat_40, lv.PART.MAIN | lv.STATE.DEFAULT )
```

It has the same structure as the temperature label and is located at the bottom of the screen .

Main function — main()

initialization

```
try:
    display, touch, stc8, dht20 = device_init()
    ui_Screen1, ui_Label1, ui_Label2 = create_lvgl_ui()
    LED.value(1)
```

Initialize all hardware devices , create the UI interface, and light up the LEDs (indicating system startup).

First reading of temperature and humidity values

```
dht20.measure()
temp = dht20.temperature()
hum = dht20.humidity()
ui_Label1.set_text(f"temperature: {round(temp, 1)} C")
ui_Label2.set_text(f"humidity: {round(hum)} %")
```

dht20.measure() : Triggers sensor measurement

temperature() : Gets the temperature value

humidity() : Gets the humidity value

round(temp, 1) : Round to one decimal place.

Display screen

```
lv.screen_load(ui_Screen1)
```

Load the screen into the display buffer and begin rendering the UI.

Main loop

```
last_dht_time = time.ticks_ms()

while True:
    current_time = time.ticks_ms()
        if time.ticks_diff(current_time, last_dht_time) >= 1000:
        dht20.measure()
        temp = dht20.temperature()
        hum = dht20.humidity()
        ui_Label1.set_text(f"temperature: {round(temp, 1)} C")
        ui_Label2.set_text(f"humidity: {round(hum)} %")
        last_dht_time = current_time
    lv.tick_inc(10)
    lv.timer_handler()
    time.sleep_ms(10)
```

The main loop has three main tasks:

1. Periodically read sensor data (non-blocking)

Use **time.ticks_ms()** to record time.

time.ticks_diff() calculates the time difference, reading it every 1000ms without

affecting UI responsiveness.

2. LVGL Heartbeat

lv.tick_inc(10) : tells LVGL that 10 milliseconds have passed.

lv.timer_handler() : Handles all UI updates, animations, and touch events.

3. Sleep mode controls frame rate

time.sleep_ms(10) : Sleep for 10ms each time the loop is repeated , which means the loop frequency is about 100Hz, and the UI refresh is smooth enough.

Exception handling

```
except Exception as e:
    sys.print_exception(e)
```

Capture all exceptions and print stack traces for easier debugging.