E32R32P&E32N32P 3.2inch MicroPython Demo Instructions

CONTENTS

1. Software and hardware platform description

Module: 3.2-inch ESP32-32E display module with 240x320 resolution and ST7789 screen driver IC.

Module master: ESP32-WROOM-32E module, the highest main frequency 240MHz, support 2.4G WIFI+ Bluetooth.

Thonny version: 4.1.6

ESP32 MicroPython firmware version: 1.23.0.

2. Pin allocation instructions

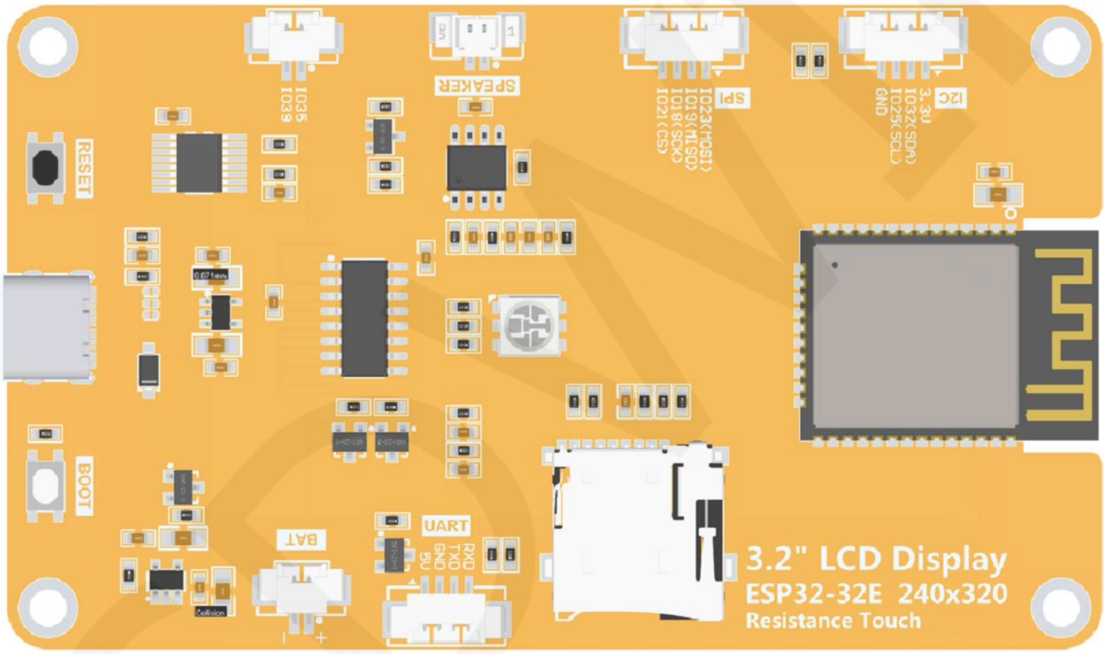

Figure 2.1 Rear view of 3.2-inch ESP32-32E display module

The main controller of the 3.2-inch ESP32 display module is ESP32-32E, and the GPIO allocation for its onboard peripherals is shown in the table below:

ESP32-32E pin allocation instructions			
On board device	**On board device pins**	**ESP32-32E connection pin**	**description**
LCD	**TFT_CS**	IO15	LCD screen chip selection control signal, low level effective
	TFT_RS	IO2	LCD screen command/data selection control signal.High level: data, low level: command

	TFT_SCK	IO14	SPI bus clock signal (shared by LCD screen and touch screen)	
	TFT_MOSI	IO13	SPI bus writes data signals (shared by LCD screen and touch screen)	
	TFT_MISO	IO12	SPI bus reading data signal (shared by LCD screen and touch screen)	
	TFT_RST	EN	LCD screen reset control signal, low level reset (shared reset pin with ESP32-32E main control)	
	TFT_BL	IO27	LCD screen backlight control signal (high level lights up the backlight, low level turns off the backlight)	
RTP	TP_SCK	IO14	SPI bus clock signal (shared by touch screen and LCD screen)	
	TP_DIN	IO13	SPI bus writes data signals (shared by touch screen and LCD screen)	
	TP_DOUT	IO12	SPI bus reading data signal (shared by touch screen and LCD screen)	
	TP_CS	IO33	Resistance touch screen chip selection control signal, low level effective	
	TP_IRQ	IO36	Resistive touch screen touch interrupt signal, when a touch is generated, input a low level to the main control	
LED	LED_RED	IO22	Red LED light	RGB tri color LED light, with a common anode, lit at low level and turned off at high level.
	LED_GREEN	IO16	Green LED light	
	LED_BLUE	IO17	Blue LED light	
SDCARD	SD_CS	IO5	SD card signal selection, low level effective	
	SD_MOSI	IO23	SD card SPI bus write data signal	
	SD_SCK	IO18	SD card SPI bus clock signal	
	SD_MISO	IO19	SD card SPI bus read data signal	
BATTERY	BAT_ADC	IO34	Battery voltage ADC value acquisition signal (input)	
Audio	Audio_ENABLE	IO4	Audio enable signal, low-level enable, high-level disable	
	Audio_DAC	IO26	Audio signal DAC output signal	
KEY	BOOT_KEY	IO0	Download mode selection button (press and hold the button to power on, then	

			release it to enter download mode)
	RESET_KEY	EN	ESP32-23E reset button, low level reset (shared with LCD screen reset)
Serial Port	**RX0**	RXD0	ESP32-32E serial port receiving signal
	TX0	TXD0	ESP32-32E serial port sends signal
POWER	**TYPE-C_POWER**	/	Type-C power interface, connected to 5V voltage.

Table 2.1 Pin allocation instructions for ESP32-32E onboard peripherals

3. Instructions for the example program

3.1. Set up ESP32 MicroPython development environment

For detailed instructions on setting up the "**MicroPython_development_environment_construction_for_ESP32**", please refer to the document

3.2. Upload files

After the development environment is set up, the relevant files need to be uploaded to the ESP32 device in order to run the testing program.

Before uploading the file, please familiarize yourself with the directory contents of the MicroPython sample program. Open the "**1-示例程序_Demo\MicroPython**" directory in the package, as shown in the following figure:

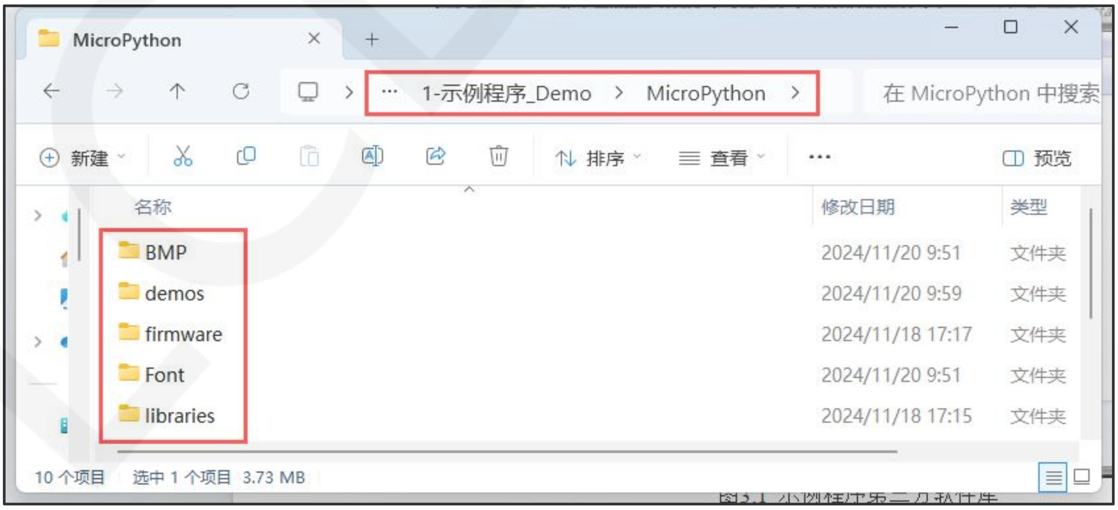

Figure 3.1 MicroPython Example Program Catalog

The contents of each folder are described as follows:

BMP: Stores BMP format images that sample programs need to use.

demos: Contains sample programs

firmware: Stores MicroPython firmware (needs to be burned when setting up the development environment)

Font: Stores the Chinese and English character modulo data that the sample program needs to use.

libraries: Stores MicroPython library files that sample programs need to use

After understanding the directory contents of the MicroPython sample program, the next step is to upload the program file to the ESP32 device. The steps are as follows:

A. Connect the ESP32 display module to the computer and power it on using a USB cable.

B. Open the Thonny software and configure the MicroPython interpreter for ESP32, as shown in the following figure:

(If already configured, this step can be omitted)

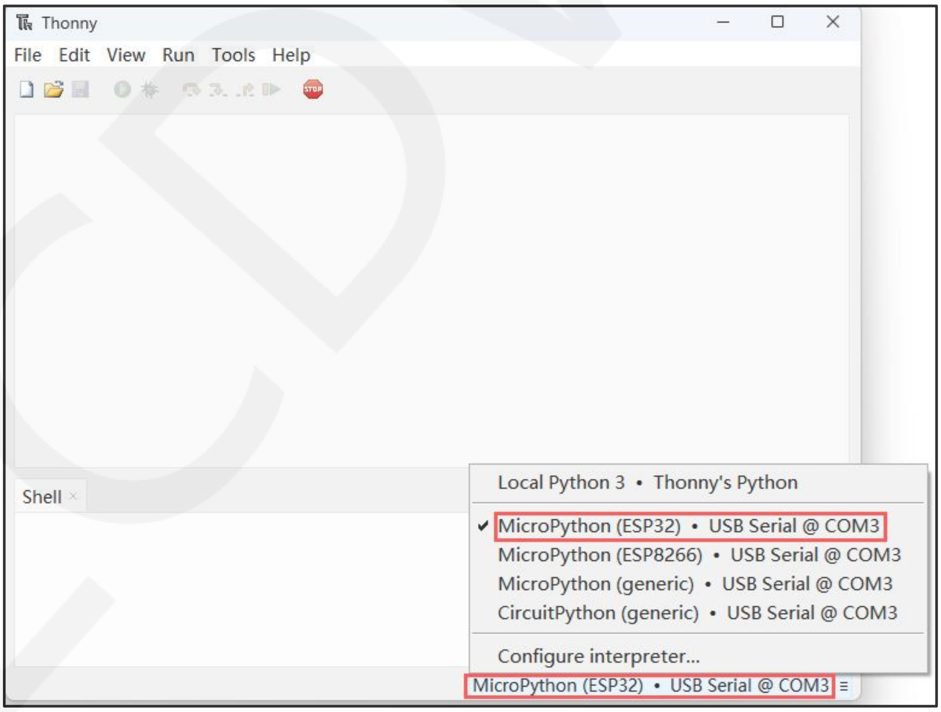

Figure 3.2 Selecting MicroPython interpreter

C. Click the toolbar  button to connect the ESP32 device. If the following prompt appears in the shell information bar, it indicates that the device connection is successful.

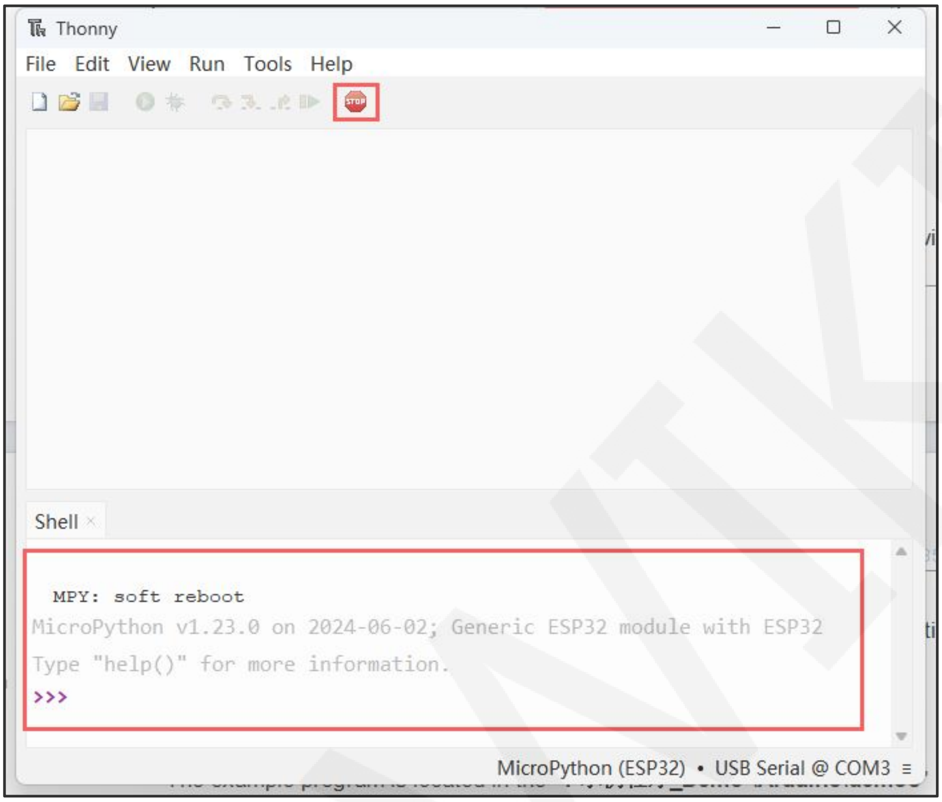

Figure 3.3 Connecting ESP32 devices

D. Click the "**View ->Files**" button to open the file window (ignore this operation if it is already open). Find the "**1-示例程序_Demo\MicroPython**" directory in the package in the window, left click the mouse to select the target file in the directory, and right-click on the standalone mouse to select "**Upload to /**" to upload the target file. As shown in the following figure:

Please note that when uploading files, ESP32 cannot run any programs, otherwise the upload will fail

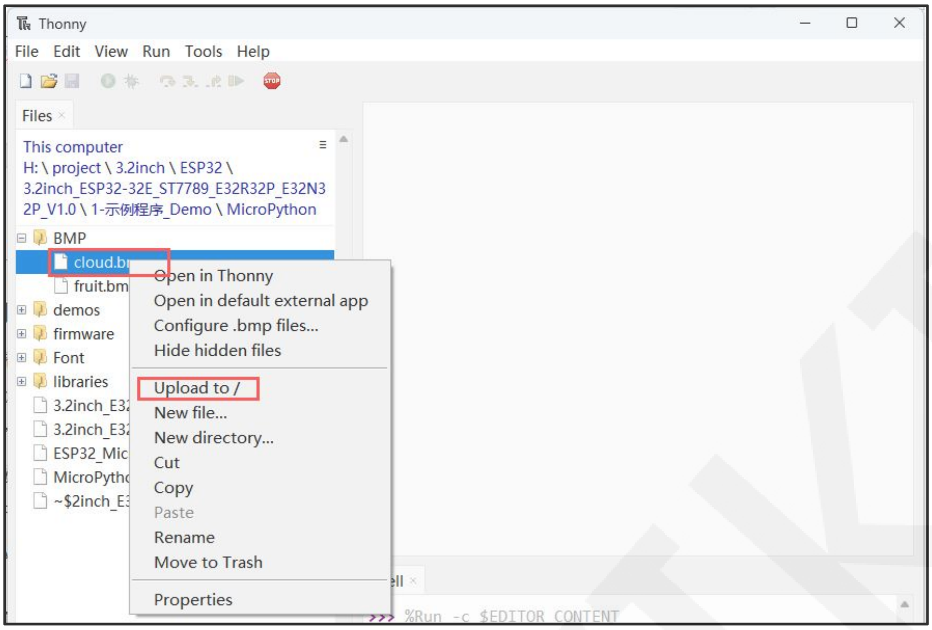

Figure 3.4 Uploading Files to ESP32 Devices

E. Upload the files from the "**BMP**", "**Font**", and "**libraries**" directories to the ESP32 device using the above method. The files in the '**demos**' directory can be transferred or not. As shown in the following figure:

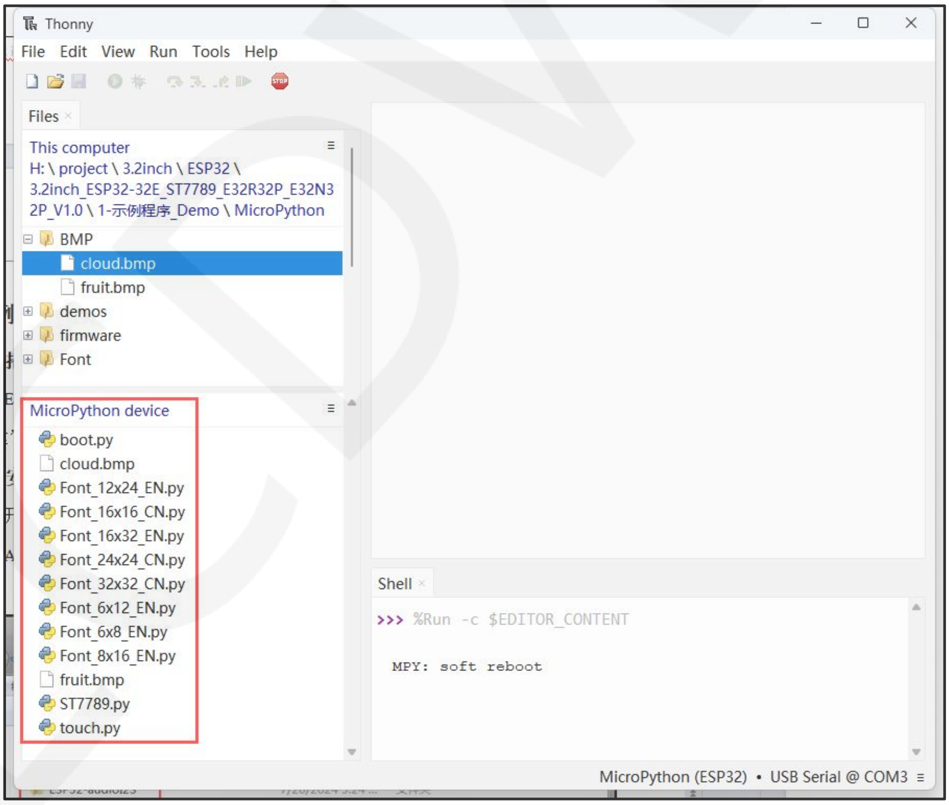

Figure 3.5 Completed file upload

3.3. Example Program Usage Instructions

The sample program is located in the "**1-示例程序_Demo\MicroPython\demos**" directory of the package, as shown in the following: figure:

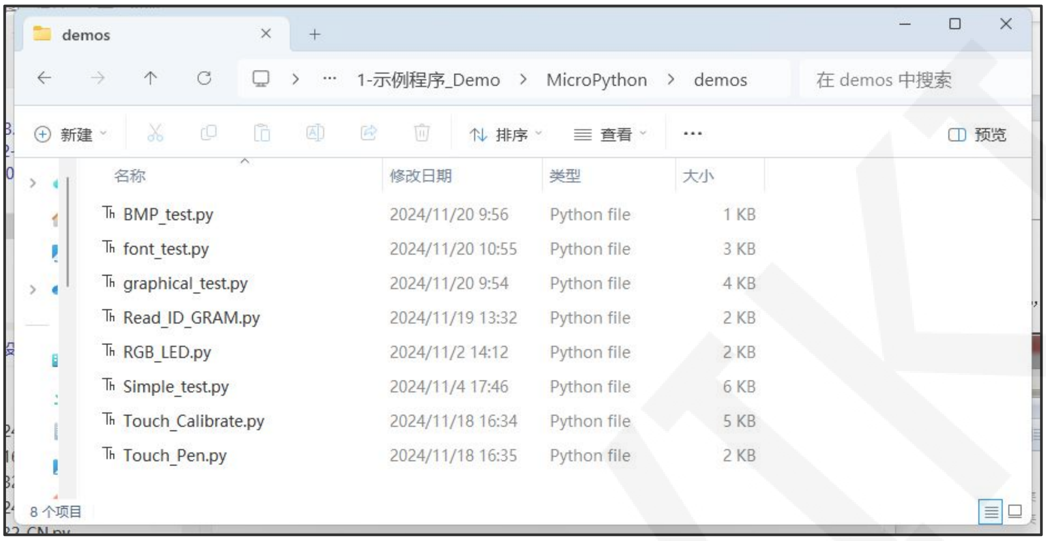

Figure 3.6 Example Program

The sample program can be uploaded to an ESP32 device to open and run, or it can be opened and run on a local computer. If you need to power on the ESP32 display module to run automatically, you need to change the sample program name to "**main. py**" and upload it to the ESP32 display module.

In the Python software, open the target sample program, click the menu bar  button, and you can run it. If the operation fails, the ESP32 device needs to be reconnected.

The introduction of each example program is as follows:

BMP_test.py

This example program relies on the ST7789.py library to display images in BMP format

font_test.py

This example program relies on the ST7789.py library to display Chinese and English characters of various sizes. The font modeling data needs to be saved in the font file according to the relevant format. For instructions on character casting, please refer to the following website:

http://www.lcdwiki.com/Chinese_and_English_display_modulo_settings

graphical_test.py

This example program relies on the ST7789.py library to display graphics such as points, lines, rectangles, rounded rectangles, triangles, circles, ellipses, etc. for drawing and filling, as well as setting display orientation.

Read_ID_GRAM.py

This example program relies on the ST7789.py library to display LCD ID and RGAM color value readings.

RGB_LED.py

This example hardware requires the use of RGB tri color lights to display the on/off and brightness adjustment of the RGB tri color lights.

Simple_test.py

This example does not rely on any software libraries and displays simple screen scrolling content.

Touch_Calibrate.py

This example relies on the ST7789.py library and the touch.exe library, displaying the calibration of a resistive touch screen. Follow the prompts displayed on the screen. After calibration is completed, the calibration parameters are output through the serial port and copied to the initialization of the sample program. Please note that the touch screen should be calibrated according to the display direction. The display direction in this program can be modified, as shown in the following figure:

```
if __name__=='__main__':
    coord = [0xFFFF, 0xFFFF]
    val = [0, 0, 0, 0, 0, 0, 0, 0]
    mylcd.LCD_Set_Rotate(1)
    mylcd.LCD_Clear(0)
    mylcd.Show_String((mylcd.lcd_width - 208) // 2, (mylcd.lcd_height-16) /
    for i in range(4):
        mylcd.Fill_Rect(0, 0, mylcd.lcd_width, 16, 0)
        mylcd.Fill_Rect(0, mylcd.lcd_height-16, mylcd.lcd_width, 16, 0)
        switch = {
            0: case_0,
            1: case_1,
            2: case_2,
```

Figure 3.7 Modifying the Touch Calibration Display Direction

Touch_Pen.py

This example relies on the ST7796.py library and the touch.exe library, displaying the operation of drawing dots and lines on the touch screen.