

ESP-IDF_LVGL porting instructions

CONTENTS

1. Create a new VS Code project.....	3
2. Copy engineering files.....	13
3. Configure LVGL.....	15
3.1. Configure lvgl through the menuconfig configuration interface.....	15
3.2. Configure lvgl through files.....	20
4. Configure LCD.....	21
5. Configure touch screen.....	24
6. Modify engineering documents.....	28
6.1. Modify the contents of the lv_porting folder.....	28
6.2. Modify the main.c file.....	35
7. Debugging project code.....	36
8. Burn and run.....	42

1. Create a new VS Code project

After the ESP-IDF VSCode development environment is set up, the next step is to create a new project project. The steps for creating a new project engineering are as follows:

- A.** Open the **Visual Studio Code (VS Code)** software, press the "**Ctrl+Shift+P**" key combination or click in the search bar at the top, and then select "**Show and Run Commands**" from the drop-down menu to enter the command Enter the status as shown in the following figure:

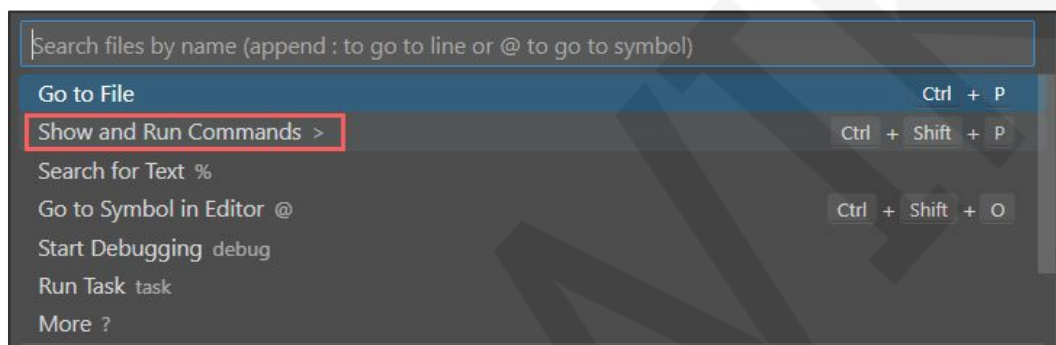


Figure 1.1 Entering Command Input State

- B.** Enter "**New Project**" in the command input bar, then press Enter to confirm or click on the **new project** option in the drop-down menu, as shown in the following figure:

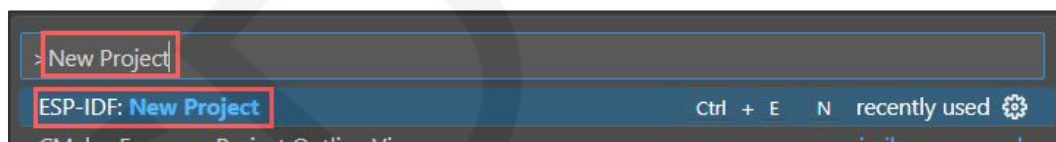
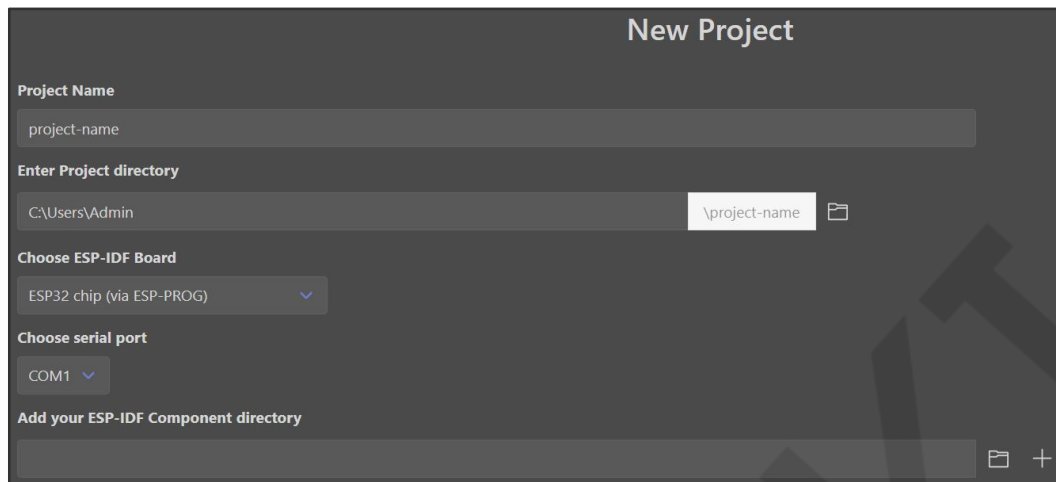


Figure 1.2 Creating a New Project

- C.** Configure the new project project on the pop-up interface, as shown in the following figure:



The screenshot shows the 'New Project' configuration window. It includes the following fields and options:

- Project Name:** A text input field containing 'project-name'.
- Enter Project directory:** A text input field containing 'C:\Users\Admin' and a file icon, followed by a sub-field containing '\project-name' and a folder icon.
- Choose ESP-IDF Board:** A dropdown menu with 'ESP32 chip (via ESP-PROG)' selected.
- Choose serial port:** A dropdown menu with 'COM1' selected.
- Add your ESP-IDF Component directory:** A text input field with a file icon and a plus sign.

Figure 1.3 Configuration of New Project Engineering

Project Name: The project name can only be named in English

Enter Project directory: The path name saved for the project engineering can only be selected in full English, otherwise an error will be reported during compilation because the path cannot be found.

Choose ESP-IDF Board: Select the type of chip to use and the type of download and debug interface. Select from the dropdown menu according to the actual situation.

Choose Serial Port: Select the serial port number. Select the actual serial port number connected to the development board. This string of slogans will be automatically selected when creating a new project when the device is already connected. When the device is not connected, it can be left unchecked.

Add your ESP-IDF Component director: Add third-party components to the project, if not available, there is no need to add them. The project engineering configuration here is shown in the following figure:

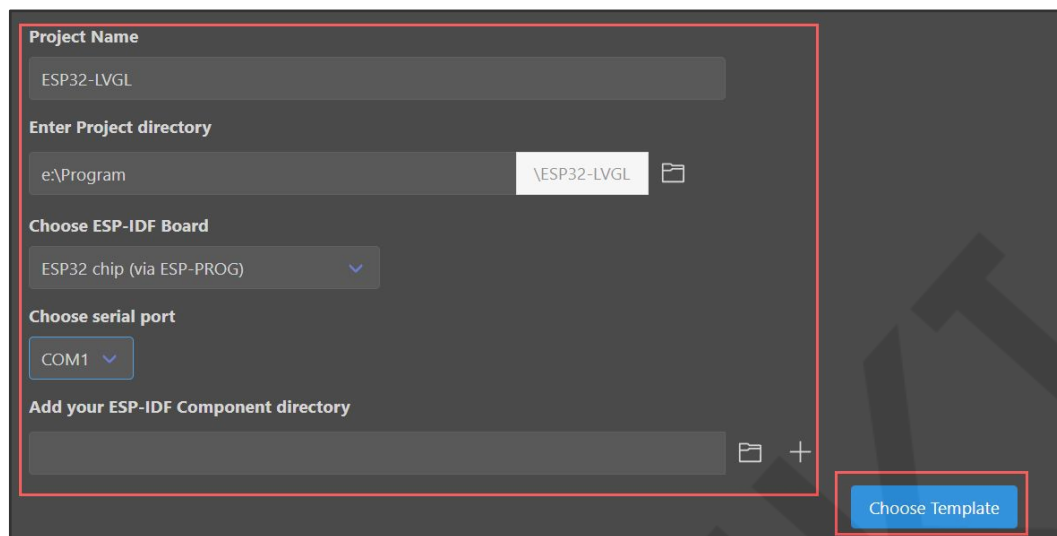


Figure 1.4 Project Configuration Example

D. After the project engineering configuration is completed, click the **"Choose Template"** button shown in Figure 1.4. In the pop-up interface, you can see many project engineering templates, which can be selected according to the actual project requirements. Choose the official basic template here. The specific operation is as follows: first, select **"ESP-IDF"** from the template category drop-down menu, then select **"sample_project"** in get started, and finally click the **"Create project using template sample_daject"** option, as shown in the following figure:

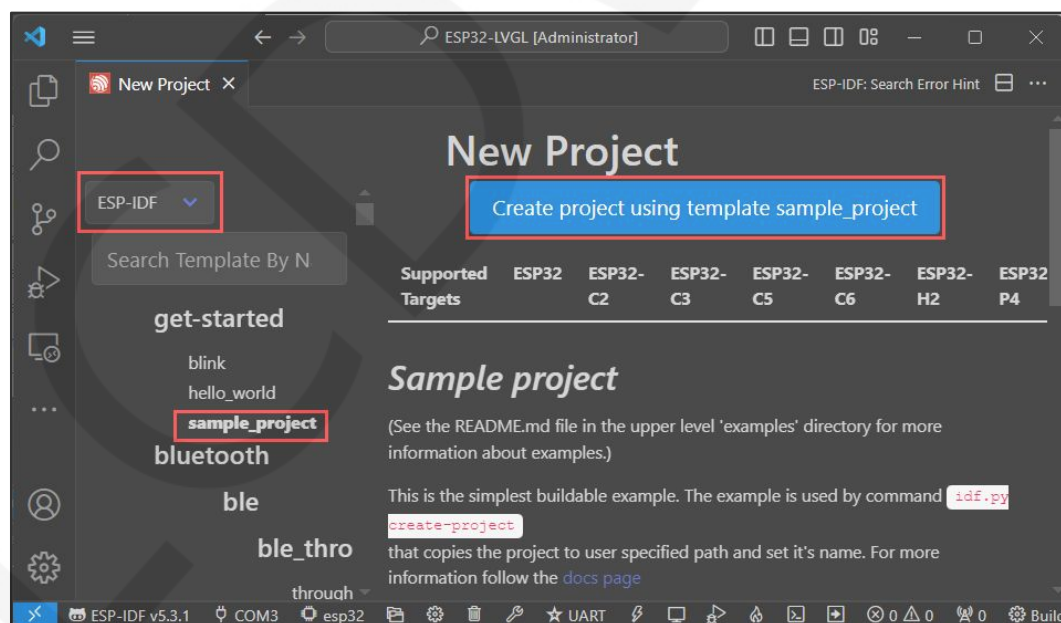


Figure 1.5: Selecting a Project Template

E. Next, click on the **"Yes"** option in the information box that pops up in the bottom right corner (as shown in the figure below), and a new window will open to display the

project engineering content.

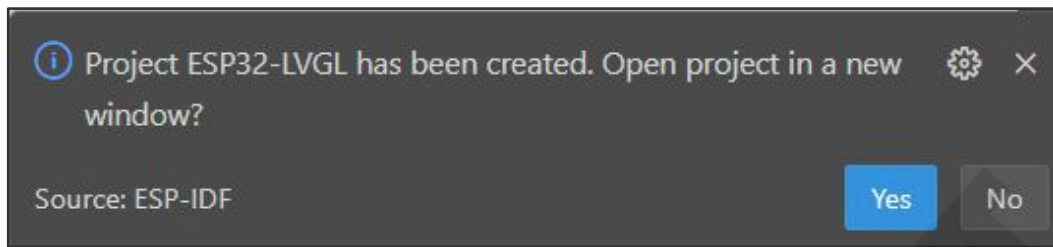


Figure 1.6 Open Project Window

F. Select '**Yes, I trust this authors**' in the pop-up interface of the project window, as shown in the following figure:

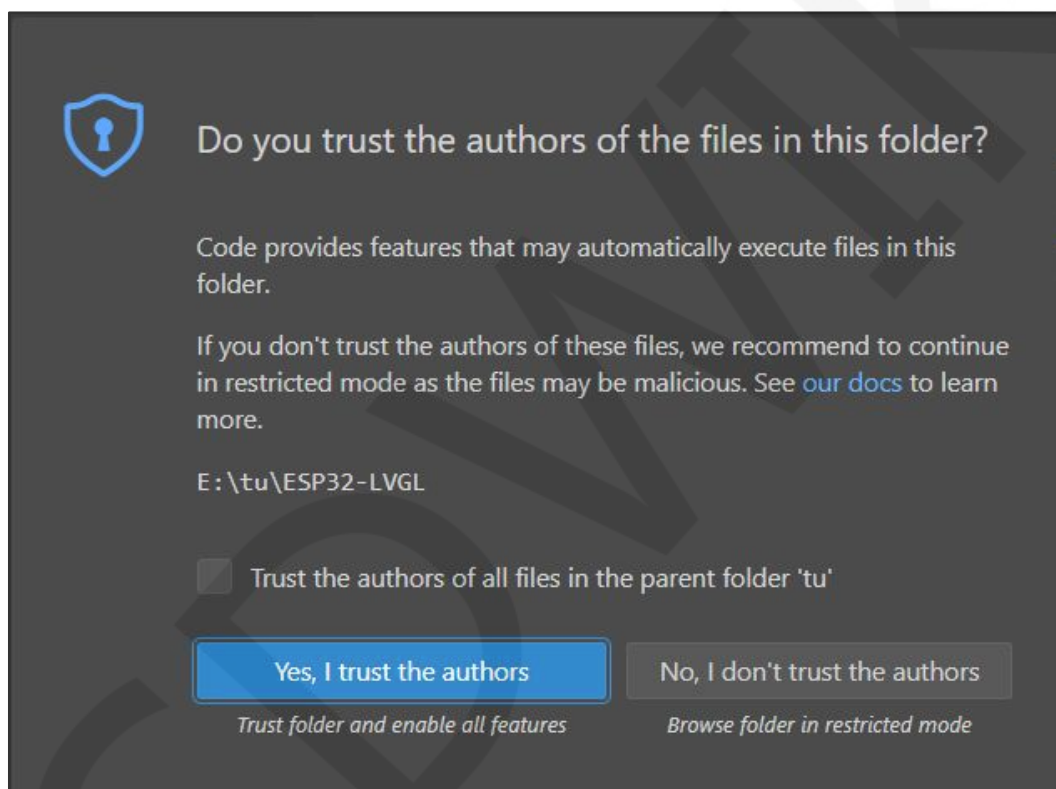


Figure 1.7: Selecting File Mode

G. After the project is created, the project engineering files can be viewed in the VS Code resource management area, as shown in the following figure:

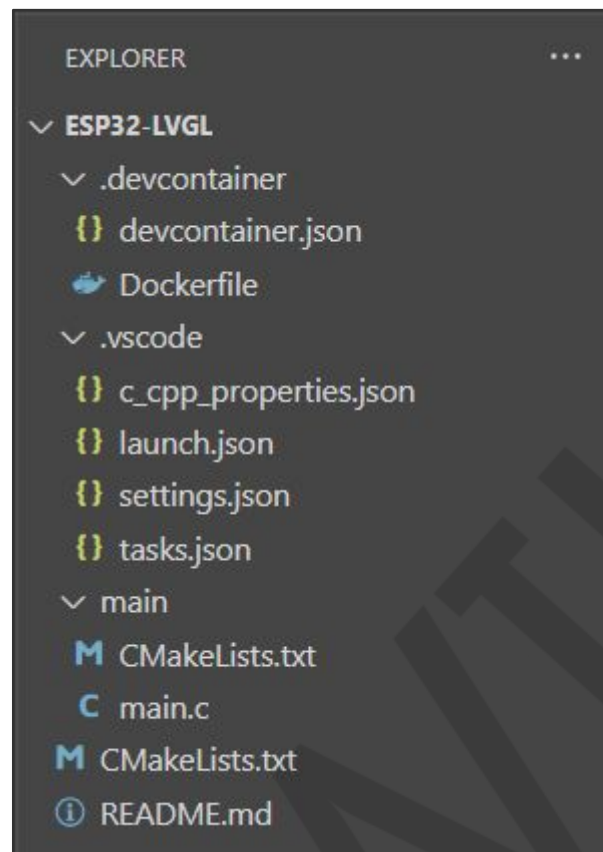


Figure 1.8 Project Files

.devcontainer: The files in this folder are used to define the configuration of the development container environment.

.vscode: The files in this folder are used for personalized configuration, workspace configuration, task configuration, debugger configuration, and compilation configuration of the project. These configurations are only valid for the current project.

The above two folders and files are automatically generated by VS Code when creating project engineering, and do not require manual writing.

main: The main application directory contains the main.c main application file and its Cmake file.

CMakeLists.txt: The main Cmake file defines the basic settings and components of the project engineering.

H. Next, configure the project engineering by clicking the settings button (gear icon) at the bottom of VS Code to enter the menuconfig menu configuration interface, as shown in the following figure:

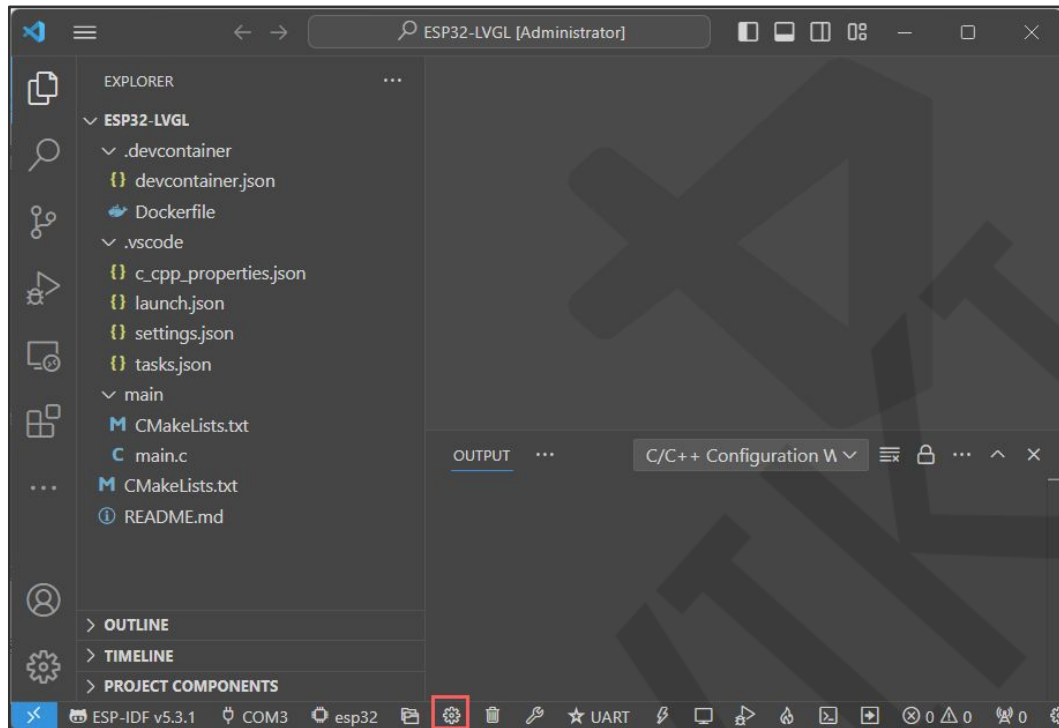


Figure 1.9 Enter the project configuration interface

- I. Enter "**flash**" in the "**Search parameter**" search bar of the menuconfig menu configuration interface to enter the flash configuration interface, as shown in the following figure:

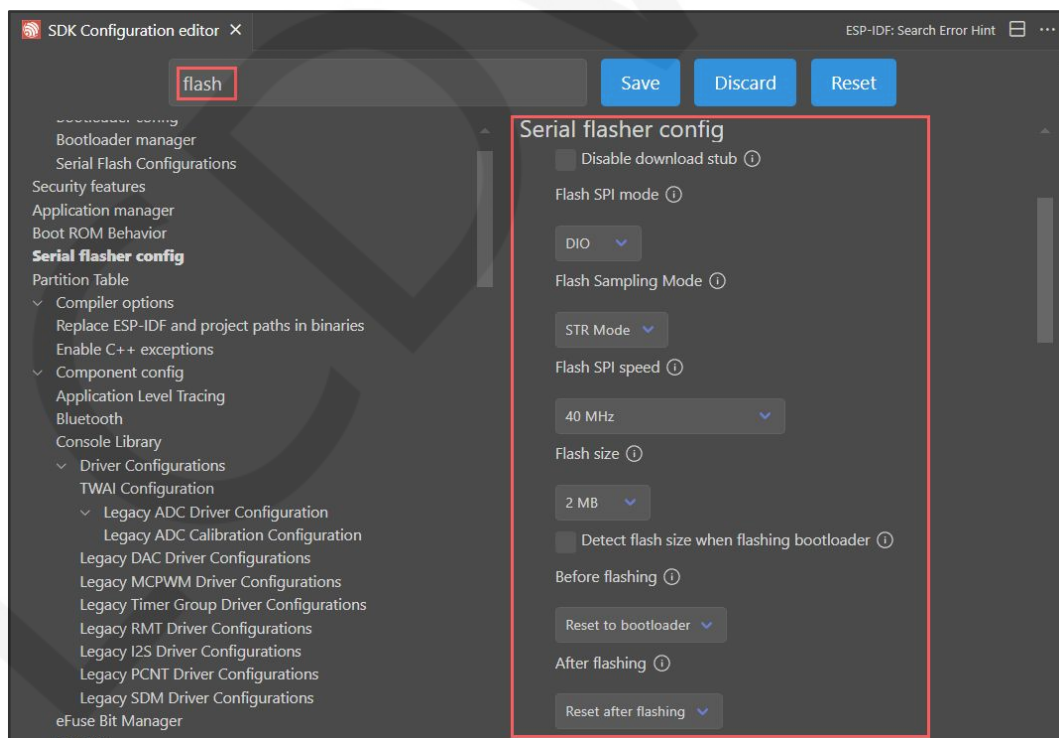


Figure 1.10: Configuring Flash

Flash SPI mode: Flash communication mode, optional parameters include: QIO、DIO、QOUT、DOUT。QIO uses 4-wire communication for both address and data; DIO uses 2-wire communication for both address and data; QOUT uses only 4 wires for data communication; DOUT only uses 2-wire communication for data. The selection should be based on the actual connection method of Flash.

Flash Sampling Mode: Flash sampling mode, only STR mode is used.

Flash SPI speed: Flash SPI speed, optional parameters include: 20M, 26M, 40M, 80M. To achieve optimal performance, an 80M speed is generally chosen.

Flash Size: Flash capacity, optional parameters include: 1M, 2M, 4M, 8M, 16M, 32M, 64M, 128M, depending on the actual capacity of Flash.

Before flashing: Choose whether to reset the ESP32 chip through the burning tool before burning Flash. Generally, choose reset because it can automatically enter the bootloader. If you choose not to reset, you will need to manually reset before entering download mode.

After flashing: Choose whether to reset the ESP32 chip through the burning tool after burning Flash. Generally, choose reset because it can automatically run the burned application program. If you choose not to reset, you need to manually reset in order to run the pre burned application.

Keep other Flash settings as default. The flash configuration here is shown in the following figure:

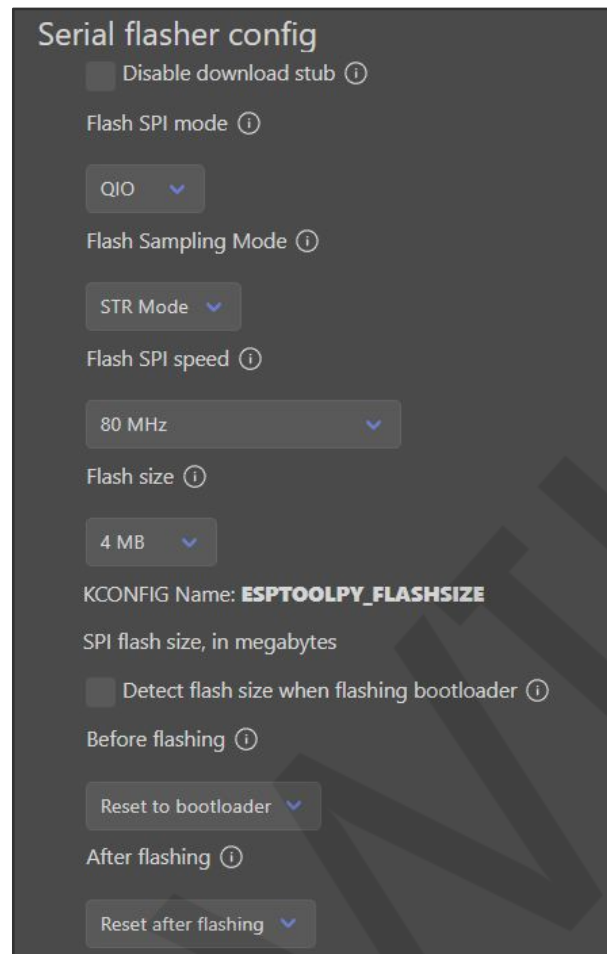


Figure 1.11 Project Flash Configuration

J. Enter "**Partition Table**" in the "**Search parameter**" search bar of the menuconfig menu configuration interface to enter the Partition Table configuration interface, as shown in the following figure:

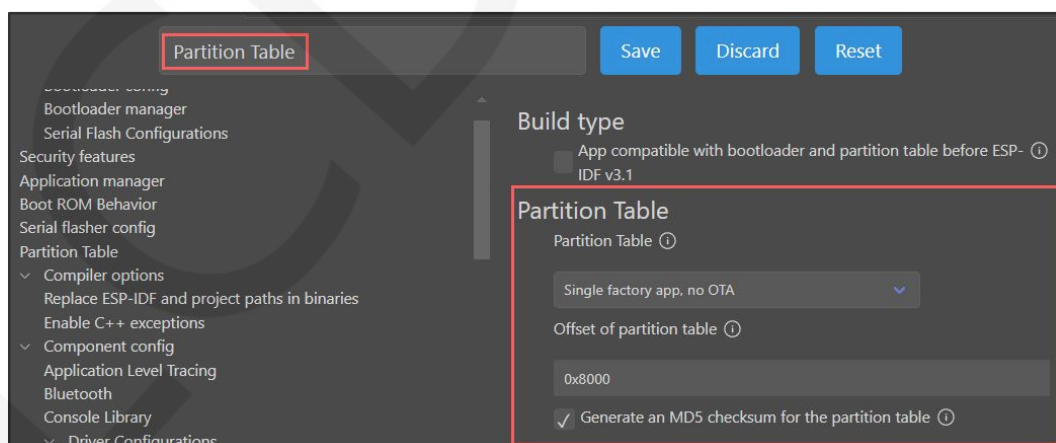


Figure 1.12 Configuring Partition Table

Partition Table : The Flash partition scheme generally chooses the default configuration. If there are many project files and the binary files

generated by compilation are relatively large, you can choose **"single factory app (large), no OTA"**. Users can also choose their own partition schemes.

Offset of partition table: The offset address stored in the Flash partition scheme table is generally set to the default value.

- K. If the ESP32 module used contains PSRAM, PSRAM can be configured. If not, this step is ignored. Enter **"PSRAM"** in the **"Search parameter"** search bar of the menuconfig menu configuration interface to enter the PSRAM configuration interface. The first choice is to check the **"Support for external, SPI connected RAM"** option in order to display the PSRAM configuration item. Configure PSRAM according to the actual situation. As shown in the following figure:

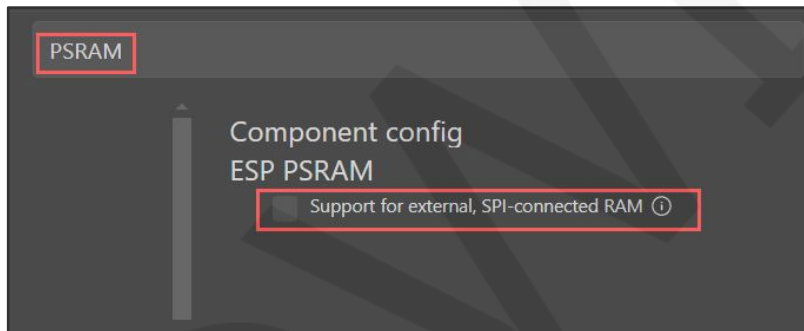


Figure 1.13 Configuring PSRAM

- L. Enter **"CPU frequency"** in the **"Search parameter"** search bar of the menuconfig menu configuration interface to enter the CPU frequency configuration interface. The CPU frequency options are 80MHz, 160MHz, and 240MHz. To maximize performance, 240MHz is generally chosen, as shown in the following figure:

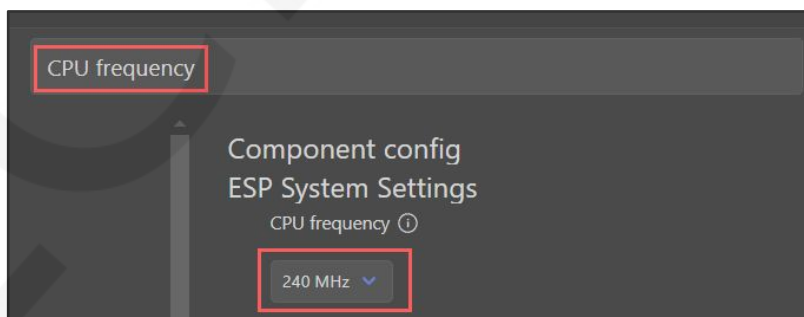


Figure 1.13: Configuring CPU Frequency

- M. Enter **"FreeRTOS"** in the **"Search parameter"** search bar of the menuconfig menu configuration interface to enter the FreeRTOS configuration interface. Change the

value of the **configTICK.RATE.HZ** option from the default value of 100 to 1000, while keeping all other values as default, as shown in the following figure:

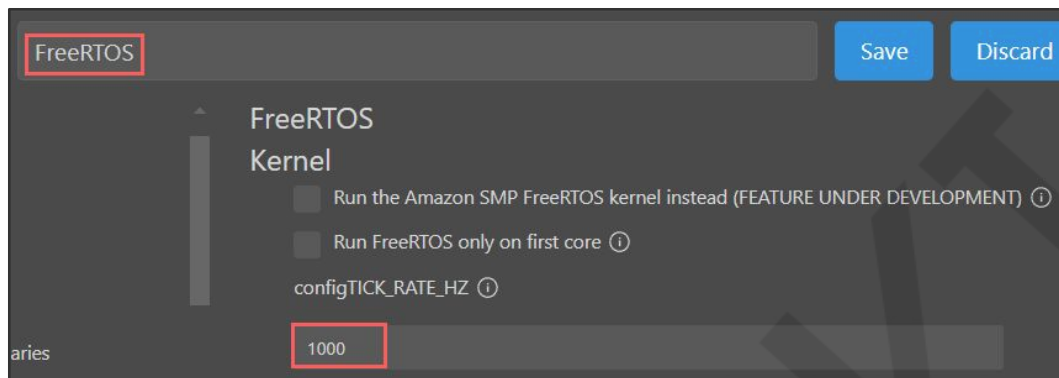


Figure 1.14: Configuring CPU Frequency

N. After the configuration is complete, click the **"Save"** button as shown in the following figure.

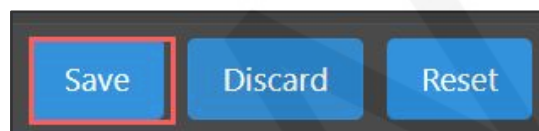


Figure 1.15 Saving Project Configuration

At this point, the basic work of the new project construction has been completed.

You can see that there are two additional files in the project, as shown in the following figure:

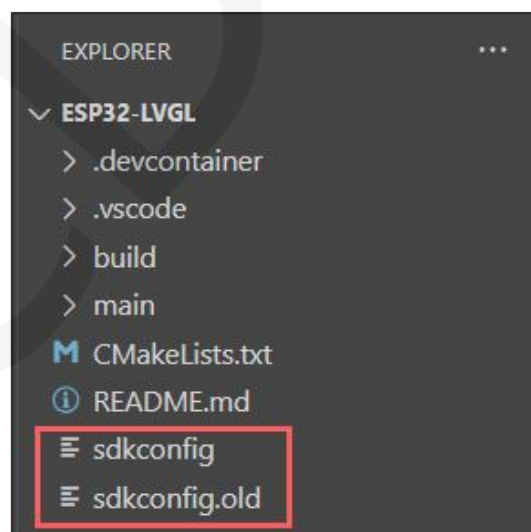


Figure 1.16 Project Document

Among them, **sdkconfig** is the latest generated system configuration file, and **sdkconfig.old** is the old system configuration file. When modifying system configuration,

it can also be done in the sdkconfig file.

2. Copy engineering files

After the completion of the project construction, it is necessary to copy the relevant engineering files. The steps are as follows:

- A. Download compressed files of LVGL v8.3.11 and LVGL esp32 drivers from the following addresses.

LVGL: <https://github.com/lvgl/lvgl/releases/tag/v8.3.11>

ESP_drivers: https://github.com/lvgl/lvgl_esp32_drivers

- B. Open the newly created project directory and create a new "**components**" folder, as shown in the following figure:

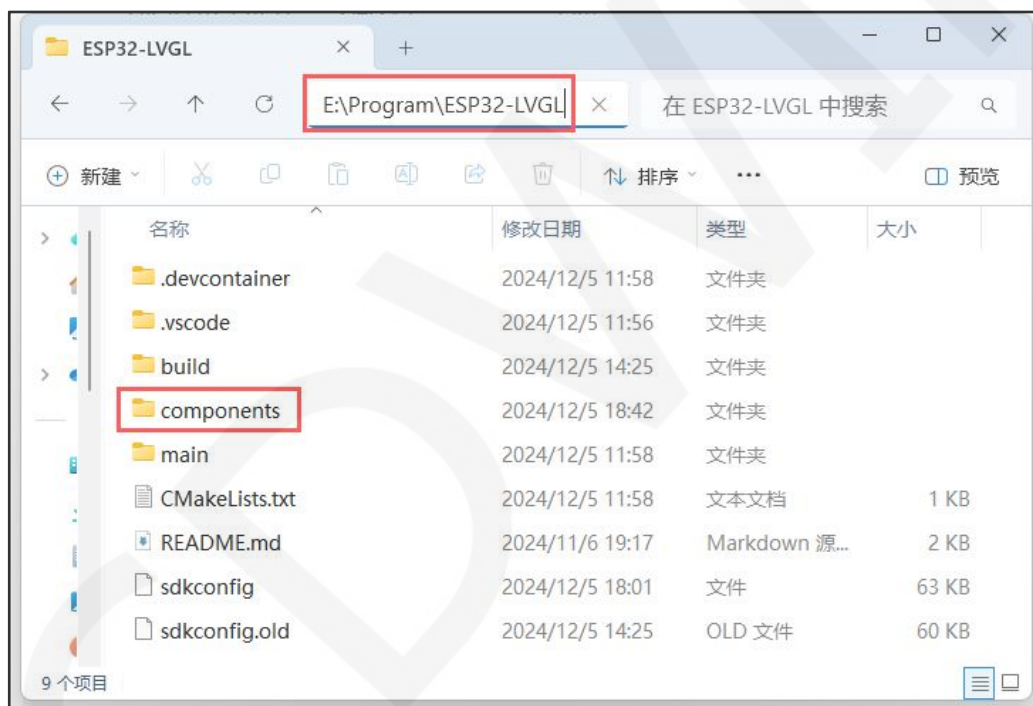


Figure 2.1 Creating a New Components Folder

- C. Copy the downloaded compressed files of **LVGL v8.3.11** and **LVGL esp32 drivers** to the "**components**" folder and extract them. After decompression is complete, rename the decompressed folder and delete the compressed file, as shown in the following figure:

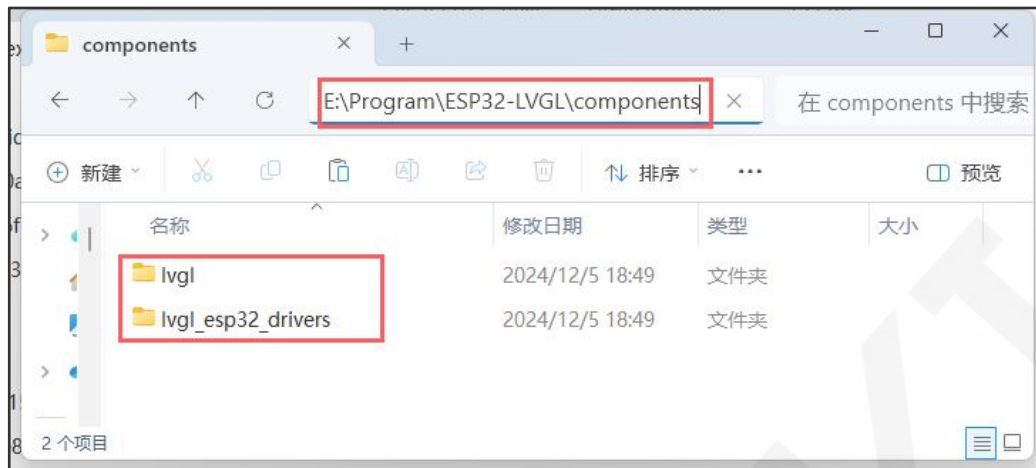


Figure 2.2 Unzip files to components folder

- D. Create a new **"lv_porting"** folder in the components folder of the project engineering, as shown in the following figure:

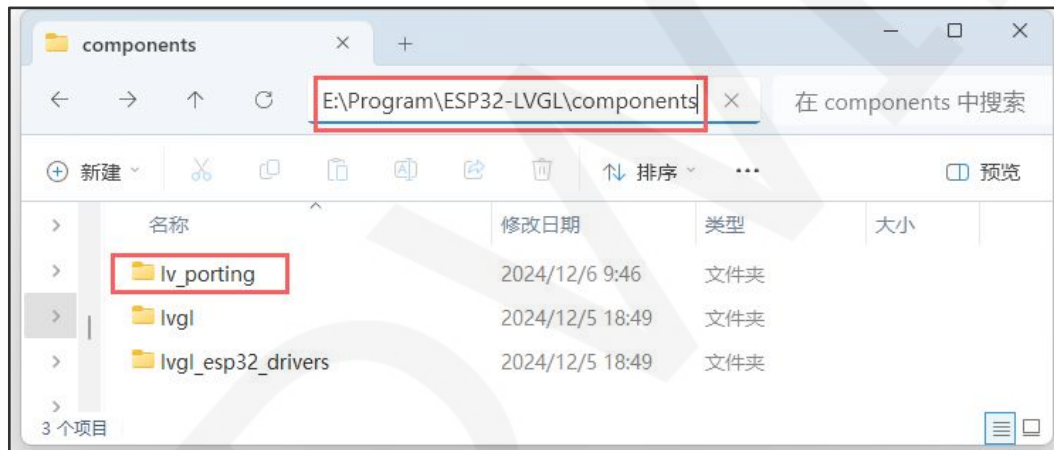


Figure 2.3 Creating a new lv_port folder

- E. Open the **"components\lvgl\examples\porting"** directory in the project directory, and add the **"lv_port-disp template. c"**, **"lv_port-disp template. h"** Copy the four files **"lv_port_indev_template. c"** and **"lv_port_indev_template. h"** to the **"components \ lv_porting"** folder in the project directory and rename them, as shown in the following figure:

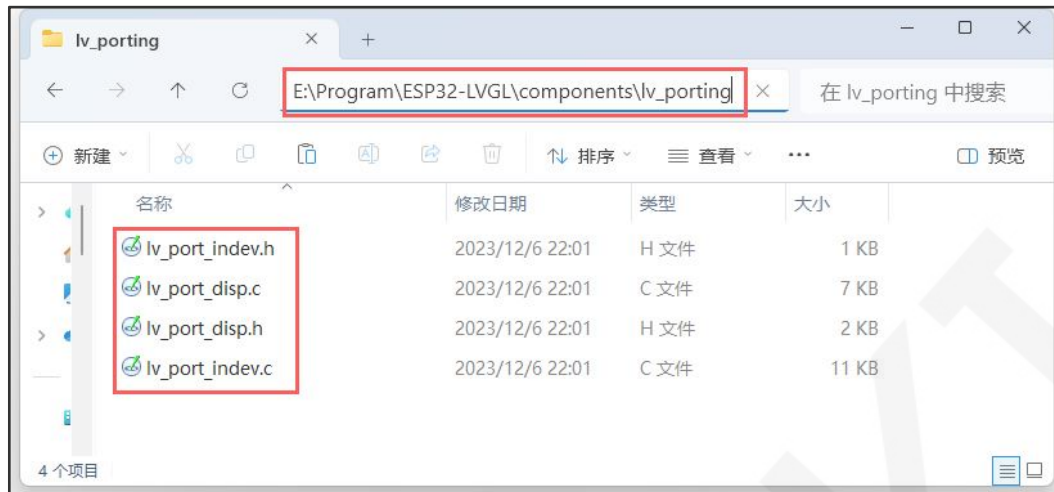



Figure 2.4 Copy files to lv_port folder

At this point, the relevant files have been copied.

3. Configure LVGL

The powerful configuration function provided by lvgl allows for correct configuration based on one's own project, so that the project can work properly. There are two ways to configure lvgl: through file configuration and through the menuconfig menu configuration interface. The focus here is on configuring lvgl through the menuconfig menu configuration interface. The options for configuring lvgl through files are the same as it is, only the operating interface is different.

3.1 Configure lvgl through the menuconfig configuration interface

- A. Click on the gear button  at the bottom of VS Code, or click on the search bar at the top, select **"Show and Run Commands"** from the drop-down menu, then enter **"Configuration Editor"** in the search bar, and click **"ESP-IDF: SDK Configuration Editor (Menuconfig)"** in the displayed results. This opens the SDK configuration editor interface.
- B. Enter **"lvgl"** in the search bar of the configuration editor interface to find the configuration options for lvgl, as shown in the following figure:

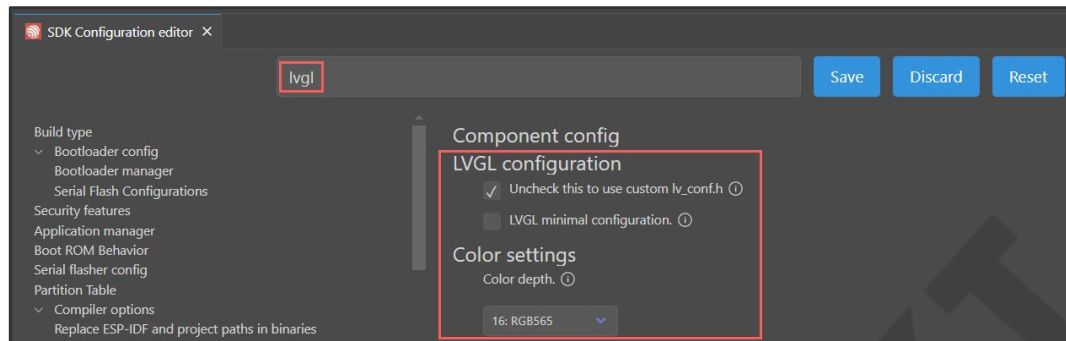


Figure 3.1 Search LVGL configuration

C. Next, you can configure LVGL. Here, we only introduce some LCD related configurations. For other configurations, if no special modifications are needed, they can be left as default. The configuration introduction is as follows:

- **LVGL Overall Configuration**

When the "**Uncheck this to use custom lv_conf. h**" option is checked, it means that lvgl is configured through the menuconfig menu configuration interface; If not checked, it means configuring lvgl through files (as described below).

When the "**LVGL minimal configuration.**" option is checked, it indicates that the LVGL is configured to minimize and is generally not checked.

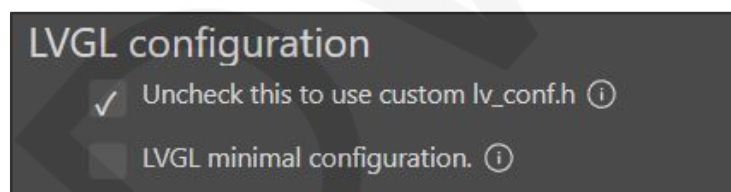


Figure 3.2 LVGL Overall Configuration

- **LCD color configuration**

Color depth is generally selected as "**16: RGB565**".

If the LCD uses SPI bus communication, the option "**Swap the 2 bytes of RGB565 color. Useful if the display has an 8-bit interface (e.g. SPI)**" needs to be checked. Leave other options as default.

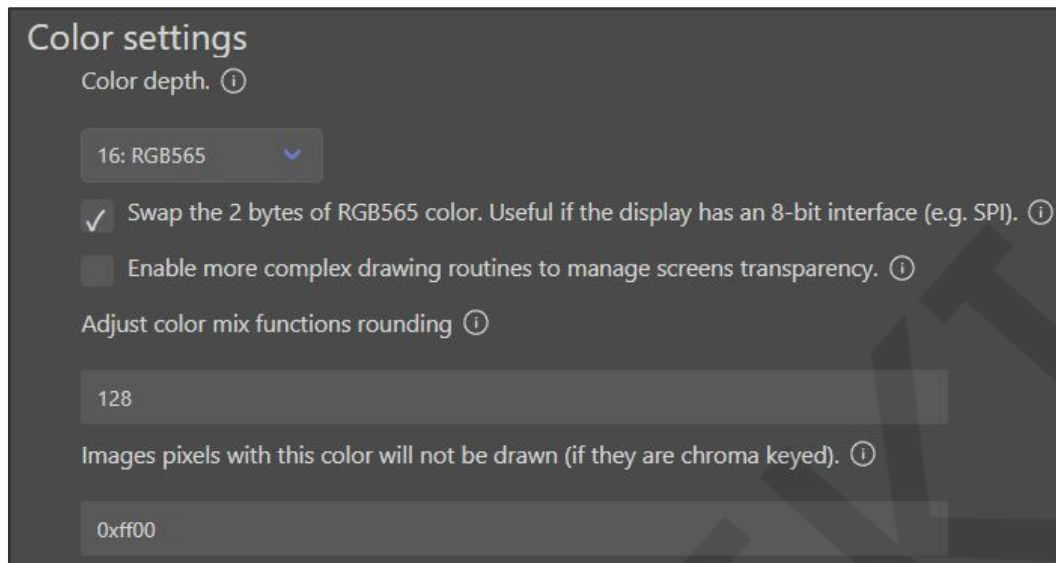


Figure 3.3 LCD color configuration

- **Memory Configuration**

Size of the memory used by 'lv_cem_alloc' in kilobytes is generally set to 64.

If the device's memory is large enough, the value can be set higher. Keep other configurations as default.

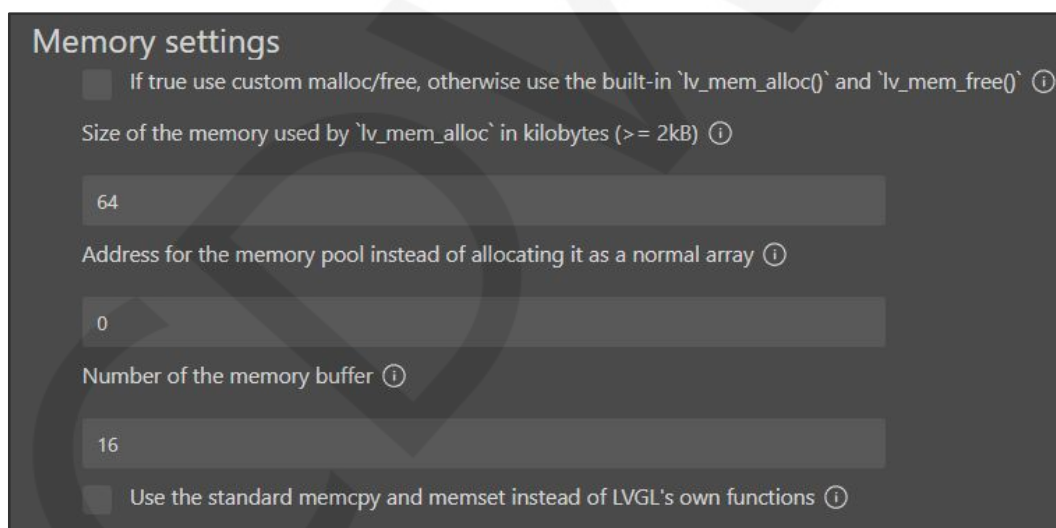


Figure 3.4 Memory Configuration

- **Other configurations**

Check the '**Show CPU usage and FPS count.**' option and set the position to '**Bottom right**'. This way, when running the application, you can see the current CPU usage and screen refresh rate in the bottom right corner of the screen. Other configurations remain default.

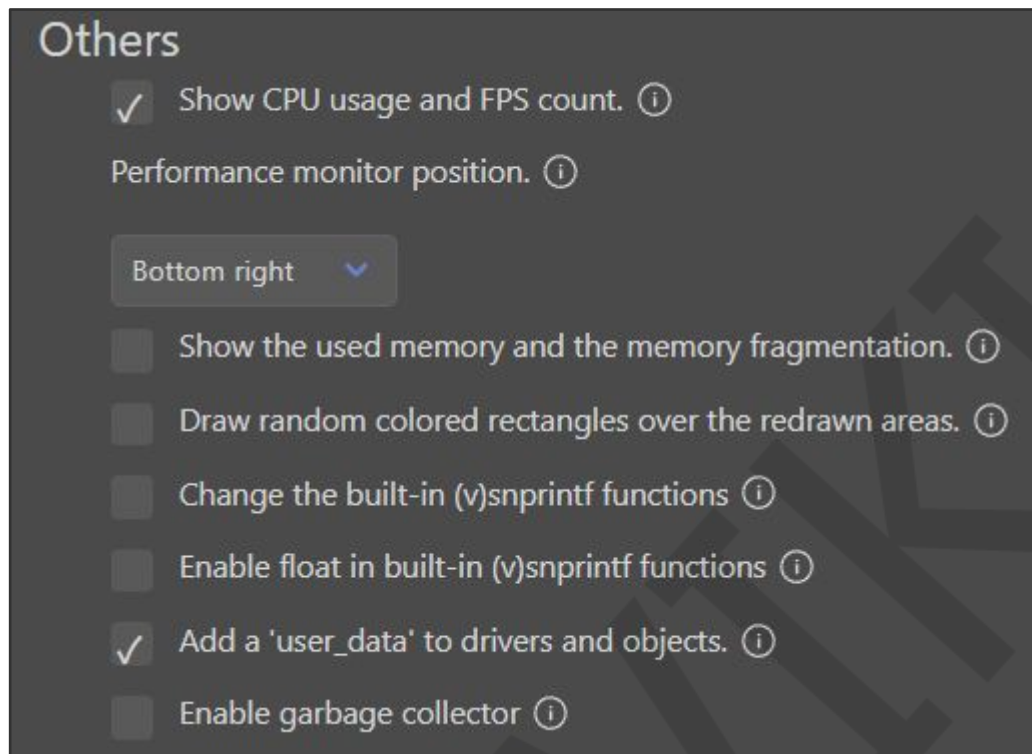


Figure 3.5 Other configurations

- Font settings

Check all font options from size 8 to size 48, so that these fonts will be included in the application and there will be no missing fonts. Keep other configurations as default.

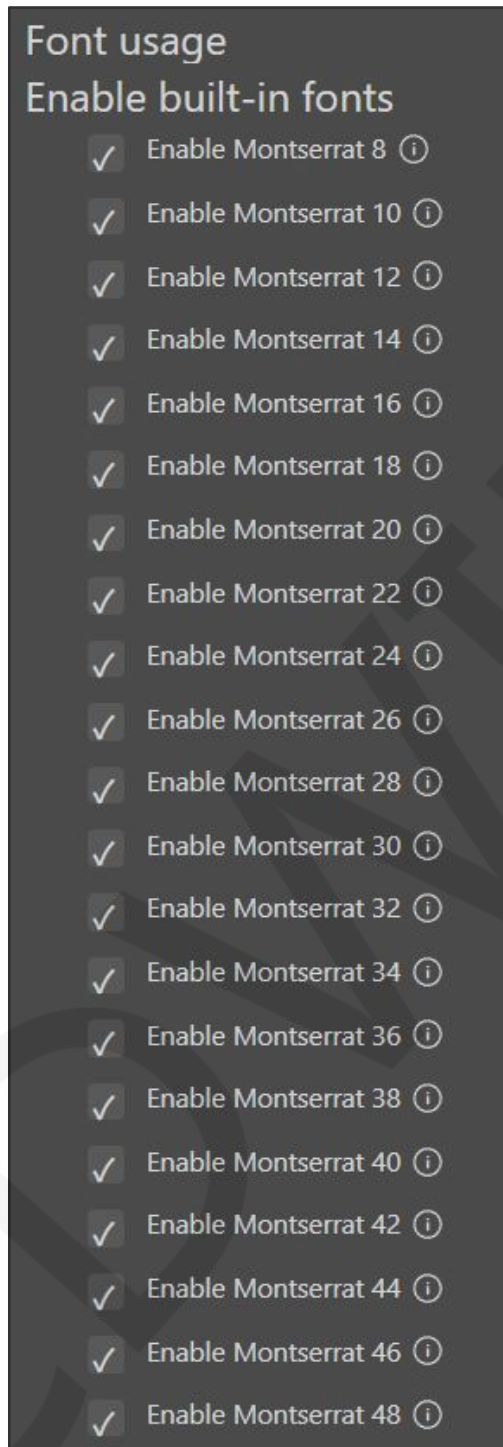


Figure 3.6 Font settings

- **Default Example Configuration**

Check all 5 default examples that come with lvgl, so that they will be compiled into the code during compilation for easy access. However, due to the limitations of the development board's Flash and memory capacity, all functions cannot be checked (they can also be checked when the capacity is large enough). Keep other

configurations as default.

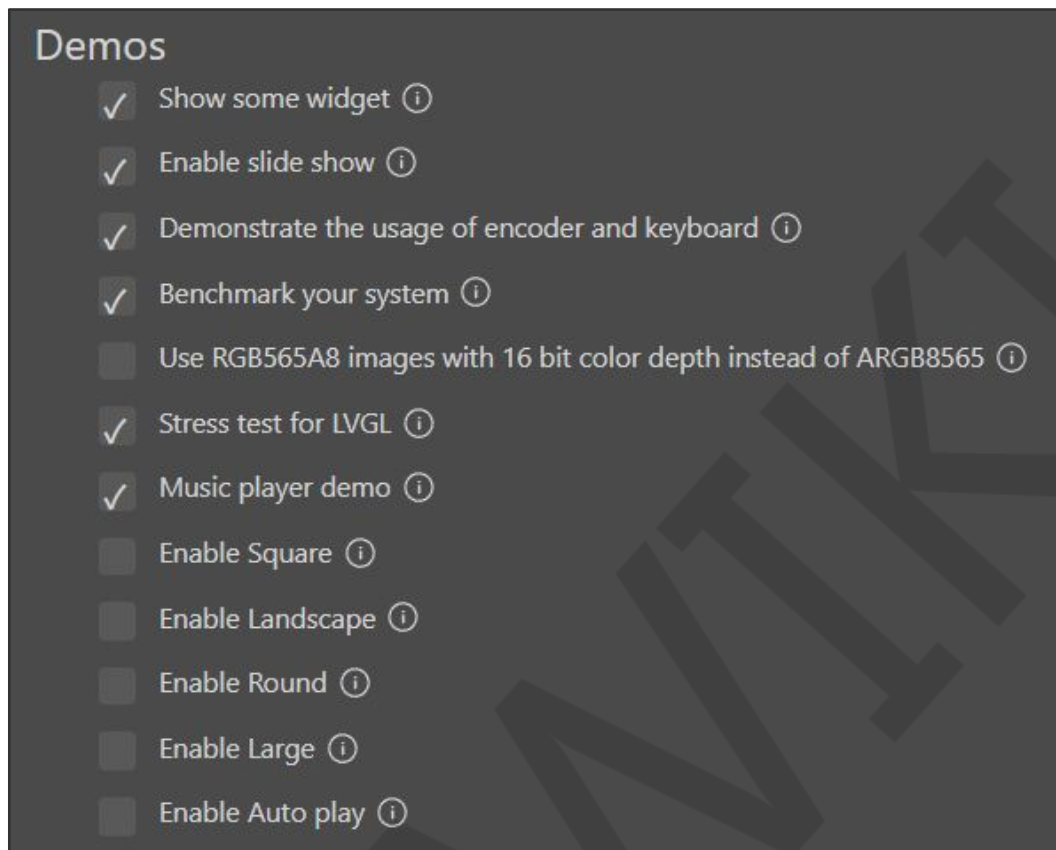


Figure 3.7 Default Example Configuration

D. After configuring lvgl, click the **"Save"** button to save the configuration.



Figure 3.8 Save lvgl configuration

3.2 Configuring lvgl through files

To configure lvgl through files, it is necessary to copy the **"lv_comf_template. h"** file from the **"components \ lvgl"** folder in the project directory to the current folder and rename it as **"lv_comf. h"**, as shown in the following figure:

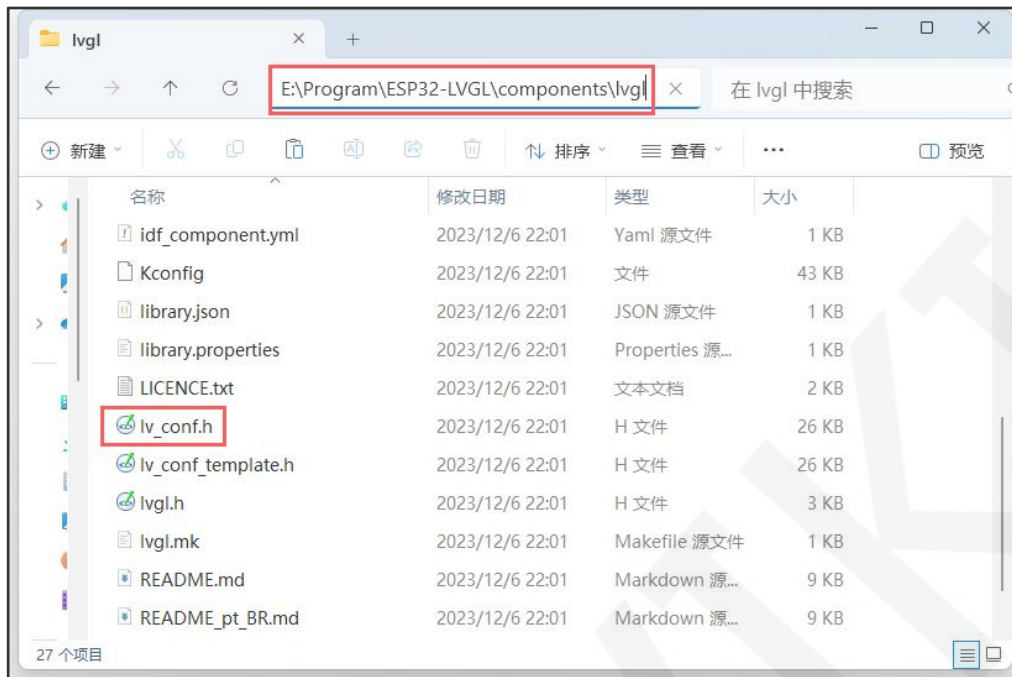


Figure 3.9 Copy and rename lvgl configuration file

Next, open the `lv_conf.h` file and configure it according to the actual situation.

4. Configure LCD

Next, configure the LCD as follows:

- A. Open the SDK configuration editor interface (refer to the LVGL configuration instructions for the method), enter **"display"** in the search bar, and you can find the display configuration options, as shown in the following figure:

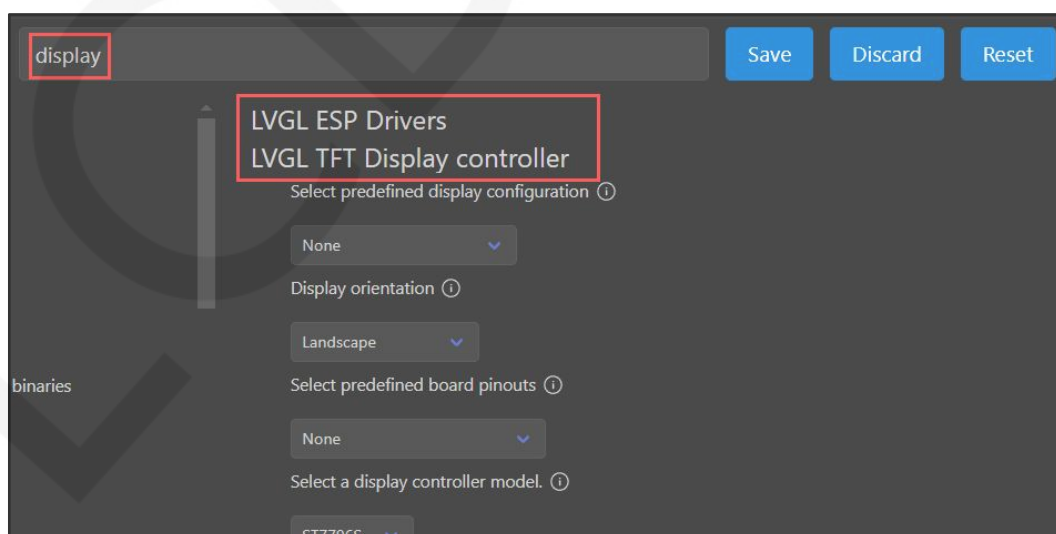


Figure 4.1 Configuring LCD

B. Next, configure the LCD. The configuration introduction is as follows:

- **LCD display controller configuration**

Select predefined display configuration: Select the already defined display configuration. If a certain configuration is selected, all LCD configurations are predefined. In general, the LCD used is not configured, so select 'None'.

Display orientation: Display direction setting, with options for landscape and portrait, depending on the implementation situation.

Select predefined board pinouts: Select the predefined pin configuration for the development board, or choose 'None' if none are available.

Select a display controller model: Select a controller model for the LCD used, which includes most commonly used LCD controller models, and choose according to the actual usage. If not, it needs to be transplanted.

TFT SPI Bus: Select the SPI bus number for ESP32, which has two sets of available SPIs according to the pin selection.

TFT Data Transfer Mode: SPI transmission mode selection, with options of 1-wire, 2-wire, and 4-wire. Usually choose 1 line.

TFT SPI Duplex Mode: SPI communication mode, with options for half duplex and full duplex. Usually choose full duplex.

Use custom SPI clock frequency: After selecting, you can define the SPI rate yourself, generally defined as 80M.

LVGL TFT Display controller

Select predefined display configuration ⓘ

None ▼

Display orientation ⓘ

Landscape ▼

Select predefined board pinouts ⓘ

None ▼

Select a display controller model. ⓘ

ST7796S ▼

☐ Use custom display buffer size (bytes) ⓘ

TFT SPI Bus. ⓘ

SPI2_HOST ▼

TFT Data Transfer Mode ⓘ

SIO (MOSI/MISO) ▼

TFT SPI Duplex Mode ⓘ

FULL DUPLEX ▼

☒ Use custom SPI clock frequency. ⓘ

Select a custom frequency. ⓘ

80 MHz ▼

Figure 4.2 LCD Controller Configuration

- **Definition of ESP32 pins for LCD**

GPIO for MOSI: SPI bus write data pin definition.

GPIO for MISO: SPI bus read data pin definition, "**GPIO for MISO**" needs to be checked to set.

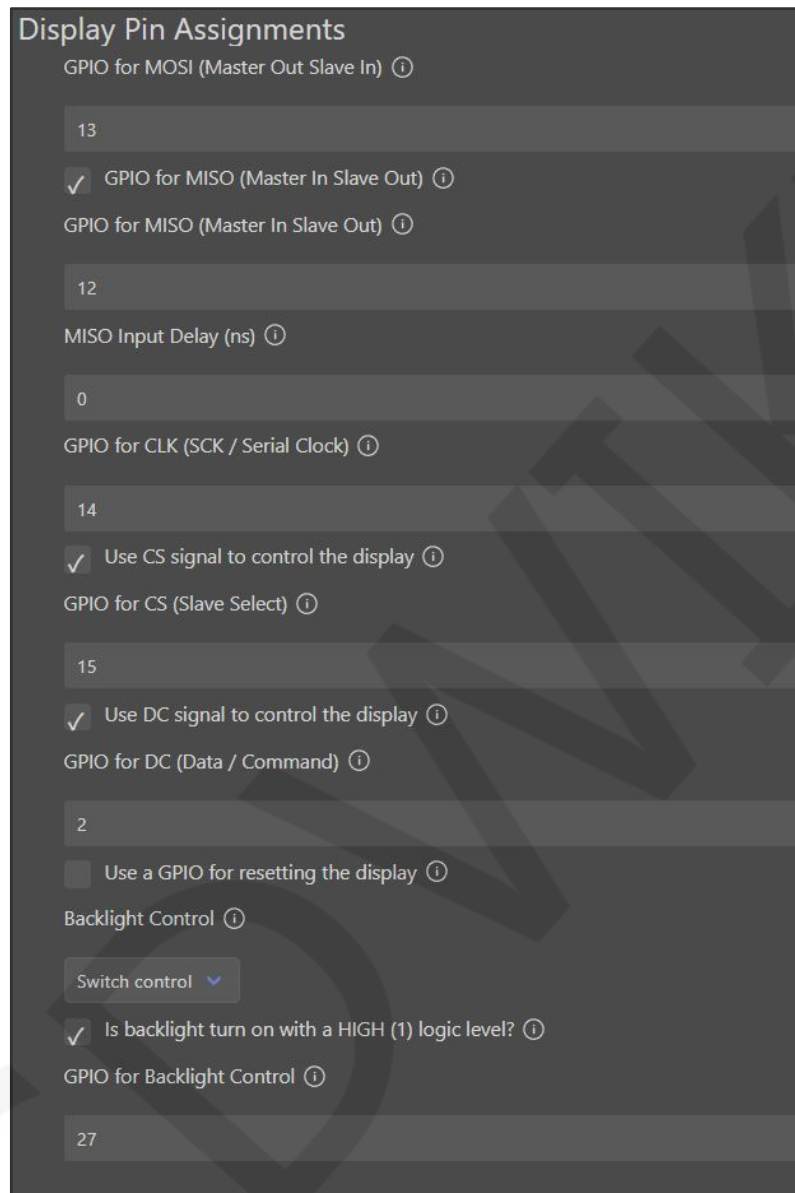
GPIO for CLK: Definition of clock pins on SPI bus

GPIO for CS: The definition of LCD enable pins requires selecting "**Use CS signal to control the display**" to set.

GPIO for DC: LCD data and command selection pin definitions require selecting "**Use DC signal to control the display**" to set.

Backlight Control: LCD backlight control, available in three modes: none, PWM, and on/off, with the option of high-level on/off.

GPIO for Backlight Control: Definition of LCD backlight control pins, PWM or switch mode must be selected to set.



Display Pin Assignments

GPIO for MOSI (Master Out Slave In) ⓘ

13

☒ GPIO for MISO (Master In Slave Out) ⓘ

GPIO for MISO (Master In Slave Out) ⓘ

12

MISO Input Delay (ns) ⓘ

0

GPIO for CLK (SCK / Serial Clock) ⓘ

14

☒ Use CS signal to control the display ⓘ

GPIO for CS (Slave Select) ⓘ

15

☒ Use DC signal to control the display ⓘ

GPIO for DC (Data / Command) ⓘ

2

☐ Use a GPIO for resetting the display ⓘ

Backlight Control ⓘ

Switch control ▾

☒ Is backlight turn on with a HIGH (1) logic level? ⓘ

GPIO for Backlight Control ⓘ

27

Figure 4.3 LCD Pin Definition

C. After completing the LCD configuration, click the **"Save"** button to save the configuration.

5. Configure touch screen

Next, configure the touch screen. The steps are as follows:

A. Open the SDK configuration editor interface (refer to LVGL configuration instructions for the method), enter **"touch"** in the search bar, and you can find the

configuration options for touch, as shown in the following figure:

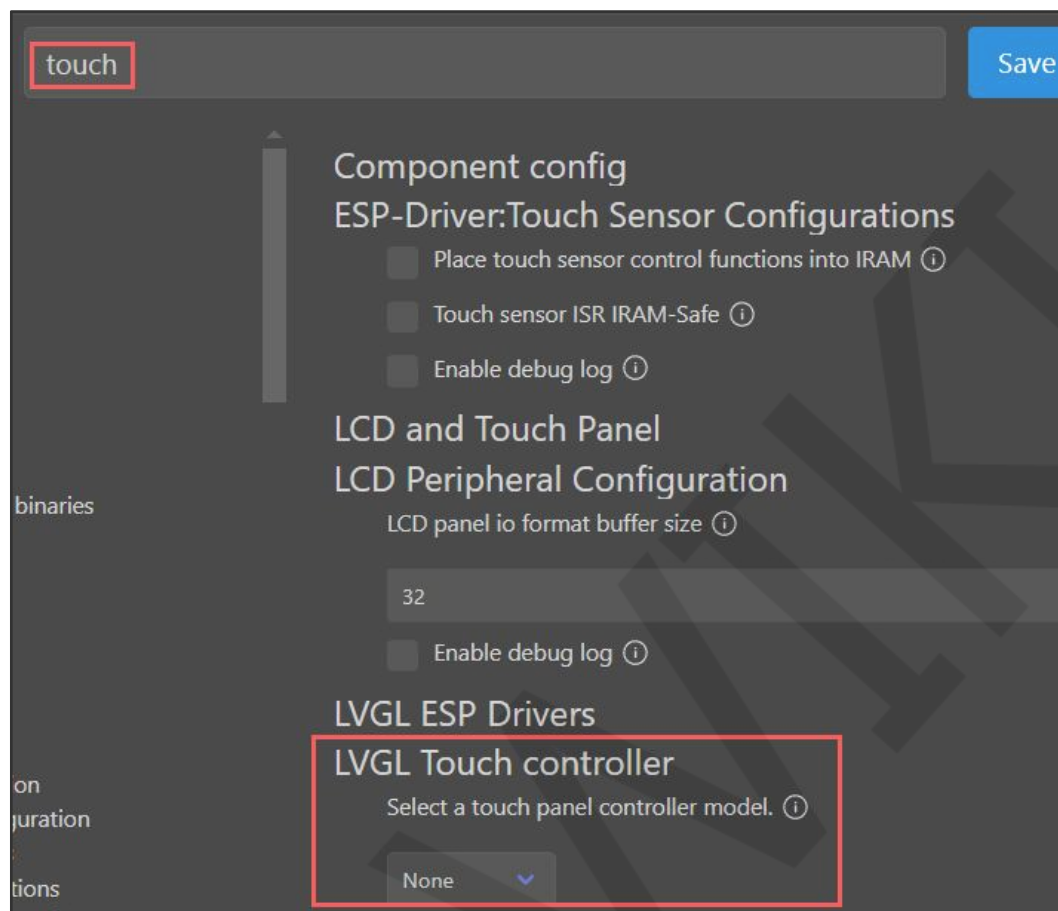


Figure 5.1 Touch screen configuration

B. Next, we will configure the touch screen. The configuration introduction is as follows:

- **Definition of touch screen ESP32 pins**

Select a touch panel controller model : Select a touchscreen controller model.

Optional models include resistive touch screen and capacitive screen controller. Choose according to the actual situation. If there is no required model, it needs to be transplanted.

Touch Controller SPI Bus : Select the SPI bus number of the ESP32 used for the touch screen. ESP32 has two sets of available SPI, which can be selected based on the pins.

GPIO for MISO : Definition of read data pins for SPI bus used in touch screen.

GPIO for MOSI : Definition of write data pins for SPI bus used in touch screen.

GPIO for CLK : Definition of clock signal pins for SPI bus used in touch screen.

GPIO for CS : Definition of touch screen enable pins.

GPIO for IRQ : Definition of touch screen interrupt pin.

LVGL Touch controller

Select a touch panel controller model. ⓘ

XPT2046 ▼

Touch Controller SPI Bus. ⓘ

SPI2_HOST ▼

Touchpanel (XPT2046) Pin Assignments

GPIO for MISO (Master In Slave Out) ⓘ

12

GPIO for MOSI (Master Out Slave In) ⓘ

13

GPIO for CLK (SCK / Serial Clock) ⓘ

14

GPIO for CS (Slave Select) ⓘ

33

GPIO for IRQ (Interrupt Request) ⓘ

36

Figure 5.2 Definition of Touchscreen Pins

- **Other parameters configuration of touch screen**

Minimum X coordinate value: The Minimum ADC value at the x-coordinate

Minimum Y coordinate value: The Minimum ADC value at the y-coordinate

Maximum X coordinate value: The maximum ADC value at the x-coordinate

Maximum Y coordinate value: The maximum ADC value at the y-coordinate

Swap XY: Choose whether to exchange the coordinate values of x and y. Check to exchange

Invert X coordinate value: Choose whether to reverse the x-coordinate value, if

checked, reverse it

Invert Y coordinate value: Choose whether to reverse the y-coordinate value, if

checked, reverse it

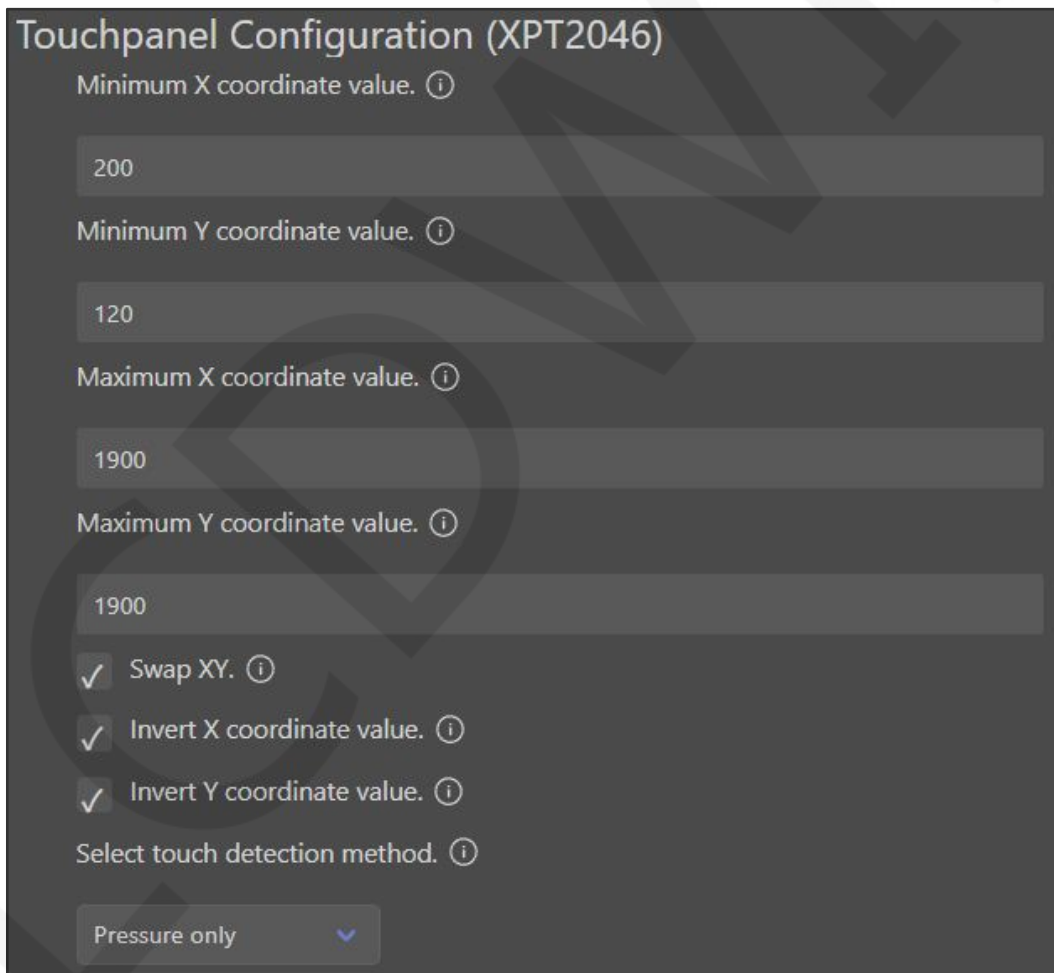
If the exact values of the above parameters are not known, they can be left unconfigured for now. After the program runs, the relevant values can be obtained from the serial port output by clicking the touch screen and then set.

Select touch detection method: Select the touch event detection method. Optional

to detect only through pressing, only through interrupt pins, or through both pressing and interrupt pins. If a

polling method is adopted, one can choose to

only detect by pressing.



Touchpanel Configuration (XPT2046)

Minimum X coordinate value. ⓘ

200

Minimum Y coordinate value. ⓘ

120

Maximum X coordinate value. ⓘ

1900

Maximum Y coordinate value. ⓘ

1900

☒ Swap XY. ⓘ

☒ Invert X coordinate value. ⓘ

☒ Invert Y coordinate value. ⓘ

Select touch detection method. ⓘ

Pressure only ▼

Figure 5.3 Other parameters configuration of touch screen

C. After completing the touch screen parameter configuration, click the **"Save"** button to save the configuration.

6. Modify engineering documents

6.1 Modify the contents of the lv_porting folder

- Add Cmake file

Open the "**components\lv_porting**" folder in the project engineering directory, create a new **CMakeLists.txt** file in that folder, open the file, enter the following content, and then save it:

```
file(GLOB_RECURSE SOURCES ./*.c
    )

idf_component_register(SRCS ${SOURCES}
    INCLUDE_DIRS
        .
    REQUIRES lvgl
        lvgl_esp32_drivers
    )
```

- Modify files related to the display screen

A. Open file content and modify header file definition

Open the "**lv_port_desp. c**" file in the "**components\lv_porting**" folder under the project directory, change the 7th line from "**# if 0**" to "**# if 1**", and then modify and add the header file. The modified content is as follows:

```
/*Copy this file as "lv_port_disp.c" and set this value to "1" to enable
content*/
#if 1

/*****
 *   INCLUDES
 *****/
#include "lv_port_disp.h"
#include <stdbool.h>
#include "lvgl_helpers.h"
#include "disp_driver.h"
```

B. Modify the definition of LCD resolution

Firstly, annotate the two warnings, and then modify the horizontal and vertical resolutions according to the actual situation. For example, if an LCD driven by ST7796 is used here for vertical display, the horizontal resolution macro definition

MY_DISP_HOR_RES is set to 320, and the vertical resolution macro definition

MY_DISP_VER_RES is set to 480; If displaying in landscape mode, swap the two resolution values. The modified content is as follows:

```

/*****
 *      DEFINES
 *****/
#ifndef MY_DISP_HOR_RES
    //warning Please define or replace the macro MY_DISP_HOR_RES with the
    actual screen width, default value 320 is used for now.
    #define MY_DISP_HOR_RES    480
#endif

#ifndef MY_DISP_VER_RES
    //warning Please define or replace the macro MY_DISP_HOR_RES with the
    actual screen height, default value 240 is used for now.
    #define MY_DISP_VER_RES    320
#endif

```

C. Modify buffer zone

We choose the double buffer method here and comment out the code for setting the other two buffer methods. Change the buffer capacity to the capacity defined in the LCD driver (defined in the lvgl_celpers. h header file, different LCD drivers correspond to different buffer capacities). The modified content is as follows:

```

/* Example for 1) */
//static lv_disp_draw_buf_t draw_buf_dsc_1;
//static lv_color_t buf_1[MY_DISP_HOR_RES *
10];          /*A buffer for 10 rows*/
//lv_disp_draw_buf_init(&draw_buf_dsc_1, buf_1, NULL, MY_DISP_HOR_RES
* 10); /*Initialize the display buffer*/

/* Example for 2) */
static lv_disp_draw_buf_t draw_buf_dsc_2;
static lv_color_t buf_2_1[DISP_BUF_SIZE];          /*A
buffer for 10 rows*/
static lv_color_t buf_2_2[DISP_BUF_SIZE];          /*An
other buffer for 10 rows*/
lv_disp_draw_buf_init(&draw_buf_dsc_2, buf_2_1, buf_2_2,
DISP_BUF_SIZE); /*Initialize the display buffer*/

/* Example for 3) also set disp_drv.full_refresh = 1 below*/
//static lv_disp_draw_buf_t draw_buf_dsc_3;

```

```
//static lv_color_t buf_3_1[MY_DISP_HOR_RES *  
MY_DISP_VER_RES];          /*A screen sized buffer*/  
//static lv_color_t buf_3_2[MY_DISP_HOR_RES *  
MY_DISP_VER_RES];          /*Another screen sized buffer*/  
//lv_disp_draw_buf_init(&draw_buf_dsc_3, buf_3_1, buf_3_2,  
//                        MY_DISP_VER_RES *  
LV_VER_RES_MAX); /*Initialize the display buffer*/
```

修改lvgl显示驱动缓冲区定义, 将其由单缓冲区改为双缓冲区, 修改后的内容如下所示:

```
/*Set a display buffer*/  
disp_drv.draw_buf = &draw_buf_dsc_2;
```

D. Modify LCD initialization function

Call the initialization function defined in the LCD driver in lvgl's LCD initialization function (defined in the lvgl_celpers. c file, different LCD drivers correspond to different initialization functions). The modified content is as follows:

```
/*Initialize your display and the required peripherals.*/  
static void disp_init(void)  
{  
    /*You code here*/  
    lvgl_driver_init();  
}
```

E. Modify the LCD screen refresh function

Call the refresh function defined in the LCD driver in the LCD refresh function of lvgl (defined in the disp_driver. c file, different LCD drivers correspond to different refresh functions). The modified content is as follows:

```
/*Flush the content of the internal buffer the specific area on the display  
*You can use DMA or any hardware acceleration to do this operation in the  
background but  
'lv_disp_flush_ready()' has to be called when finished.*/  
static void disp_flush(lv_disp_drv_t * disp_drv, const lv_area_t * area,  
lv_color_t * color_p)  
{  
    disp_driver_flush(disp_drv, area, color_p);  
    #if 0  
    if(disp_flush_enabled) {  
        /*The most simple case (but also the slowest) to put all pixels to  
the screen one-by-one*/  
  
        int32_t x;  
        int32_t y;  
        for(y = area->y1; y <= area->y2; y++) {
```

```

        for(x = area->x1; x <= area->x2; x++) {
            /*Put a pixel to the display. For example:*/
            /*put_px(x, y, *color_p)*/
            color_p++;
        }
    }
}

/*IMPORTANT!!!
 *Inform the graphics library that you are ready with the flushing*/
lv_disp_flush_ready(disp_drv);
#endif
}

```

At this point, the lv_port-disp. c file has been modified and the modified file has been saved.

F. Modify header file

Open the "lv_port_desp.h" file in the "components \ lv_porting" folder of the project engineering directory, change line 7 from "# if 0" to "# if 1", and then save.

The modified file content is as follows:

```

/*Copy this file as "lv_port_disp.h" and set this value to "1" to enable
content*/
#if 1

```

● Modify touch screen related files

A. Open file content and modify header file definition

Open the "lv_port_indev.c" file in the "components\lv_porting" folder under the project directory, change line 7 from "# if 0" to "# if 1", and then modify and add the header file. The modified content is as follows:

```

/*Copy this file as "lv_port_indev.c" and set this value to "1" to enable
content*/
#if 1

/*****
 *   INCLUDES
 *****/
#include "lv_port_indev.h"
#include "../lvgl.h"
#include "touch_driver.h"

```

B. Annotate function declarations and variable definitions for other input

devices

Because only touch screens are used here, it is recommended to comment out function declarations and variable definitions related to other input devices, otherwise errors may occur during compilation. The modified file content is as follows:

```

/*****
 *  STATIC PROTOTYPES
 *****/

static void touchpad_init(void);
static void touchpad_read(lv_indev_drv_t * indev_drv, lv_indev_data_t *
data);
static bool touchpad_is_pressed(void);
static void touchpad_get_xy(lv_coord_t * x, lv_coord_t * y);
//static void mouse_init(void);
//static void mouse_read(lv_indev_drv_t * indev_drv, lv_indev_data_t *
data);
//static bool mouse_is_pressed(void);
//static void mouse_get_xy(lv_coord_t * x, lv_coord_t * y);
//static void keypad_init(void);
//static void keypad_read(lv_indev_drv_t * indev_drv, lv_indev_data_t *
data);
//static uint32_t keypad_get_key(void);
//static void encoder_init(void);
//static void encoder_read(lv_indev_drv_t * indev_drv, lv_indev_data_t *
data);
//static void encoder_handler(void);
//static void button_init(void);
//static void button_read(lv_indev_drv_t * indev_drv, lv_indev_data_t *
data);
//static int8_t button_get_pressed_id(void);
//static bool button_is_pressed(uint8_t id);
/*****
 *  STATIC VARIABLES
 *****/

lv_indev_t * indev_touchpad;
//lv_indev_t * indev_mouse;
//lv_indev_t * indev_keypad;
//lv_indev_t * indev_encoder;
//lv_indev_t * indev_button;
//static int32_t encoder_diff;
//static lv_indev_state_t encoder_state;

```

C. Annotate the initialization code of other input devices

Comment the initialization code of other devices in the `lv_port_indev_init` function, otherwise the touch screen will not work properly. The modified file content is as follows:

```
#if 0
/*-----
 * Mouse
 * -----*/
/*Initialize your mouse if you have*/
mouse_init();
/*Register a mouse input device*/
lv_indev_drv_init(&indev_drv);
indev_drv.type = LV_INDEV_TYPE_POINTER;

.....

/*-----
 * Button
 * -----*/
/*Initialize your button if you have*/
button_init();
/*Register a button input device*/
lv_indev_drv_init(&indev_drv);
indev_drv.type = LV_INDEV_TYPE_BUTTON;
indev_drv.read_cb = button_read;
indev_button = lv_indev_drv_register(&indev_drv);
/*Assign buttons to points on the screen*/
static const lv_point_t btn_points[2] = {
    {10, 10}, /*Button 0 -> x:10; y:10*/
    {40, 100}, /*Button 1 -> x:40; y:100*/
};
lv_indev_set_button_points(indev_button, btn_points);
#endif
```

D. Modify touch screen initialization function

Call the initialization function defined in the touch screen driver in lvgl's touch screen initialization function (defined in the `touch_driver.c` file, different touch screen drivers correspond to different initialization functions). The modified content is as follows:

```
/*Initialize your touchpad*/
static void touchpad_init(void)
{
    /*Your code comes here*/
    touch_driver_init();
}
```

```
}
```

E. Modify the touch screen coordinate reading function

Call the coordinate reading function defined in the touch screen driver in lvgl's touch screen coordinate reading function (defined in the touch_driver.c file, different touch screen drivers correspond to different coordinate reading functions). The modified content is as follows:

```
/*Will be called by the library to read the touchpad*/
static void touchpad_read(lv_indev_drv_t * indev_drv, lv_indev_data_t *
data)
{
    touch_driver_read(indev_drv, data);
#if 0
    static lv_coord_t last_x = 0;
    static lv_coord_t last_y = 0;

    /*Save the pressed coordinates and the state*/
    if(touchpad_is_pressed()) {
        touchpad_get_xy(&last_x, &last_y);
        data->state = LV_INDEV_STATE_PR;
    }
    else {
        data->state = LV_INDEV_STATE_REL;
    }
    /*Set the last pressed coordinates*/
    data->point.x = last_x;
    data->point.y = last_y;
#endif
}
```

F. Annotate the related function definitions of other input devices

Because only the touch screen is used here, it is recommended to comment out the function definitions related to other input devices, otherwise errors may occur during compilation. The modified file content is as follows:

```
#if 0
/*-----
 * Mouse
 * -----*/
/*Initialize your mouse*/
```

```
static void mouse_init(void)
{
    /*Your code comes here*/
}

.....

/*Test if `id` button is pressed or not*/
static bool button_is_pressed(uint8_t id)
{
    /*Your code comes here*/
    return false;
}
#endif
```

At this point, the `lv_port_indev.c` file has been modified and the modified file has been saved.

G. Modify header file

Open the "`lv_port_indev.h`" file in the "`components\lv_porting`" folder of the project directory, change the 7th line from "`# if 0`" to "`# if 1`", and then save it. The modified file content is as follows:

```
/*Copy this file as "lv_port_indev.h" and set this value to "1" to enable
content*/
#if 1
```

6.2 Modify the main.c file

Open the "`main.c`" file in the "main" folder of the project engineering directory, modify the file content as shown below, and then save it.

```
#include <stdio.h>
#include "sdkconfig.h"
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "esp_system.h"
#include "esp_spi_flash.h"

#include "lvgl.h"
#include "lv_port_disp.h"
#include "lv_port_indev.h"
#include "esp_timer.h"
#include "lv_demos.h"
```

```
static void inc_lvgl_tick(void *arg)
{
    lv_tick_inc(10);
}

void app_main(void)
{
    lv_init();           //init lvgl
    lv_port_disp_init(); //init display
    lv_port_indev_init(); //init touch screen
    /* 为 LVGL 提供时基单元 */
    const esp_timer_create_args_t lvgl_tick_timer_args = {
        .callback = &inc_lvgl_tick,
        .name = "lvgl_tick"
    };
    esp_timer_handle_t lvgl_tick_timer = NULL;
    ESP_ERROR_CHECK(esp_timer_create(&lvgl_tick_timer_args,
    &lvgl_tick_timer));
    ESP_ERROR_CHECK(esp_timer_start_periodic(lvgl_tick_timer, 10 *
    1000));

    lv_demo_widgets(); //LVGL Demo

    while (1)
    {
        vTaskDelay(pdMS_TO_TICKS(10));
        lv_task_handler();
    }
}
```

7. Debugging project code

Before compiling and debugging engineering code, let's familiarize ourselves with the compilation and debugging tools of VS Code, as shown in the following figure:

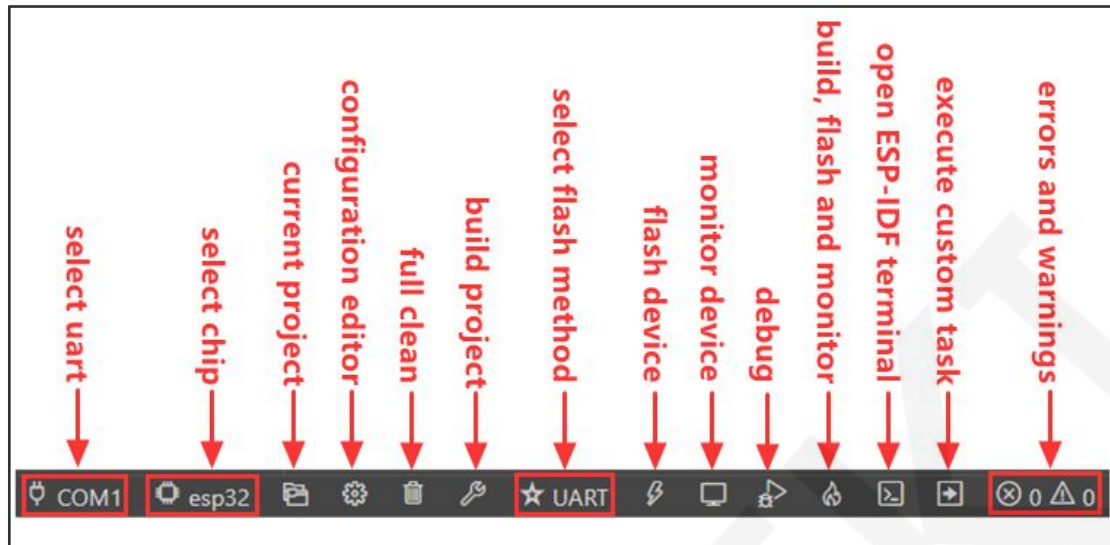


Figure 7.1 Compilation and Debugging Tools for VS Code

select uart : Select the serial port number connected to the current development board.

select chip : Select the main control chip used by the current development board.

current project : Choose to open the required project, usually not selected because it defaults to the current project.

configuration editor : Open the project menu editor (menuconfig) to configure some features of the current project.

full clean : Delete files generated by project compilation (mainly delete the build folder).

build project : Compile the code for the current project.

select flash method : Choose the burning method for engineering code, including serial port, JTAG, etc. Generally, serial port burning is chosen.

flash device : Burn the compiled code of the current project. After modifying the code, it needs to be compiled and burned before it becomes effective.

monitor device : Open the serial port output window to display the serial port output information of the development board.

debug : Debug the code of the current project.

build, flash and monitor : Combining the operations of compiling, burning, and opening the serial output window together, clicking once to achieve


these three functions, is the most commonly used operation.

open ESP-IDF terminal : Open the ESP idf command line terminal window, where you can enter the IDF command and execute it.

Execute custom task : Execute some personalized tasks that are generally not needed.

Errors and warnings : Display the number of errors and warnings generated during the compilation of engineering code.

After becoming familiar with the compilation and debugging tools of VS Code, the next step is to compile and debug the engineering code, detect errors in the code, and correct them until the compilation is successful. The steps are as follows:

- A. Click on the Build Project button  at the bottom of the VS Code software, You can see the serial terminal window outputting compilation information.
- B. When the compilation stops halfway, it indicates that there is an error in the code.

Click the bottom **error and warning number button** to view the result, as shown in the following figure:

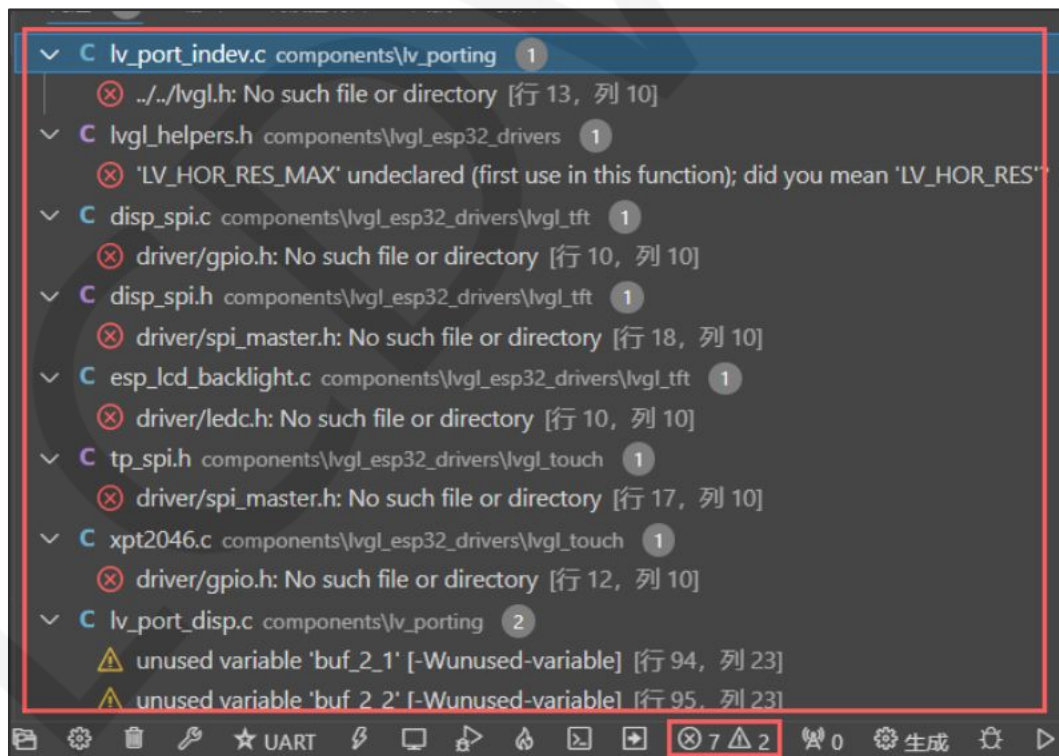
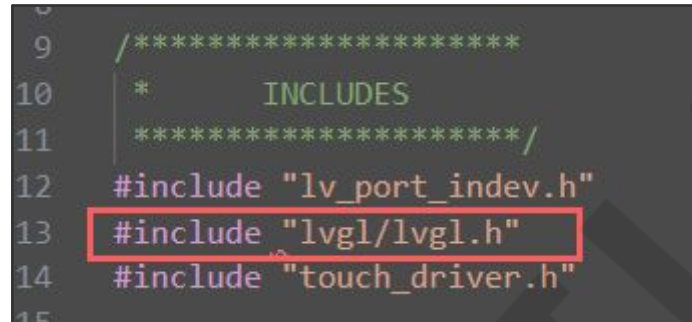


Figure 7.2 Engineering code compilation errors and warning display

C. Next, fix the error. First, check the cause of the error, then click on the error message to enter the location where the error occurred.

- The first error is that the header file `"../lvgl.h"` cannot be found. Simply modify the header file path in the `lv_port_index.c` file. As shown in the following figure:

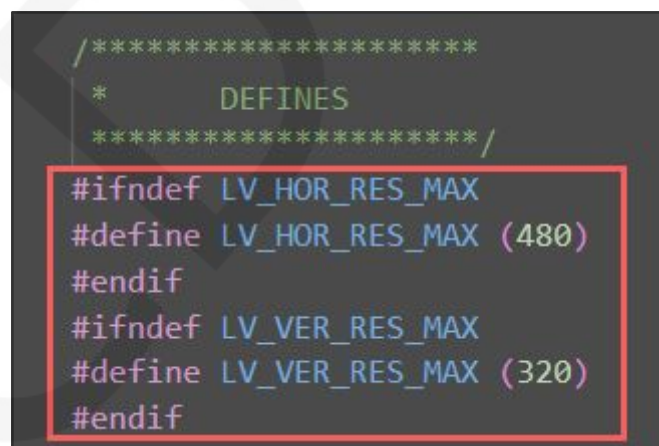


```
9  /*****
10  *      INCLUDES
11  *****/
12  #include "lv_port_index.h"
13  #include "lvgl/lvgl.h"
14  #include "touch_driver.h"
15
```

Figure 7.3 Modified File 1

- The second error is that `"LV_HOR_RES_MAX"` is not defined. Add the relevant macro definition in `lv_helpers.h`.

Please define the horizontal and vertical resolutions based on the actual situation. For example, if an LCD driven by ST7796 is used here for a vertical display, the horizontal resolution macro is defined as `LV_HOR_RES_MAX` set to 320, and the vertical resolution macro is defined as `LV_VER_RES_MAX` set to 480; If displaying in landscape mode, swap the two resolution values..



```
 /*****
 *      DEFINES
 *****/
13  #ifndef LV_HOR_RES_MAX
14  #define LV_HOR_RES_MAX (480)
15  #endif
16  #ifndef LV_VER_RES_MAX
17  #define LV_VER_RES_MAX (320)
18  #endif
```

Figure 7.4 Modify File 2

- The third to seventh errors are all header files that cannot be found. This is because the functions in `idf` are used in `lvgl_esp32_drivers`, so the `idf` file needs to be referenced. Add `"REQUIRES driver"` to line 89 of the `"components/lvgl_esp32_drivers/CMakeLists.txt"` file in the project project,

as shown in the figure below, and then save the file.

```

87 idf_component_register(SRCS ${SOURCES}
88                        INCLUDE_DIRS ${LVGL_INCLUDE_DIRS}
89                        REQUIRES driver
90                        REQUIRES lvgl)
91

```

Figure 7.5 Modified File 3

After the error is corrected, continue compiling. Compile or error, as shown in the following figure:

```

C esp_lcd_backlight.c components\lvgl_esp32_drivers\lvgl_tft 4
  x 'ledc_timer_config_t' has no member named 'bit_num' [行 52, 列 14]
  x implicit declaration of function 'gpio_matrix_out'; did you mean 'gpio_iomux_out'? [-Werror=implicit-function-decl
  x implicit declaration of function 'gpio_pad_select_gpio'; did you mean 'esp_rom_gpio_pad_select_gpio'? [-Werror=ir
  x 'SIG_GPIO_OUT_IDX' undeclared (first use in this function) [行 67, 列 43]
C st7796s.c components\lvgl_esp32_drivers\lvgl_tft 2
  x implicit declaration of function 'gpio_pad_select_gpio'; did you mean 'esp_rom_gpio_pad_select_gpio'? [-Werror=ir
  x 'portTICK_RATE_MS' undeclared (first use in this function); did you mean 'portTICK_PERIOD_MS'? [行 109, 列 42]
C lv_port_indev.c components\lv_porting 2
  y 'touchpad_is_pressed' defined but not used [-Wunused-function] [行 216, 列 13]
  y 'touchpad_get_xy' defined but not used [-Wunused-function] [行 224, 列 13]
C disp_spi.c components\lvgl_esp32_drivers\lvgl_tft 1
  y ignoring attribute 'section (".iram1.1")' because it conflicts with previous 'section (".iram1.0")' [-Wattributes] [行 30

```

Figure 7.6 Compilation Error 1

- By checking the error messages, it was found that these errors were caused by missing declarations or definitions. This is because ESP-IDF_5.0 has updated some API definitions, while lvgl_esp32_drivers still uses the old API, resulting in the inability to find the API definitions. The error has been corrected as follows:
- Open "**lvgl_esp32_drivers\lvgl_tft\esp_lcd_backlight. c**", modify according to the line number in the corresponding place by referring to the following three contents, and then save.

```

9 #include "esp_lcd_backlight.h"
10 #include "driver/ledc.h"
11 #include "rom/gpio.h"
12 #include "esp_log.h"
13 #include "soc/ledc_periph.h" // to invert LEDC output on IDF version < v4.3
14 #include "soc/gpio_sig_map.h"

```

Figure 7.7 Modified File 4


```

51         const ledc_timer_config_t LCD_backlight_timer = {
52             .speed_mode = LEDC_LOW_SPEED_MODE,
53             .duty_resolution = LEDC_TIMER_10_BIT,
54             .timer_num = config->timer_idx,
55             .freq_hz = 5000,
56             .clk_cfg = LEDC_AUTO_CLK};
57

```

Figure 7.8 Modified File 5

```

64         // Configure GPIO for output
65         bckl_dev->index = config->gpio_num;
66         esp_rom_gpio_pad_select_gpio(config->gpio_num);
67         ESP_ERROR_CHECK(gpio_set_direction(config->gpio_num, GPIO_MODE_OUTPUT));
68         gpio_matrix_out(config->gpio_num, SIG_GPIO_OUT_IDX, config->output_invert, false);
69     }

```

Figure 7.9 Modified File 6

- Open "**components\lvgl_esp32_drivers\lvgl_tft\st7796s.c**" (taking st7796s.c as an example, if using another driver, you need to modify it in the corresponding file), modify it according to the line number in the corresponding place by referring to the following content, and then save it.

```

84         //Initialize non-SPI GPIOs
85         esp_rom_gpio_pad_select_gpio(ST7796S_DC);
86         gpio_set_direction(ST7796S_DC, GPIO_MODE_OUTPUT);
87
88     #if ST7796S_USE_RST
89         esp_rom_gpio_pad_select_gpio(ST7796S_RST);
90         gpio_set_direction(ST7796S_RST, GPIO_MODE_OUTPUT);
91
92         //Reset the display
93         gpio_set_level(ST7796S_RST, 0);
94         vTaskDelay(100 / portTICK_PERIOD_MS);
95         gpio_set_level(ST7796S_RST, 1);
96         vTaskDelay(100 / portTICK_PERIOD_MS);
97     #endif
98
99     ESP_LOGI(TAG, "Initialization.");
100
101     //Send all the commands
102     uint16_t cmd = 0;
103     while (init_cmds[cmd].databytes != 0xff)
104     {
105         st7796s_send_cmd(init_cmds[cmd].cmd);
106         st7796s_send_data(init_cmds[cmd].data, init_cmds[cmd].databytes & 0x1F);
107         if (init_cmds[cmd].databytes & 0x80)
108         {
109             vTaskDelay(100 / portTICK_PERIOD_MS);
110         }
111         cmd++;
112     }

```

Figure 7.10 Modify File 7

D. After the modification is complete, continue compiling. When the **"Memory Type**

"Usage Summary" table appears and the "Build Successful" prompt appears in the bottom right corner, it indicates that the compilation is successful.

Memory Type/Section	Used [bytes]	Used [%]	Remain [bytes]	Total [bytes]
Flash Code	316713	9.48	3025591	3342304
.text	316713	9.48		
Flash Data	269380	6.42	3924892	4194272
.rodata	269124	6.42		
.appdesc	256	0.01		
DRAM	156524	86.6	24212	180736
.bss	146336	80.97		
.data	10188	5.64		
IRAM	77695	59.28	53377	131072
.text	76667	58.49		
.vectors	1027	0.78		
RTC SLOW	24	0.29	8168	8192
.rtc_slow_reserved	24	0.29		



Total image size: 673975 bytes (.bin may be padded larger)

Build Successfully

Figure 7.11 Compilation Successful

8. Burn and run

After successful compilation, the program can be burned to verify if it can run normally. The steps are as follows:

- A. Connect the display module to the ESP32 device according to the pin definitions configured earlier (if the ESP32 is an onboard device, no wiring is required), then connect the ESP32 device to the computer, and select the chip, serial port (USB to serial port driver needs to be installed), and burning method in the bottom toolbar of the VS code software.
- B. Click the burn button  on the bottom toolbar of the VS code software (if it has already been compiled) or click the build/burn/monitor button  (if it has not been compiled or the build has been deleted), and you can burn it now.
- C. After successful burning, if all configurations are correct, you can see the LVGL demo interface appearing on the display screen, as shown in the following figure:

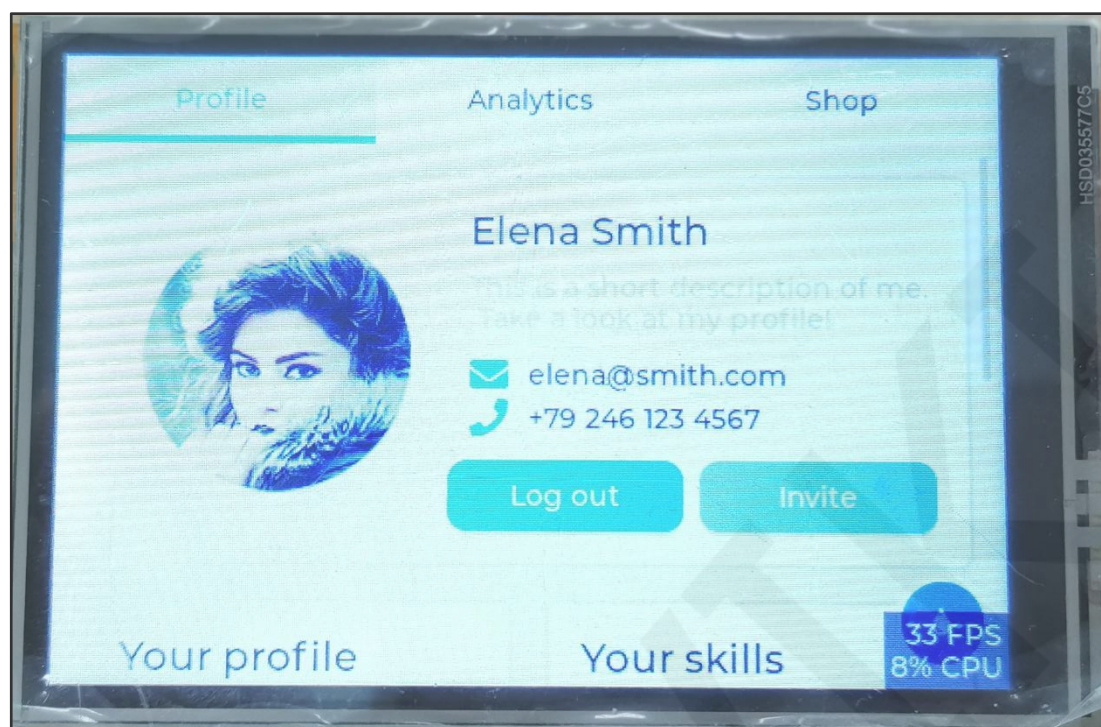


Figure 8.1: The program is running normally

D. If the display module is equipped with a touch screen, there may be touch abnormalities. For example, capacitive touch screens have issues with no touch and running errors, while resistive touch screens have problems with inaccurate positioning, X and Y direction swapping, reversed X direction coordinate values, and reversed Y direction coordinate values due to the lack of calibration parameters. In general, it is necessary to solve the problem by modifying the code and configuration parameters.

E. Here is an explanation on how to set the calibration parameters for a resistive touch screen. The steps are as follows:

- **Calibrate touch direction**

Firstly, slide horizontally and vertically on the touch screen. If the content on the display screen also scrolls horizontally and vertically, then

The horizontal and vertical directions of the touch screen are not swapped, otherwise it is necessary to swap the horizontal and vertical directions of the touch screen. The method is as follows:

Open the SDK configuration editor interface (refer to LVGL configuration instructions for instructions), enter "**touch**" in the search bar, and then find the

"Swap XY" option. If this option was previously checked, remove the check; If not checked, check it and save the settings. As shown in the following figure:

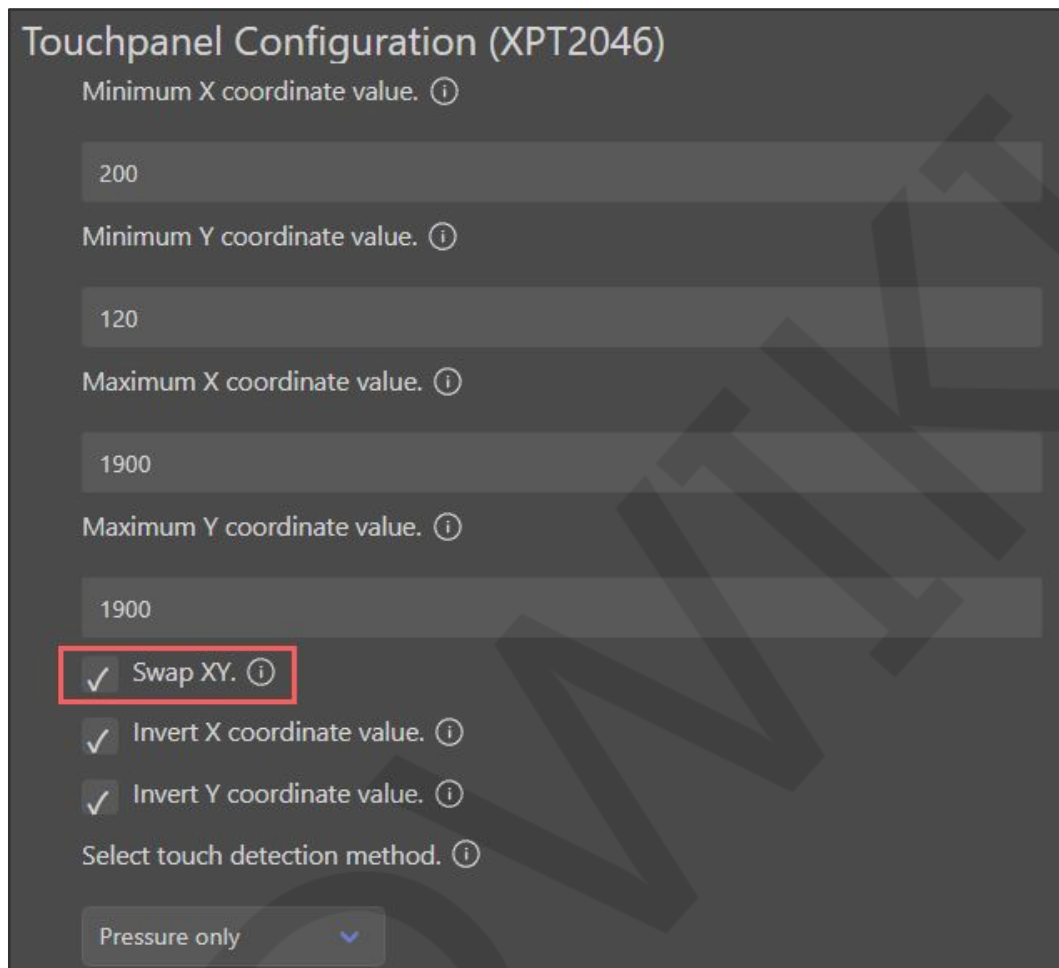
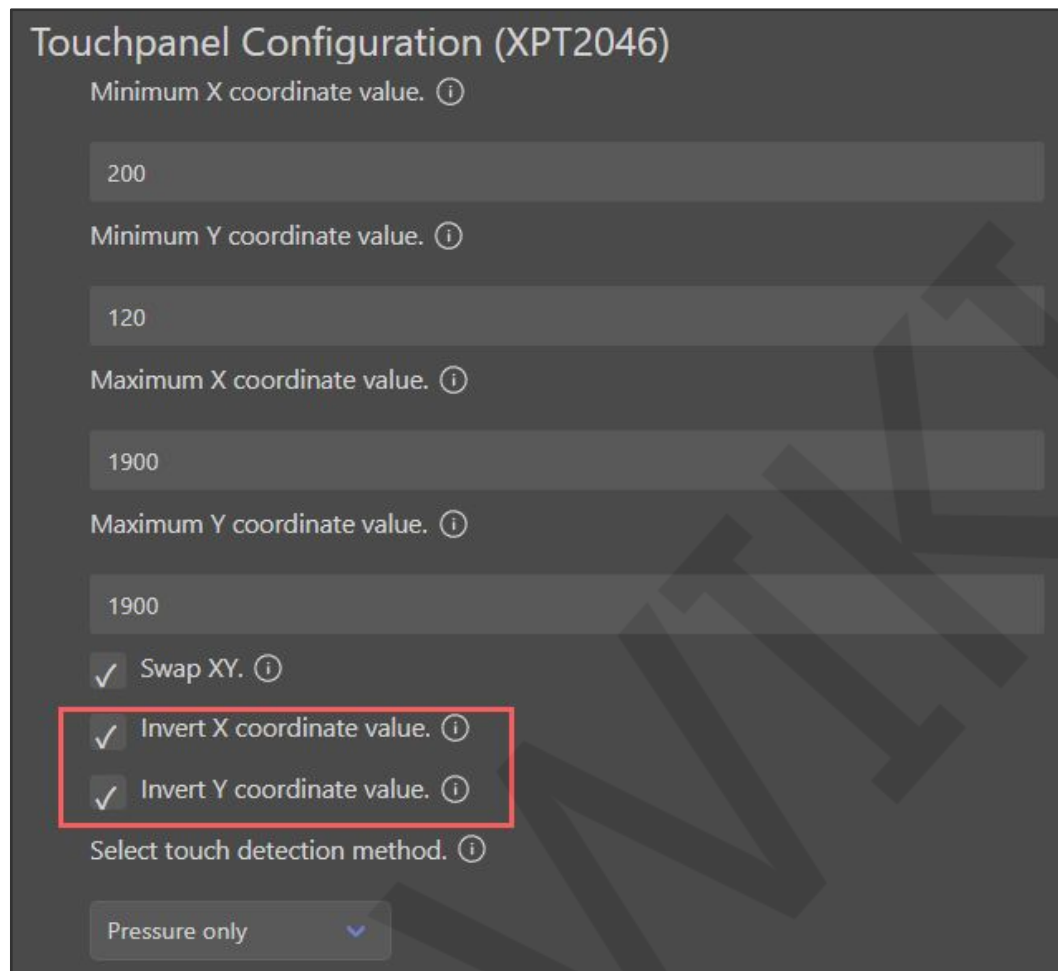


Figure 8.2 Exchanging X and Y directions

After saving the settings, recompile and burn the program. After the program runs, slide horizontally and vertically on the touch screen. If the horizontal and vertical scrolling directions of the display screen content are consistent with the sliding direction of the touch screen, it means that the horizontal and vertical coordinates of the touch screen are consistent with the display screen. Otherwise, the horizontal or vertical coordinates need to be reversed. The modification method is as follows:

Install the above method and find the "**Invert X coordinate value**" and "**Invert Y coordinate value**" options in the SDK configuration editor interface. If the options were previously checked, uncheck them; If not checked, check it and save the settings. As shown in the following figure:



Touchpanel Configuration (XPT2046)

Minimum X coordinate value. ⓘ

200

Minimum Y coordinate value. ⓘ

120

Maximum X coordinate value. ⓘ

1900

Maximum Y coordinate value. ⓘ

1900

☒ Swap XY. ⓘ

☒ Invert X coordinate value. ⓘ

☒ Invert Y coordinate value. ⓘ


Select touch detection method. ⓘ

Pressure only ▼

Figure 8.3 Setting X or Y direction coordinates

● Set calibration parameters

After successful touch direction calibration, it is necessary to set calibration parameters to ensure accurate touch.

Firstly, open the SDK configuration editor interface, click on "**Bootloader manager**", and then set "**Bootloader log verbose**" to "**Info**", as shown in the following figure. This way, the log information of the code can be output through the serial port (if it is already set by default, this step is ignored). Finally, click the button  to recompile and burn the program.

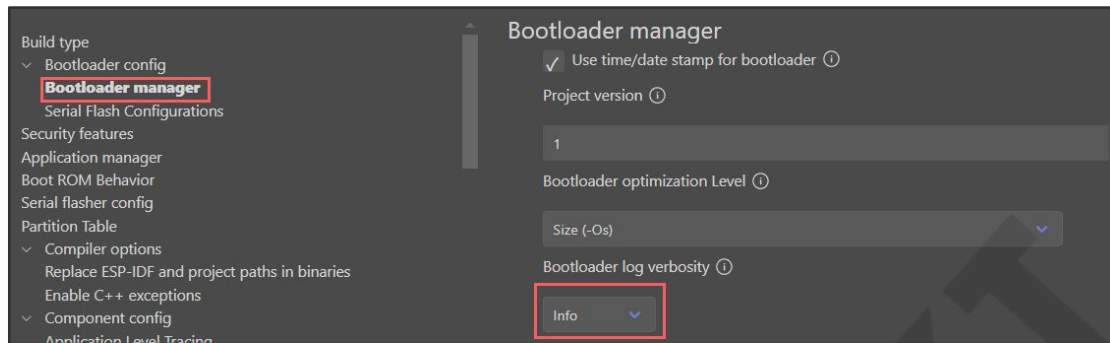


Figure 8.4 Setting log output level

After the program is burned, run the program. Using the display direction as a reference, click on the top left corner of the touch screen. At this point, a set of touch screen ADC values and display screen coordinate values can be obtained through the serial terminal, as shown in the following figure:

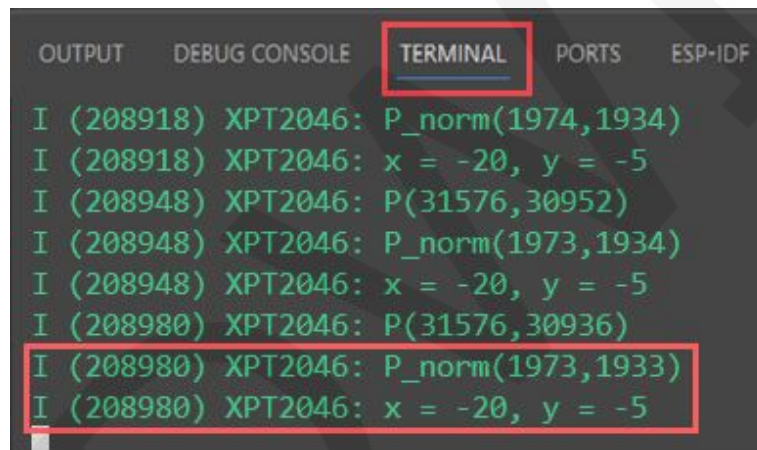


Figure 8.5 Obtaining Calibration Value 1

Then click on the bottom right corner to obtain another set of touch screen ADC values and display screen coordinate values, as shown in the following figure:

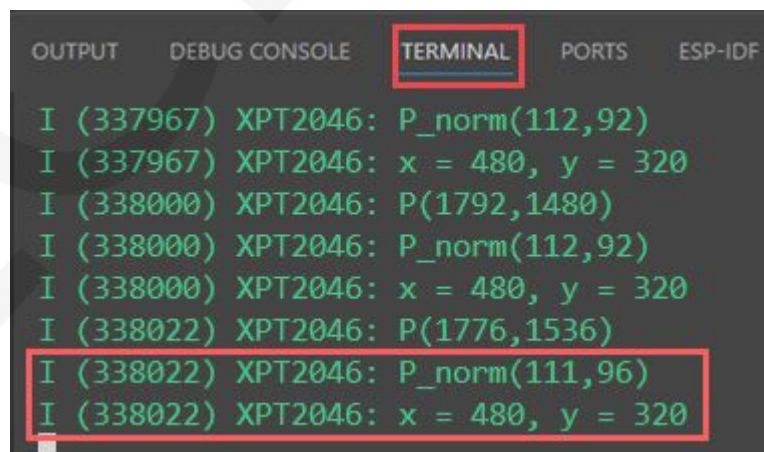


Figure 8.6 Obtaining Calibration Values 2

By comparing these two sets of calibration values, the minimum and maximum ADC values in the X direction and the minimum and maximum ADC values in the Y direction of the touch screen can be obtained. For example, the minimum ADC value in the X direction here is 111, and the maximum ADC value is 1973; The minimum ADC value in the Y direction is 96, and the maximum ADC value is 1933.

Next, open the SDK configuration editor interface, enter "**touch**" in the search bar, find the corresponding position, fill in these four values, and then click the "**save**" button to save the configuration. As shown in the following figure:

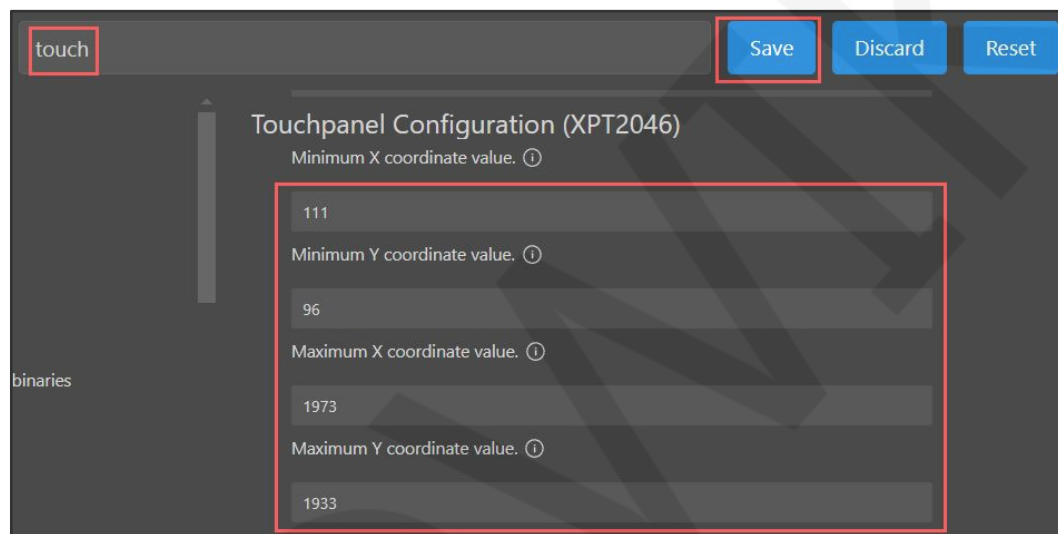


Figure 8.7 Setting Calibration Values

Finally, recompile and burn.

Please note that after changing the display direction, the touch parameters need to be reconfigured.