

ESP-IDF LVGL 移植说明

目 录

- 1. 新建 VS Code 工程..... 3
- 2. 拷贝工程文件..... 12
- 3. 配置 LVGL..... 13
 - 3.1. 通过 menuconfig 菜单配置界面配置 lvgl..... 14
 - 3.2. 通过文件配置 lvgl..... 18
- 4. 配置 LCD..... 19
- 5. 配置触摸屏..... 22
- 6. 修改工程文件..... 24
 - 6.1. 修改 lv_porting 文件夹内容..... 24
 - 6.2. 修改 main.c 文件..... 32
- 7. 调试工程代码..... 33
- 8. 烧录并运行..... 38

1. 新建VS Code工程

ESP-IDF VSCode开发环境搭建完成后，接下来新建项目工程。新建项目工程的步骤如下：

- A、打开Visual Studio Code（VS Code）软件，按“Ctrl+Shift+P”组合键或者在顶部的搜索栏里点击，然后在下拉菜单里选择“显示并运行命令”，这样就进入了命令输入状态，如下图所示：

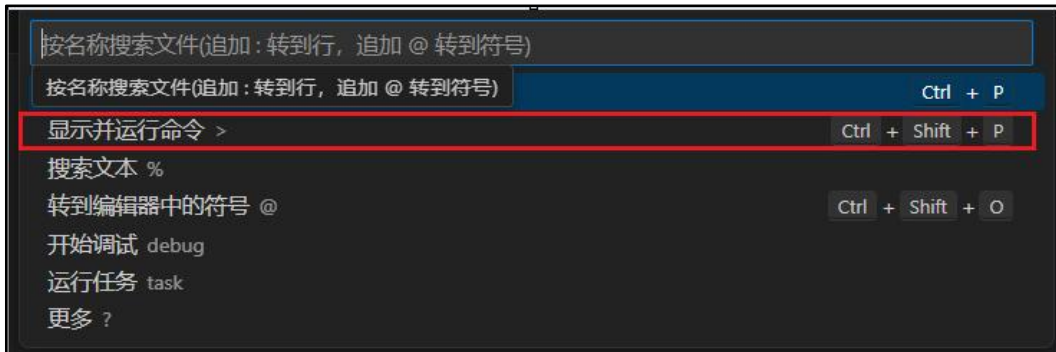


图1.1 进入命令输入状态

- B、在命令输入栏里输入“新项目”，然后按Enter键确认或者点击下拉菜单的新项目选项，如下图所示：

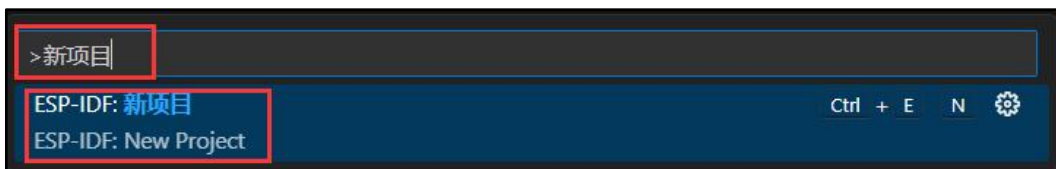


图1.2 创建新项目

- C、在弹出的界面对新项目工程进行配置，配置的内容如下图所示：

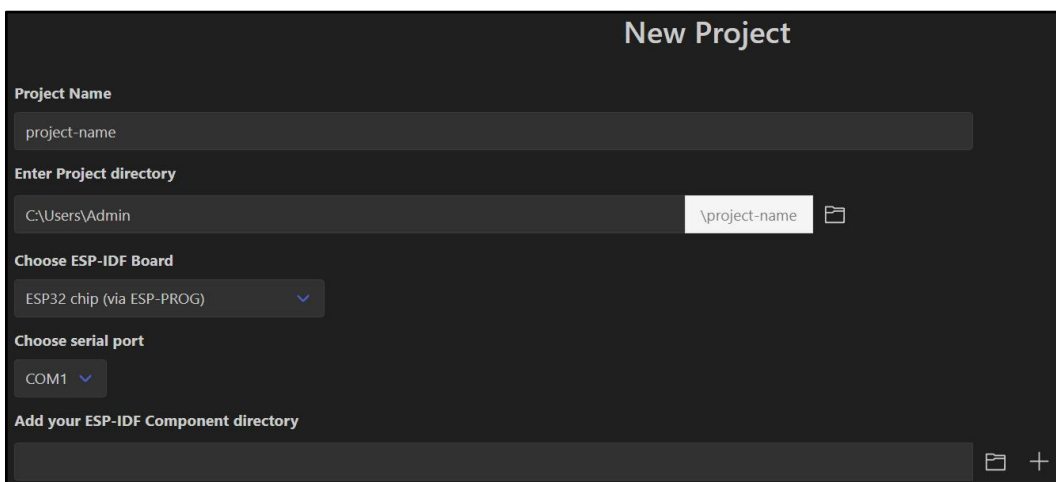


图1.3 配置新项目工程

Project Name: 项目工程名称，只能使用英文命名

Enter Project directory: 项目工程保存的路径名称，只能选择全英文的路径，否则编译时会因为找不到路径而报错。

Choose ESP-IDF Board: 选择使用的芯片类型和下载以及调试接口类型。从下拉菜单里根据实际情况选择。

Choose Serial Port: 选择串口号。选择开发板实际连接的串口号。此串口号在已经连接设备时新建项目工程会自动选择。没连接设备时，可不选。

Add your ESP-IDF Component director: 添加项目第三方组件，如没有，则无需添加。

这里项目工程配置如下图所示：

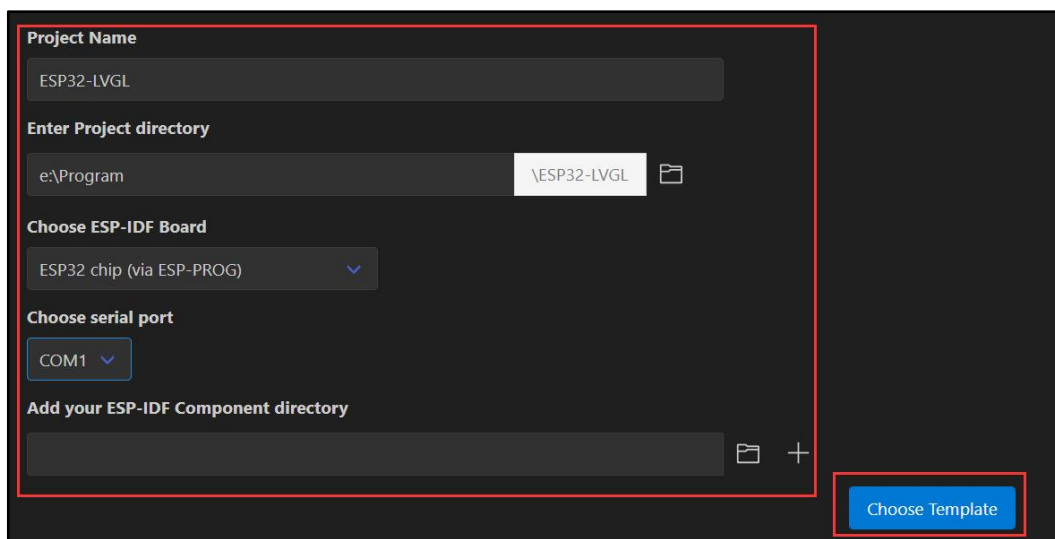


图1.4 项目配置示例

D、项目工程配置完成后，点击图1.4所示的“**Choose Template**”按钮，在弹出的界面里，可以看到很多项目工程模板，可以根据实际的项目需求来选择相应的模板。这里选择官方的基本模板。具体操作为：首先在模板类别下拉菜单里选择“**ESP-IDF**”，然后在 get-started 里选择“**sample_project**”，最后点击“**Create project using template sample_project**”选项，如下图所示：

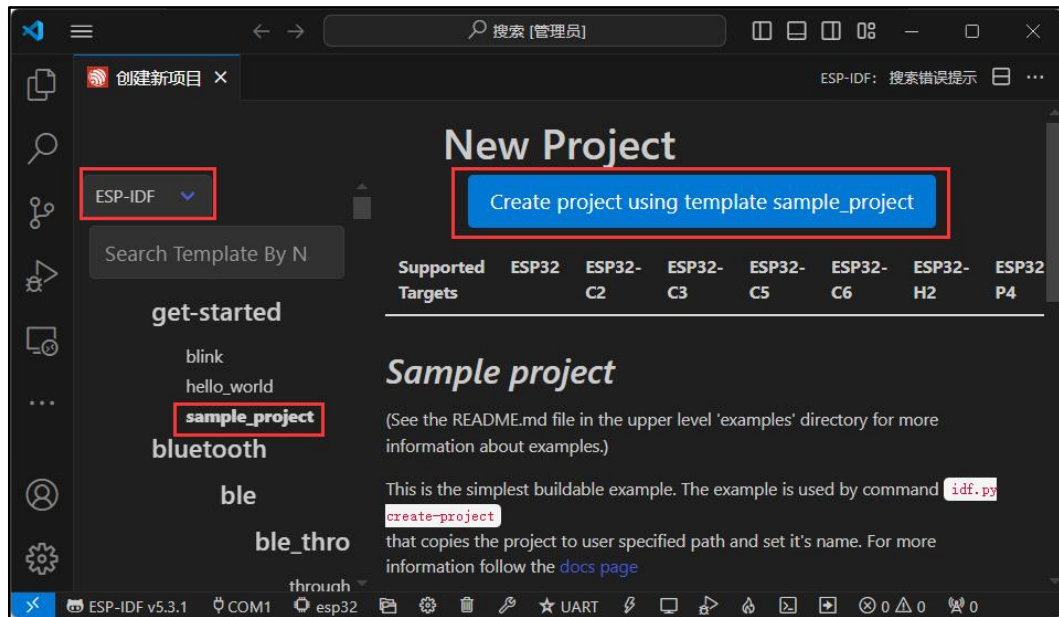


图1.5 选择项目模板

E、接下来在右下角弹出来的信息框里点击“**Yes**”选项（如下图所示），此时会打开一个新的窗口来显示项目工程内容。

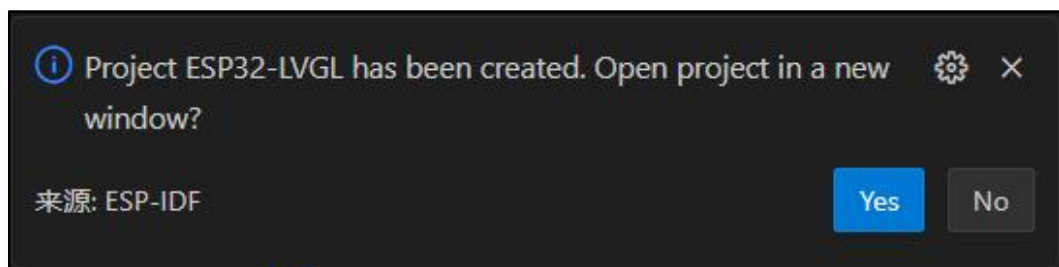


图1.6 打开项目窗口

F、在项目窗口弹出的界面里选择“**是，我信任此作者**”，如下图所示：



图1.7 选择文件模式

G、项目创建后，在VS Code资源管理区域可以看到项目工程文件，如下图所示：

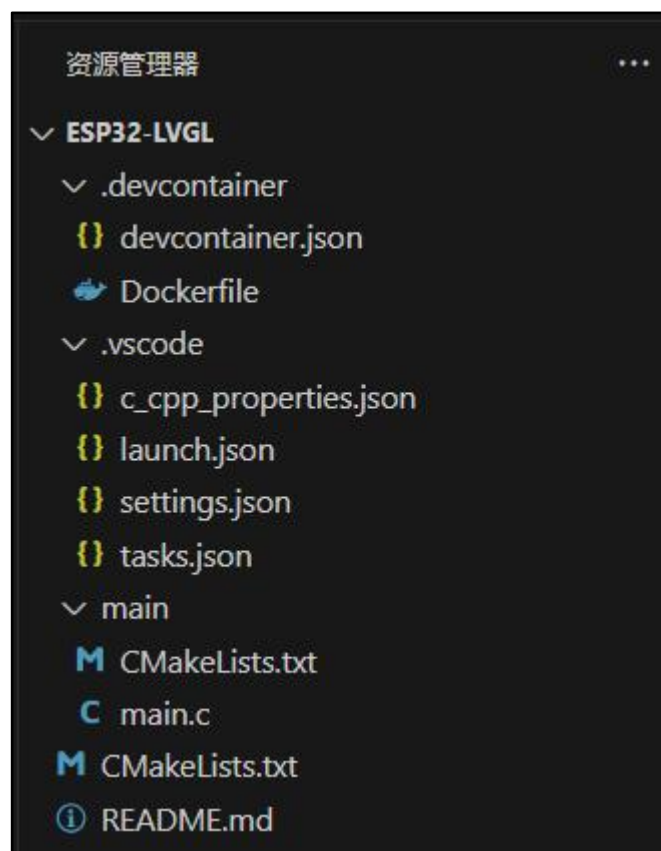


图1.8 项目文件

- .devcontainer**: 此文件夹的文件用来定义开发容器环境的配置。
- .vscode**: 此文件夹的文件用来对项目进行个性配置、工作环境配置、任务配置、调试器配置、编译配置。这些配置只针对当前项目有效。

以上两个文件夹及文件是VS Code创建项目工程时，自动生成的，不需要手动编写。

main: 主应用程序目录，包含main.c主应用程序文件和其Cmake文件。

CMakeLists.txt: 主Cmake文件，定义了项目工程基本设置和组件。

- H、接下来配置项目工程，点击VS Code底部的设置按钮（齿轮图标）进入menuconfig菜单配置界面，如下图所示：

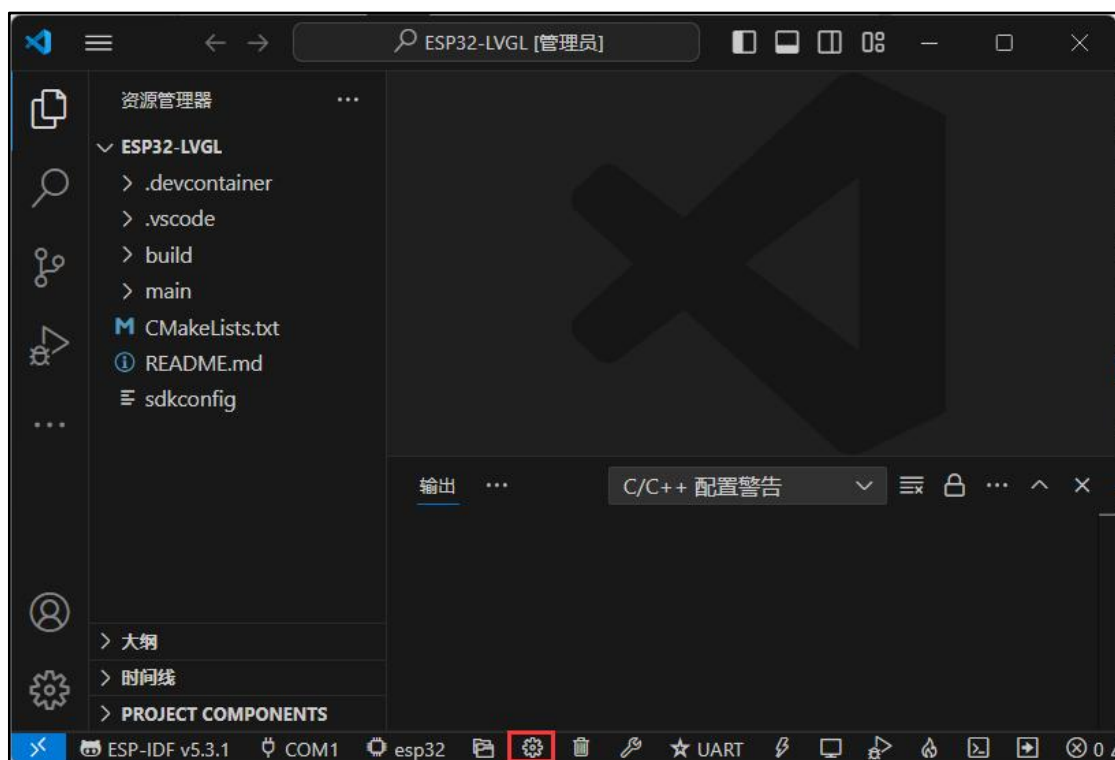


图1.9 进入项目配置界面

- I、在menuconfig菜单配置界面的“Search parameter”搜索栏里输入“flash”，进入flash配置界面，如下图所示：

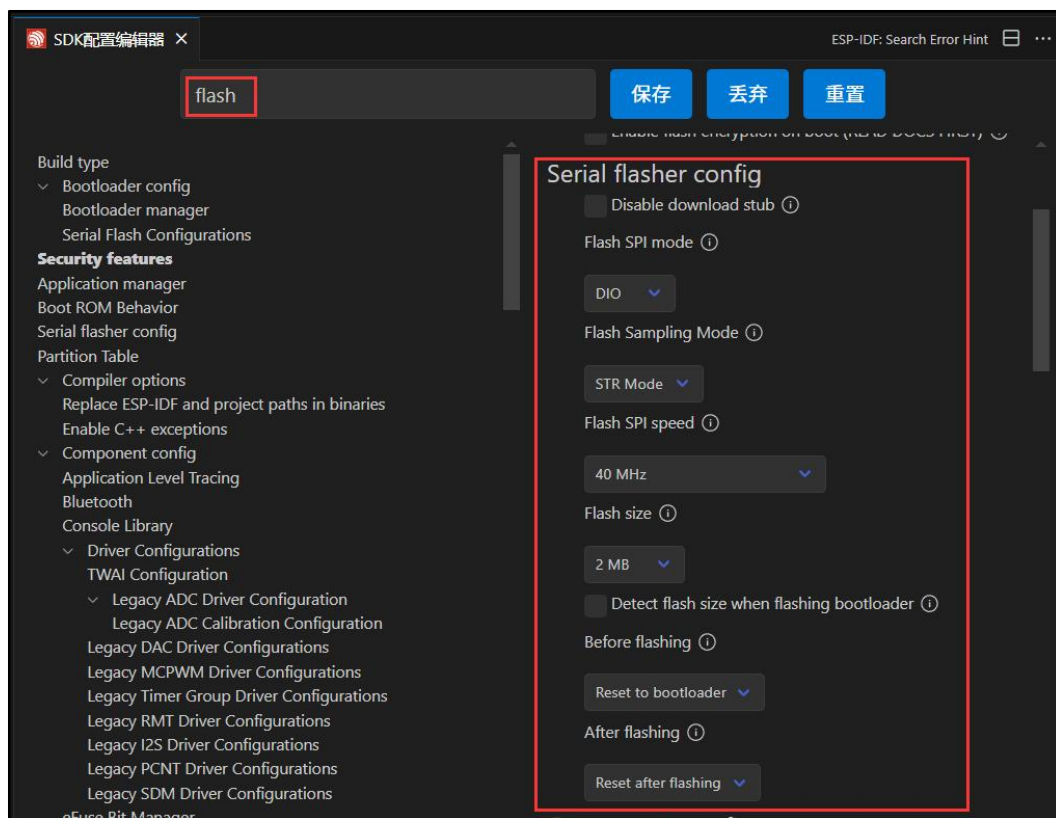


图1.10 配置flash

Flash SPI mode: Flash通信模式，可选参数有：QIO、DIO、QOUT、DOUT。

其中QIO为地址和数据都使用4线进行通信；DIO为地址和数据都使用2线进行通信；QOUT为只有数据使用4线进行通信；DOUT为只有数据使用2线进行通信。需根据Flash实际连接方式进行选择。

Flash Sampling Mode: Flash采样模式，只有STR Mode这一种采用模式。

Flash SPI speed: Flash SPI速率，可选参数有：20M、26M、40M、80M。为了获取最佳性能，一般选择80M速率。

Flash Size: Flash容量，可选参数有：1M、2M、4M、8M、16M、32M、64M、128M，需根据Flash实际容量选择。

Before flashing: 选择在烧录Flash之前是否通过烧录工具复位ESP32芯片，一般情况下选择复位，因为这样可以自动进入bootloader。如果选择不复位，则需要手动复位，才能进入下载模式。

After flashing: 选择在烧录Flash之后是否通过烧录工具复位ESP32芯片，一般情况下选择复位，因为这样可以自动运行烧录好的应用程序。如果选择不复位，则需手动复位，才能运行烧录好的应用程序。

其他的Flash设置保持默认即可。这里flash配置如下图所示：

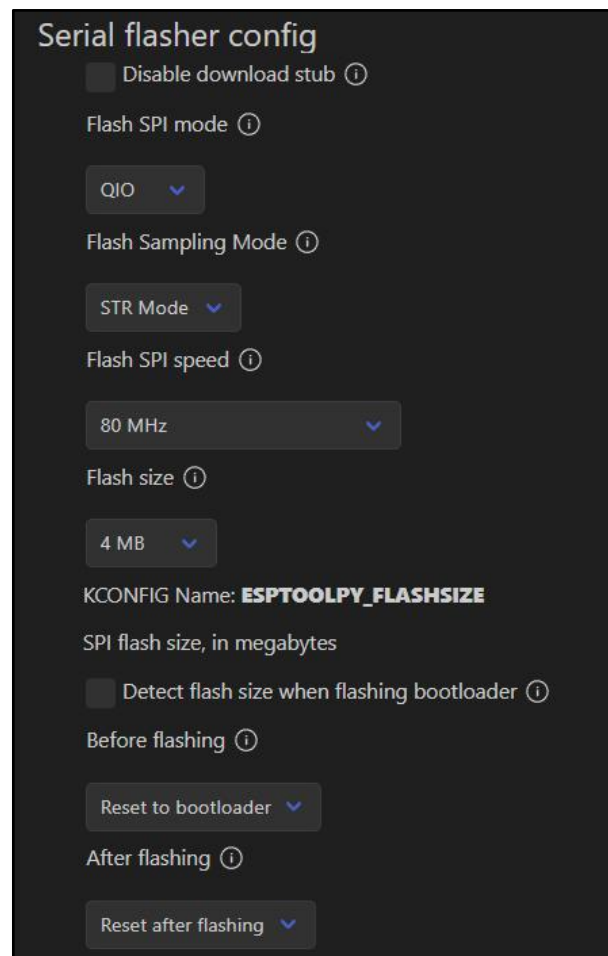


图1.11 项目flash配置

J、在menuconfig菜单配置界面的“Search parameter”搜索栏里输入“Partition Table”，进入Partition Table配置界面，如下图所示：

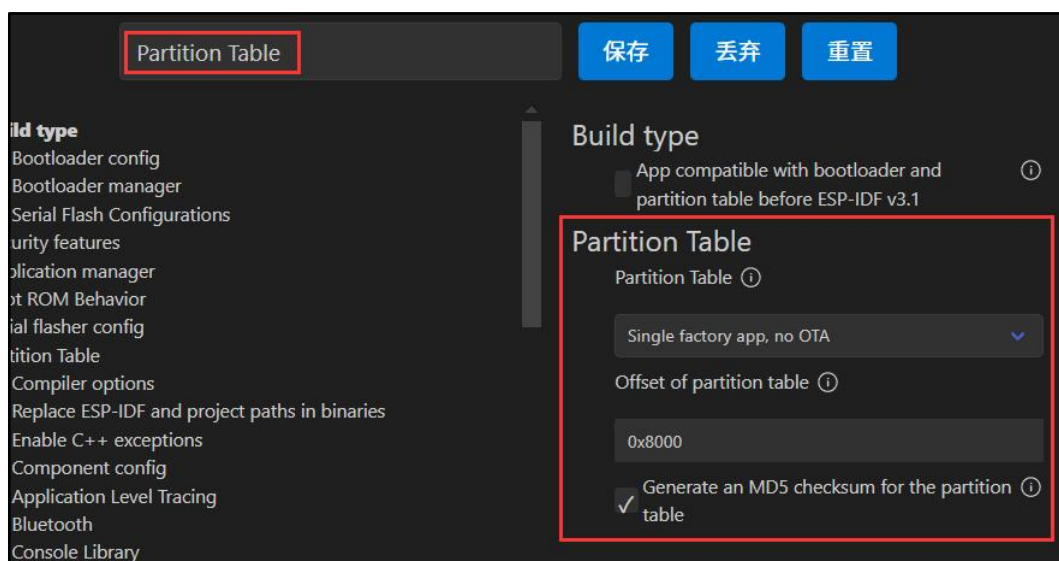


图1.12 配置Partition Table

Partition Table: Flash分区方案，一般选择默认配置，如果项目文件多，编译生成的二进制文件比较大，可以选择“**single factory app(large), no OTA**”。还可以选择用户自己配置的分区方案。

Offset of partition table: Flash分区方案表存放的偏移地址，一般选择默认值。

K、如果使用的ESP32模组含有PSRAM，则可以对PSRAM进行配置，如果没有，则忽略该步骤。

在menuconfig菜单配置界面的“**Search parameter**”搜索栏里输入“**PSRAM**”，进入PSRAM配置界面。首选需要勾选“**Support for external, SPI-connected RAM**”选项才能出现PSRAM配置项，根据实际情况对PSRAM进行配置。如下图所示：

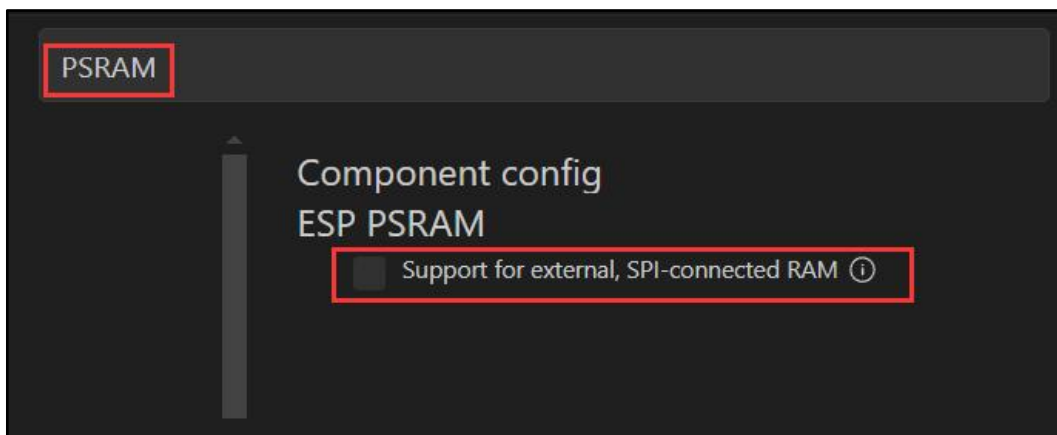


图1.13 配置PSRAM

L、在menuconfig菜单配置界面的“**Search parameter**”搜索栏里输入“**CPU frequency**”，进入CPU frequency配置界面。CPU频率可选项有：80MHz、160MHz、240MHz。为了发挥最大性能，一般选择240MHZ，如下图所示：

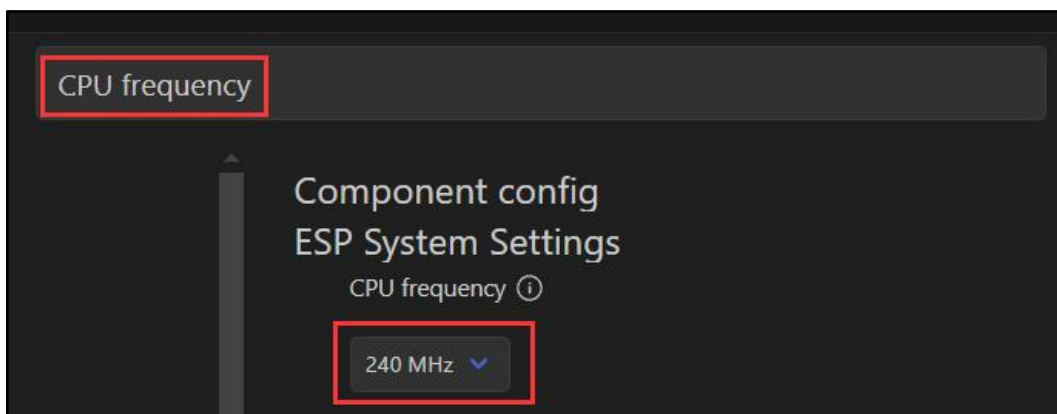


图1.13 配置CPU frequency

M、在menuconfig菜单配置界面的“**Search parameter**”搜索栏里输入“**FreeRTOS**”，进入FreeRTOS配置界面。将**configTICK_RATE_HZ**选项值由默认值100改为1000，其他值保持默认，如下图所示：

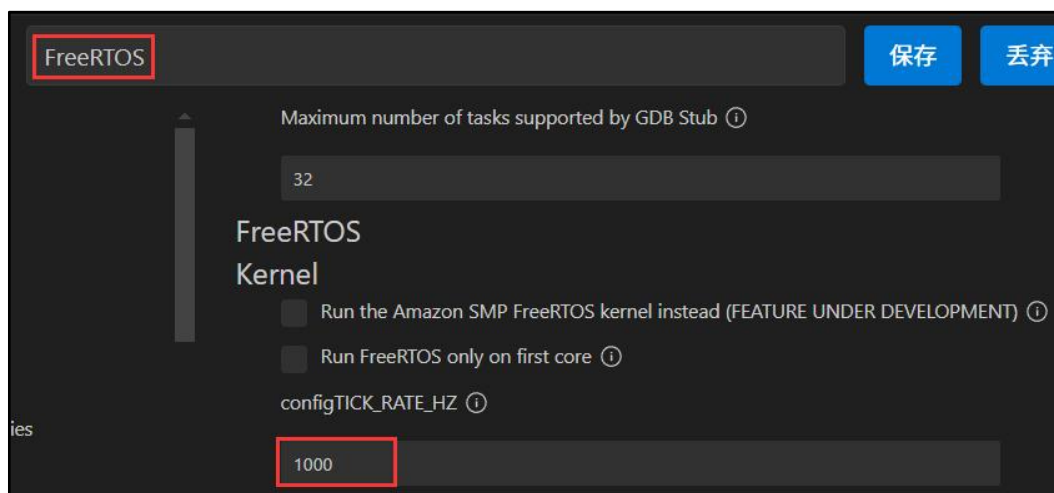


图1.14 配置CPU frequency

N、配置完成后，点击“保存”按钮，如下图所示。



图1.15 保存项目配置

至此，新建项目工程的基本工作已经完成。可以看到项目里多了两个文件，如下图所示：

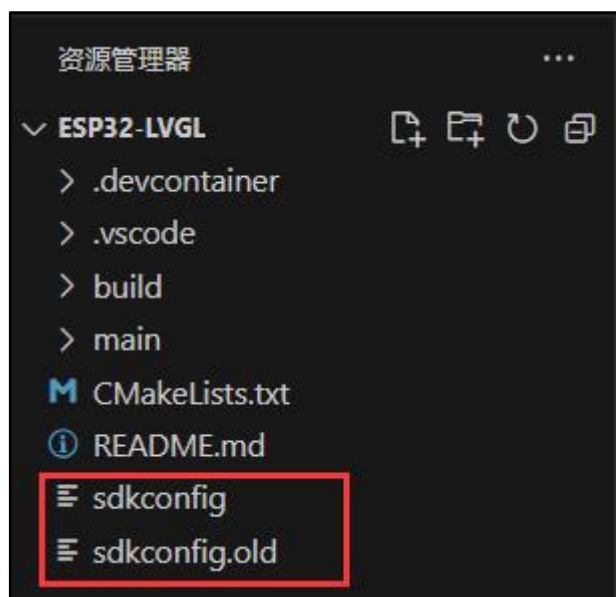


图1.16 项目文件

其中**sdkconfig**为最新生成的系统配置文件，**sdkconfig.old**为旧的系统配置文件。
修改系统配置时，也可以在**sdkconfig**文件里修改。

2. 拷贝工程文件

项目工程新建完成后，接下来需要拷贝相关的工程文件。步骤如下：

A、从如下地址分别下载LVGL v8.3.11和lvgl esp32 drivers文件压缩包。

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

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

B、打开新建的项目工程目录，新建“components”文件夹，如下图所示：

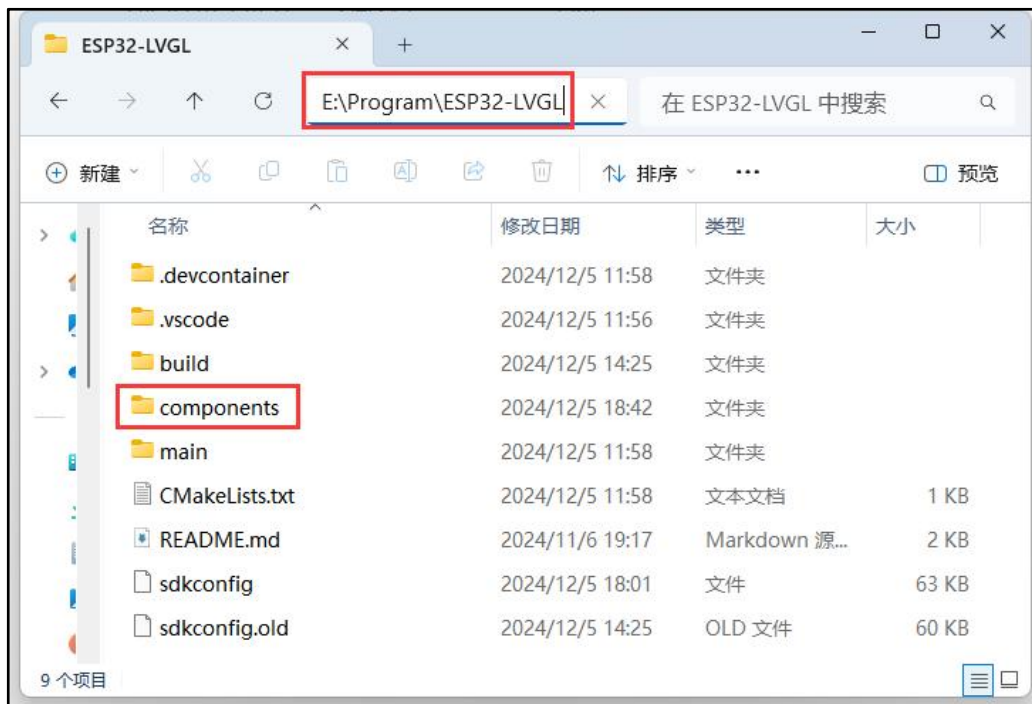


图2.1 新建components文件夹

C、将下载的LVGL v8.3.11和lvgl esp32 drivers文件压缩包都拷贝到“components”文件夹并解压。解压完成后对解压后的文件夹重命名，并删除压缩包，如下图所示：

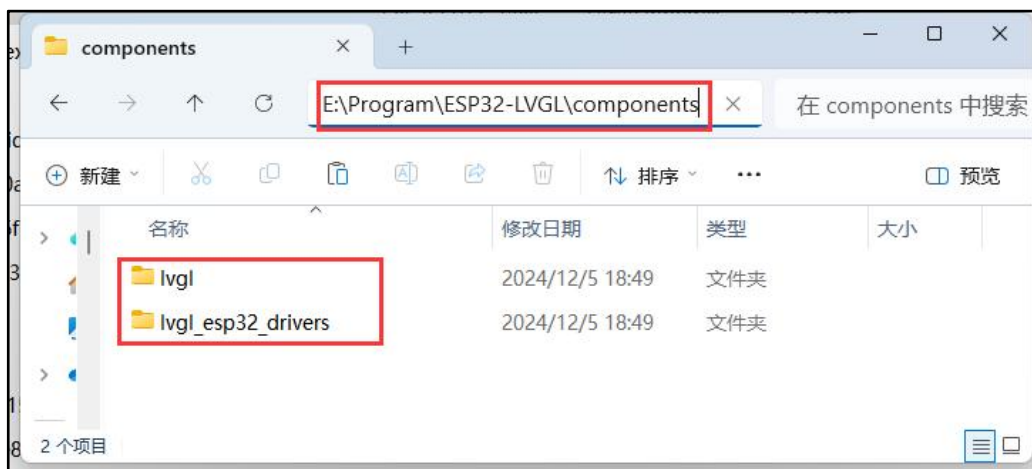


图2.2 解压文件到components文件夹

D、在项目工程的components文件夹下新建“lv_porting”文件夹，如下图所示：

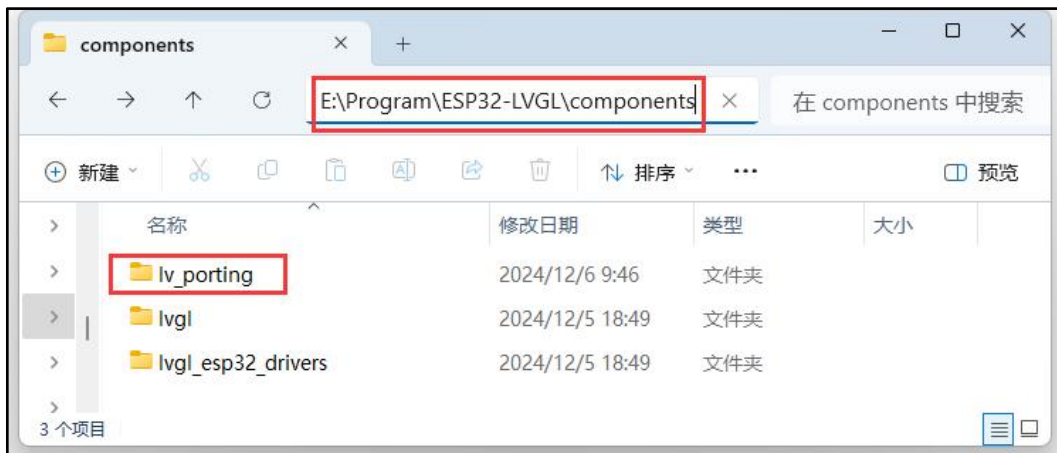


图2.3 新建lv_porting文件夹

E、打开工程目录下的“components\lvgl\examples\porting”目录，将该目录下的“lv_port_disp_template.c”、“lv_port_disp_template.h”、“lv_port_indev_template.c”、“lv_port_indev_template.h”四个文件拷贝到工程目录下的“components\lv_porting”文件夹并重命名，如下图所示：

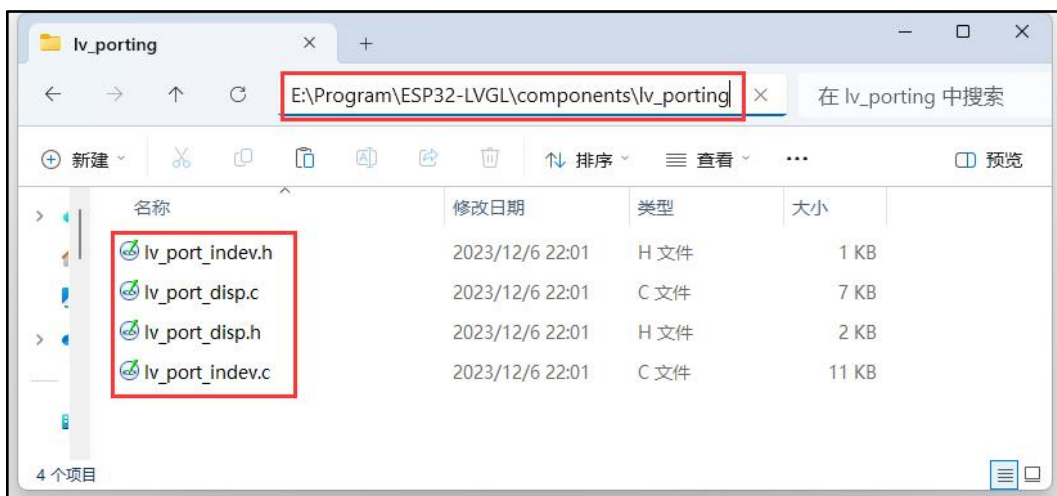



图2.4 拷贝文件到lv_porting文件夹

至此相关的文件拷贝完毕。

3. 配置LVGL

lvgl提供的强大的配置功能，可以根据自己的项目做正确的配置，以便项目可以正常工作。lvgl配置有两种方式：通过文件配置和通过menuconfig菜单配置界面配置。这里重点介绍通过menuconfig菜单配置界面配置lvgl。通过文件配置lvgl的选项和其是一致的，只是操作界面不一样。

3.1 通过menuconfig菜单配置界面配置lvgl

- A、点击VS Code底部的齿轮按钮，或者点击顶部的搜索栏，在下拉菜单里选择“显示并运行命令”，然后在搜索栏输入“配置编辑器”，在显示的结果里点击“ESP-IDF:SDK配置编辑器(Menuconfig)”。这样就打开SDK配置编辑器界面。
- B、在配置编辑器界面的搜索栏里输入“lvgl”，就可以找到lvgl的配置选项了，如下图所示：

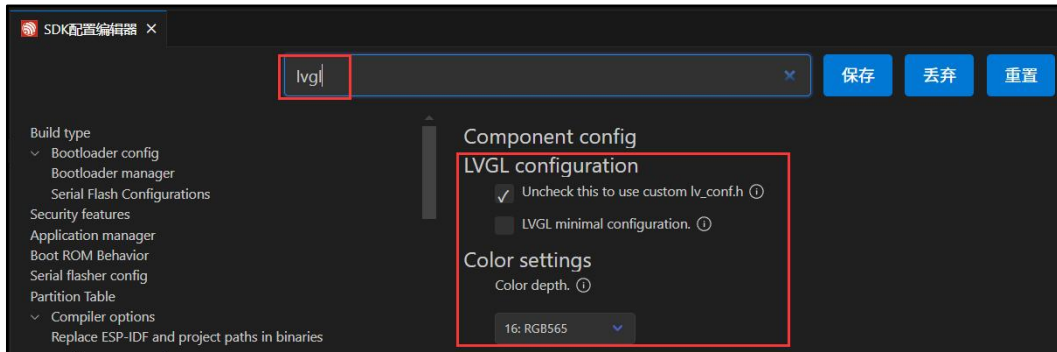


图3.1 搜索LVGL配置

- C、接下来就可以对LVGL进行配置，这里只介绍LCD相关的一些配置，其他配置，如果不需要特意修改，保持默认即可。配置介绍如下：

● lvgl总配置

当勾选“Uncheck this to use custom lv_conf.h”选项，则表示通过menuconfig菜单配置界面配置lvgl；不勾选则表示通过文件配置lvgl（下文有介绍）。

当勾选“LVGL minimal configuration.”选项，则表示对lvgl进行最小化配置，一般不勾选。

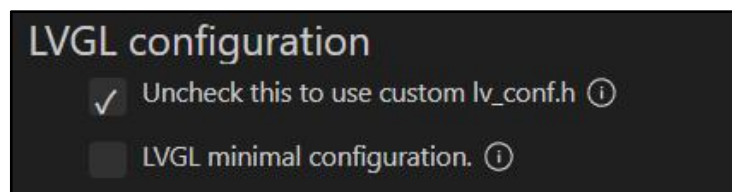


图3.2 LVGL总配置

● LCD颜色配置

Color depth一般选择“16:RGB565”

如果LCD使用SPI总线通信，则需要勾选“Swap the 2 bytes of RGB565 color. Useful if the display has an 8-bit interface (e.g. SPI).”选项。其他选项保持默认即可。

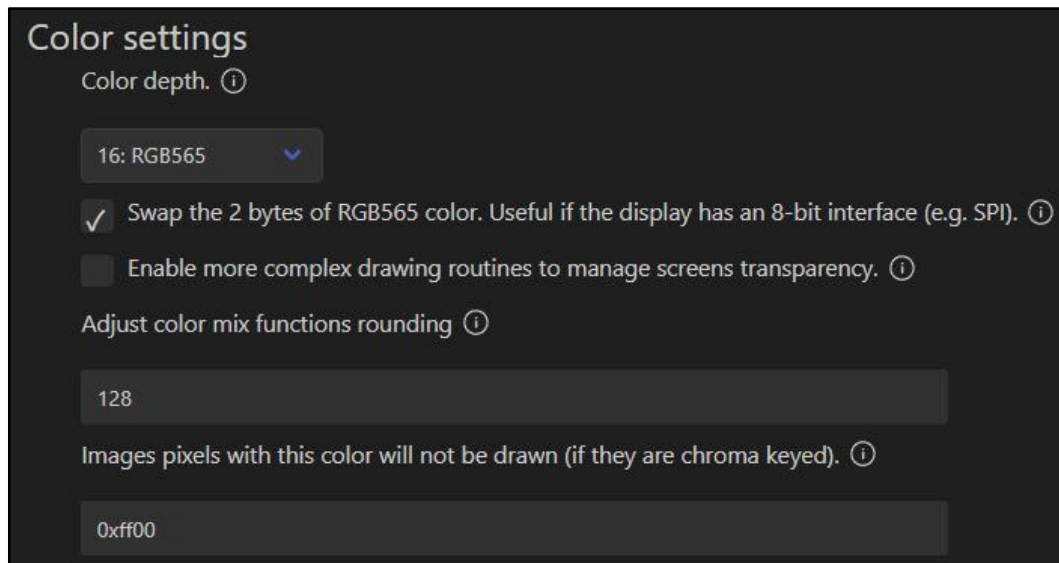


图3.3 lcd颜色配置

- 内存配置

Size of the memory used by ``lv_mem_alloc`` in kilobytes 一般设为64。如果所使用的设备内存足够大，可以将该数值设大点。其他配置保持默认即可。

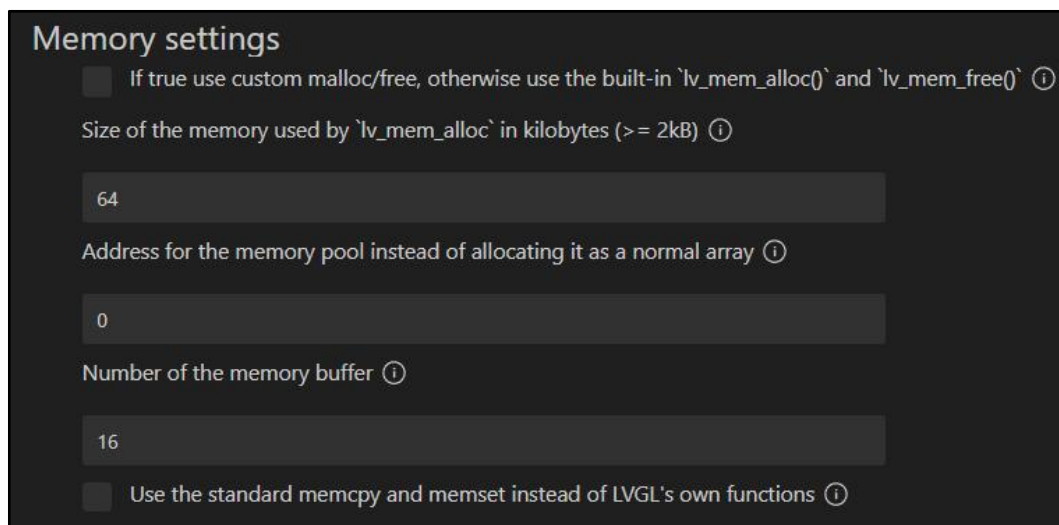


图3.4 内存配置

- 其他配置

勾选“Show CPU usage and FPS count.”选项，并把位置设为“Bottom right”。这样在运行应用程序时，可以在屏幕右下角看到当前CPU使用率和屏幕刷新率。其他配置保持默认。

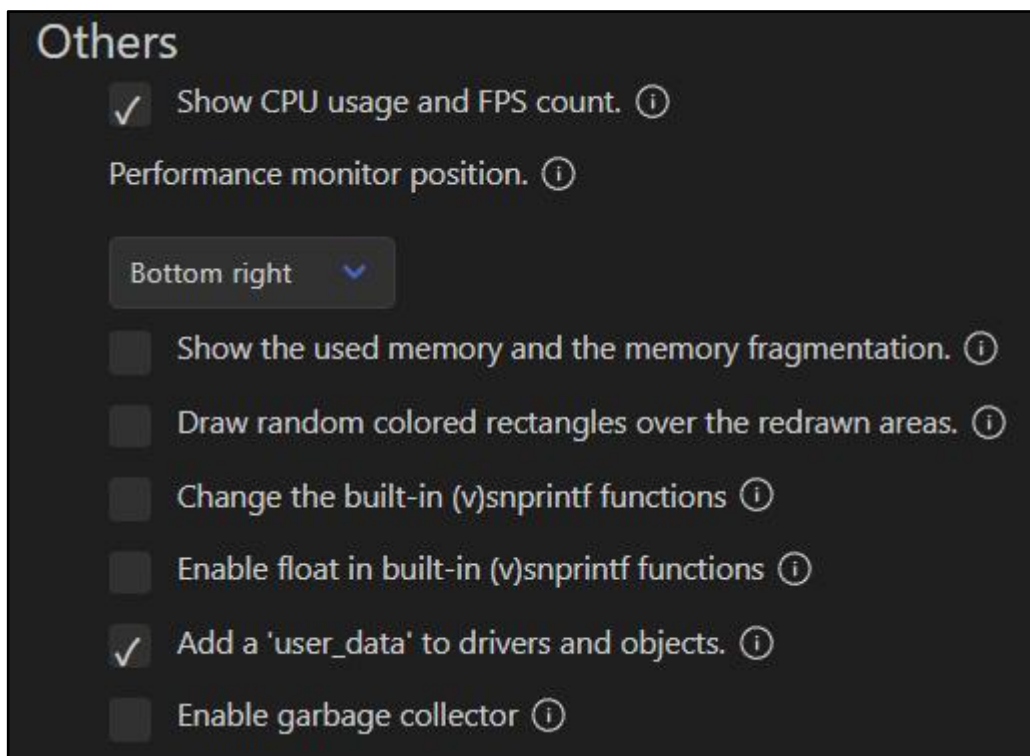


图3.5 其他配置

- 字体设置

将8号到48号字体选项全部勾选，这样应用程序里就会包含这些字体，不会出现字体缺失的情况。其他配置保持默认即可。

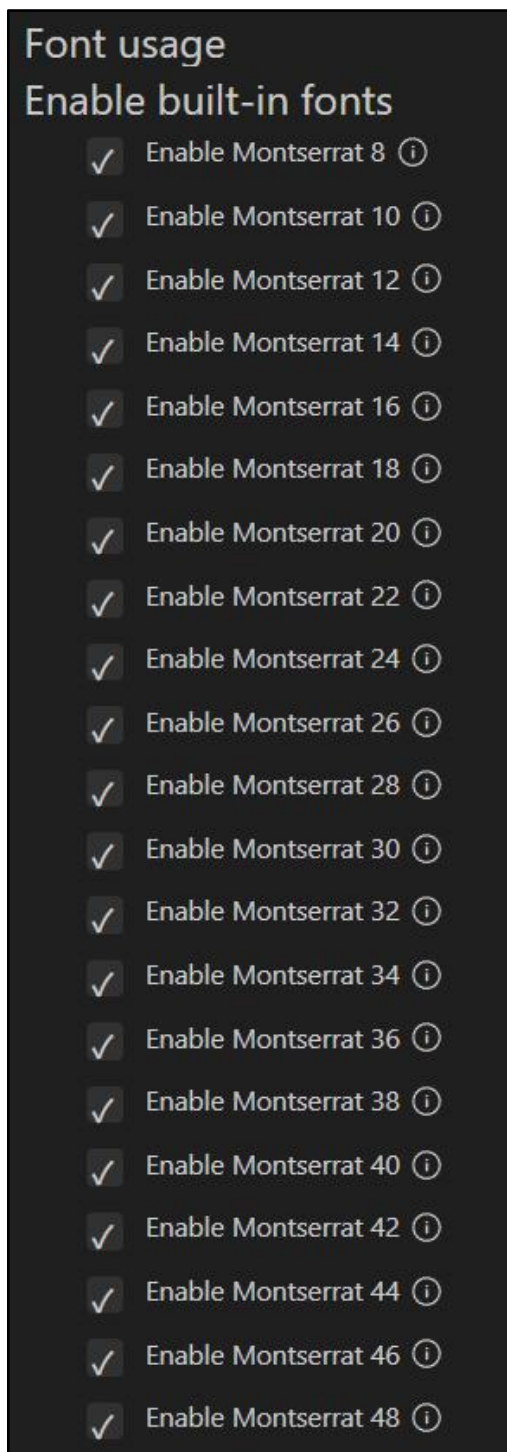


图3.6 字体配置

- 默认示例配置

将lvgl默认自带的5个示例都勾选，这样编译时就会将这些示例代码都编译进去，方便调用。但是由于受开发板Flash和内存容量限制，不能将所有的功能都勾选（容量足够大时，也可以勾选）。其他配置保持默认即可。

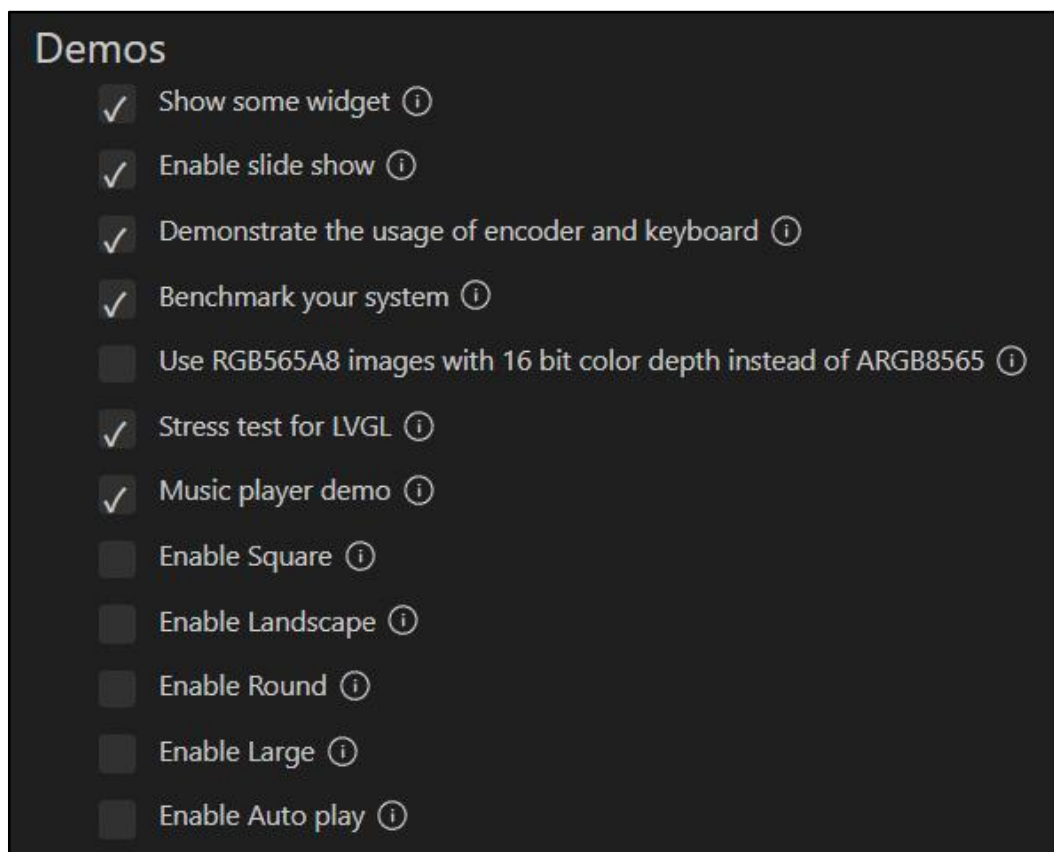


图3.7 默认示例配置

D、lvgl配置完成后，点击“保存”按钮，将配置保存。



图3.8 保存lvgl配置

3.2 通过文件配置lvgl

通过文件配置lvgl，需要将项目工程目录下“components\lvgl”文件夹里

“lv_conf_template.h”拷贝到当前文件夹下并重名为“lv_conf.h”，如下图所示：

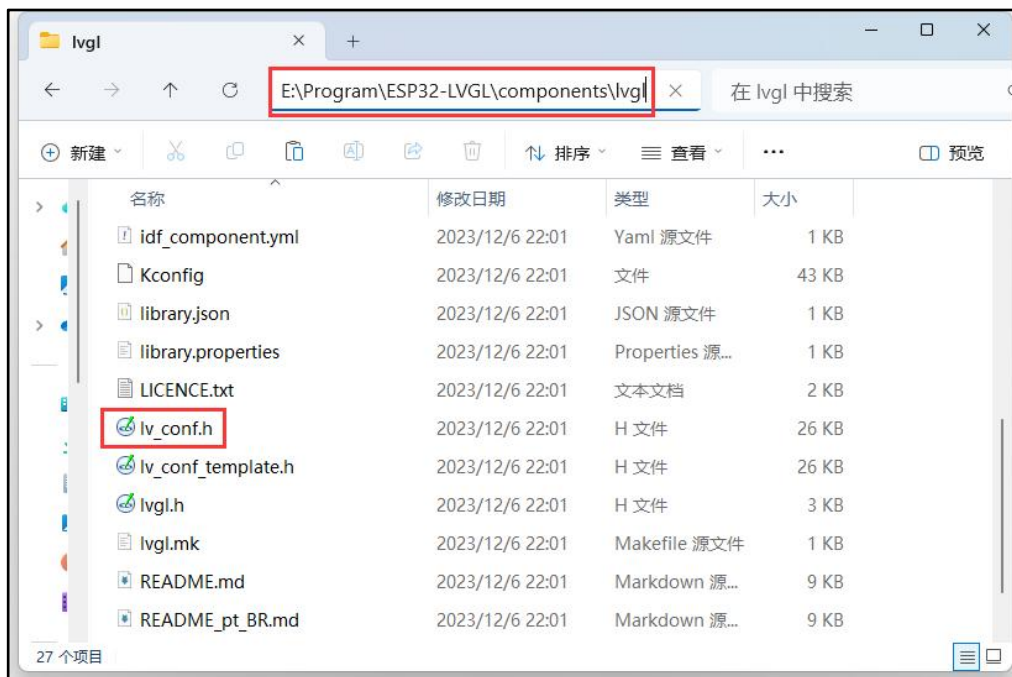


图3.9 拷贝并重命名lvgl配置文件

接下来打开`lv_conf.h`文件，根据实际情况配置。

4. 配置LCD

接下来配置LCD，步骤如下：

A、打开SDK配置编辑器界面（方法参照LVGL配置说明），搜索栏里输入“**display**”，就可以找到display的配置选项了，如下图所示：

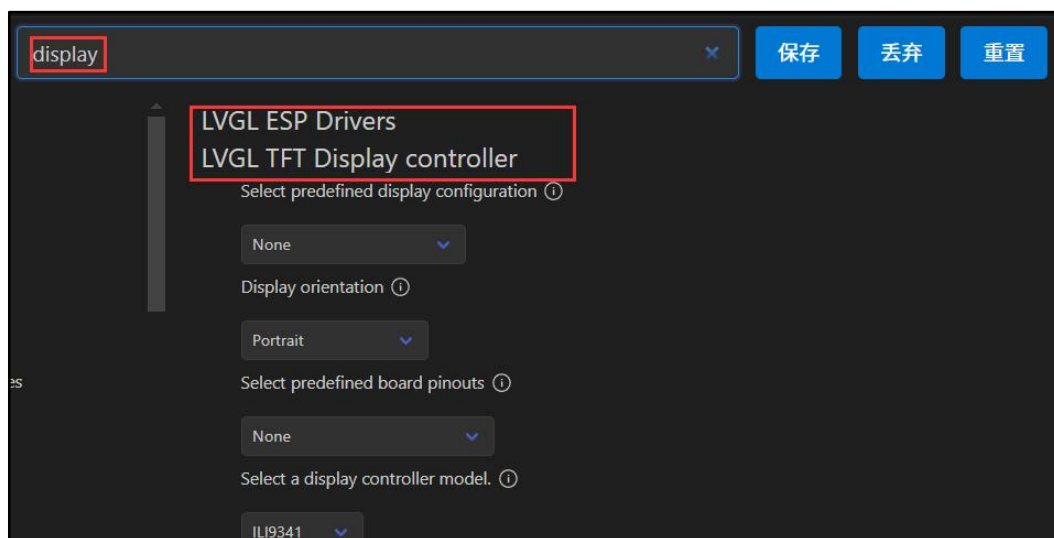


图4.1 配置LCD

B、接下来对LCD进行配置，配置介绍如下：

- LCD显示控制器配置

Select predefined display configuration: 选择已经定义好的显示配置。如果选择了某个配置，则所有的LCD配置都被预定义了。一般情况下，使用的LCD是没有配置的，所以选择“None”。

Display orientation: 显示方向设置，可选横屏和竖屏，根据实际情况选择。

Select predefined board pinouts: 选择预定义的开发板引脚配置，无则选“None”。

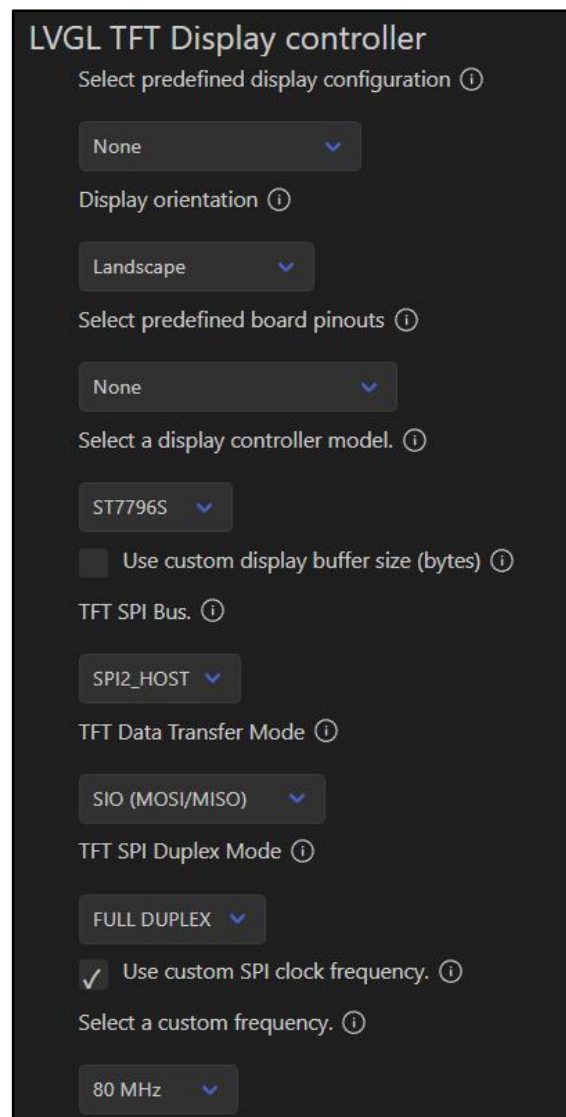
Select a display controller model: 选择一个所使用LCD的控制器型号，这里包含了大部分常用的LCD控制器型号，根据实际所用的去选择。如没有，则需移植。

TFT SPI Bus: 选择ESP32的SPI总线编号，ESP32有两组可用的SPI，根据引脚选择。

TFT Data Transfer Mode: SPI传输模式选择，可选1线、2线、4线。一般选1线。

TFT SPI Duplex Mode: SPI通信模式，可选半双工和全双工。一般选全双工。

Use custom SPI clock frequency: 勾选后可以自己定义SPI速率，一般定义为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 ▼

图4.2 LCD控制器配置

● LCD的ESP32引脚定义

GPIO for MOSI: SPI总线写数据引脚定义。

GPIO for MISO: SPI总线读数据引脚定义，需勾选“GPIO for MISO”才能设置。

GPIO for CLK: SPI总线时钟引脚定义

GPIO for CS: LCD使能引脚定义，需勾选“Use CS signal to control the display”才可设置。

GPIO for DC: LCD数据和命令选择引脚定义，需勾选“Use DC signal to control the display”才可设置。

Backlight Control: LCD背光控制，可选无、PWM、开关三种方式，可选高电平开。

GPIO for Backlight Control: LCD背光控制引脚定义，需选PWM或开关方式才可设。

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

图4.3 LCD引脚定义

C、LCD配置完成后，点击“保存”按钮，保存配置。

5. 配置触摸屏

接下来配置触摸屏，步骤如下：

A、打开SDK配置编辑器界面（方法参照LVGL配置说明），搜索栏里输入“touch”，就可以找到touch的配置选项了，如下图所示：

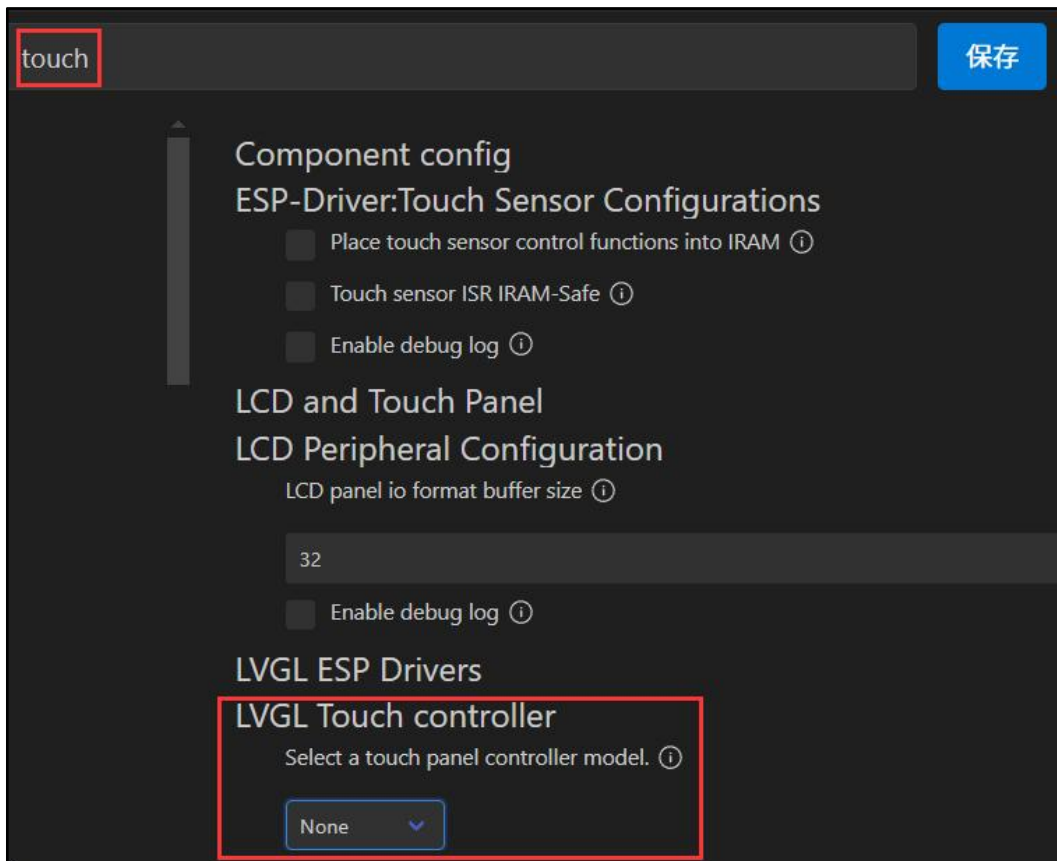


图5.1 触摸屏配置

B、接下来对触摸屏进行配置，配置介绍如下：

● 触摸屏ESP32引脚定义

Select a touch panel controller model: 选择一个触摸屏控制器型号。可选的有电阻触摸屏和电容触摸屏控制器型号。根据实际情况选择。如果没有需要的型号，则需移植。

Touch Controller SPI Bus: 选择触摸屏所使用ESP32的SPI总线编号，ESP32有两组可用的SPI，根据引脚选择。

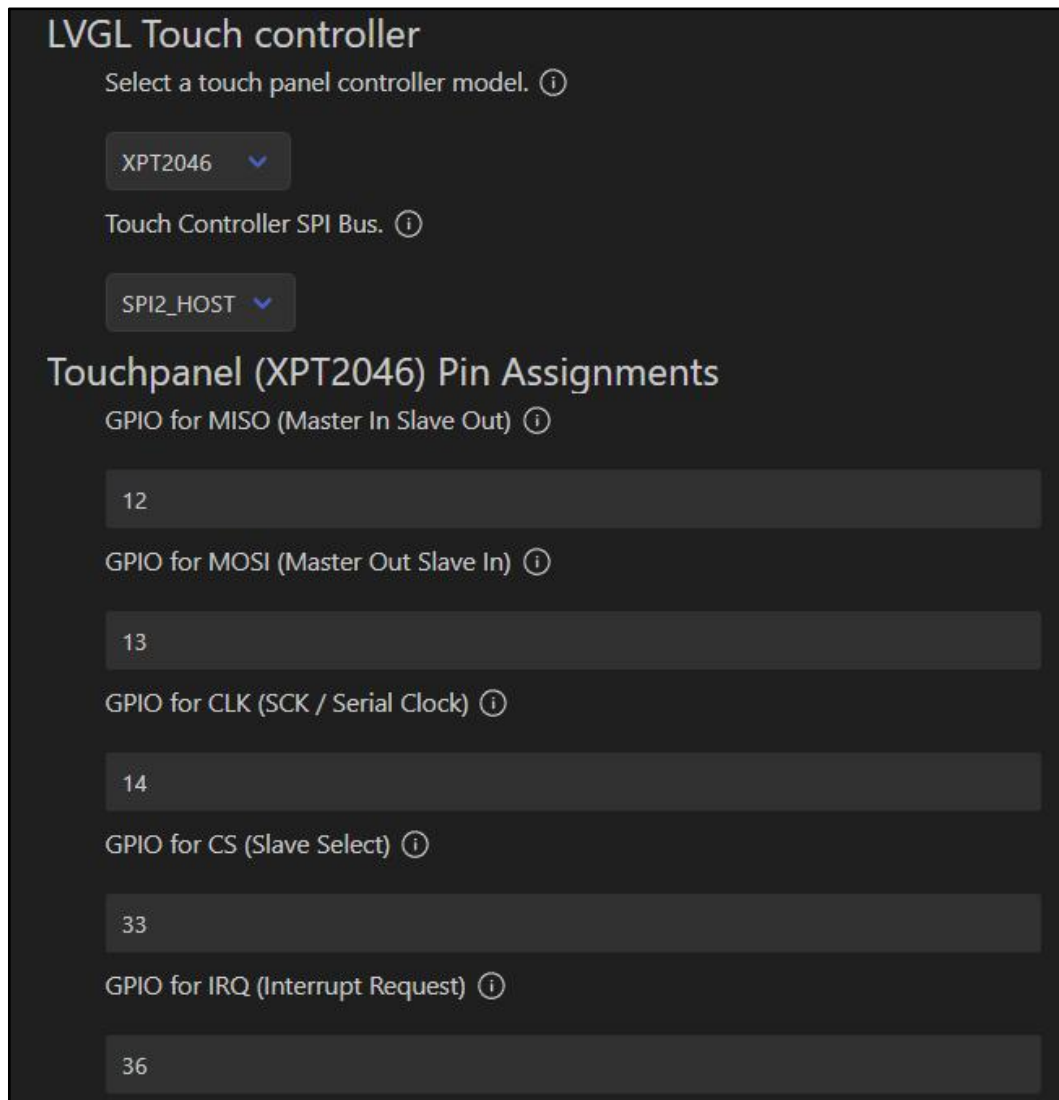
GPIO for MISO: 触摸屏所用SPI总线的读数据引脚定义

GPIO for MOSI: 触摸屏所用SPI总线的写数据引脚定义

GPIO for CLK: 触摸屏所用SPI总线时钟信号引脚定义

GPIO for CS: 触摸屏使能引脚定义

GPIO for IRQ: 触摸屏中断引脚定义



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

图5.2 触摸屏引脚定义

- 触摸屏其他参数配置

Minimum X coordinate value: x坐标最小的ADC值

Minimum Y coordinate value: y坐标最小的ADC值

Maximum X coordinate value: x坐标最大的ADC值

Maximum Y coordinate value: y坐标最大的ADC值

Swap XY: 选择是否交换x和y的坐标值。勾选则交换

Invert X coordinate value: 选择是否反向x坐标值，勾选则反向

Invert Y coordinate value: 选择是否反向y坐标值，勾选则反向

以上参数，如果不知道确切值，可以先不配置，待程序运行起来后，通过点击触摸屏，从串口输出获取相关值再设置。

Select touch detection method: 选择触摸事件检测方式。可选只通过按压检测、只通过中断引脚检测、通过按压和中断引脚一起检测。如果采取轮询的方式，可以选择只通过按压检测。

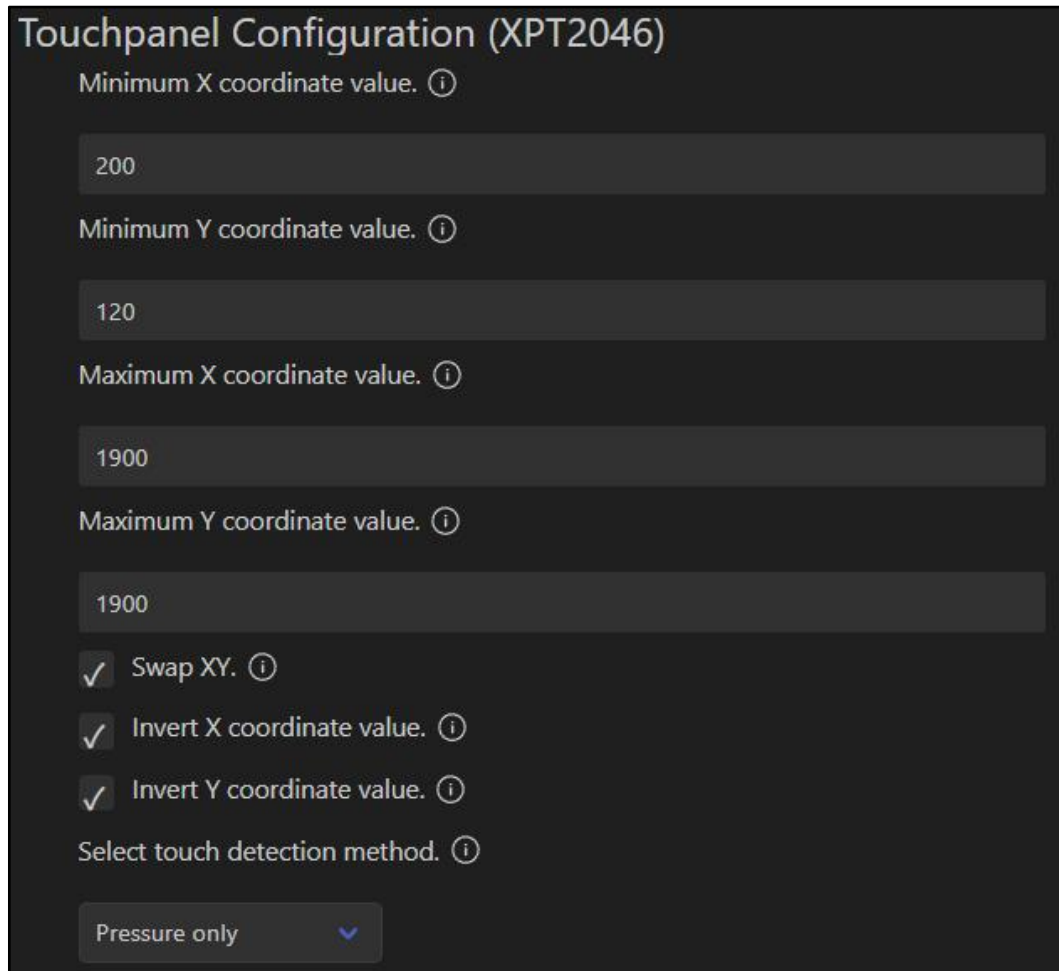


图5.3 触摸屏其他参数配置

C、触摸屏参数配置完成后，点击“保存”按钮，保存配置。

6. 修改工程文件

6.1 修改lv_porting文件夹内容

- 添加Cmake文件

打开项目工程目录下“components\lv_porting”文件夹，在该文件夹下新建

CMakeLists.txt文件，打开该文件后，输入如下内容，然后保存：

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

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

- 修改显示屏相关的文件

- A、开启文件内容并修改头文件定义

打开项目工程目录下“components\lv_porting”文件夹的“lv_port_disp.c”文件，将第7行“#if 0”改为“#if 1”，再修改和添加头文件。修改后的内容如下所示：

```
/*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、修改LCD分辨率定义

首先将两个warning注释掉，然后根据实际情况修改水平和垂直分辨率。例如这里使用ST7796驱动的LCD，如果为竖屏显示，则水平分辨率宏定义MY_DISP_HOR_RES设为320，垂直分辨率宏定义MY_DISP_VER_RES设为480；如果为横屏显示，则将两个分辨率值交换设置。修改后的内容如下所示：

```
/*****
 *      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、修改缓冲区

这里选用双缓冲区方式，将其他两种缓冲区方式设置代码注释掉。再将缓冲区容量改为LCD驱动里定义的容量（在lvgl_helpers.h头文件里定义，不同的LCD驱动对应不同的缓冲区容量），修改后的内容如下所示：

```
/* 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、修改LCD初始化函数

在lvgl的LCD初始化函数里调用LCD驱动里定义的初始化函数（在lvgl_helpers.c文件里定义，不同的LCD驱动对应不同的初始化函数），修改后的内容如下所示：

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

E、修改LCD刷屏函数

在lvgl的LCD刷屏函数里调用LCD驱动里定义的刷屏函数（在disp_driver.c文件里定义，不同的LCD驱动对应不同的刷屏函数），修改后的内容如下所示：

```
/*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
}
```

至此lv_port_disp.c文件修改完毕，保存好修改后的文件。

F、修改头文件

打开项目工程目录下“components\lv_porting”文件夹的“lv_port_disp.h”文

件，将第7行“#if 0”改为“#if 1”，然后保存。修改后的文件内容如下所示：

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

- 修改触摸屏相关的文件

- A、开启文件内容并修改头文件定义

打开项目工程目录下“components\lv_porting”文件夹的“lv_port_indev.c”

文件，将第7行“#if 0”改为“#if 1”，再修改和添加头文件。修改后内容如下所示：

```
/*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、注释其他输入设备的函数申明和变量定义

因为这里只用到了触摸屏，所以其他输入设备相关的函数申明和变量定义都建议注

释掉，否则编译时可能报错。修改后的文件内容如下所示：

```
/*****
 *      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、注释其他输入设备的初始化代码

在lv_port_indev_init函数里将其他设备的初始化代码注释，否则触摸屏不能正常工作。修改后的文件内容如下所示：

```
#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、修改触摸屏初始化函数

在lvgl的触摸屏初始化函数里调用触摸屏驱动里定义的初始化函数（在touch_driver.c文件里定义，不同的触摸屏驱动对应不同的初始化函数），修改后的内容如下所示：

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

E、修改触摸屏坐标读取函数

在lvgl的触摸屏坐标读取函数里调用触摸屏驱动里定义的坐标读取函数（在touch_driver.c文件里定义，不同的触摸屏驱动对应不同的坐标读取函数），修改后的内容如下所示：

```
/*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、注释其他输入设备的相关函数定义

因为这里只用到了触摸屏，所以其他输入设备相关的函数定义都建议注释掉，否则编译时可能报错。修改后的文件内容如下所示：

```
#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
```

至此lv_port_indev.c文件修改完毕，保存好修改后的文件。

G、修改头文件

打开项目工程目录下“components\lv_porting”文件夹的“lv_port_indev.h”

文件，将第7行“#if 0”改为“#if 1”，然后保存。修改后的文件内容如下所示：

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

6.2 修改main.c文件

打开项目工程目录下“main”文件夹的“main.c”文件，将文件内容按照如下所示修改，然后保存。

```
#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. 编译并调试工程代码

在编译和调试工程代码之前，先来熟悉一下VS Code的编译和调试工具，如下图所示：

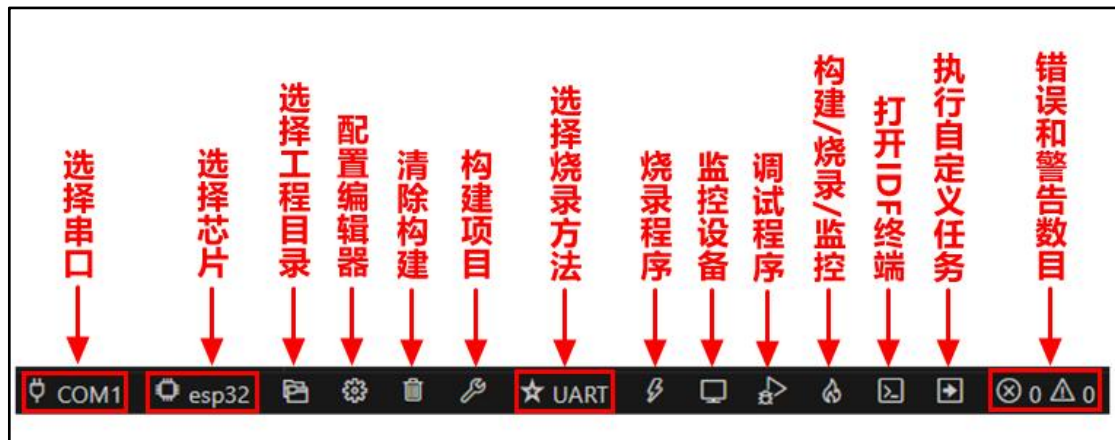


图7.1 VS Code的编译和调试工具

选择串口：选择当前所用开发板连接的串口号。

选择芯片：选择当前所用开发板使用的主控芯片。

选择工程目录：选择打开需要的工程，一般不选择，因为默认就是当前工程。

配置编辑器：打开工程菜单编辑器(menuconfig)，用来配置当前工程的一些功能。

清除构建：删除工程编译生成的文件（主要删除build文件夹）。

构建项目：编译当前工程的代码。

选择烧录方法：选择工程代码的烧录方法，可选串口、JTAG等等，一般选择串口烧录。

烧录程序：烧录当前工程编译好的代码。修改代码后，需要先编译，烧录后才有效。

监控设备：打开串口输出窗口，显示开发板串口输出信息。

调试程序：调试当前工程的代码。


构建/烧录/监控：将编译、烧录以及打开串口输出窗口的操作合在一起，点击一次实现三个功能，是最常用的操作。

打开IDF终端：打开esp-idf命令行终端窗口，可以输入idf命令并执行。

执行自定义任务：执行一些个人自定义的任务，一般用不到。

错误和警告数目：显示编译工程代码时产生的错误和警告数目。

熟悉了VS Code的编译和调试工具后，接下来编译并调试工程代码，检测代码中的错误并修正，直到编译通过。步骤如下：

A、点击VS Code软件底部的构建项目按钮，可以看到串口终端窗口输出编译信息。

B、当编译中途停止，则说明代码有错误，点击底部**错误和警告数目**按钮查看结果，如

下图所示：

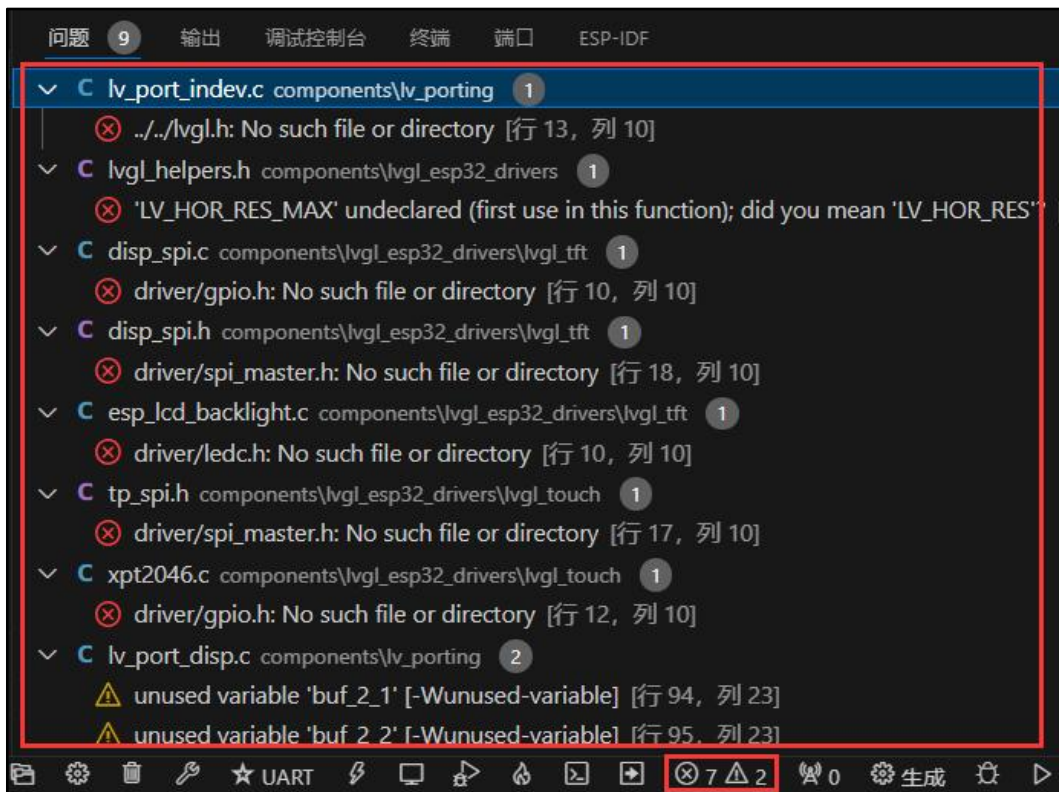


图7.2 工程代码编译错误和警告显示

C、接下来修正错误。首先查看错误原因，然后点击错误信息，进入出现错误的地方。

- 第一个错误是“../../lvgl.h”头文件找不到，在lv_port_index.c文件里修改头文件路径即可。如下图所示：

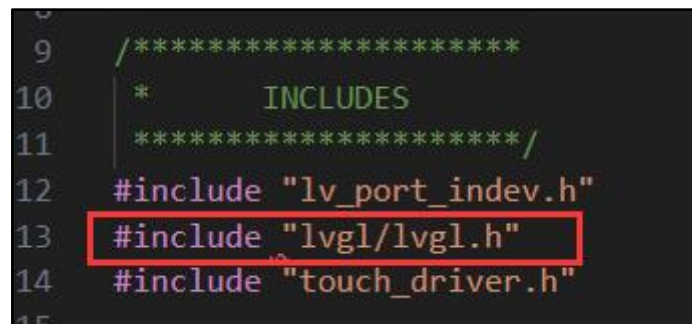


图7.3 修改文件1

- 第二个错误是“LV_HOR_RES_MAX”没定义，在lv_helpers.h里添加相关的宏定义。注意根据实际情况定义水平和垂直分辨率。例如这里使用ST7796驱动的LCD，如果为竖屏显示，则水平分辨率宏定义LV_HOR_RES_MAX设为320，垂直分辨率宏定义LV_VER_RES_MAX设为480；如果为横屏显示，则将两个分辨率值交换设置。

```

/*****
 *      DEFINES
 *****/

#ifndef LV_HOR_RES_MAX
#define LV_HOR_RES_MAX (480)
#endif
#ifndef LV_VER_RES_MAX
#define LV_VER_RES_MAX (320)
#endif

```

图7.4 修改文件2

- 第三到第七错误都是头文件找不到，这是因为lvgl_esp32_drivers里用了idf里的函数，所以得引用idf文件，在项目工程

“components\lvgl_esp32_drivers\CMakeLists.txt”文件里的89行添加“REQUIRES driver”，如下图所示，然后保存文件。

```

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

```

图7.5 修改文件3

错误修改完毕后，继续编译。编译还是报错，如下图示：

```

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

```

图7.6 编译报错1

- 通过查看错误信息，发现这些错误都是申明缺失或者定义缺失导致的，这是因为ESP-IDF_5.0已经更新了部分API定义，而lvgl_esp32_drivers依然再用老的API，

所以导致API定义找不到。错误修改如下：

- 打开“lvgl_esp32_drivers\lvgl_tft\esp_lcd_backlight.c”，根据行号在相应的地方参照以下三处内容修改, 然后保存。

```
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"
```

图7.7 修改文件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
```

图7.8 修改文件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 }
```

图7.9 修改文件6

- 打开“components\lvgl_esp32_drivers\lvgl_tft\st7796s.c”（这里以st7796s.c为例，如果使用其他驱动，则要到相应的文件下去修改），根据行号在相应的地方参照以下内容修改, 然后保存。


```

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 }

```

图7.10 修改文件7

D、修改完毕后，继续编译，当出现“Memory Type Usage Summary”表格，且右下角出现“Build Successfully”提示时，说明编译成功。

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

图7.11 编译成功

8. 烧录并运行

编译成功后，就可以烧录程序，验证程序是否可以正常运行。步骤如下：

- A、将显示模块按照前面配置的引脚定义连接到ESP32设备（如果为ESP32为板载设备，则不需接线），再将ESP32设备连接到电脑，然后在VS code软件底部工具栏选择芯片、选择串口（需要安装USB转串口驱动）、选择烧录方式。
- B、点击VS code软件底部工具栏烧录按钮 （如果已经编译完成）或者点击构建/烧录/监控按钮 （还没编译或者构建已经删除），此时就可以烧录了。
- C、烧录成功后，如果配置都是正确的，就可以看到显示屏出现LVGL demo的界面，如下图所示：

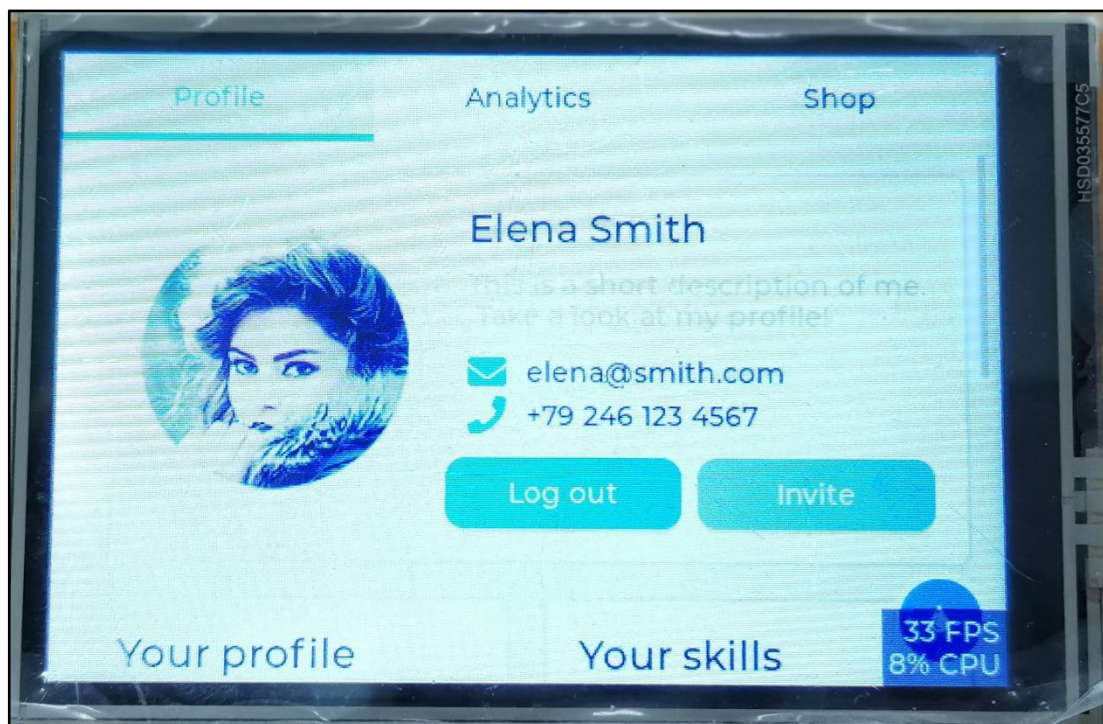


图8.1 程序运行正常

- D、如果显示模块带有触摸屏，则可能会出现触摸异常的情况。例如电容触摸屏出现无触摸且运行报错的问题，电阻触摸屏因为没设置校准参数而出现位置不准确、X和Y方向交换、X方向坐标值反了、Y方向坐标值反了等问题。一般情况下需要通过修改代码和配置参数来解决。
- E、这里讲解一下怎样设置电阻触摸屏校准参数，步骤如下：

● 校准触摸方向

首先在触摸屏上进行水平和垂直滑动，如果显示屏内容也会水平和垂直滚动，则说

明触摸屏水平方向和垂直方向没有交换，否则需要交换触摸屏水平方向和垂直方向，方法如下：

打开SDK配置编辑器界面（方法参照LVGL配置说明），搜索栏里输入“touch”，然后找到“Swap XY”选项。如果该选项之前被勾选，则将勾选去掉；如果没有勾选，则勾选上，然后保存设置。如下图所示：

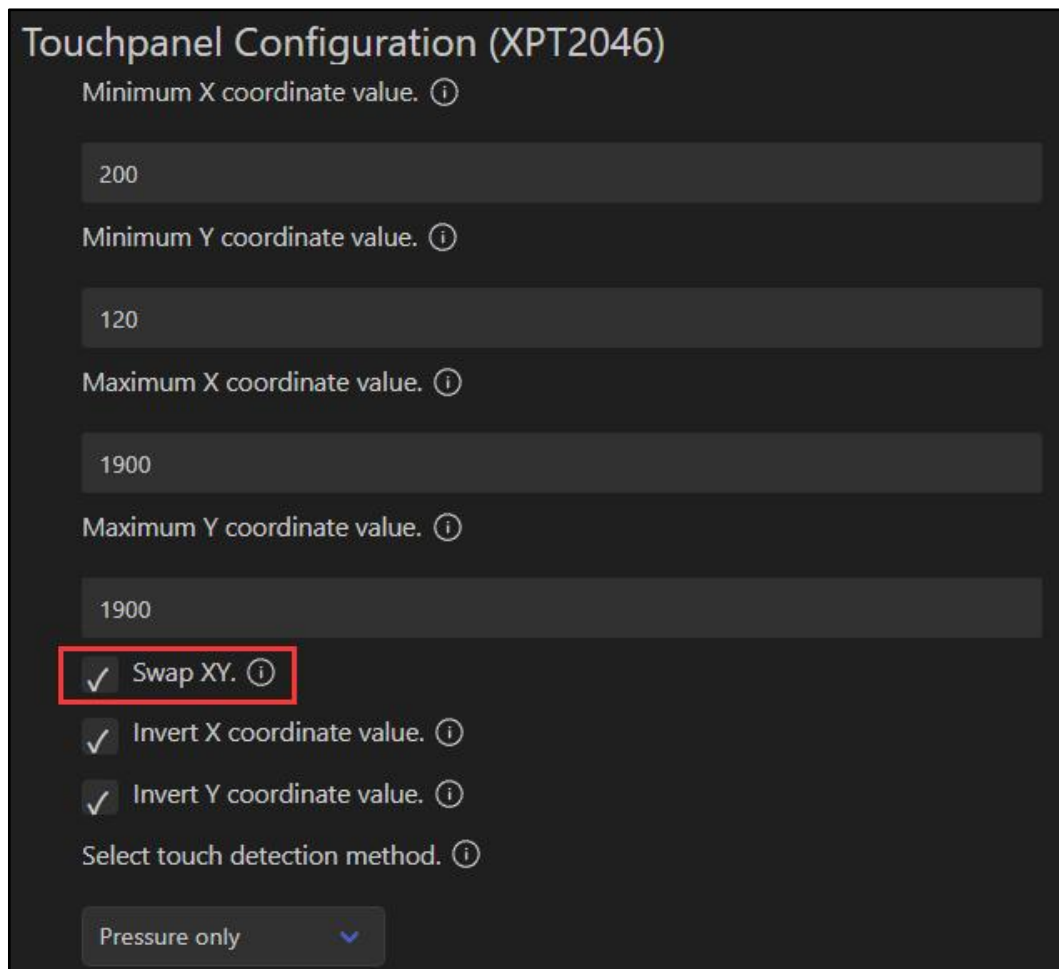


图8.2 交换X和Y方向

设置保存好后，重新编译并烧录程序。程序运行后，再在触摸屏上进行水平和垂直滑动。如果显示屏内容水平和垂直滚动的方向和触摸屏滑动方向一致，则说明触摸屏水平和垂直坐标和显示屏一致，否则需要将水平或者垂直坐标反向，修改方法如下：

安装上述方法在SDK配置编辑器界面找到“**Invert X coordinate value**”和“**Invert Y coordinate value**”选项，如果选项之前被勾选，则将勾选去掉；如果没有勾选，则勾选上，然后保存设置。如下图所示：

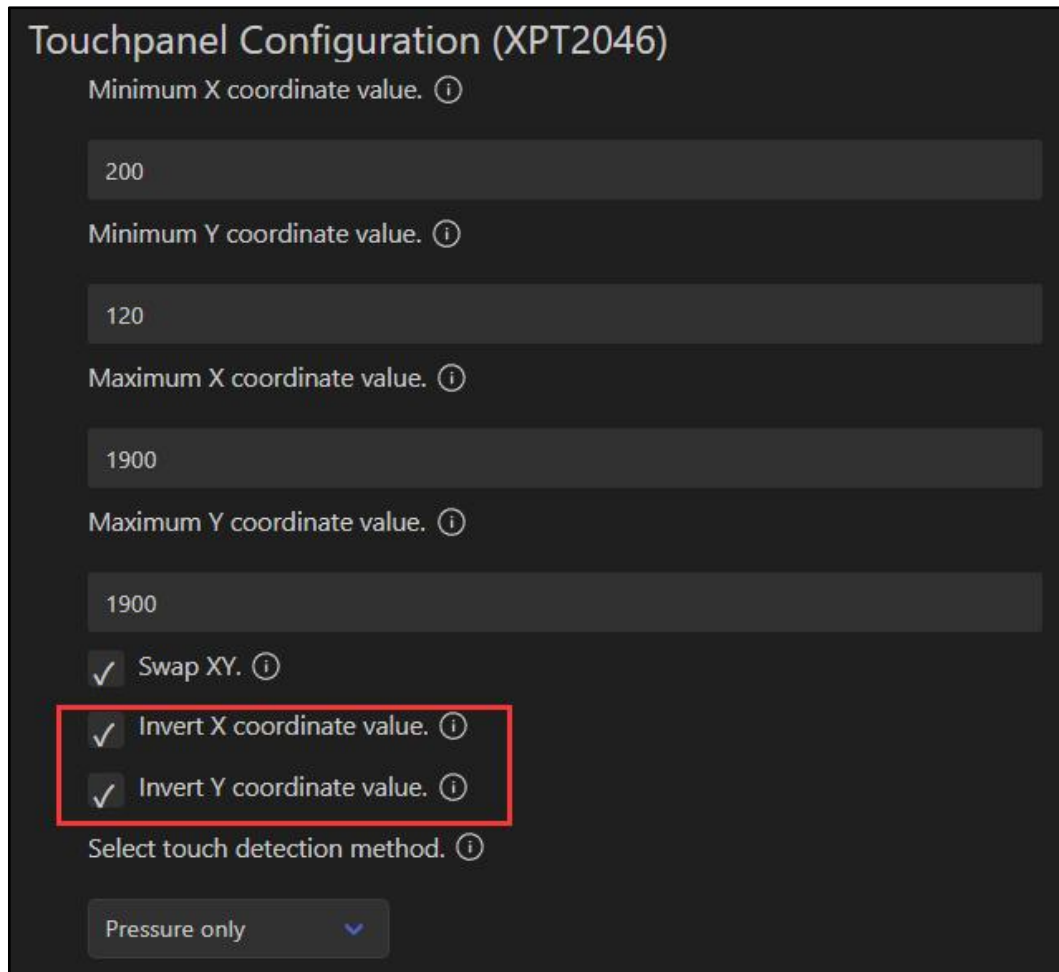



图8.3 设置X或Y方向坐标

● 设置校准参数

触摸方向校准成功后，接下来需要设置校准参数，确保触摸准确。

首先打开SDK配置编辑器界面，点击“**Bootloader manager**”，然后将“**Bootloader log verbosity**”设置为“**Info**”，如下图所示，这样就可以通过串口输出代码的log

信息（如默认已经设置，则忽略该步骤）。最后点击  按钮重新编译并烧录程序。

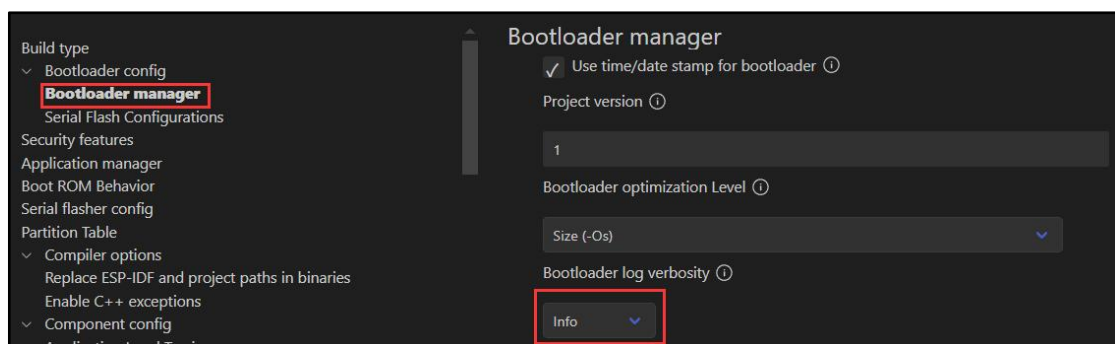
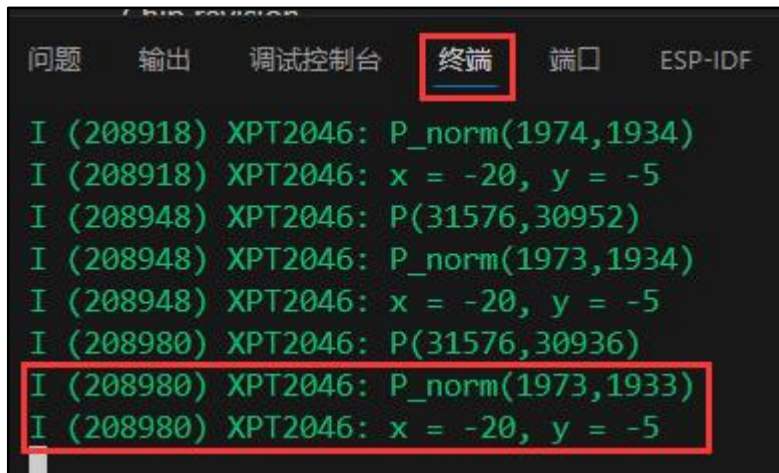


图8.4 设置log输出等级

程序烧录完成后，运行程序。以显示方向作为参考，点击触摸屏的左上角，此时通过串口终端可以获取一组触摸屏ADC值和显示屏坐标值，如下图所示：



```
I (208918) XPT2046: P_norm(1974,1934)
I (208918) XPT2046: x = -20, y = -5
I (208948) XPT2046: P(31576,30952)
I (208948) XPT2046: P_norm(1973,1934)
I (208948) XPT2046: x = -20, y = -5
I (208980) XPT2046: P(31576,30936)
I (208980) XPT2046: P_norm(1973,1933)
I (208980) XPT2046: x = -20, y = -5
```

图8.5 获取校准值1

然后再点击右下角获取另一组触摸屏ADC值和显示屏坐标值，如下图所示：



```
I (337967) XPT2046: P_norm(112,92)
I (337967) XPT2046: x = 480, y = 320
I (338000) XPT2046: P(1792,1480)
I (338000) XPT2046: P_norm(112,92)
I (338000) XPT2046: x = 480, y = 320
I (338022) XPT2046: P(1776,1536)
I (338022) XPT2046: P_norm(111,96)
I (338022) XPT2046: x = 480, y = 320
```

图8.6 获取校准值2

通过对比这两组校准值，可以获取触摸屏X方向最小和最大的ADC值，以及Y方向最小和最大的ADC值。例如这里X方向最小ADC值为111，最大ADC值为1973；Y方向最小ADC值为96，最大ADC值为1933。

接下来打开SDK配置编辑器界面，在搜索栏里输入“touch”，找到相应的位置，将此四个值填入，然后点击“保存”按钮保存配置。如下图所示：

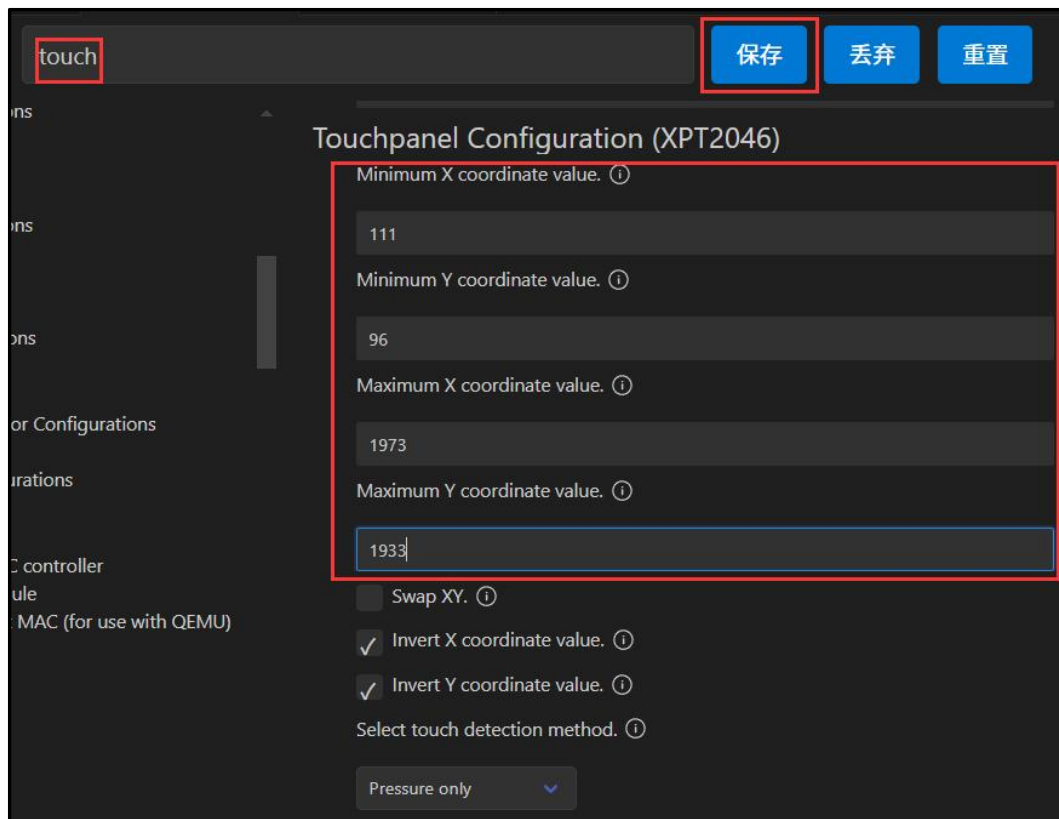


图8.7 设置校准值

最后重新编译并烧录。

需要注意，改变显示方向后，触摸参数需要重新配置。