ESP32-C6-LCD-1.9

From Waveshare Wiki
Jump to: navigation, search

Overview

[Expand]

ESP32-C6-LCD-1.9
Without touch version

(https://www.waveshare.com/esp32-c6-lcd-1.9.htm?sku=30937)

ESP32-C6, UART/I2C/USB

ESP32-C6-Touch-LCD-1.9
With touch version

(https://www.waveshare.com/esp32-c6-lcd-1.9.htm?sku=30941)

ESP32-C6, UART/I2C/USB

Usage Instructions

Currently there are two development tools and frameworks, **Arduino IDE** and **ESP-IDF**, providing flexible development options, you can choose the right development tool according to your project needs and personal habits.

Development Tools

Arduino IDE

(/wiki/File:180px-Arduino-IDE-logo.jpg)

Arduino IDE is an open source electronic prototyping platform, convenient and flexible, easy to get started. After a simple learning, you can start to develop quickly. At the same time, Arduino has a large global user community, providing an abundance of open source code, project examples and tutorials, as well as rich library resources, encapsulating complex functions, allowing developers to quickly implement various functions.

ESP-IDF

(/wiki/File:180px-ESP-IDF-logo.jpg)

ESP-IDF, or full name Espressif IDE, is a professional development framework introduced by Espressif Technology for the ESP series chips. It is developed using the C language, including a compiler, debugger, and flashing tool, etc., and can be developed via the command lines or through an integrated development environment (such as Visual Studio Code with the Espressif IDF plugin). The plugin offers features such as code navigation, project management, and debugging, etc.

Each of these two development approaches has its own advantages, and developers can choose according to their needs and skill levels. Arduino are suitable for beginners and non-professionals because they are easy to learn and quick to get started. ESP-IDF is a better choice for developers with a professional background or high performance requirements, as it provides more advanced development tools and greater control capabilities for the development of complex projects.

Components Preparation

- ESP32-C6-LCD-1.9 x1
- TF card x 1 (Optional)

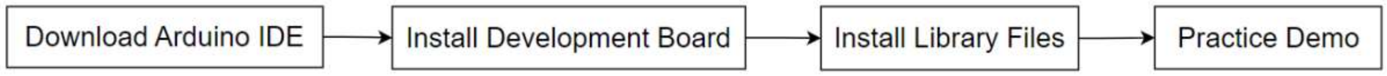
- USB cable (Type A male to Type C male) x 1

(/wiki/File:500px-Esp32_c6_lcd_1.9_3.png)

Before operating, it is recommended to browse the table of contents to quickly understand the document structure. For smooth operation, please read the FAQ carefully to understand possible problems in advance. All resources in the document are provided with hyperlinks for easy download.

Working with Arduino

This chapter introduces setting up the Arduino environment, including the Arduino IDE, management of ESP32 boards, installation of related libraries, program compilation and downloading, as well as testing demos. It aims to help users master the development board and facilitate secondary development.

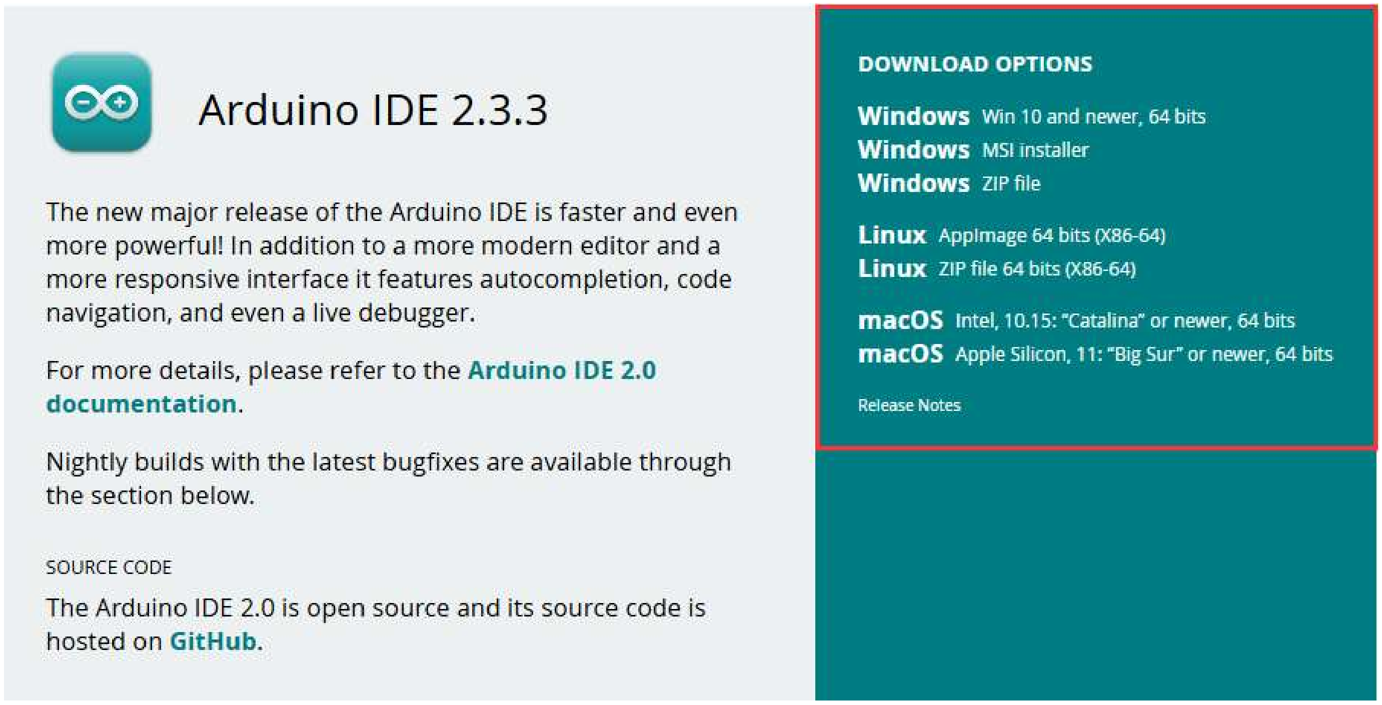


(/wiki/File:Arduino-flow-04.png)

Environment Setup

Download and Install Arduino IDE

- Click to visit the Arduino official website (https://www.arduino.cc/en/software), select the corresponding system and system bit to download



Arduino IDE 2.3.3

The new major release of the Arduino IDE is faster and even more powerful! In addition to a more modern editor and a more responsive interface it features autocompletion, code navigation, and even a live debugger.

For more details, please refer to the **Arduino IDE 2.0 documentation**.

Nightly builds with the latest bugfixes are available through the section below.

SOURCE CODE

The Arduino IDE 2.0 is open source and its source code is hosted on **GitHub**.

DOWNLOAD OPTIONS

Windows Win 10 and newer, 64 bits
Windows MSI installer
Windows ZIP file

Linux AppImage 64 bits (X86-64)
Linux ZIP file 64 bits (X86-64)

macOS Intel, 10.15: "Catalina" or newer, 64 bits
macOS Apple Silicon, 11: "Big Sur" or newer, 64 bits

Release Notes

(/wiki/File:ESP32-S3-AMOLED-1.91-Ar-software-01.png)

- Run the installer and install all by default

The environment setup is carried out on the Windows 10 system, Linux and Mac users can access Arduino-esp32 environment setup (https://docs.espressif.com/projects/arduino-esp32/en/latest/installing.html) for reference

Install ESP32 Development Board

- Before using ESP32-related motherboards with the Arduino IDE, you must first install the software package for the **esp32 by Espressif Systems** development board
- According to **board installation requirement**, it is generally recommended to use **Install Online**. If online installation fails, use **Install Offline**.
- For the installation tutorial, please refer to Arduino board manager tutorial (https://www.waveshare.com/wiki/Arduino_Board_Managers_Tutorial)

- ESP32-C6-LCD-1.9 Development board installation instructions

Board name	Board installation requirement	Version number requirement
ESP32-C6-LCD-1.9/ESP32C6 Dev Module	"Install Offline" / "Install Online"	≥3.0.7

Install Library

- When installing Arduino libraries, there are usually two ways to choose from: **Install online** and **Install offline**. **If the library installation requires offline installation, you must use**

the provided library file
For most libraries, users can easily search and install them through the online library manager of the Arduino software. However, some open-source libraries or custom libraries are not synchronized to the Arduino Library Manager, so they cannot be acquired through online searches. In this case, users can only manually install these libraries offline.

- For library installation tutorial, please refer to Arduino library manager tutorial (https://www.waveshare.com/wiki/Arduino_Library_Manager_Tutorial)
- **ESP32-C6-LCD-1.9 library file is stored in Demo at the following path:**

```
..\ESP32-C6-LCD-1.9-Demo\Arduino\libraries
```

- ESP32-C6-LCD-1.9-Demo library file installation description

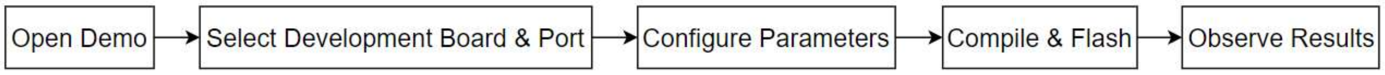
Library Name	Description	Version	Library Installation Requirement
LVGL	Graphical library	v8.4.0	"Install Offline"
Arduino_GFX_Library	Graphics library	v1.5.6	"Install Online" or "Install Offline"
Freenove_WS2812_Lib_for_ESP32	WS2812 driver library	v2.0.0	"Install Online" or "Install Offline"

Run the First Arduino Demo

If you are just getting started with ESP32 and Arduino, and you don't know how to create, compile, flash, and run Arduino ESP32 programs, then please expand and take a look. Hope it can help you!

[Expand]

Demo



(/wiki/File:Demo-flow-01.png)

- ESP32-C6-LCD-1.9 demo

Demo	Basic Description	Dependency Library
01_ADC_Test	Read the current voltage value of the system	-
02_I2C_QMI8658	Print the original data sent by the IMU	-
03_SD_Card	Load and display the information of the TF card	-
04_WS2812_Test	Drive WS2812RGB LED beads	Freenove_WS2812_Lib_for_ESP32
05_WIFI_AP	Set to AP mode to obtain the IP address of the access device	-
06_WIFI_STA	Set to STA mode to connect to WiFi and obtain an IP address	-
07_Hello_World_GFX	GFX basic demo	Arduino_GFX_Library
08_LVGL_Test	LVGL demo	LVGL

01_ADC_Test

[Collapse]

Demo description

- The analog voltage connected through the GPIO is converted to digital by the ADC, and then the actual system voltage is calculated and printed to the terminal.

Hardware connection

- Connect the board to the computer using a USB cable

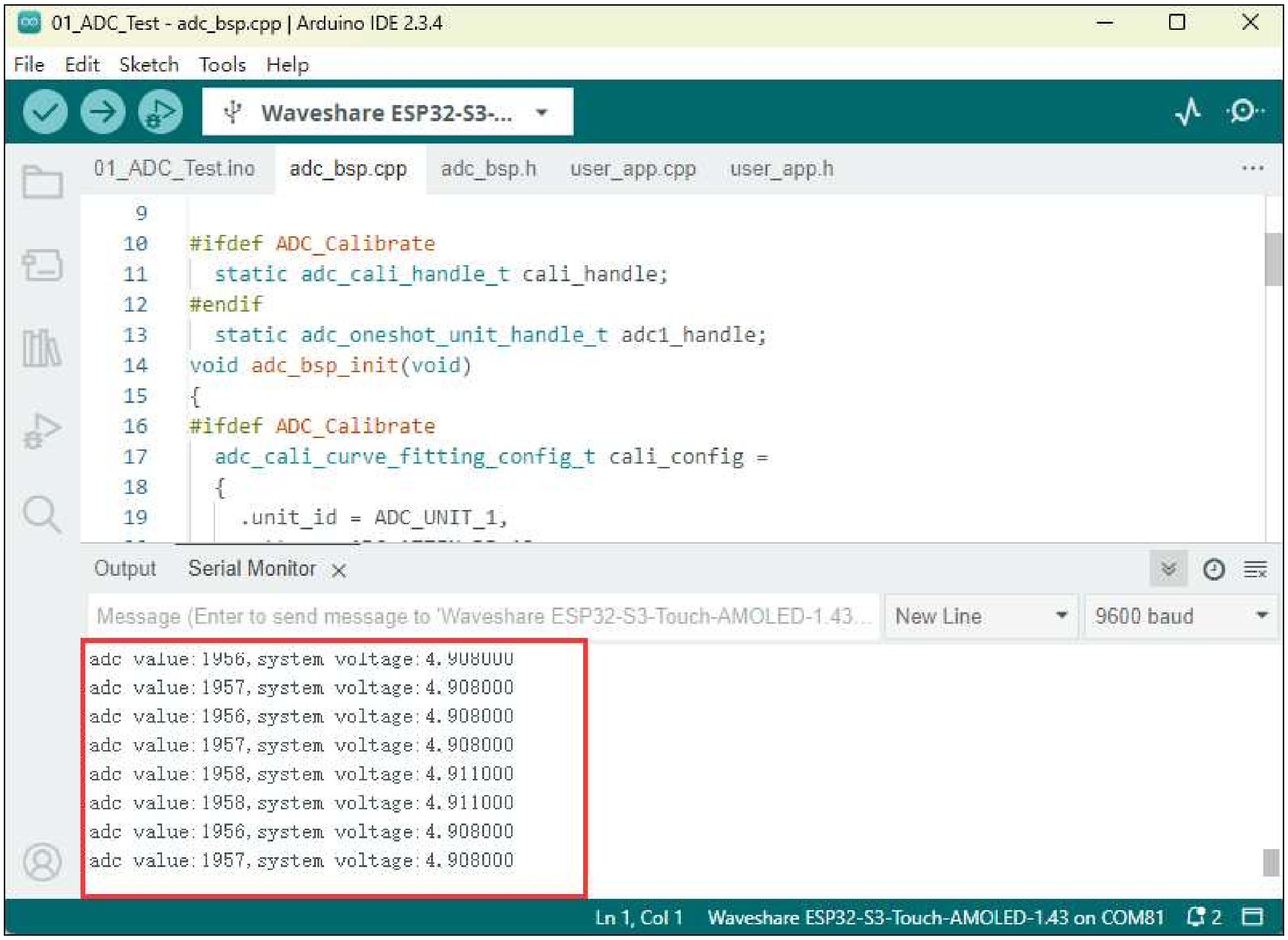
(/wiki/File:300px-Esp32_c6_lcd_1.9_5.png)

Code analysis

- **adc_bsp_init(void)** : Initializes ADC1, including creating an ADC one-time trigger unit and configuring channel 0 for ADC1.
- **adc_get_value(float *value,int *data)** : Reads the value of ADC1 channel 0 and calculates the corresponding voltage value based on the reference voltage and resolution, stores it at the position where the incoming pointer points to, and stores 0 if the read fails.
- **adc_example(void* parameter)**: Initialize ADC1 and then create an ADC task that reads the ADC value every 1 second and calculates the system's voltage based on the raw ADC value.

Result demonstration

- The program compilation download is complete, and opening the serial port monitor will show the printed ADC values and voltage, as shown in the following figure:


01_ADC_Test - adc_bsp.cpp | Arduino IDE 2.3.4
File Edit Sketch Tools Help
Waveshare ESP32-S3-...
01_ADC_Test.ino adc_bsp.cpp adc_bsp.h user_app.cpp user_app.h
Output Serial Monitor
Message (Enter to send message to 'Waveshare ESP32-S3-Touch-AMOLED-1.43...
New Line
9600 baud
Ln 1, Col 1 Waveshare ESP32-S3-Touch-AMOLED-1.43 on COM81

```

#ifdef ADC_Calibrate
  static adc_cali_handle_t cali_handle;
#endif
  static adc_oneshot_unit_handle_t adc1_handle;
void adc_bsp_init(void)
{
#ifdef ADC_Calibrate
  adc_cali_curve_fitting_config_t cali_config =
  {
    .unit_id = ADC_UNIT_1,
```

```
adc value:1956,system voltage:4.908000
adc value:1957,system voltage:4.908000
adc value:1956,system voltage:4.908000
adc value:1957,system voltage:4.908000
adc value:1958,system voltage:4.911000
adc value:1958,system voltage:4.911000
adc value:1956,system voltage:4.908000
adc value:1957,system voltage:4.908000
```

(/wiki/File:ESP32-S3-Touch-AMOLED-1.43-demo-03.png)

- The ADC sampling value is about 1900, and the system voltage is about 4.9V. For a detailed analysis, you can refer to the Schematic Diagram.

02_I2C_QMI8658

[Collapse]

Demo description

- Through I2C protocol, initialize the QMI8658 chip, then read and print the corresponding attitude information every 1 second to the terminal.

Hardware connection

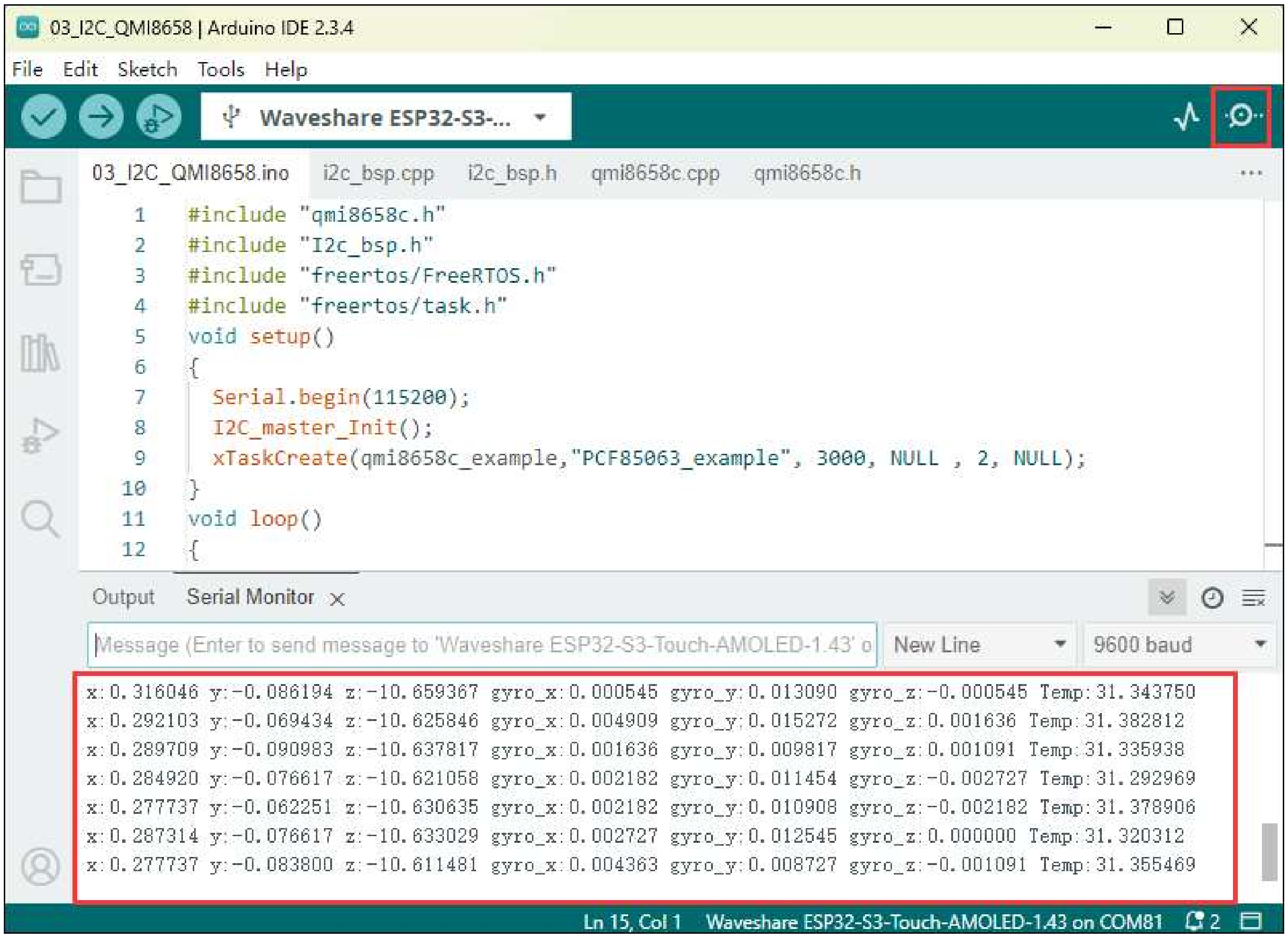
- Connect the board to the computer using a USB cable (refer to demo 01)

Code analysis

- **qmi8658c_example(void* parameter)**: The function initializes the QMI8658 device, reading and printing accelerometer data, gyroscope data, and temperature data in an infinite loop, once every second. During the rotation of the board, the gyroscope data increases with greater rotation speed, and the accelerometer calculates the corresponding acceleration based on the current position.

Result demonstration

- Open the serial port monitoring, and you can see the original data output from the IMU (Euler angles need to be converted by yourself), as shown in the following figure:


03_I2C_QMI8658 | Arduino IDE 2.3.4
File Edit Sketch Tools Help
Waveshare ESP32-S3-...
03_I2C_QMI8658.ino i2c_bsp.cpp i2c_bsp.h qmi8658c.cpp qmi8658c.h
Output Serial Monitor
Message (Enter to send message to 'Waveshare ESP32-S3-Touch-AMOLED-1.43' o
New Line
9600 baud
Ln 15, Col 1 Waveshare ESP32-S3-Touch-AMOLED-1.43 on COM81 2

```
#include "qmi8658c.h"
#include "I2c_bsp.h"
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
void setup()
{
  Serial.begin(115200);
  I2C_master_Init();
  xTaskCreate(qmi8658c_example,"PCF85063_example", 3000, NULL , 2, NULL);
}
void loop()
{
```

```
x:0.316046 y:-0.086194 z:-10.659367 gyro_x:0.000545 gyro_y:0.013090 gyro_z:-0.000545 Temp:31.343750
x:0.292103 y:-0.069434 z:-10.625846 gyro_x:0.004909 gyro_y:0.015272 gyro_z:0.001636 Temp:31.382812
x:0.289709 y:-0.090983 z:-10.637817 gyro_x:0.001636 gyro_y:0.009817 gyro_z:0.001091 Temp:31.335938
x:0.284920 y:-0.076617 z:-10.621058 gyro_x:0.002182 gyro_y:0.011454 gyro_z:-0.002727 Temp:31.292969
x:0.277737 y:-0.062251 z:-10.630635 gyro_x:0.002182 gyro_y:0.010908 gyro_z:-0.002182 Temp:31.378906
x:0.287314 y:-0.076617 z:-10.633029 gyro_x:0.002727 gyro_y:0.012545 gyro_z:0.000000 Temp:31.320312
x:0.277737 y:-0.083800 z:-10.611481 gyro_x:0.004363 gyro_y:0.008727 gyro_z:-0.001091 Temp:31.355469
```

(/wiki/File:ESP32-S3-Touch-AMOLED-1.43-demo-05.png)

- Data is output once every second. If you need to modify or refer to it, you can directly access the qmi source file for operations.

03_SD_Card

[Collapse]

Demo description

- By using the SPI bus to drive the TF card, after the TF card is successfully mounted, the TF card information is printed to the terminal, and the data is read and written to the **`writeTest.txt`** file every 1 second.

Hardware connection

- Connect the fat32 format TF card to the board and use a USB cable to connect the board to the computer (refer to demo 01)

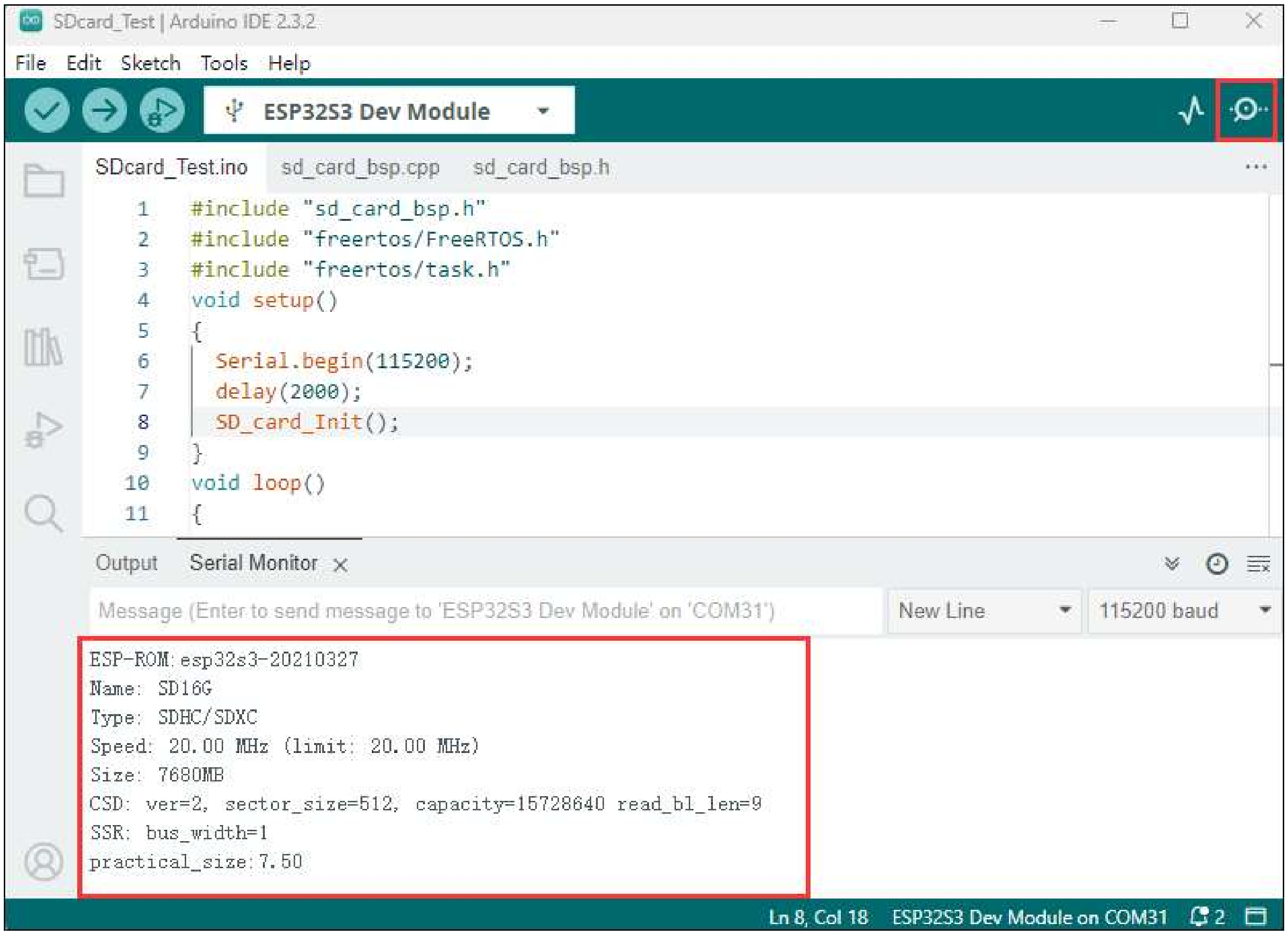
Code analysis

- In the **`03_SD_Card.ino`** file, find the **`#define sdcard_write_Test`** and cancel the macro definition comment, you can test the TF card read and write function.

```
//#define sdcard_write_Test
```

Result demonstration

- Click on the serial port monitoring device, you can see the information of the output TF card, practical_size is the actual capacity of the TF card, as shown in the figure below:


SDcard_Test | Arduino IDE 2.3.2
File Edit Sketch Tools Help
ESP32S3 Dev Module
SDcard_Test.ino
sd_card_bsp.cpp
sd_card_bsp.h
Output
Serial Monitor
Message (Enter to send message to 'ESP32S3 Dev Module' on 'COM31')
New Line
115200 baud
Ln 8, Col 18 ESP32S3 Dev Module on COM31 2

```
#include "sd_card_bsp.h"
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
void setup()
{
  Serial.begin(115200);
  delay(2000);
  SD_card_Init();
}
void loop()
{
```

```
ESP-ROM:esp32s3-20210327
Name: SD16G
Type: SDHC/SDXC
Speed: 20.00 MHz (limit: 20.00 MHz)
Size: 7680MB
CSD: ver=2, sector_size=512, capacity=15728640 read_bl_len=9
SSR: bus_width=1
practical_size:7.50
```

(/wiki/File:ESP32-S3-AMOLED-1.91-Ar-demo-07.png)

04_WS2812_Test

[Collapse]

Demo description

- Drive WS2812RGB LED beads to achieve a flowing rainbow effect, this demo is only applicable to ESP32-C6-LCD-1.9.

Hardware connection

- Connect the board to the computer using a USB cable (refer to demo 01)

Code analysis

- First analyze the macro definitions of the file.

```
#define LEDS_COUNT  2 // Number of WS2812-RGB LED beads
#define LEDS_PIN    3 //WS2812-GPIO control pin
#define CHANNEL     0 //WS2812 control channel
```

- Analyze the main functions.

```
strip.setLedColorData(id, color); //id: The ID of the RGB beads, color: The actual color
0XFF0000 which means red
strip.setLedColorData(id, r, g, b); //r, g, b: The actual values of the three primary co
lors
strip.show();//Push the color data to WS2812
```

Result demonstration

- The actual effect is as follows:

(/wiki/File:Esp32_s3_lcd_1.9_vid_-middle-original.gif)

05_WIFI_AP

[Collapse]

Demo description

- This demo can set the development board as a hotspot, allowing phones or other devices in STA mode to connect to the development board.

Hardware connection

- Connect the board to the computer using a USB cable (refer to demo 01)

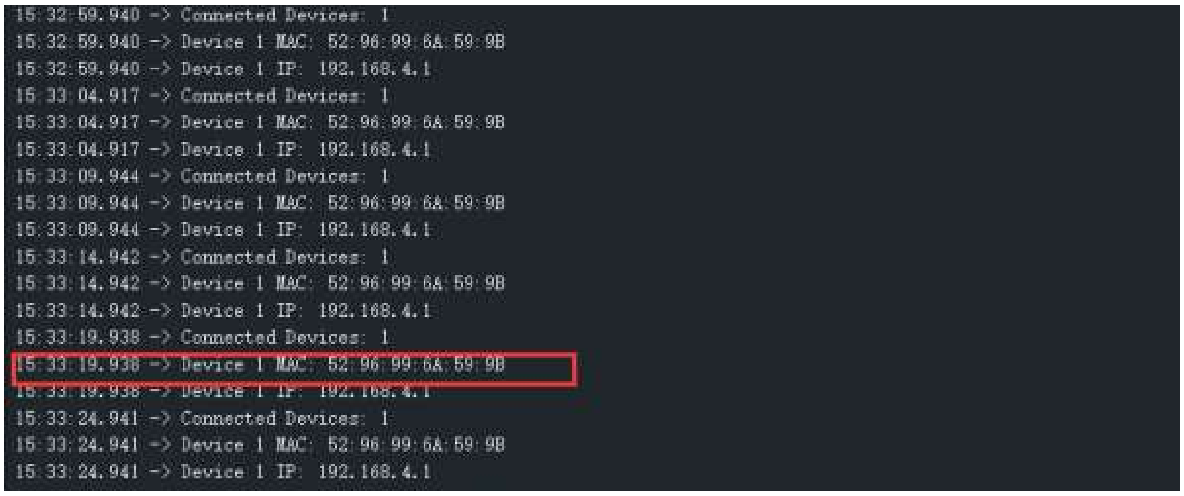
Code analysis

- In the file `05_WIFI_AP.ino`, find `ssid` and `password`, then a phone or other device in STA mode can connect to the development board using these ssid and password.

```
const char *ssid = "ESP32_AP";
const char *password = "12345678";
```

Result demonstration

After flashing the program, open the serial terminal, if the device is successfully connected to the hotspot, the MAC address of the device will be output, as shown in the figure:



```
15:32:59.940 -> Connected Devices: 1
15:32:59.940 -> Device 1 MAC: 52:96:99:6A:59:9B
15:32:59.940 -> Device 1 IP: 192.168.4.1
15:33:04.917 -> Connected Devices: 1
15:33:04.917 -> Device 1 MAC: 52:96:99:6A:59:9B
15:33:04.917 -> Device 1 IP: 192.168.4.1
15:33:09.944 -> Connected Devices: 1
15:33:09.944 -> Device 1 MAC: 52:96:99:6A:59:9B
15:33:09.944 -> Device 1 IP: 192.168.4.1
15:33:14.942 -> Connected Devices: 1
15:33:14.942 -> Device 1 MAC: 52:96:99:6A:59:9B
15:33:14.942 -> Device 1 IP: 192.168.4.1
15:33:19.938 -> Connected Devices: 1
15:33:19.938 -> Device 1 MAC: 52:96:99:6A:59:9B
15:33:19.938 -> Device 1 IP: 192.168.4.1
15:33:24.941 -> Connected Devices: 1
15:33:24.941 -> Device 1 MAC: 52:96:99:6A:59:9B
15:33:24.941 -> Device 1 IP: 192.168.4.1
```

(/wiki/File:600px-ESP32-S3-LCD-1.3-1006.png)

06_WIFI_STA

[Collapse]

Demo description

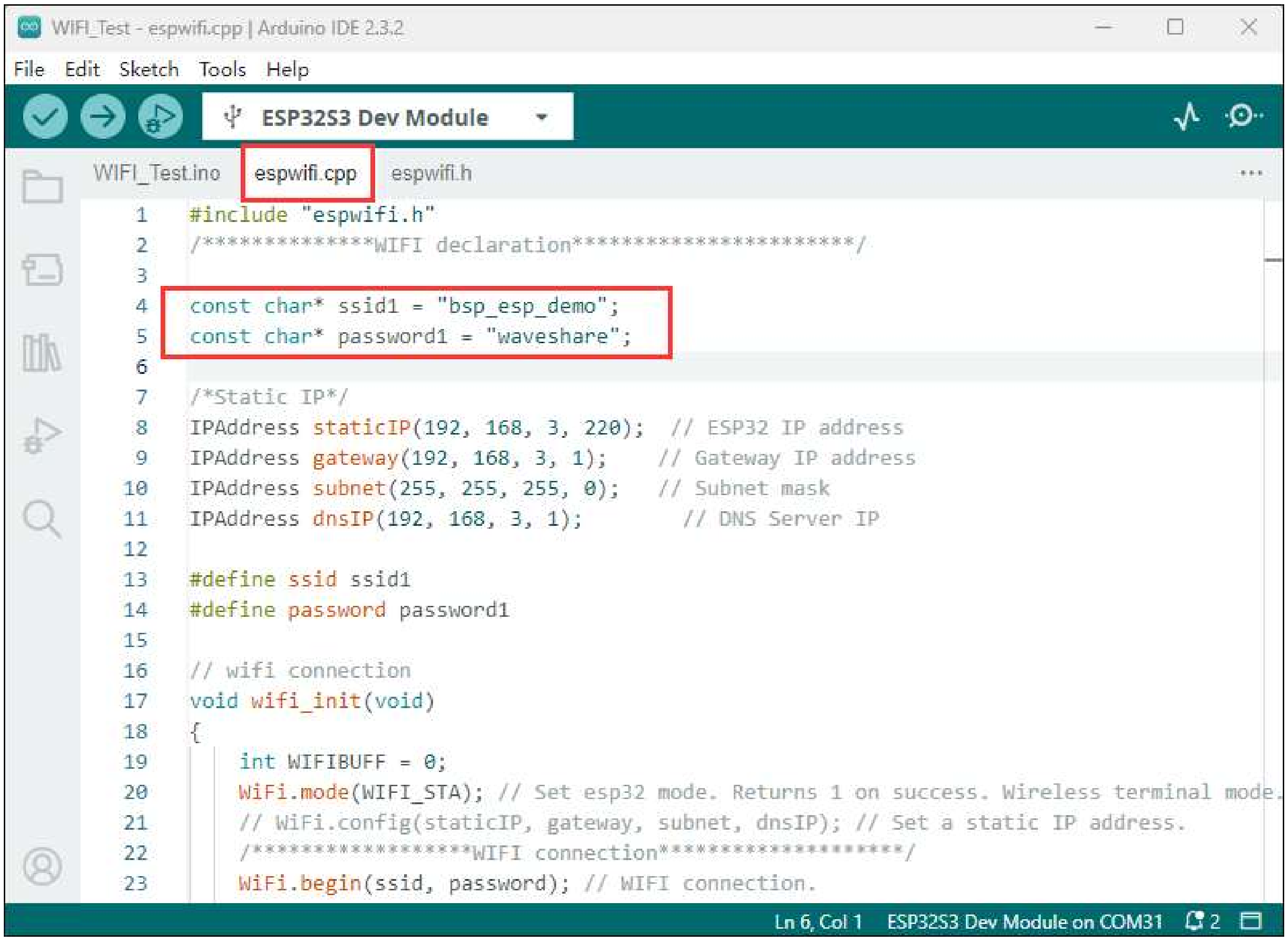
- The development board is used as a terminal role, which can connect to the AP available in the environment, and print the obtained IP information to the terminal after successful connection.

Hardware connection

- Connect the board to the computer using a USB cable (refer to demo 01)

Code modification

The project realizes that the chip is connected to WIFI in STA mode and obtains the IP address, before compiling and downloading the firmware, some code needs to be modified, specifically changing the name and password of the WIFI router to those suitable for the environment.



WIFI_Test - espwifi.cpp | Arduino IDE 2.3.2
File Edit Sketch Tools Help
ESP32S3 Dev Module
WIFI_Test.ino
espwifi.cpp
espwifi.h
Ln 6, Col 1 ESP32S3 Dev Module on COM31 2

```
#include "espwifi.h"
/*************WIFI declaration**********************/

const char* ssid1 = "bsp_esp_demo";
const char* password1 = "waveshare";

/*Static IP*/
IPAddress staticIP(192, 168, 3, 220);  // ESP32 IP address
IPAddress gateway(192, 168, 3, 1);    // Gateway IP address
IPAddress subnet(255, 255, 255, 0);   // Subnet mask
IPAddress dnsIP(192, 168, 3, 1);        // DNS Server IP

#define ssid ssid1
#define password password1

// wifi connection
void wifi_init(void)
{
    int WIFIBUFF = 0;
    WiFi.mode(WIFI_STA); // Set esp32 mode. Returns 1 on success. Wireless terminal mode.
    // WiFi.config(staticIP, gateway, subnet, dnsIP); // Set a static IP address.
    /*****************WIFI connection*******************/
    WiFi.begin(ssid, password); // WIFI connection.
```

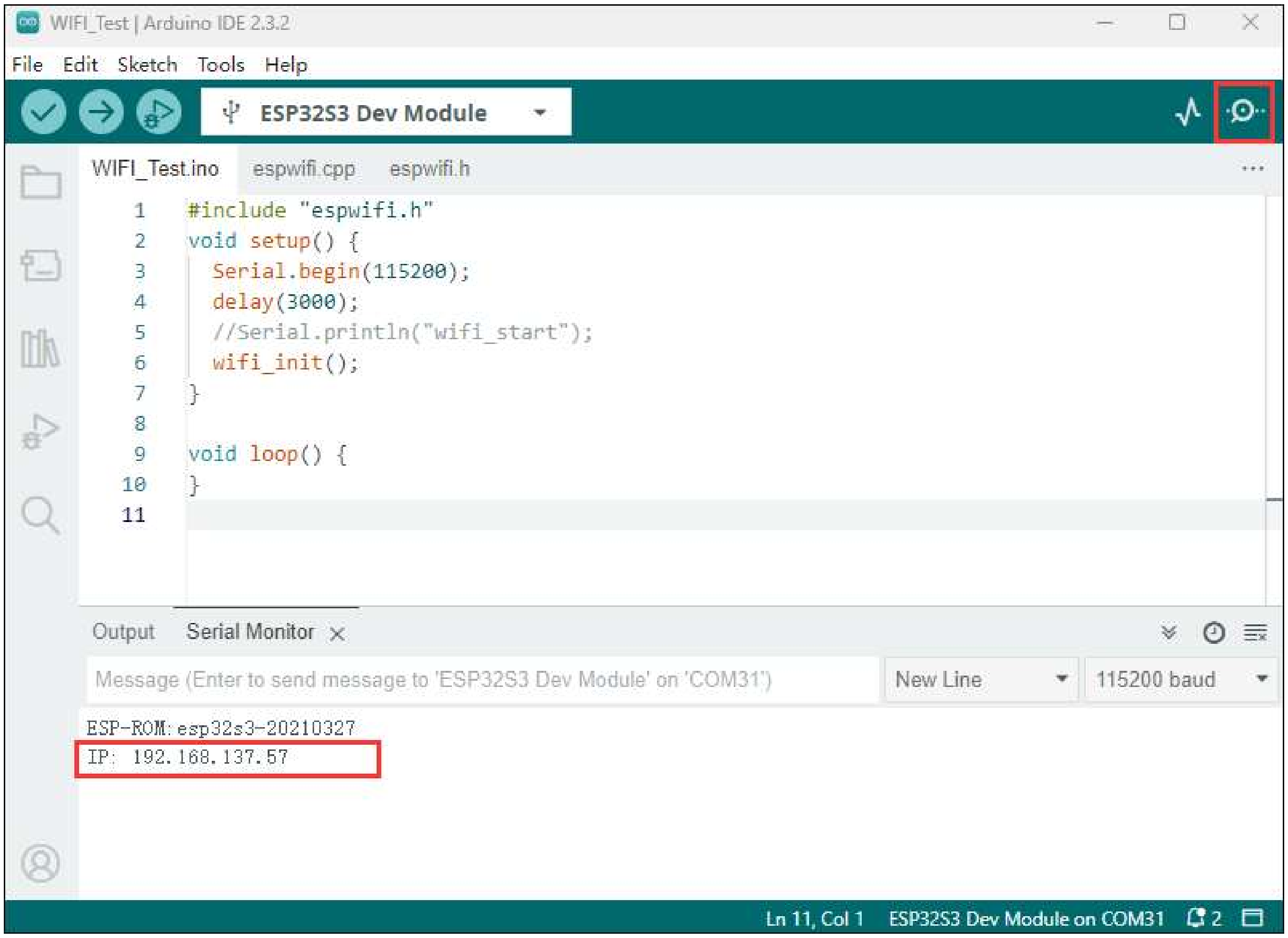
(/wiki/File:ESP32-S3-AMOLED-1.91-Ar-demo-08.png)

Code analysis

- **wifi_init(void)**: This function is used to initialize the Wi-Fi connection of the ESP32. It sets the ESP32 to Wi-Fi site mode and tries to connect to the specified Wi-Fi network (via the `ssid` and `password`). If the connection is successful, it prints the local IP address; if the connection fails within a certain period (20 * 500 milliseconds), it prints the connection failure message. At the same time, the function can also set the auto-connection and auto-reconnect functions.

Result demonstration

- The chip is successfully connected to WIFI in STA mode, and you can see the obtained IP address by clicking on the serial port monitoring device, as shown in the figure:



```
#include "espwifi.h"
void setup() {
  Serial.begin(115200);
  delay(3000);
  //Serial.println("wifi_start");
  wifi_init();
}

void loop() {
}
```

Message (Enter to send message to 'ESP32S3 Dev Module' on 'COM31')

```
ESP-ROM:esp32s3-20210327
IP: 192.168.137.57
```

(/wiki/File:ESP32-S3-AMOLED-1.91-Ar-demo-09.png)

07_Hello_World_GFX

[Collapse]

Demo description

- Implement some basic GUI interface on the screen by porting the Arduino_GFX_Library.

Hardware connection

- Connect the board to the computer using a USB cable (refer to demo 01)

Code analysis

- LCD can be driven by configuring the two classes **`Arduino_DataBus`** and **`Arduino_GFX`**.

```
Arduino_DataBus *bus = new Arduino_HWSPI(6 /* DC */, 7 /* CS */, 5 /* SCK */, 4 /* MOSI
*/);
Arduino_GFX *gfx = new Arduino_ST7789(bus, 14 /* RST */,0 /*rotation*/,0/*IPS*/,170/*w
*/,320/*h*/,35/*offsetx1*/,0/*offsety1*/,35/*offsetx2*/,0/*offsety2*/);
```

Result demonstration

After flashing the program, you can see multiple formats of Hello World displayed on the screen, as shown in the figure:

(/wiki/File:300px-ESP32-S3-LCD-1.9-details-intro200.png)

08_LVGL_Test

[Collapse]

Demo description

- Implement some multifunctional GUI interfaces on the screen by porting LVGL.

Hardware connection

- Connect the board to the computer using a USB cable (refer to demo 01)

Code analysis

For LVGL, lvgl_conf.h is its configuration file, and below are explanations for some commonly used contents.

```
/*Color depth: 1 (1 byte per pixel), 8 (RGB332), 16 (RGB565), 32 (ARGB8888)*/
#define LV_COLOR_DEPTH 16//Color depth, a macro definition that must be concerned with p
orting LVGL


#define LV_MEM_CUSTOM 0
#if LV_MEM_CUSTOM == 0
    /*Size of the memory available for `lv_mem_alloc()` in bytes (>= 2kB)*/
    #define LV_MEM_SIZE (48U * 1024U)          /*[bytes]*/

    /*Set an address for the memory pool instead of allocating it as a normal array. Can
be in external SRAM too.*/
    #define LV_MEM_ADR 0     /*0: unused*/
    /*Instead of an address give a memory allocator that will be called to get a memory
pool for LVGL. E.g. my_malloc*/
    #if LV_MEM_ADR == 0
        #undef LV_MEM_POOL_INCLUDE
        #undef LV_MEM_POOL_ALLOC
    #endif

#else       /*LV_MEM_CUSTOM*/
    #define LV_MEM_CUSTOM_INCLUDE <stdlib.h>   /*Header for the dynamic memory function
*/
    #define LV_MEM_CUSTOM_ALLOC   malloc
    #define LV_MEM_CUSTOM_FREE    free
    #define LV_MEM_CUSTOM_REALLOC realloc
#endif     /*LV_MEM_CUSTOM*/
//The above section is mainly for LVGL memory allocation,
//which defaults to lv_mem_alloc() versus lv_mem_free().
```

There are also some LVGL demos and file systems that can be set in the conf configuration file.

Code modification

- If you need to rotate the display by 90 degrees, you can find the macro definition of `Direction` in the **lcd_config.h** file, and choose one of the two.

```
#define Rotate 1    //Rotation 90
#define Normal 0    //Normal
#define Direction   Normal
```

- In order to be compatible with ESP32-C6-LCD-1.9, Touch is not enabled by default in the demo. If you need to enable it, you can find the macro definition of `EXAMPLE_USE_TOUCH` in **lcd_config.h** file and set it to 1.

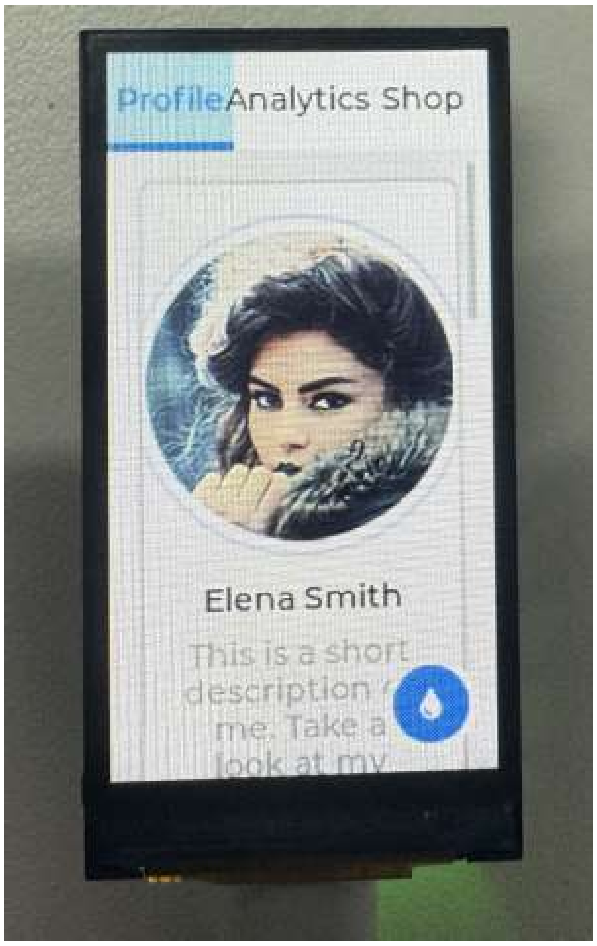
```
#define EXAMPLE_USE_TOUCH      0
```

- The TF card and screen on the board share SPI. To enable coexistence, you can find the macro definition of `EXAMPLE_USE_SDCARD` in **lcd_config.h** file and set it to 1.

```
#define EXAMPLE_USE_SDCARD     0
```

Result demonstration

- After the demo is flashed, the running result of the device is as follows:



(/wiki/File:300px-ESP32-C6-LCD-1.9-20.png)

For more learning and use of LVGL, please refer to LVGL official documentation (https://docs.lvgl.io/master/intro/introduction/index.html)

Working with ESP-IDF

[Collapse]