

ESP32-S3-AMOLED-1.91

Overview

Usage instructions

ESP32-S3-AMOLED-1.91 currently provides 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

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

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 tools, 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.

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.

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.

Environment setup

Download and install Arduino IDE

Click to visit the official website, select the corresponding system and system bit to download.

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](#)

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 for reference

Install Arduino-ESP32

Regarding ESP32-related motherboards used with the Arduino IDE, the esp32 by Espressif Systems library must be installed first.

It is generally recommended to use Install Online. If online installation fails, use Install Offline.

To install the Arduino-ESP32 tutorial, please refer to [Arduino board manager tutorial](#)

The ESP32-S3-AMOLED-1.91 development board comes with an offline package. [Click here to download: esp32_package_3.0.7_arduino_offline_package](#)

ESP32-S3-AMOLED-1.91 Development board installation instructions

Board name	Board installation requirements	Version number requirements
ESP32-S3-AMOLED -1.91	"Install Offline" / "Install Online"	3.0.7 and above

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](#)
- ESP32-S3-AMOLED-1.91 library file is stored in the sample program

ESP32-S3-AMOLED-1.91 library installation description

Library Name	Description	Version	Library Installation Requirements
LVGL	Graphical library	v8.4.0	"Install Offline"

Run the first Arduino demo

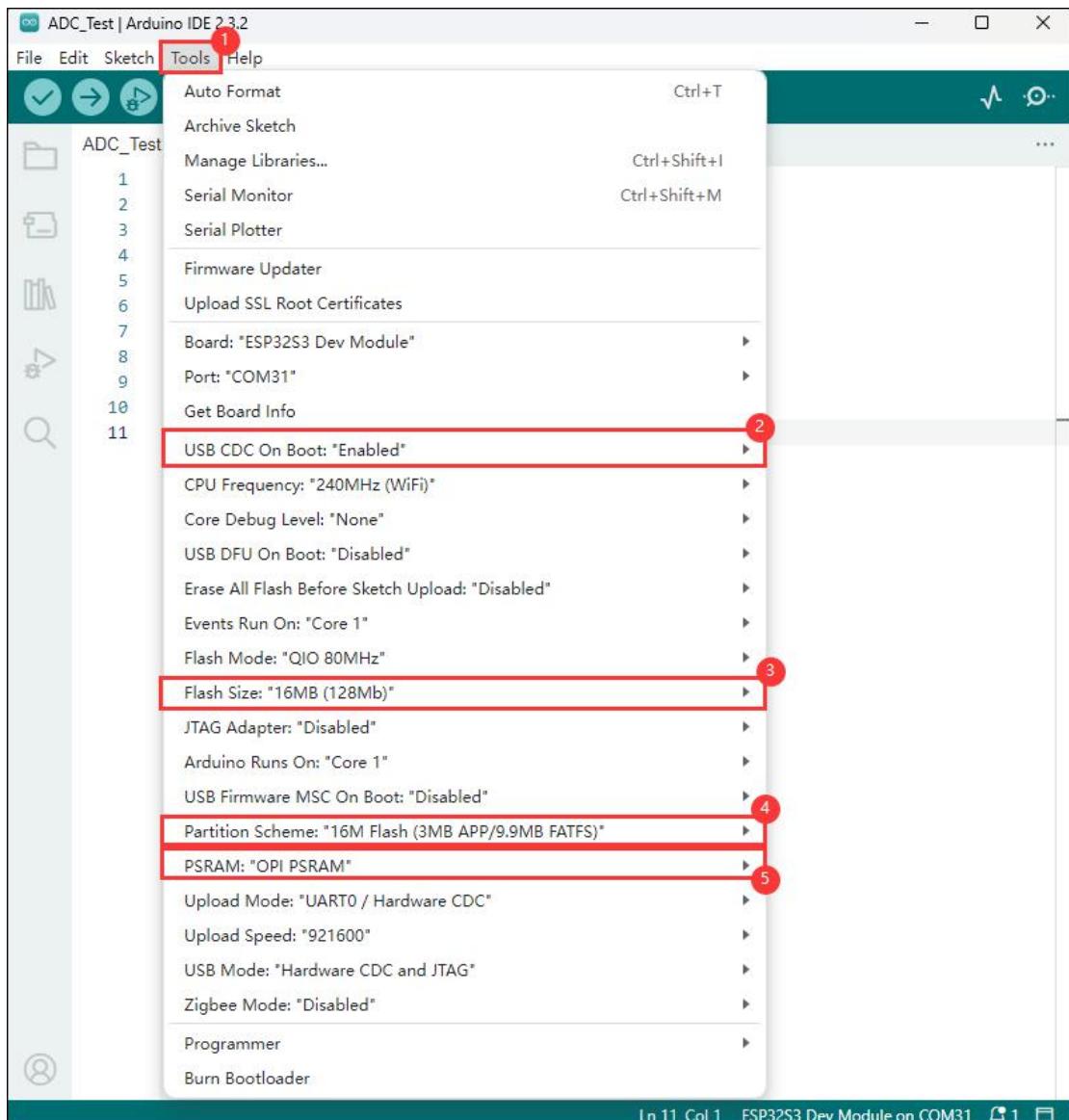
If you are just getting started with ESP32 and Arduino, and you don't know how to create, compile, flash, and run Arduino ESP32 programs, then please expand and take a look. Hope it can help you!

Demos

ESP32-S3-AMOLED-1.91 demos

Demo	Basic Description	Dependency Library
ADC	Read the current voltage value of the system	-
I2C_QMI8658	Print the original data sent by the IMU	-
LVGL	LVGL demo	LVGL
SPI_SD	Load and display the information of the TF card	-
WIFI_STA	Set to STA mode to connect to WiFi and obtain an IP address	-
WIFI_AP	Set to AP mode to obtain the IP address of the access device	-
Arduino_Playability	Playability program demo	-

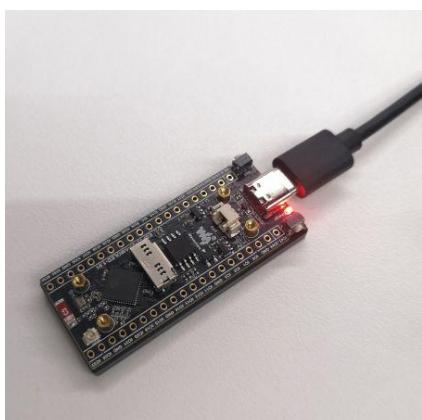
Arduino project parameter settings



ADC

Hardware connection

Connect the board to the computer using a USB cable



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)`: After initializing the ADC, in an infinite loop, read the value of the ADC channel on GPIO1, print the original ADC value, and calculate the system voltage value, performing this operation once every second.

Result demonstration

Open the serial port monitoring, you can see the ADC value and voltage in the printed output, as shown in the following figure:

The screenshot shows the Arduino IDE interface with the following details:

- Title Bar:** ADC_Test - adc_bsp.cpp | Arduino IDE 2.3.2
- Toolbar:** File, Edit, Sketch, Tools, Help, a refresh icon, and a red-bordered power icon.
- Sketch Explorer:** Shows files: ADC_Test.ino, adc_bsp.cpp, adc_bsp.h, user_app.cpp, user_app.h.
- Code Editor:** Displays the C++ code for the `adc_bsp_init` function.
- Serial Monitor:** Set to 115200 baud. It shows the output of the `adc_example` function, with the last few lines highlighted by a red box:

```
18:49:25.663 -> adc-example run
18:49:25.783 -> adc value:2520,system voltage:4.060547
18:49:26.814 -> adc value:2515,system voltage:4.052490
18:49:27.791 -> adc value:2517,system voltage:4.055713
18:49:28.785 -> adc value:2521,system voltage:4.062158
18:49:29.828 -> adc value:2518,system voltage:4.057324
```

The ADC sampling value is around 2518, and the actual voltage is greater than 3.3V, which is due to the voltage boost through the battery chip. For a detailed analysis, you can refer to the schematic diagram

I2C_QMI8658

Hardware connection

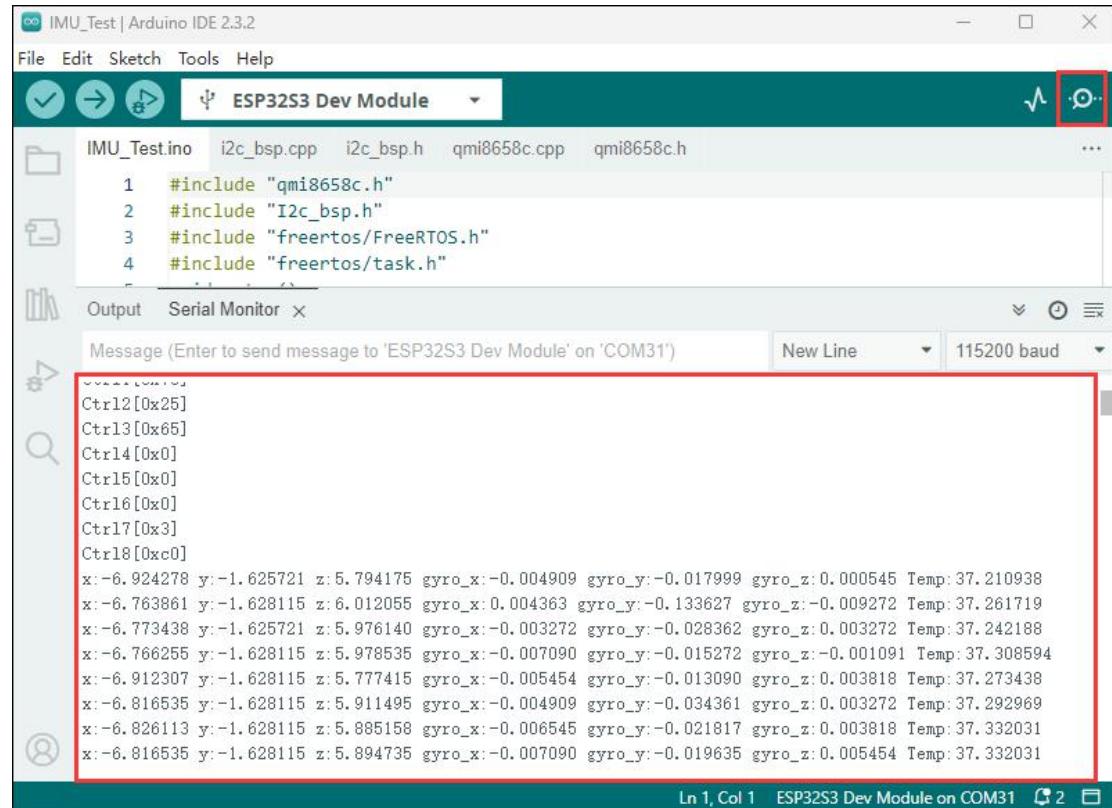
Connect the board to the computer using a USB cable

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:



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.

LVGL

Hardware connection

Connect the board to the computer using a USB cable

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 defi
nition that must be concerned with porting 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*/
#endif LV_MEM_ADR == 0

#ifndef LV_MEM_POOL_INCLUDE
#ifndef LV_MEM_POOL_ALLOC
#endif
#endif

#else /*LV_MEM_CUSTOM*/
#define LV_MEM_CUSTOM_INCLUDE <stdlib.h> /*Header for the d
ynamic memory function*/
#define LV_MEM_CUSTOM_ALLOC malloc
#define LV_MEM_CUSTOM_FREE free
#define LV_MEM_CUSTOM_REALLOC realloc#endif /*LV_MEM_CUS
TOM*////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

For boards without touch, you need to find **EXAMPLE_USE_TOUCH** in the file where the main function is located, and change 1 to 0

For boards with touch, you need to find **EXAMPLE_USE_TOUCH** in the file where the main

function is located, and change 0 to 1

```
#define EXAMPLE_USE_TOUCH 0
```

Result demonstration

The LVGL demo has high requirements for RAM and ROM, so it is necessary to configure the demo according to the requirements of environment setup, and after the program is flashed, the running effect of the device is as follows (if it is a board with touch, you need to modify the macro definition EXAMPLE_USE_TOUCH to 1):

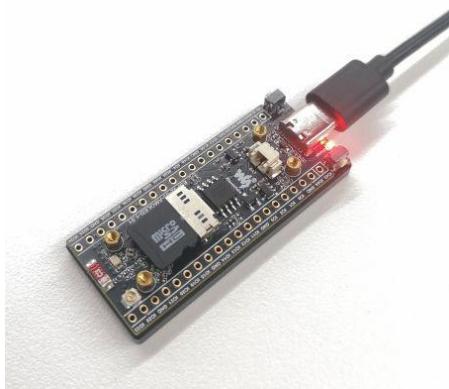


For more learning and use of LVGL, [please refer to LVGL official documentation](#)

SPI_SD

Hardware connection

Install a TF card on the board (you must insert a TF card with a capacity of less than 64G first), and use a USB cable to connect the board to the computer (if it is a V1 board, you need to comment out the macro definition VersionControl_V2, which is enabled by default)



Code analysis

SD_card_Init(void): This function initializes the TF card according to different configurations, including configuring the mounting parameters, host and card slot parameters, and then tries to mount the TF card, if successful, the card information and capacity are printed.

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:

The screenshot shows the Arduino IDE interface with the following details:

- Title Bar:** SDcard_Test | Arduino IDE 2.3.2
- File Menu:** File Edit Sketch Tools Help
- Sketch Name:** ESP32S3 Dev Module
- Code Area:** Contains the following C++ code:

```
1 #include "sd_card_bsp.h"
2 #include "freertos/FreeRTOS.h"
3 #include "freertos/task.h"
4 void setup()
5 {
6     Serial.begin(115200);
7     delay(2000);
8     SD_card_Init();
9 }
10 void loop()
11 {
```
- Output Area:** Shows the serial monitor output:

```
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
```
- Serial Monitor Settings:** Baud rate is set to 115200 baud.

How to know more about the Arduino ESP32 libraries which explain the use of TF card? Please refer to [Arduino ESP32 library TF card use](#).

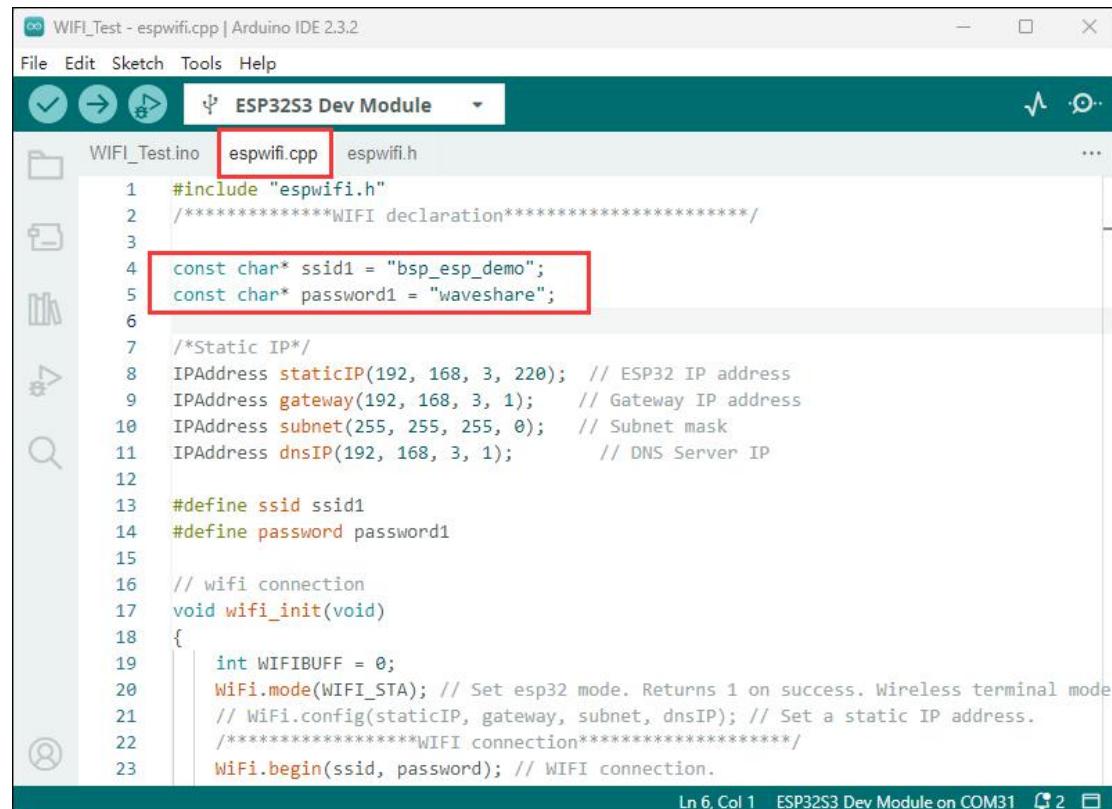
WIFI_STA

Hardware connection

Connect the board to the computer using a USB cable

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.



The screenshot shows the Arduino IDE interface with the following details:

- Title Bar:** WIFI_Test - espwifi.cpp | Arduino IDE 2.3.2
- File Menu:** File Edit Sketch Tools Help
- Sketch Name:** WIFI_Test
- Selected File:** espwifi.cpp
- Board:** ESP32S3 Dev Module
- Code Area:** The code is displayed in a monospaced font. A red box highlights the following lines:

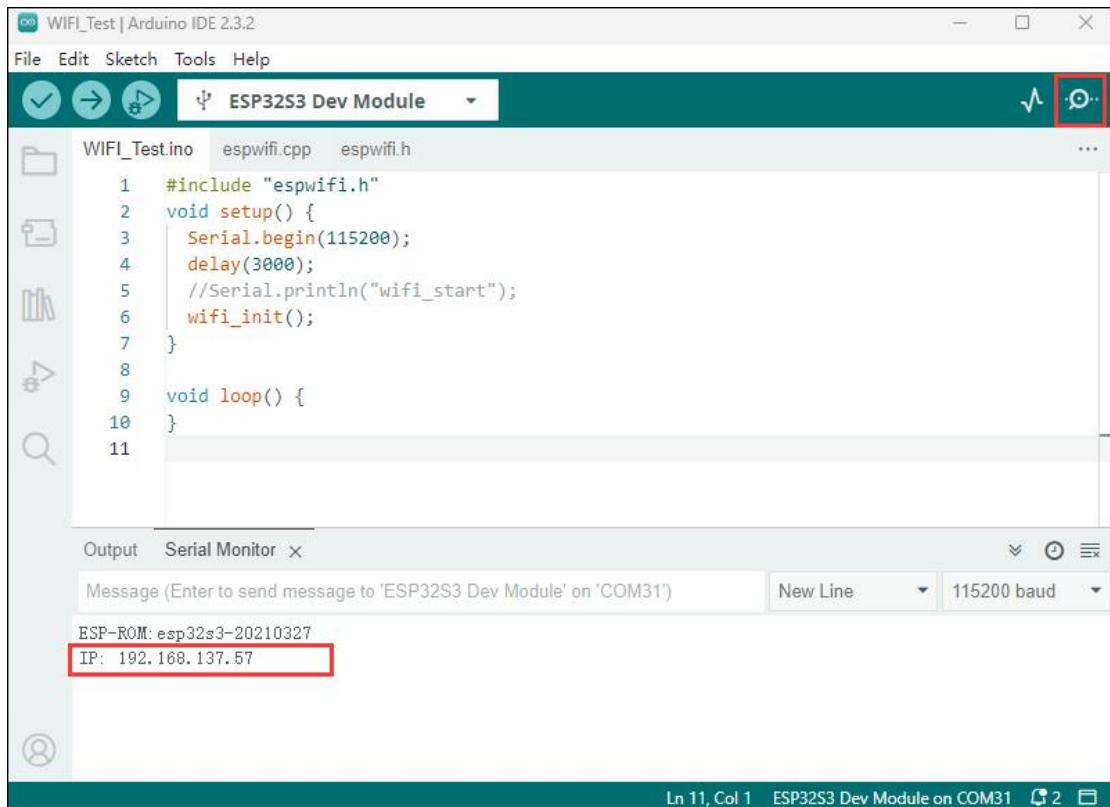
```
4 const char* ssid1 = "bsp_esp_demo";
5 const char* password1 = "waveshare";
```
- Bottom Status:** Ln 6, Col 1 ESP32S3 Dev Module on COM31

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.



WIFI_AP

Hardware connection

Connect the board to the computer using a USB cable

Code analysis

The code initializes serial communication on the ESP32 and then creates a WiFi access point named "bsp_esp_demo" with the password "waveshare", and no other continuous operations are performed in a loop while the program runs.

```
const char* ssid      = "bsp_esp_demo";
const char* password = "waveshare";

WiFi.softAP(ssid,password);
```

Result demonstration

Use your mobile phone or other device to connect to WIFI, the WiFi name is "bsp_esp_demo", and the password is "waveshare"

Arduino_Playability

We also provide some playable demos for your reference, but it should be noted that the

following demos are based on `ESP32_Arduino` versions below V3.0. Environment setup can be referred to below

Install the 2.0.9 environment directly by downloading the [Arduino2.0.9 offline package](#), double-click to install it after the download is completed

Click to download the required library files, the library file is installed in the same way as the Arduino library file installation above

Hardware connection

Connect the board to the computer using a USB cable

Result demonstration



Working with ESP-IDF

This chapter introduces setting up the ESP-IDF environment setup, including the installation of Visual Studio and the Espressif IDF plugin, program compilation, downloading, and testing of example programs, to assist users in mastering the development board and facilitating secondary development.

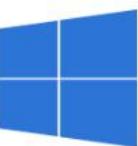
Environment setup

Download and install Visual Studio

Open the download page of VScode official website, choose the corresponding system and system bit to download

Download Visual Studio Code

Free and built on open source. Integrated Git, debugging and extensions.



↓ Windows

Windows 10, 11



↓ .deb

Debian, Ubuntu



↓ Mac

macOS 10.15+

User Installer x64 Arm64

.deb x64 Arm32 Arm64

.zip Intel chip Apple silicon Universal

System x64 Arm64

.rpm x64 Arm32 Arm64

CLI Intel chip Apple silicon

Installer x64 Arm64

.tar.gz x64 Arm32 Arm64

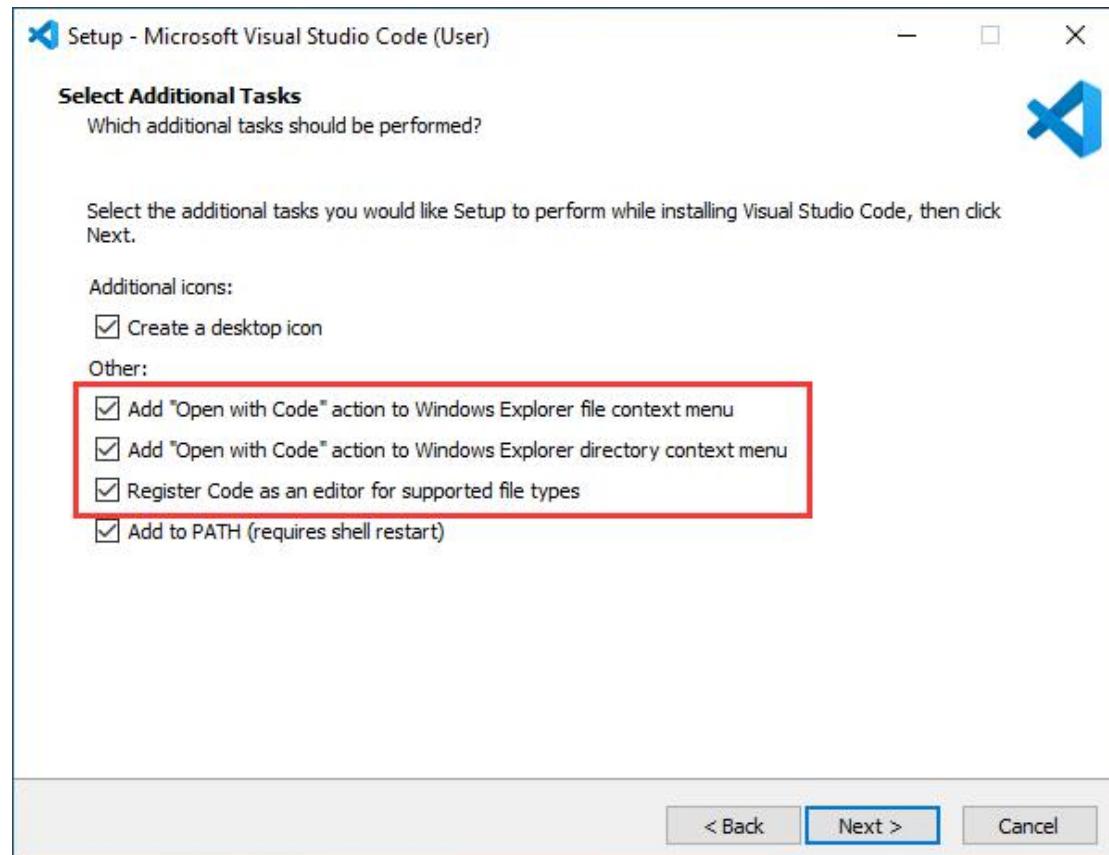
.zip x64 Arm64

Snap Snap Store

CLI x64 Arm64

CLI x64 Arm32 Arm64

After running the installation package, the rest can be installed by default, but here for the subsequent experience, it is recommended to check boxes 1, 2, and 3



After the first two items are enabled, you can open VSCode directly by right-clicking files or directories, which can improve the subsequent user experience.

After the third item is enabled, you can select VSCode directly when you choose how to open it.

The environment setup is carried out on the Windows 10 system, Linux and Mac users can access [ESP-IDF environment setup](#) for reference

Install Espressif IDF Plug-in

It is generally recommended to use Install Online. If online installation fails due to network factor, use Install Offline.

For more information about how to install the Espressif IDF plugin, see [Install Espressif IDF Plugin](#)

Run the first ESP-IDF demo

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

Demos

ESP32-S3-AMOLED-1.91 demos

Demo	Basic Description	Dependency Library
ADC	Read the current voltage value of the system	-
I2C_QMI8658	Print the original data sent by the IMU	-
LVGL	LVGL demo	LVGL
SPI_SD	Load and display the information of the TF card	-
WIFI_STA	Set to STA mode to connect to WiFi and obtain an IP address	-
WIFI_AP	Set to AP mode to obtain the IP address of the access device	-
FactoryProgram	Comprehensive project	LVGL

ADC

Hardware connection

Connect the board to the computer using a USB cable

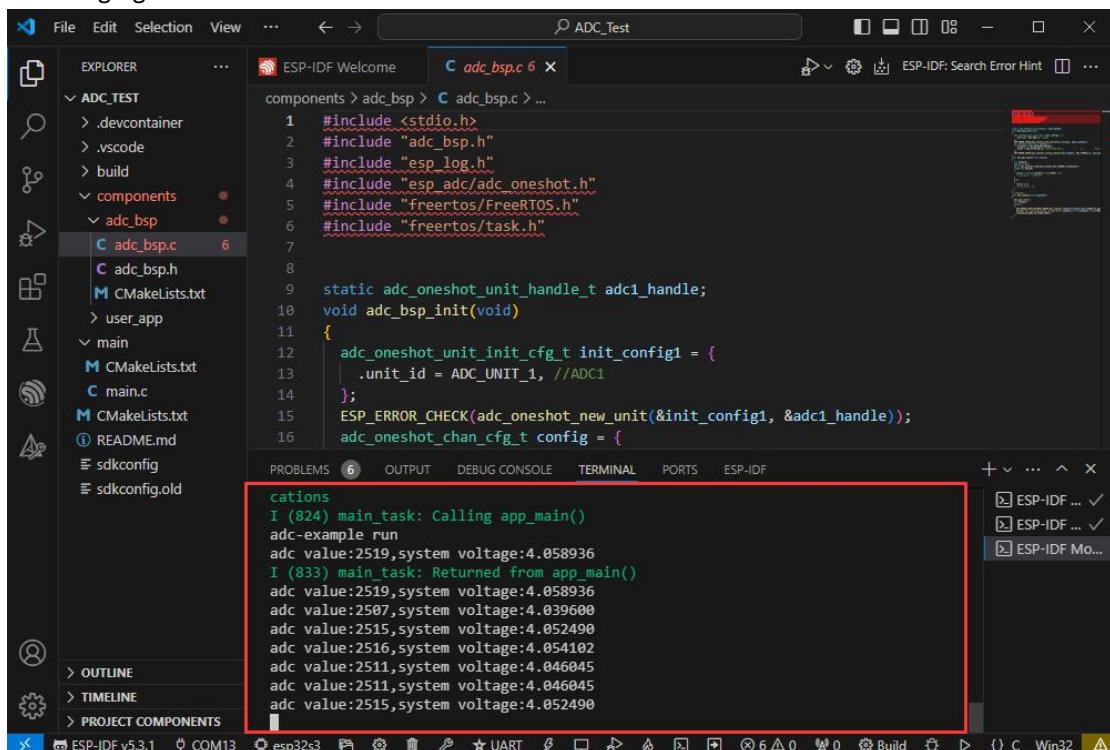
Code analysis

- `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)`: 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)`: After initializing the ADC, in an infinite loop, read the value of the ADC channel on GPIO1, print the original ADC value, and calculate the system voltage value, performing this operation once every second.

Result demonstration

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

Open the serial port monitoring, you can see the output ADC value and voltage, as shown in the following figure:



The screenshot shows the Visual Studio Code interface with the ESP-IDF extension. The left sidebar shows the project structure for 'ADC_TEST'. The main area shows the code editor with `adc_bsp.c` open. The bottom right is the terminal window displaying serial port output. A red box highlights the terminal window, which contains the following text:

```

I (824) main_task: Calling app_main()
adc-example run
adc value:2519,system voltage:4.058936
I (833) main_task: Returned from app_main()
adc value:2519,system voltage:4.058936
adc value:2507,system voltage:4.039600
adc value:2515,system voltage:4.052490
adc value:2516,system voltage:4.054102
adc value:2511,system voltage:4.046045
adc value:2511,system voltage:4.046045
adc value:2515,system voltage:4.052490

```

It can be seen that the ADC sampling value is about 2511, and the actual voltage is greater than 3.3V, because this is the voltage after the battery chip is boosted. For specific analysis, you can click to view the schematic diagram

I2C_QMI8658

Hardware connection

Connect the board to the computer using a USB cable

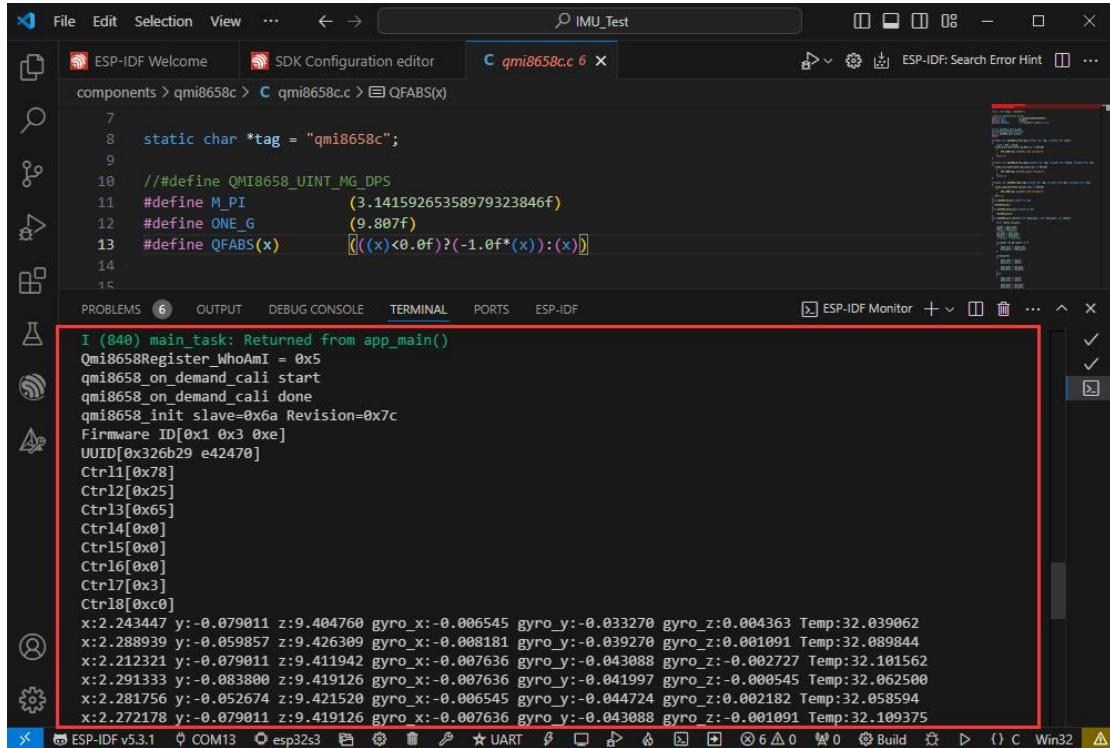
Code analysis

`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.

Result demonstration

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

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:



The screenshot shows the ESP-IDF terminal window titled "IMU_Test". The terminal tab is active, displaying the output of the "main_task" function. The output includes initialization messages like "Qmi8658Register_WhoAmI = 0x5" and "qmi8658_on_demand_cali start", followed by a series of Euler angle readings for each axis over time. The data is formatted as "x:y:z" where x, y, and z represent the three axes. A red box highlights the terminal output area.

```
I (840) main_task: Returned from app_main()
Qmi8658Register_WhoAmI = 0x5
qmi8658_on_demand_cali start
qmi8658_on_demand_cali done
qmi8658_init slave=0x6a Revision=0x7c
Firmware ID[0x1 0x3 0xe]
UUID[0x326b29 e42470]
Ctrl1[0x78]
Ctrl2[0x25]
Ctrl3[0x65]
Ctrl4[0x0]
Ctrl5[0x0]
Ctrl6[0x0]
Ctrl7[0x3]
Ctrl8[0xc0]
x:2.243447 y:-0.079011 z:9.404760 gyro_x:-0.006545 gyro_y:-0.033270 gyro_z:0.004363 Temp:32.039062
x:2.288939 y:-0.059857 z:9.426389 gyro_x:-0.008181 gyro_y:-0.039270 gyro_z:0.001091 Temp:32.089844
x:2.212321 y:-0.079011 z:9.411942 gyro_x:-0.007636 gyro_y:-0.043088 gyro_z:-0.002727 Temp:32.101562
x:2.291333 y:-0.083800 z:9.419126 gyro_x:-0.007636 gyro_y:-0.041997 gyro_z:-0.000545 Temp:32.062500
x:2.281756 y:-0.052674 z:9.421520 gyro_x:-0.006545 gyro_y:-0.044724 gyro_z:0.002182 Temp:32.058594
x:2.272178 y:-0.079011 z:9.419126 gyro_x:-0.007636 gyro_y:-0.043088 gyro_z:-0.001091 Temp:32.109375
```

You can see that it is output every 1 second. If you need to modify or refer to it, you can directly go to the qmi source file to modify it

LVGL

Hardware connection

Connect the board to the computer using a USB cable

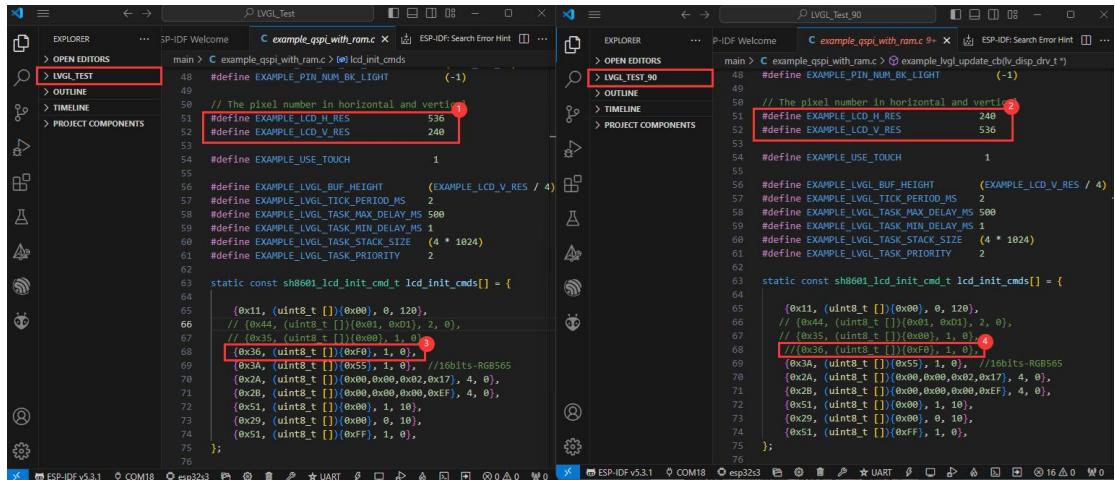
Code analysis

There are two projects in the LVGL catalog, one is the original display, and the other is the 90-degree rotation display, which can be selected according to your needs.

The original display and the 90-degree rotation display are mainly different from two points:

- ① and ②: There is a difference in the number of pixels in the horizontal and vertical directions
- ③ and ④: The register at the 0x36 address under the `lcd_init_cmds` array is responsible for rotation, and modifying its value can achieve a rotation effect. If no rotation is needed, you can

directly delete {0x36,(uint8_t [])}{0xF0,1,0}}



```
main > C example_lvgl_with_ram.c x [e] lcd_init_cmds
48 #define EXAMPLE_PIN_NUM_BK_LIGHT (-1)
49
50 // The pixel number in horizontal and vertical
51 #define EXAMPLE_LCD_H_RES 536
52 #define EXAMPLE_LCD_V_RES 240
53
54 #define EXAMPLE_USE_TOUCH 1
55
56 #define EXAMPLE_LVGL_BUF_HEIGHT (EXAMPLE_LCD_V_RES / 4)
57 #define EXAMPLE_LVGL_TICK_PERIOD_MS 2
58 #define EXAMPLE_LVGL_TASK_MAX_DELAY_MS 500
59 #define EXAMPLE_LVGL_TASK_MIN_DELAY_MS 1
60 #define EXAMPLE_LVGL_TASK_STACK_SIZE (4 * 1024)
61 #define EXAMPLE_LVGL_TASK_PRIORITY 2
62
63 static const sh8601_lcd_init_cmd_t lcd_init_cmds[] = {
64
65     {0x11, (uint8_t []){0x00}, 0, 120},
66     // {0x44, (uint8_t []){0x01, 0x01}, 2, 0},
67     // {0x35, (uint8_t []){0x00}, 1, 0},
68     {0x36, (uint8_t []){0xF0}, 1, 0},
69
70     {0xA3, (uint8_t []){0x55}, 1, 0}, // 16bits-RGB565
71     {0xA2, (uint8_t []){0x00, 0x00, 0x02, 0x17}, 4, 0},
72     {0xB8, (uint8_t []){0x00, 0x00, 0x00, 0xEF}, 4, 0},
73     {0x53, (uint8_t []){0x00}, 1, 10},
74     {0x29, (uint8_t []){0x00}, 0, 10},
75     {0x51, (uint8_t []){0xFF}, 1, 0},
76 };

```

```
main > C example_lvgl_with_ram.c x [e] example_lvgl_update_cbfv_disp_div.t
48 #define EXAMPLE_PIN_NUM_BK_LIGHT (-1)
49
50 // The pixel number in horizontal and vertical
51 #define EXAMPLE_LCD_H_RES 240
52 #define EXAMPLE_LCD_V_RES 536
53
54 #define EXAMPLE_USE_TOUCH 1
55
56 #define EXAMPLE_LVGL_BUF_HEIGHT (EXAMPLE_LCD_V_RES / 4)
57 #define EXAMPLE_LVGL_TICK_PERIOD_MS 2
58 #define EXAMPLE_LVGL_TASK_MAX_DELAY_MS 500
59 #define EXAMPLE_LVGL_TASK_MIN_DELAY_MS 1
60 #define EXAMPLE_LVGL_TASK_STACK_SIZE (4 * 1024)
61 #define EXAMPLE_LVGL_TASK_PRIORITY 2
62
63 static const sh8601_lcd_init_cmd_t lcd_init_cmds[] = {
64
65     {0x11, (uint8_t []){0x00}, 0, 120},
66     // {0x44, (uint8_t []){0x01, 0x01}, 2, 0},
67     // {0x35, (uint8_t []){0x00}, 1, 0},
68     {0x3A, (uint8_t []){0x55}, 1, 0}, // 16bits-RGB565
69     {0xA2, (uint8_t []){0x00, 0x00, 0x02, 0x17}, 4, 0},
70     {0xB8, (uint8_t []){0x00, 0x00, 0x00, 0xEF}, 4, 0},
71     {0x51, (uint8_t []){0x00}, 1, 10},
72     {0x29, (uint8_t []){0x00}, 0, 10},
73     {0x51, (uint8_t []){0xFF}, 1, 0},
74     {0x51, (uint8_t []){0xFF}, 1, 0},
75 };

```

Code modification

For boards without touch, you need to find **EXAMPLE_USE_TOUCH** in the file where the main function is located, and change 1 to 0

For boards with touch, you need to find **EXAMPLE_USE_TOUCH** in the file where the main function is located, and change 0 to 1

```
#define EXAMPLE_USE_TOUCH 0
```

Result demonstration



Want to know the description of the screen driver IC register? Please refer to [RM67162 Datasheet](#)

SPI_SD

Hardware connection

Install a TF card on the board (you must insert a TF card with a capacity of less than 64G first), and use a USB cable to connect the board to the computer (if it is a V1 board, you need to comment out the macro definition VersionControl_V2 in sd_card_bsp.c, which is enabled by default)

Code analysis

SD_card_Init(void): This function initializes the TF card according to different configurations, including configuring the mounting parameters, host and card slot parameters, and then tries to mount the TF card, if successful, the card information and capacity are printed.

Result demonstration

You can see the output TF card information

The screenshot shows the ESP-IDF IDE interface. The left sidebar displays the project structure under 'EXPLORER'. The main area shows the code for `sd_card_bsp.c`. The terminal tab at the bottom shows the following output:

```
I (841) gpio: GPIO[9] InputEn: 0| OutputEn: 1| OpenDrain: 0| Pullup: 0| PULLDOWN: 0| Intr: 0
I (850) sdspi_transaction: cmd=52, R1 response: command not supported
I (898) sdspi_transaction: cmd=5, R1 response: command not supported
Name: SD16G
Type: SDHC/SDXC
Speed: 20.00 MHz (limit: 20.00 MHz)
Size: 7680MB
CSD: ver=2, sector_size=512, capacity=15728640 read_b1_len=9
SSR: bus width=1
SIOs:7.50
I (924) main_task: Returned from app_main()
```

How to know more about the Arduino ESP32 libraries which explain the use of TF card? Please refer to Arduino ESP32 library TF card use.

WIFI_STA

Hardware connection

Connect the board to the computer using a USB cable

Code analysis

espwifi_Init(void): This function is used for WiFi initialization on ESP32. It sequentially initializes

non-volatile storage, the TCP/IP stack, creates a default event loop and a default WiFi site network interface, initializes WiFi with the default configuration, registers event handlers to handle WiFi and IP-related events, sets WiFi connection parameters, and starts WiFi.

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.

The screenshot shows the Visual Studio Code interface with the ESP-IDF extension. The left sidebar displays the project structure under 'OPEN EDITORS' and 'WIFI_TEST'. The main editor area shows the file 'esp_wifi_bsp.c' with code for initializing WiFi. A red box highlights the configuration section where the SSID and password are set:

```
static void event_handler(void *arg, esp_event_base_t event_base, int32_t event_id, void *event_data)
{
    void espwifi_Init(void)
    {
        nvs_flash_init(); // Initialize the default NVS area
        esp_netif_init(); // Initialize the TCP/IP stack
        esp_event_loop_create_default(); // Create the default event loop
        esp_netif_create_default_wifi_sta(); // Add the TCP/IP protocol stack to the default event loop
        wifi_init_config_t cfg = WIFI_INIT_CONFIG_DEFAULT(); // Default config
        esp_wifi_init(&cfg); // Initialize WiFi
        esp_event_handler_instance_t Instance_WIFI_IP;
        esp_event_handler_instance_register(WIFI_EVENT, ESP_EVENT_ANY_ID, &event_handler, NULL,
        esp_event_handler_instance_register(IP_EVENT, IP_EVENT_STA_GOT_IP, &event_handler, NULL,
        wifi_config_t wifi_config = {
            .sta = {
                .ssid = "bsp_esp_demo",
                .password = "waveshare",
            },
        );
        esp_wifi_set_mode(WIFI_MODE_STA); // Mode STA
        esp_wifi_set_config(WIFI_IF_STA, &wifi_config); // Set WIFI
        esp_wifi_start(); // Start WIFI
    }
}
```

Result demonstration

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

* The chip successfully connects to WIFI and obtains an IP address while in STA mode.

```

10 void espwifi_Init(void)
31
32 static void event_handler(void *arg, esp_event_base_t event_base, int32_t event_id)
33 {
34     if (event_id == WIFI_EVENT_STA_START)
35     {
36         esp_wifi_connect(); // Connect to WIFI
37     }
38     else if (event_id == IP_EVENT_STA_GOT_IP)
39     {
40         ip_event_got_ip_t *event = (ip_event_got_ip_t *)event_data;
41         char ip[25];
42         uint32_t pxip = event->ip_info.ip.addr;
43         sprintf(ip, "%d.%d.%d.%d", (uint8_t)(pxip), (uint8_t)(pxip >> 8), (uint8_t)(pxip >> 16), (uint8_t)(pxip >> 24));
44         printf("IP:%s\n", ip);
45     }
}

```

PROBLEMS 5 OUTPUT DEBUG CONSOLE TERMINAL PORTS ESP-IDF ESP-IDF Monitor

I (1137) wifi:new:<1,0>, old:<1,0>, ap:<255,255>, sta:<1,0>, prof:1, snd_ch_cfg:0x0
I (1138) wifi:state: init -> auth (0xb0)
I (1166) wifi:state: auth -> assoc (0x0)
I (1177) wifi:state: assoc -> run (0x10)
I (1254) wifi:connected with bsp_esp_demo, aid = 10, channel 1, BW20, bssid = de:45:46:b8:2f:6d
I (1254) wifi:security: WPA2-PSK, phy: bgn, rssi: -81
I (1258) wifi:pm start, type: 1
I (1259) wifi:dp: 1, bi: 102400, li: 3, scale listen interval from 307200 us to 307200 us
I (1267) wifi:set rx beacon pti, rx_bcn_pti: 0, bcn_timeout: 25000, mt_pti: 0, mt_time: 10000
I (1443) wifi:AP's beacon interval = 102400 us, DTIM period = 3
I (2277) esp netif handlers: sta ip: 192.168.137.57, mask: 255.255.255.0, gw: 192.168.137.1
IP:192.168.137.57

ESP-IDF v5.3.1 COM31 esp32s3 UART Build

WIFI_AP

Hardware connection

Connect the board to the computer using a USB cable

Code analysis

`wifi_init_softap(void)`: This function is used to initialize the ESP32's Wi-Fi soft access point, including setting up the network interface, registering event handling functions, configuring soft AP parameters, and starting the soft AP.

Result demonstration

When the chip is in AP mode, use the mobile phone to successfully connect to WIFI and the serial port will print the MAC address of the connected device and the IP address assigned to the device.

```

File Edit Selection View ... ← → ⌘ WIFIAPIP
EXPLORER OPEN EDITORS C softap_example_main.c 9+ X ESP-IDF Welcome
main > C softap_example_main.c > EXAMPLE_ESP_WIFI_PASS
1 /* Wifi softAP Example
2  * CONDITIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.
3  */
4 #include <string.h>
5 #include "freertos/FreeRTOS.h"
6 #include "freertos/task.h"
7 #include "esp_mac.h"
8 #include "esp_wifi.h"
9 #include "esp_event.h"
10 #include "esp_log.h"
11 #include "nvs_flash.h"
12
13 #include "lwip/err.h"
14 #include "lwip/sys.h"
15
16 #define EXAMPLE_ESP_WIFI_SSID "waveshare_esp32"
17 #define EXAMPLE_ESP_WIFI_PASS "Wav123456"
18
19 #define EXAMPLE_ESP_WIFI_CHANNEL 1
20 #define EXAMPLE_MAX_STA_CONN 4
21
22 static const char *TAG = "wifi softAP";
23 uint8_t mac[6];
24 static void wifi_event_handler(void* arg, esp_event_base_t event_base,
25 int32_t event_id, void* event_data)
26
PROBLEMS 11 OUTPUT DEBUG CONSOLE TERMINAL PORTS ESP-IDF ESP-IDF Monitor + ...
I (91) esp_image: segment 0: paddr=00010020 vaddr=3c090020 size=1d2c4h (119492) map
I (121) esp_image: segment 1: paddr=0002d2e0 vaddr=3fc99300 size=02d2ch ( 1156) load
I (123) esp_image: segment 2: paddr=00030020 vaddr=42000020 size=802dch (525020) map
I (221) esp_image: segment 3: paddr=000b0304 vaddr=3fc9c02c size=01bc4h ( 7108) load
I (222) esp_image: segment 4: paddr=000b1ed0 vaddr=40374000 size=152a0h ( 86688) load
I (254) boot: Loaded app from partition at offset 0x10000
I (254) boot: Disabling RNG early entropy source...
MAC: 194:17:41:209:76:199 IP: 192.168.4.2

```

FactoryProgram

Hardware connection

Use a USB cable to connect the board to the computer (if it is a V1 version, you need to comment out the CMake statement target_compile_definitions which is enabled by default in the CMakeLists.txt file under the ui_bsp directory)

Code modification

For boards without touch, you need to find EXAMPLE_USE_TOUCH in the file where the main function is located, and change 1 to 0

For boards with touch, you need to find EXAMPLE_USE_TOUCH in the file where the main function is located, and change 0 to 1

```
#define EXAMPLE_USE_TOUCH 0
```

Result demonstration

The touch version supports BOOT button and touch switching

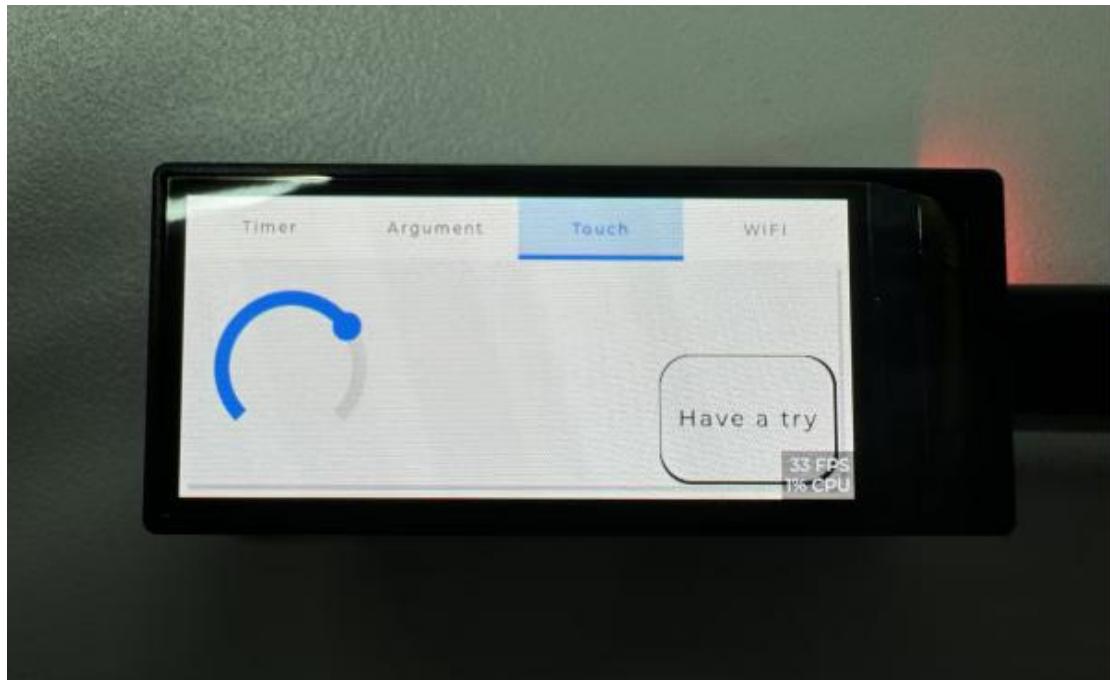
Switch pages by using the BOOT button on the side of the board to display the analog clock interface first



Switch to the Argument interface by pressing the BOOT button, where you can see a page with information on the onboard hardware



Click the BOOT button again to jump to the Touch interface, which is the component interface



Click the BOOT button again to jump to the WIFI interface



Resources

Schematic diagram

[ESP32-S3-AMOLED-1.91 Schematic diagram](#)

Demos

[ESP32-S3-AMOLED-1.91 Demo](#)

[ESP32-S3-AMOLED-1.91-Arduino Playability](#)

Datasheets

[ESP32-S3](#)

[ESP32-S3 Datasheet](#)

[ESP32-S3 Technical reference manual](#)

Other components

[QMI8658 Datasheet](#)

[RM67162 Datasheet](#)

Software tools

Arduino

[Arduino IDE Official download link](#)

[ESP32-Arduino official documentation](#)

[esp32_package_3.0.7_arduino_offline_package](#)

[esp32_package_2.0.9_arduino_offline_package](#)

VScode

[VScode official website](#)

Other resource link

[LVGL official documentation](#)

FAQ

Question1: After the module downloads the demo and re-downloads it, why sometimes it can't connect to the serial port or the flashing fails?

Answer1:

Long press the BOOT button, press RESET at the same time, then release RESET, then release the BOOT button, at this time the module can enter the download mode, which can solve most of the problems that can not be downloaded.

Question2: Failed to set up the VSCode environment?

Answer2:

First consider the network issue, try switching to another network

Question3: Error when compiling an Arduino program?

Answer3:

Check if the Arduino IDE -> Tools is correctly configured

Question4: Is it stuck when sliding pictures displayed?

Answer4:

Modify LVGL display cache to the full screen size

Modify the LV_IMG_CACHE_DEF_SIZE option in the configuration to 1000 to achieve some optimization

Question5: Can't display Chinese?

Answer5:

Basic Chinese can be displayed, but if it is a rare character, it cannot be displayed

You can transcode the required rare characters through the transcoding software, and then add them to the project font library

Question6: How to deal with the first compilation of the program being extremely slow?

Answer6:

It's normal for the first compilation to be slow, just be patient

Question7: How to handle the display "waiting for download..." on the serial port after successfully ESP-IDF flashing?

Answer7:

If there is a reset button on the development board, press the reset button; if there is no reset button, please power it on again

Question8: What should I do if I can't find the AppData folder?

Answer8:

Some AppData folders are hidden by default and can be set to show.

English system: Explorer->View->Check "Hidden items"

Chinese system: File Explorer -> View -> Display -> Check "Hidden Items"

Question9: How do I check the COM port I use?

Answer9:

Windows system:

- ① View through Device Manager: Press the Windows + R keys to open the "Run" dialog box; input devmgmt.msc and press Enter to open the Device Manager; expand the "Ports (COM and LPT)" section, where all COM ports and their current statuses will be listed.
- ② Use the command prompt to view: Open the Command Prompt (CMD), enter the “mode” command, which will display status information for all COM ports.
- ③ Check hardware connections: If you have already connected external devices to the COM port, the device usually occupies a port number, which can be determined by checking the connected hardware.

Linux system:

- ① Use the dmesg command to view: Open the terminal.
- ② Use the ls command to view: Enter ls /dev/ttyS* or ls /dev/ttyUSB* to list all serial port devices.
- ③ Use the setserial command to view: Enter setserial -g /dev/ttyS* to view the configuration information of all serial port devices.

Question10: Why does the program flashing fail when using a MAC device?

Answer10:

Install MAC Driver and flash again.