AI Camera Development Board

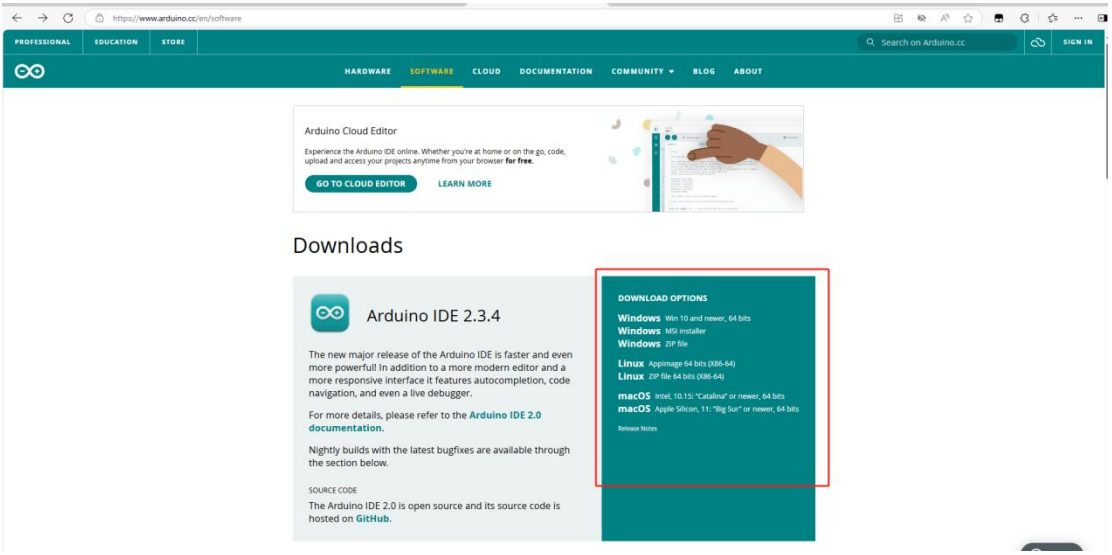
Preface

Installing Arduino IDE

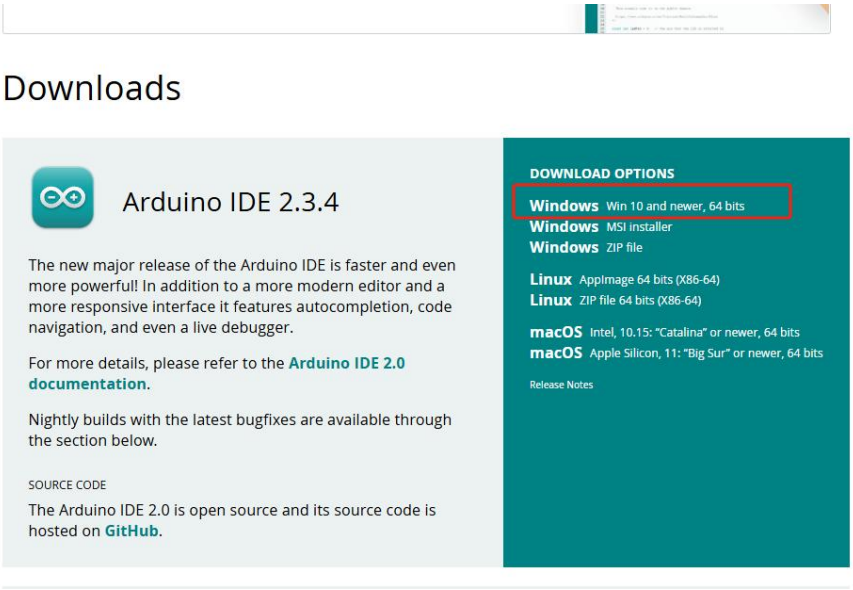
1. Open the official Arduino website

https://www.arduino.cc/en/software

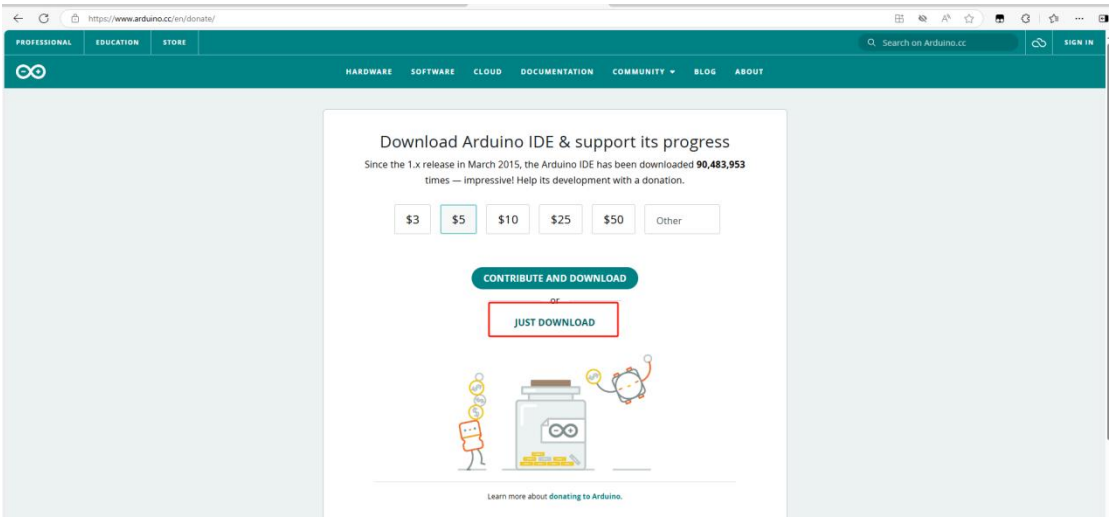
2. Select and install the appropriate version according to your operating system.



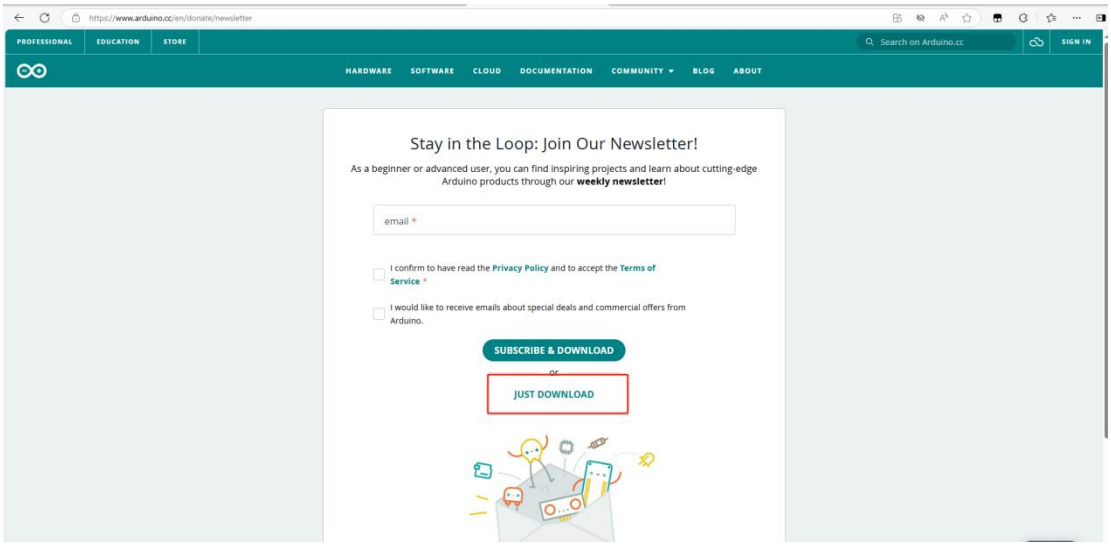
2.1 Here I choose the first one.



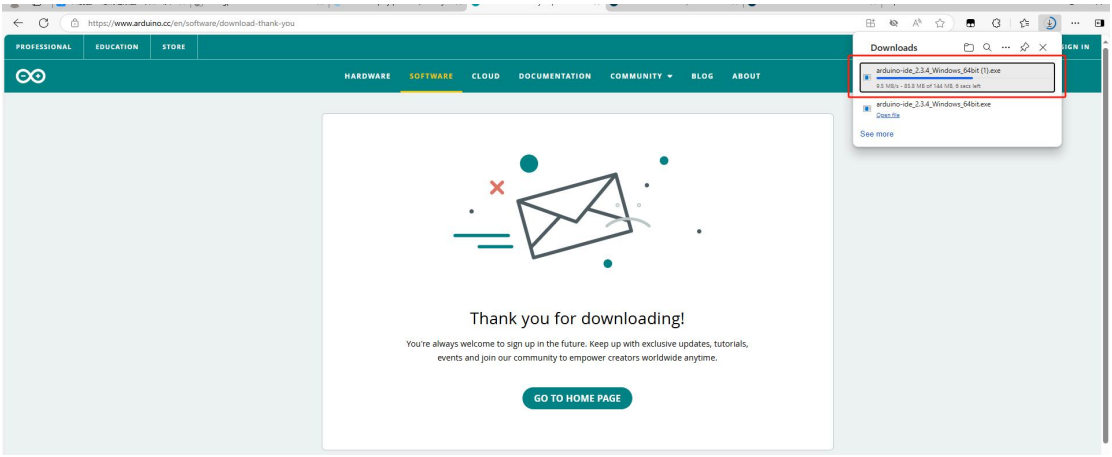
3. Select "Download Only" if you wish.



3.3 The same as above.

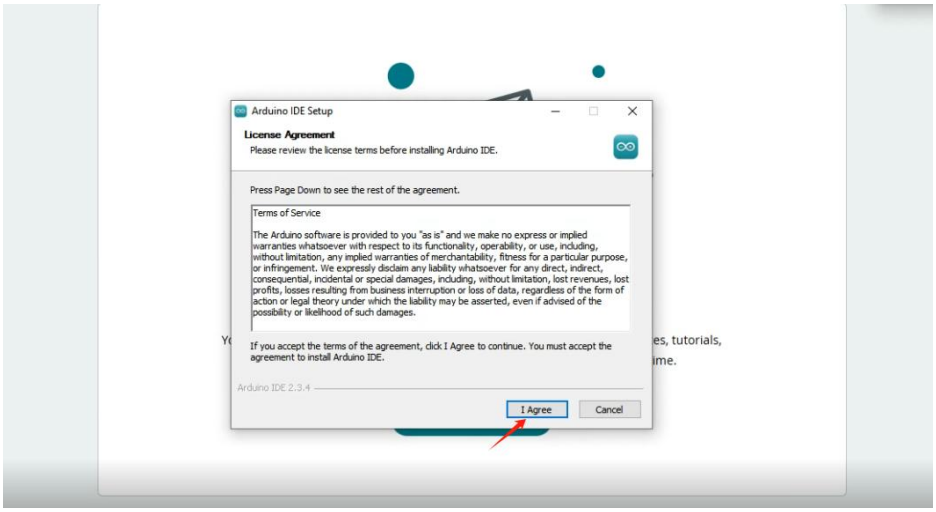


4. Wait for the download to complete

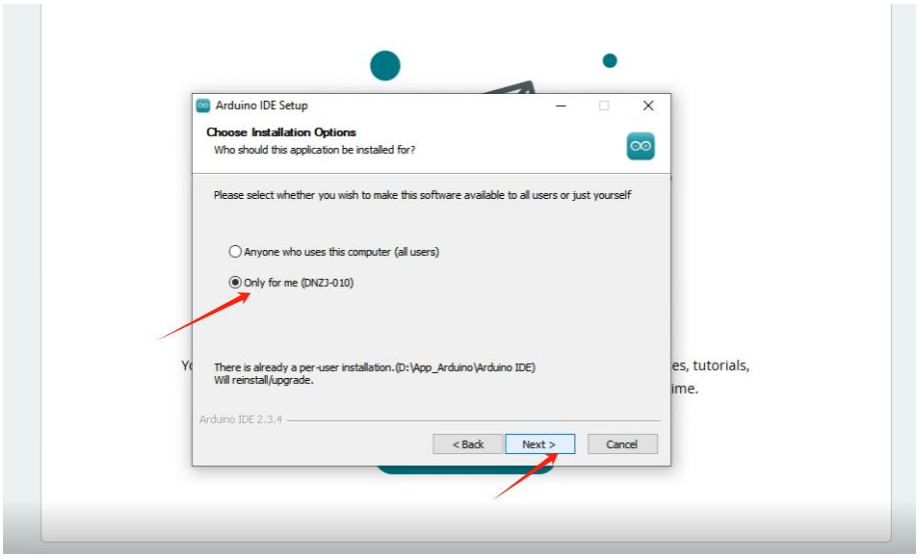


5. Confirm the download process

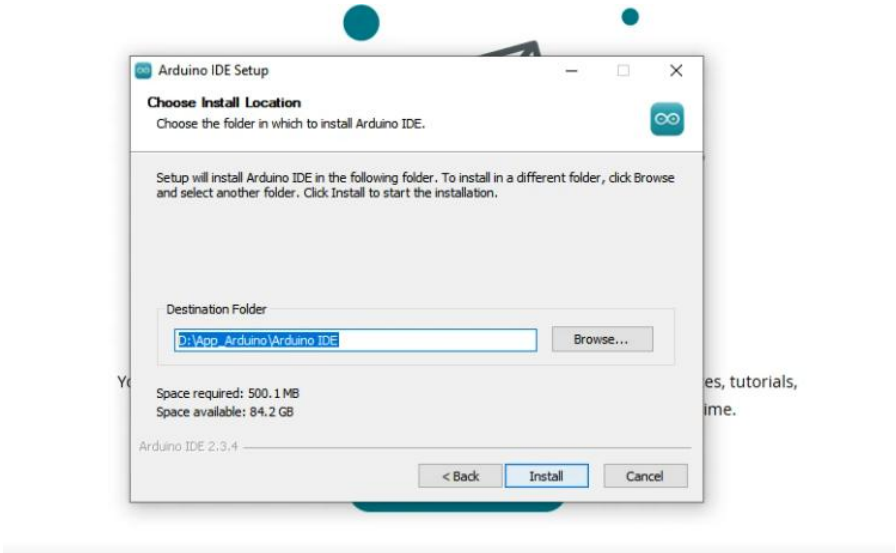
5.1 Accept the options



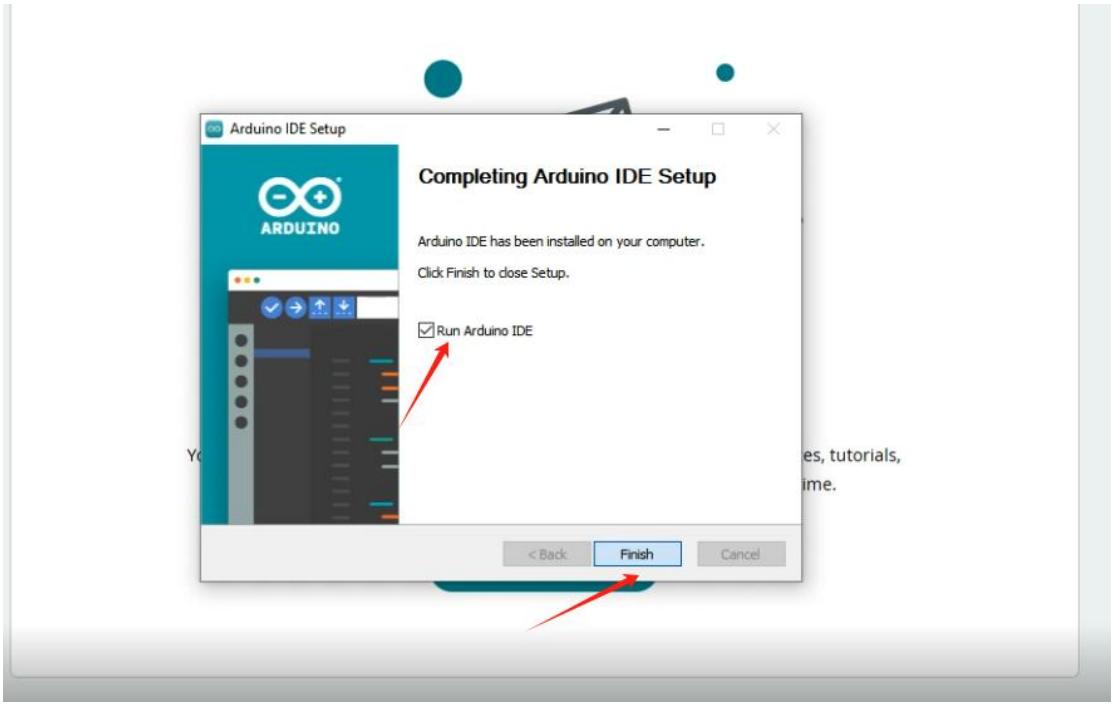
5.2 As shown in the figure, determine



5.3 Customize your path and click "Install"

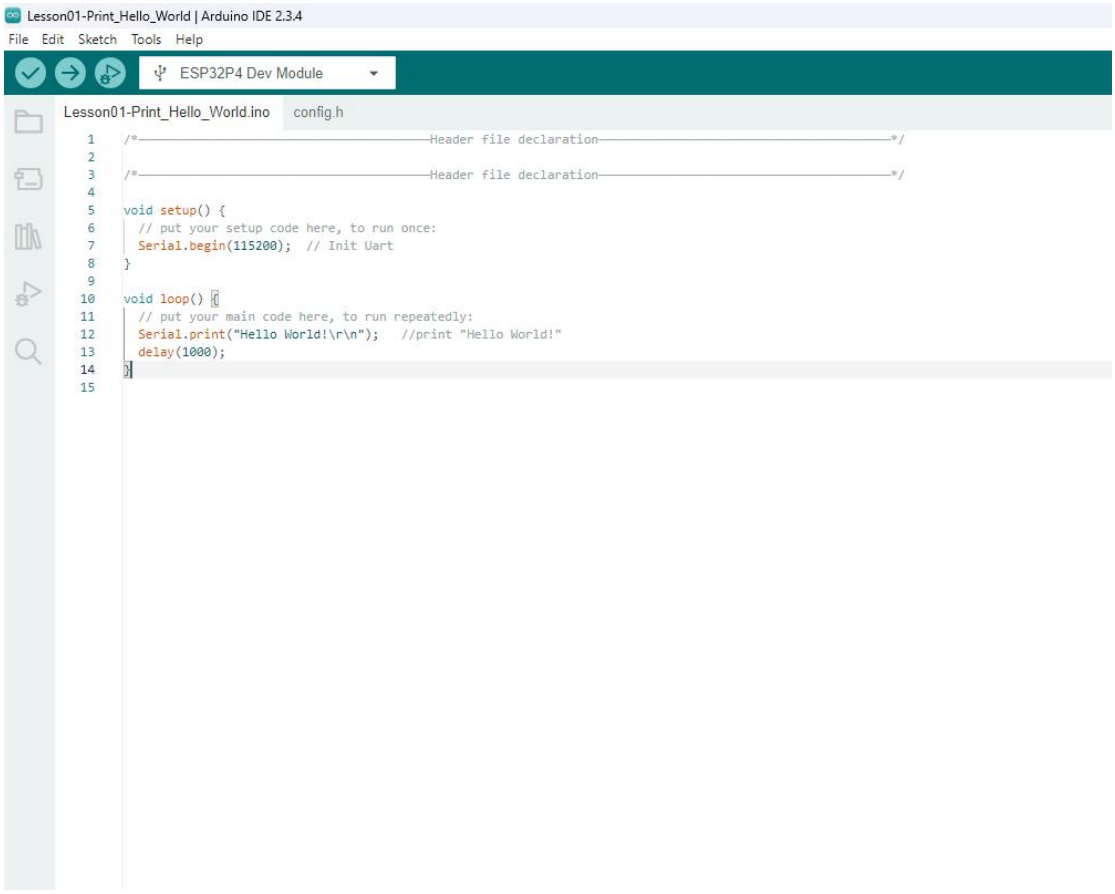


5.4 After the installation is completed, you can start using the Arduino IDE.

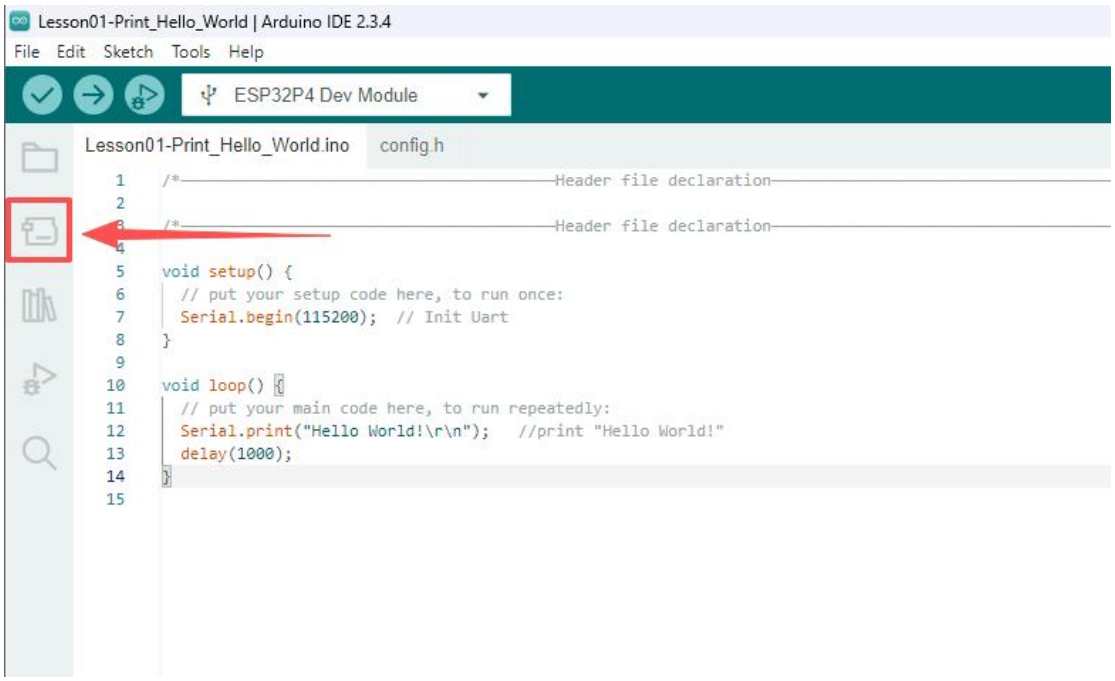


Installing the ESP32 Development Board

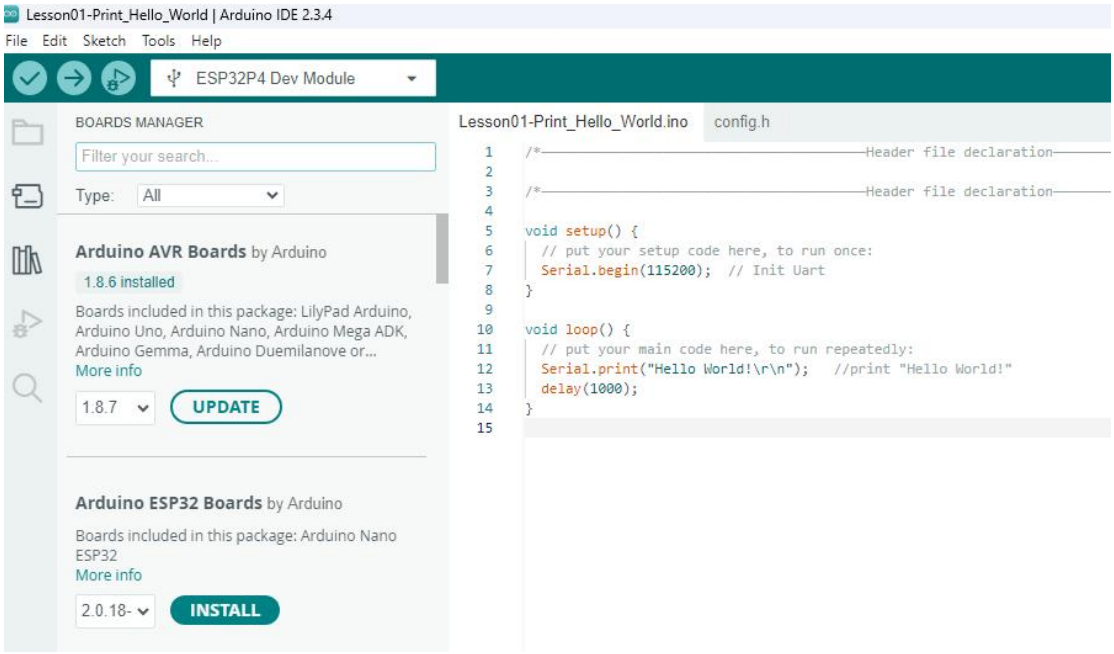
After the installation is completed, open the Arduino IDE.
First, randomly open a project.



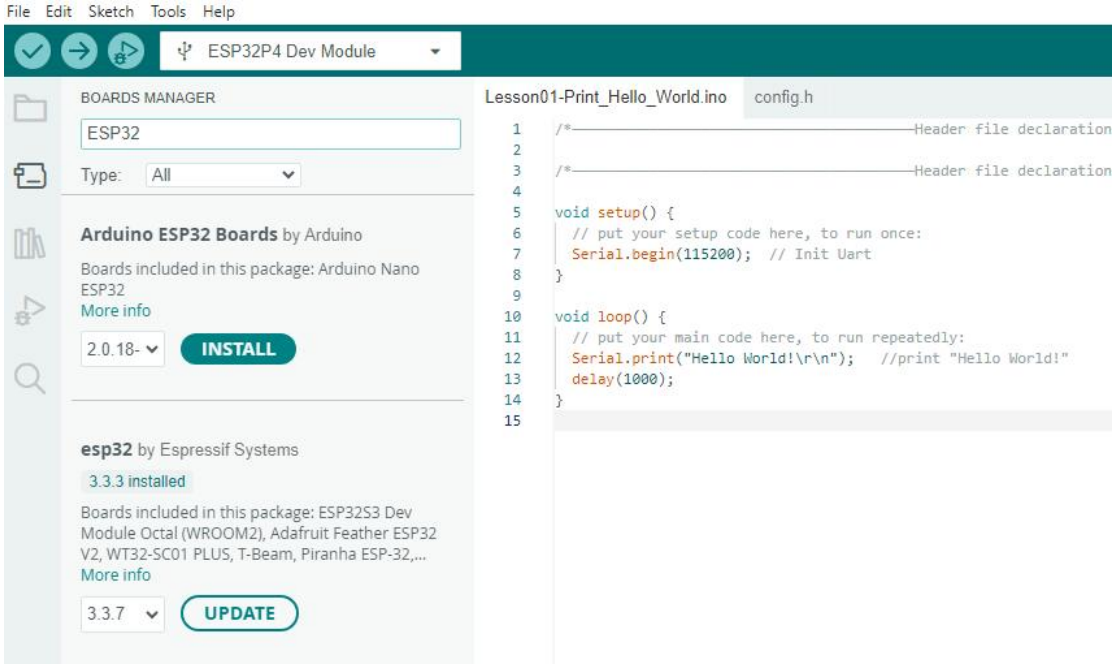
First, click on "board manage" on the left side.



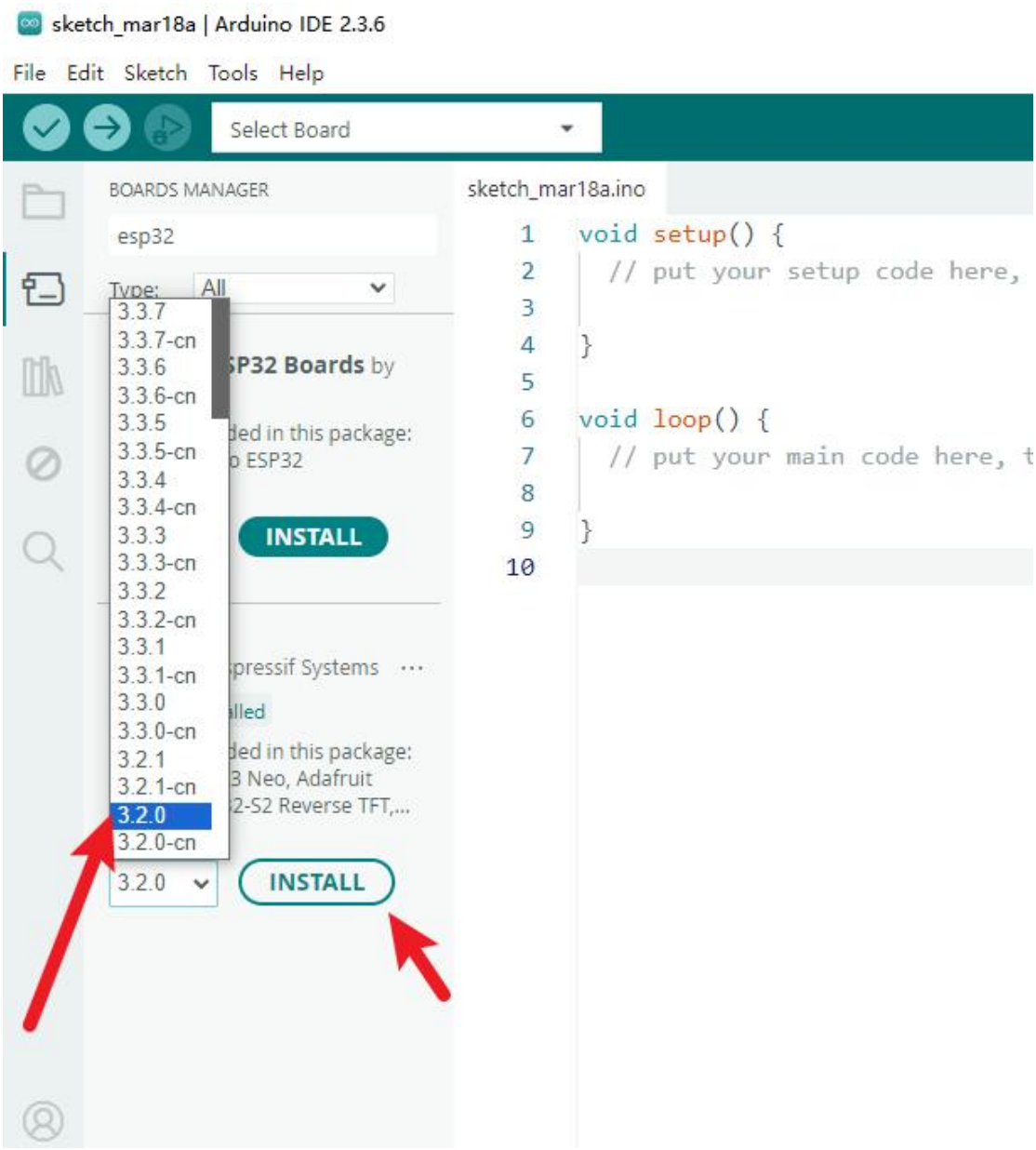
After opening



Search for ESP32 here



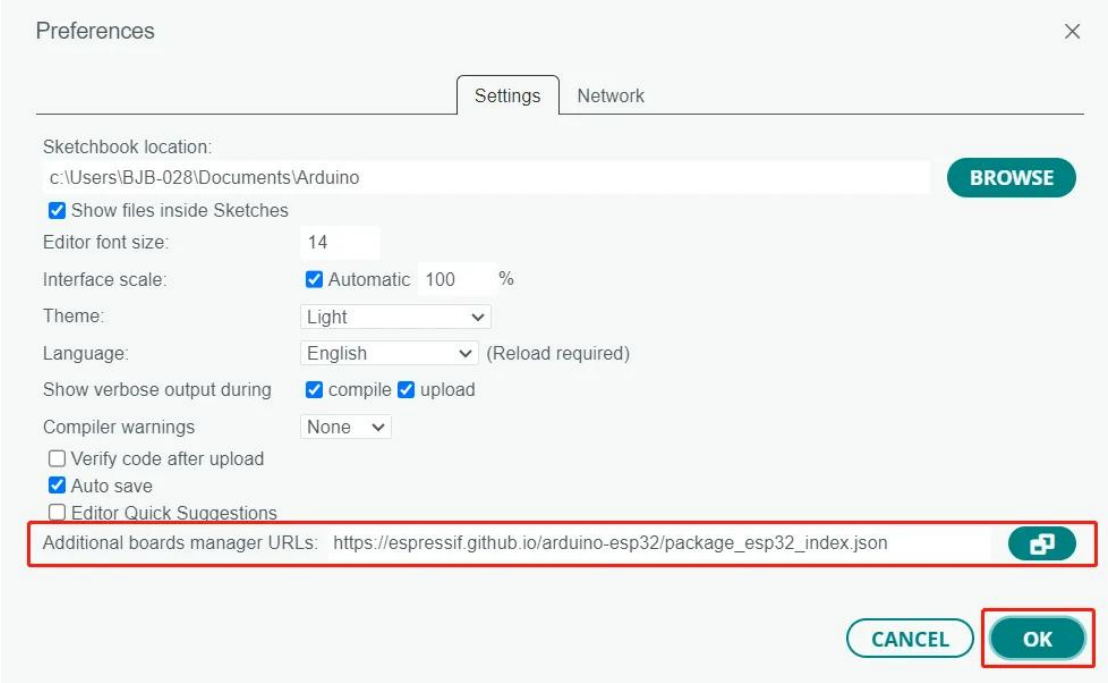
Select version 3.2.0 here and proceed with the installation.



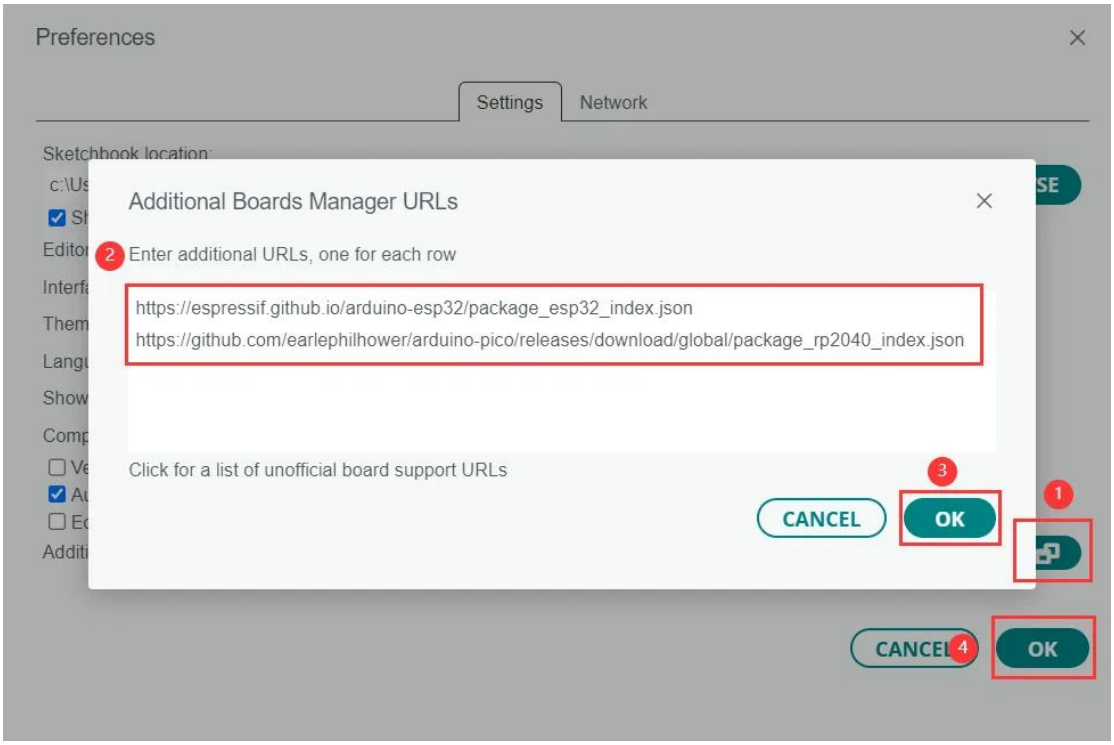
If there's no result for the manager of your development board model, you can add the development board manager URL in "Preferences", and then return to "Board Manager" to search for your model and install it.

ESP32: https://espressif.github.io/arduino-esp32/package_esp32_index.json

Click "File"→“Preferences”, then paste the URL to "Additional Boards Manager URLs" and click "OK".



If you need to add multiple URLs, click the icon on the right to open the pop-up window and add URLs. Different URLs need to be distinguished by line breaks or by comma, and then click OK.



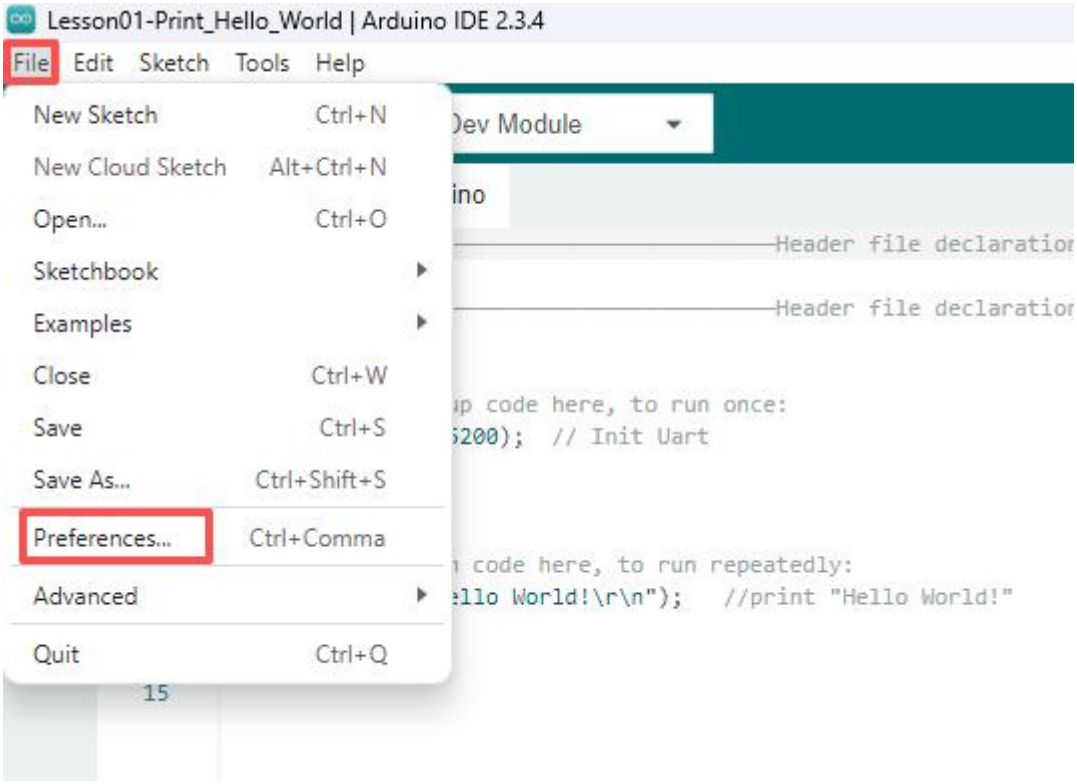
Importing Library Files (Important)

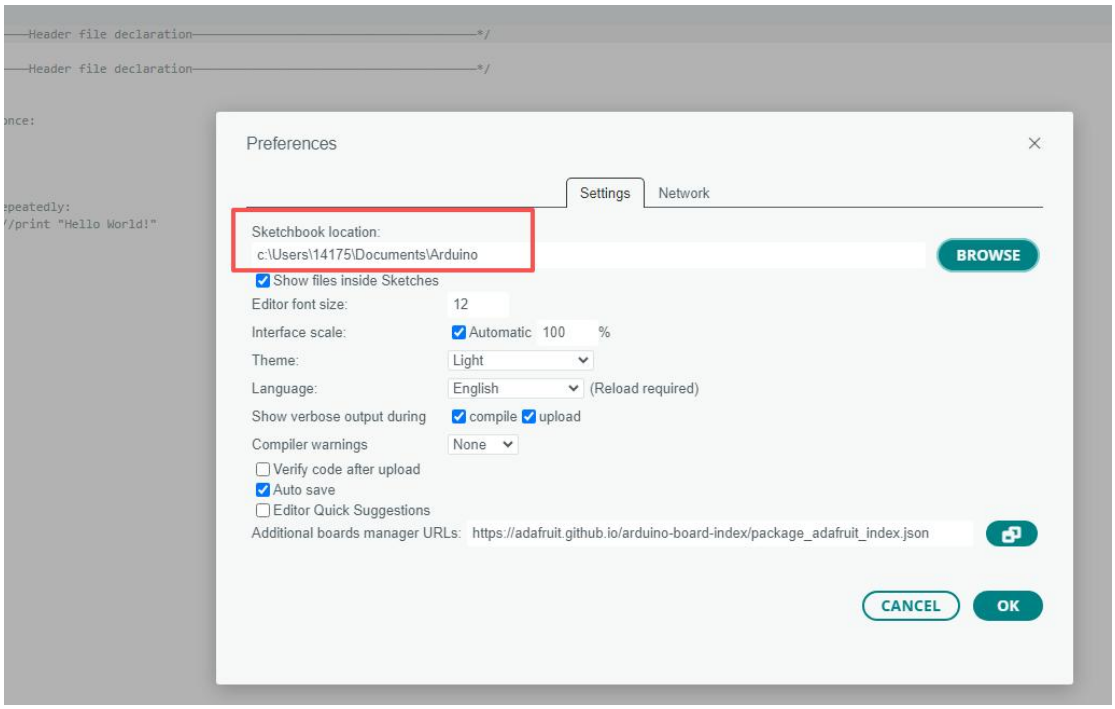
Before starting to use it, we need to first download the library files required for the

subsequent functions of the development board to the local machine and place them in the corresponding "/Arduino/libraries" directory on your computer.

Download link:
https://github.com/Elecrow-RD/AI_Camera_Development_Board_Vision_Sensor_Board_Powered_By_ESP32/tree/master/libraries

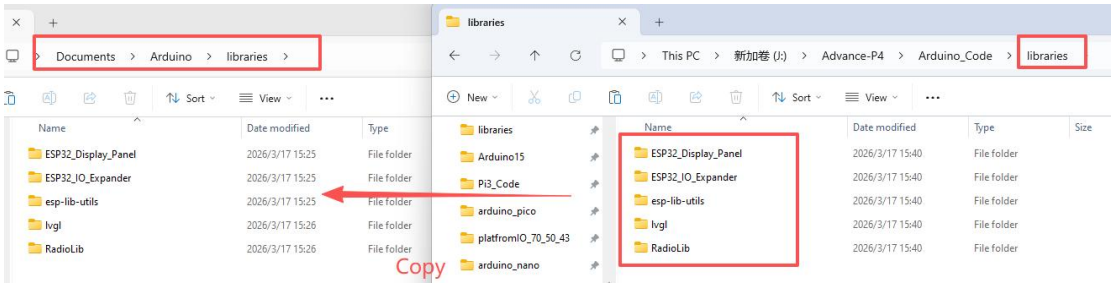
Enter this path and check the location where the library files are stored.





After determining the location where the library files are stored

Copy all the library files contained in the downloaded "libraries" file to the "C:\Users\Username\Documents\Arduino\libraries" folder on your local computer.



Note: If there is no "libraries" folder in your path, please create one yourself.

Lesson 1 — LED && Button

Overview

In this lesson, we will learn how to use the AW9523 I/O expansion chip. By using the I²C bus, we can expand the GPIO resources of the MCU and use the AW9523 to control the on and off state of LEDs. The course will focus on the basic working principle of the AW9523, register configuration methods, and how to output control signals to the expanded pins through software.

In practical applications, the AW9523 is often used for multi-channel LED control, I/O expansion, and peripheral driving. Therefore, this lesson will also explain the logic flow of output state control, helping you understand the complete implementation process from "I²C communication → expanded I/O control → LED output".

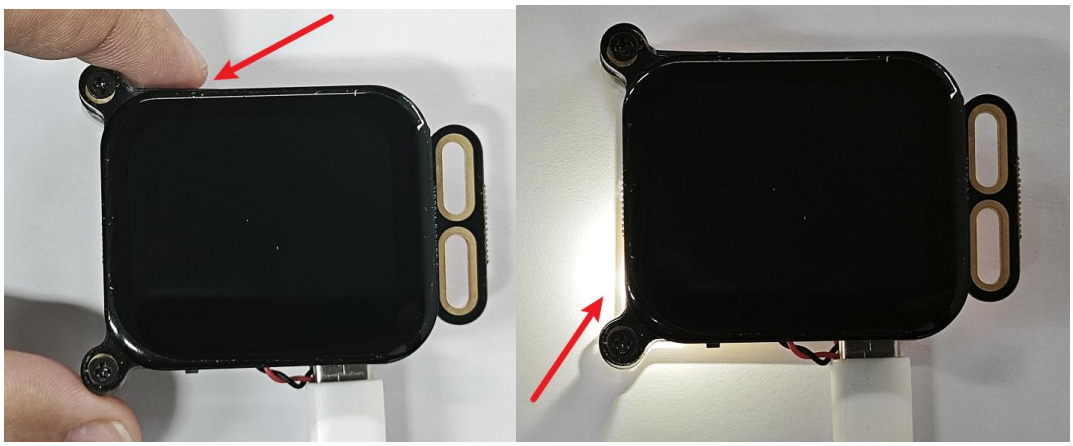
Through this lesson, you will master the basic method of using the AW9523 expansion chip to control LEDs, and finally achieve controlling LED on and off through software, laying the foundation for more complex multi-channel LED or peripheral control in subsequent lessons.

Learning Goals

1. Master the reading method of digital input (Button)
2. Understand and implement the button debounce (Debounce) principle
3. Learn button edge detection (trigger once per press)
4. Master the basic method of controlling LEDs with the AW9523 expansion chip and the concept of I2C
5. Establish a complete control logic of "Input → Judgment → Output (I²C expanded I/O)"

Project Result:

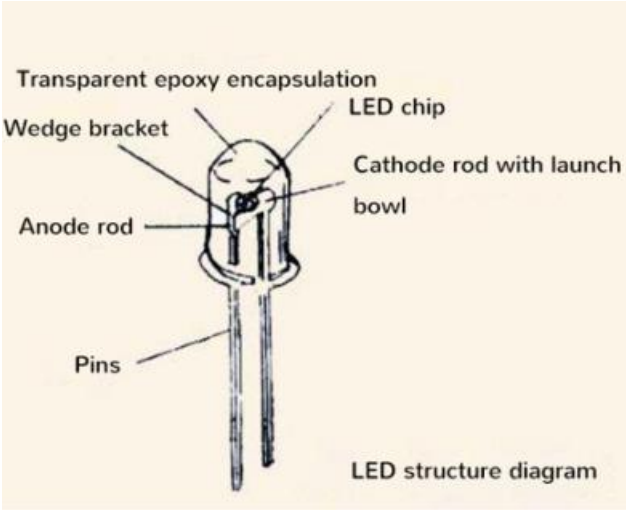
Press the button, the LED lights up.



一、I. Principle Explanation

1. Principle of LED

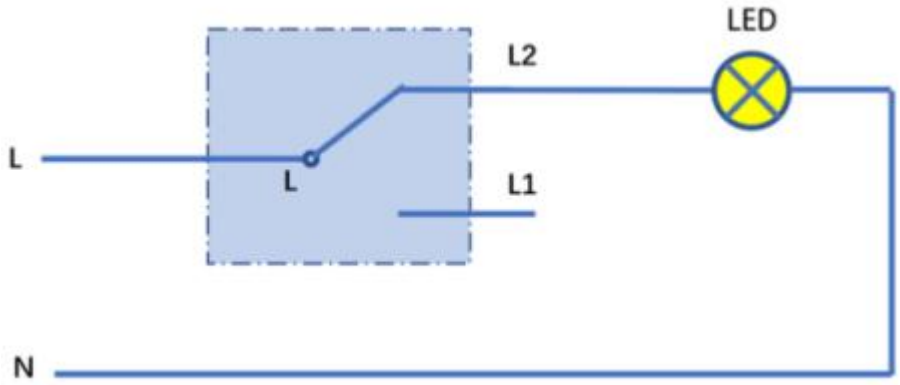
LED stands for Light Emitting Diode. It is a solid-state semiconductor device that can convert electrical energy into visible light, directly converting electrical energy into light energy. The core of an LED is a semiconductor chip. One end of the chip is fixed to a bracket and connected to the negative terminal of the power supply, while the other end is connected to the positive terminal. Usually, the longer pin is the positive terminal, and the shorter pin is the negative terminal.



2. Principle of Button

A button module is equivalent to a single-pole double-throw switch. As shown in the figure below: when the button is pressed, it is equivalent to the switch connecting to terminal L2, and the circuit is closed; when

the button is released, it is equivalent to the switch connecting to terminal L1, and the circuit is disconnected.



A button is essentially a mechanical switch. When pressed or released, it will generate short-term oscillation (bounce), causing the signal to switch back and forth between HIGH and LOW. This phenomenon is known as button bounce (Bounce). To avoid misjudgment by the program, debounce processing is required to ensure that the read button value is stable and reliable.

3. Introduction to AW9523 I/O Expansion Chip

The AW9523 is an I/O expansion chip based on I²C communication. Its main features include:

It can expand the number of GPIOs of the MCU;

It supports LED driving functions;

It controls the I/O output state through register configuration.

In this lesson:

The MCU sends control commands to the AW9523 via I²C;

The AW9523 is responsible for actually driving the LED on and off.

4. Introduction to I2C

What is I2C? I2C (Inter-Integrated Circuit) is a commonly used serial communication bus protocol, mainly used for microcontrollers (such as Arduino), various sensors, displays, memory chips, expansion modules, and more.

Its biggest feature is that multiple devices can be connected using only two signal lines.

The I2C bus requires only two signal lines:

Signal Line	Name	Function
SDA	Serial Data	Data line, used to transmit data
SCL	Serial Clock	Clock line, used to synchronize communication

In addition, each module also needs VCC (power) and GND (ground). A typical I2C device therefore usually uses four wires. On the I2C bus, each device has a unique address.

For example, Arduino is the Master, and BH1750 is the Slave. Arduino specifies which device to communicate with through the address.

In Arduino, I2C communication is implemented through the Wire library. Once initialized, all I2C devices can work normally.

```
#include <Wire.h>
static TwoWire* wi = &Wire;    // I2C bus
// Configure I2C pins
   wi->setPins(I2C_SDA, I2C_SCL);
   wi->begin();
   delay(100);
```

5. Introduction to ESP32-S3 Pins:

ESP32-S3 is a high-performance, low-power 32-bit MCU with rich and highly multiplexed GPIO resources, capable of meeting the requirements of display, communication, and human–machine interaction in various application scenarios.

Commonly used pins include:

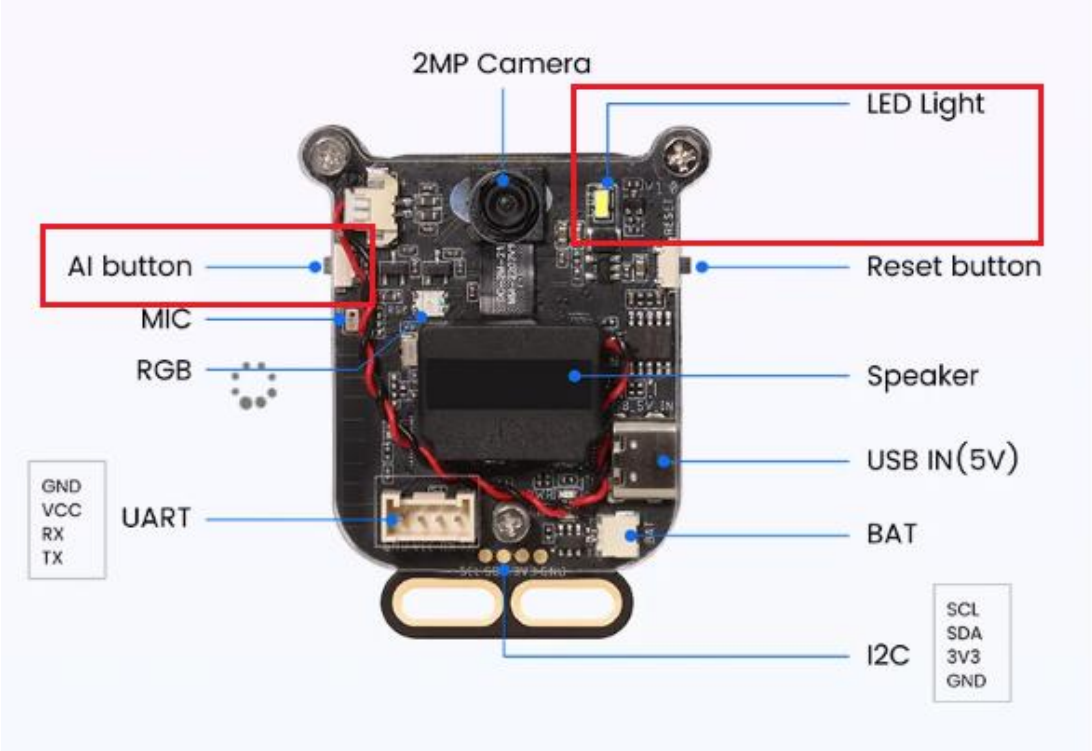
Digital Pins: used to read or output digital signals with only HIGH/LOW (1 or 0) states, which can control LEDs, buttons, relays, and other devices, such as the button and LED used here.

Analog Pins: used to read continuously changing analog signals, such as photoresistors and potentiometers, which are converted into values from 0 to 1023 through ADC.

I2C Pins: the I2C interface area, dedicated SDA (data) and SCL (clock) pins for I2C communication, used to connect multiple sensors or modules, such as the AW9523 chip on the AI camera.

II. Required Modules

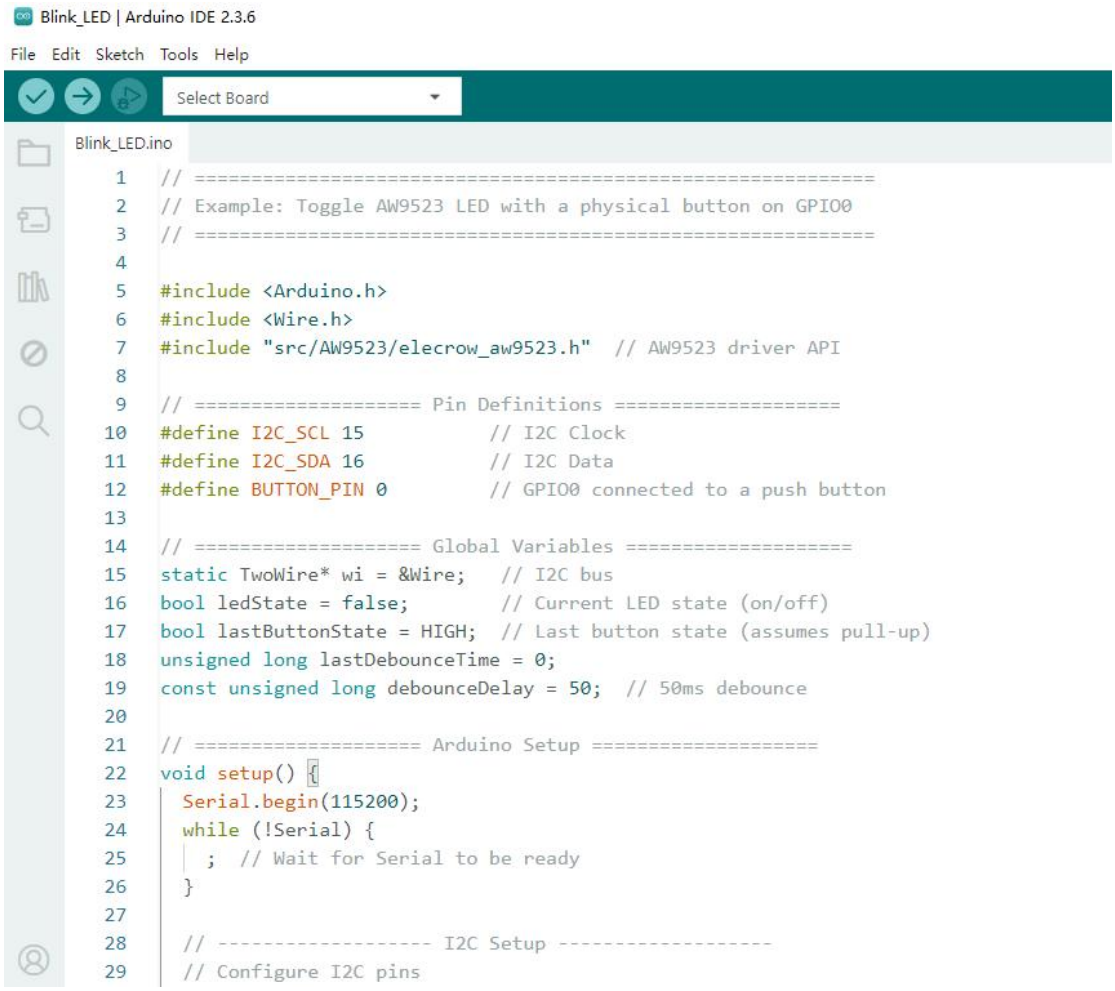
AI button and LED of the AI camera:



III. Example Explanation

Click the link to download the official example code: (GitHub link)

Open the course program using Arduino IDE:



Basic Arduino Code Knowledge:

(1) #define is used to create a name for a number or content, performing text replacement before compilation, making the program clearer and easier to maintain.

(2) setup() is the initialization function of an Arduino program.

After Arduino is powered on or reset, setup() is executed only once.

It is commonly used to set pin modes, initialize serial communication, and initialize sensors.

(3) The pinMode(pin, mode) function: pin mode setting function.

Pin: pin number, according to the hardware pin used, such as 2, 3, or 4,

Mode: mode, which can be INPUT / INPUT_PULLUP / OUTPUT;

Why set the mode?

Because each GPIO of the MCU can be:

INPUT: uses the pin as input, requiring external pull-up or pull-down resistors. The signal may float if unstable. It is used to read external signals, such as buttons and sensor outputs.

INPUT_PULLUP: input mode with internal pull-up to HIGH, which is the most convenient for reading buttons (pressed = LOW).

OUTPUT: uses the pin as output, actively outputting HIGH or LOW to control peripherals, such as LEDs, motor drivers, and relays.

(4) pinMode(BUTTON_PIN, INPUT_PULLUP) means setting BUTTON_PIN (pin 0) to input pull-up mode.

(5) loop() is the main loop function, which repeatedly executes from beginning to end.

When Arduino is running, loop() never stops.

This is the core execution structure of Arduino.

(6) digitalRead(PIN) is a function that reads the state of a digital pin.

Pin: pin number, such as pin 0;

The returned value is HIGH or LOW;

(7) delay(ms): delay function;

The parameter unit is milliseconds (ms). delay(1000) represents 1 second. During delay(), the program pauses execution.

delay(500): pauses the program for 500 milliseconds (0.5 seconds).

Code Explanation

(1) First, add the necessary libraries.

```
#include <Arduino.h>
#include <Wire.h>
#include "src/AW9523/elecrow_aw9523.h"
```

Arduino.h: Arduino core header file, providing basic functions (pinMode, digitalRead, millis(), etc.);

Wire.h: Arduino I²C communication library, used for communication between the MCU and AW9523

elecrow_aw9523.h: AW9523 driver library, encapsulating chip initialization and LED control interfaces

(2) Define digital pins for easier reading.

```
#define I2C_SCL 15
#define I2C_SDA 16
#define BUTTON_PIN 0
```

The purpose of #define I2C_SCL 15 is to give the pin number a “name”.

I2C_SCL represents digital pin 15, I2C_SDA represents digital pin 16, and BUTTON_PIN represents digital pin 0. This indicates that the I2C signal lines are connected to pins 15 and 16, and the button is connected to pin 0.

The advantage of writing it this way is:

If you want to move the module or other hardware to different pins later, you only need to modify the pin numbers here. The rest of the code does not need to be changed, making the program clearer and easier to maintain.

```
bool ledState = false;
bool lastButtonState = HIGH;
unsigned long lastDebounceTime = 0;
const unsigned long debounceDelay = 50;
```

ledState: records the current LED state;

lastButtonState: records the current button state;

lastDebounceTime: the time point when the button state last changed;

debounceDelay: debounce delay of 50 ms.

(3) Serial Initialization

```
Serial.begin(115200);
while (!Serial) {
  ;
}
```

Enable serial communication for debugging and outputting button / LED status information.

(4) I²C and AW9523 Initialization

```
wi->setPins(I2C_SDA, I2C_SCL);
wi->begin();
delay(100);
aw9523.AW_init();
aw9523.AW_set_POWER(true);
```

Specify the SDA / SCL pins used by I²C, start the I²C bus, and add a short delay to ensure bus stability. Initialize the AW9523 chip and enable the LED power domain so that the LED can be controlled.

(4) Button Initialization

```
pinMode(BUTTON_PIN, INPUT_PULLUP);
```

Use the internal pull-up resistor. The default reading is HIGH, and it reads LOW when the button is pressed.

(5) Read Button State and Detect State Changes

When the button is pressed, short-term bounce occurs, and the program may mistakenly think the button was pressed multiple times.

Solution:

```
bool reading = digitalRead(BUTTON_PIN);
if (reading != lastButtonState) {
   lastDebounceTime = millis();
}
```

If the current state is not equal to the previous state, it means the button state has just changed and may be bouncing.

Record the current time as the debounce timing starting point.

```
if ((millis() - lastDebounceTime) > debounceDelay) {
```

Only when the state remains stable for more than 50 ms is the button input considered reliable.

(6) Edge Detection and LED Toggle Control

```
static bool buttonPressed = false;
if (reading == LOW && !buttonPressed) {
   buttonPressed = true;
   ledState = !ledState;
   aw9523.AW_set_SLED(ledState);
}
```

If the current state is LOW (pressed) and it has not been triggered before, this indicates a new button press event.

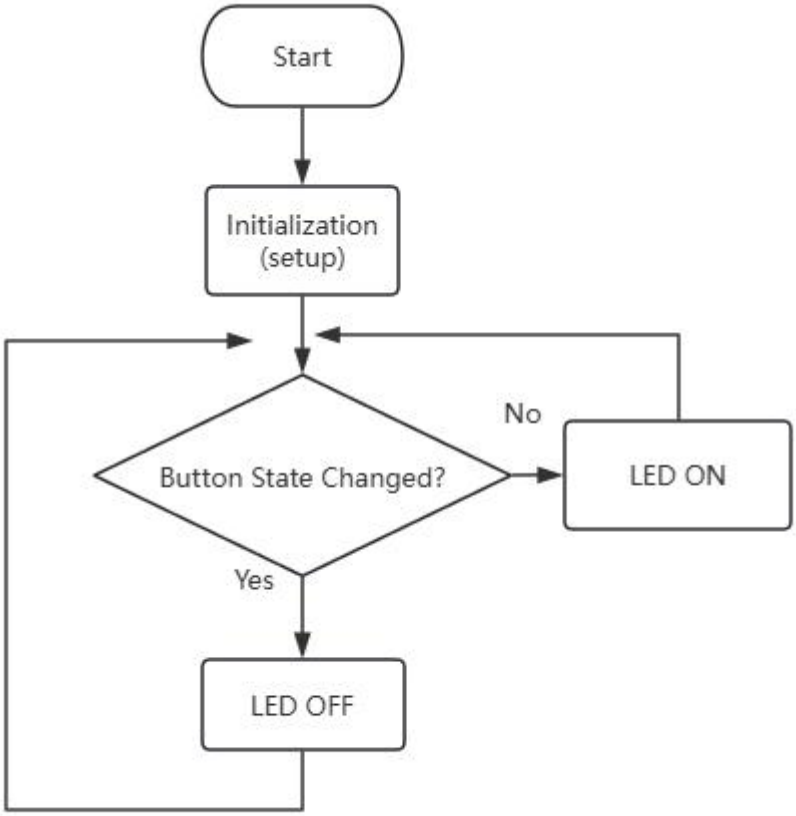
Each press toggles the LED state once, and the AW9523 is controlled via I²C to change the LED on/off state.

```
else if (reading == HIGH) {
   buttonPressed = false;
}
```

```
lastButtonState = reading;   // save the last reading
```

Clear the flag to allow the next press to trigger again, preparing for the next press.

Program Flowchart:



IV. Download and Run the Program, Observe the Result

(1) Connect the board to the computer according to the Arduino IDE operation process introduced in the preface, and perform the basic download setup.

Click Download:

Blink_LED | Arduino IDE 2.3.6

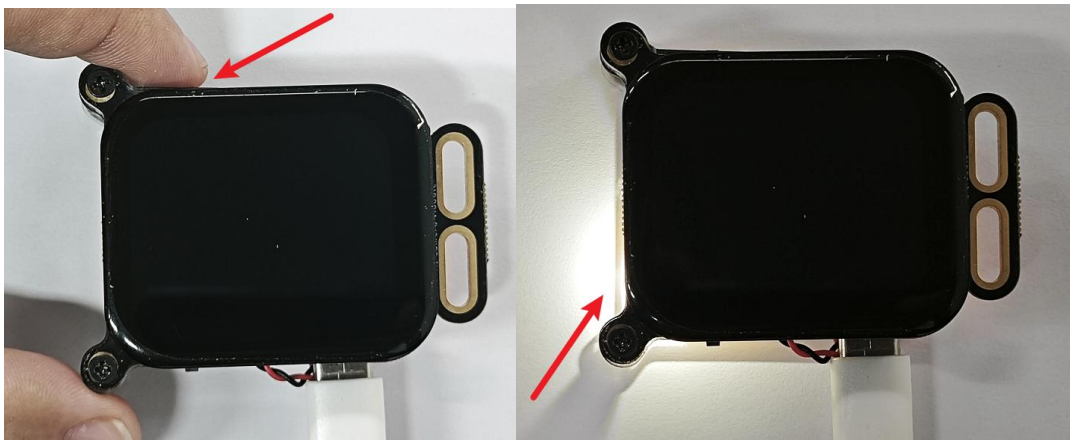
File Edit Sketch Tools Help

ESP32S3 Dev Module Upload

Upload

```
// ============================================================
// Example: Toggle AW9523 LED with a physical button on GPIO0
// ============================================================

#include <Arduino.h>
#include <Wire.h>
#include "src/AW9523/elecrow_aw9523.h"  // AW9523 driver API

// ==================== Pin Definitions ====================
#define I2C_SCL 15           // I2C Clock
#define I2C_SDA 16           // I2C Data
#define BUTTON_PIN 0         // GPIO0 connected to a push button

// ==================== Global Variables ====================
static TwoWire* wi = &Wire;   // I2C bus
bool ledState = false;        // Current LED state (on/off)
bool lastButtonState = HIGH;  // Last button state (assumes pull-up)
unsigned long lastDebounceTime = 0;
const unsigned long debounceDelay = 50;  // 50ms debounce

// ==================== Arduino Setup ====================
void setup() {
  Serial.begin(115200);
  while (!Serial) {
    ;  // Wait for Serial to be ready
  }
```

When the button is pressed, the LED turns on; when the button is pressed again, the LED turns off.



Lesson 2 — RGB LED

Overview

In this lesson, we will further study the application of the AW9523 I/O expansion chip in controlling RGB LED lights.

Through the I²C bus, the MCU can not only control the on/off state of a single LED but also achieve RGB color mixing, color gradients, and dynamic lighting effects via the AW9523.

This lesson will focus on:

The basic principle of RGB LED illumination

The concept of color mixing (Red / Green / Blue)

How to generate rainbow colors and smooth gradient effects through programming

The complete control process: MCU → I²C → AW9523 → RGB LED

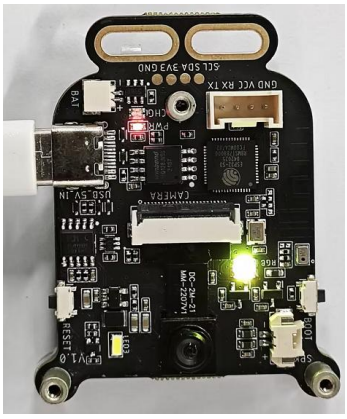
Through this lesson, you will master the basic methods of using AW9523 to control RGB lights and implement dynamic lighting effects, laying the foundation for subsequent lighting designs, status indicators, and human-machine interface applications.

Learning Objectives

1. Understand the working principle of RGB LEDs and the method of color mixing
2. Master the basic representation of 24-bit RGB color data
3. Learn how to use AW9523 to control RGB LED output
4. Understand the logic of implementing color fading and rainbow lighting effects
5. Establish a complete process of “program algorithm → I²C communication → RGB light output”

Project Effect:

The RGB light automatically displays rainbow gradient effects, with the lighting modes cycling continuously to create a dynamic RGB ambient light effect.



二、Principle Explanation

1. Principle of RGB LED

An RGB LED consists of red (R), green (G), and blue (B) LED chips packaged together. By controlling the brightness ratio of the three colors, almost all visible colors can be mixed.

R	G	B	Display Color
255	0	0	Red
0	255	0	Green
0	0	255	Blue
255	255	255	White
255	255	0	Yellow

In this lesson, the brightness range for each color channel is 0 ~ 255.

2. Color Mixing and 24-bit RGB

In programming, a complete RGB color is usually represented by 24-bit data:

0xRRGGBB:

High 8 bits: Red (R)

Middle 8 bits: Green (G)

Low 8 bits: Blue (B)

For example:

RGB(255, 0, 0) → Red, RGB(0, 255, 0) → Green, RGB(0, 0, 255) → Blue.

3. Role of AW9523 in RGB Control

AW9523 is an I/O expansion chip based on I²C communication, with LED driving capability.

In this lesson:

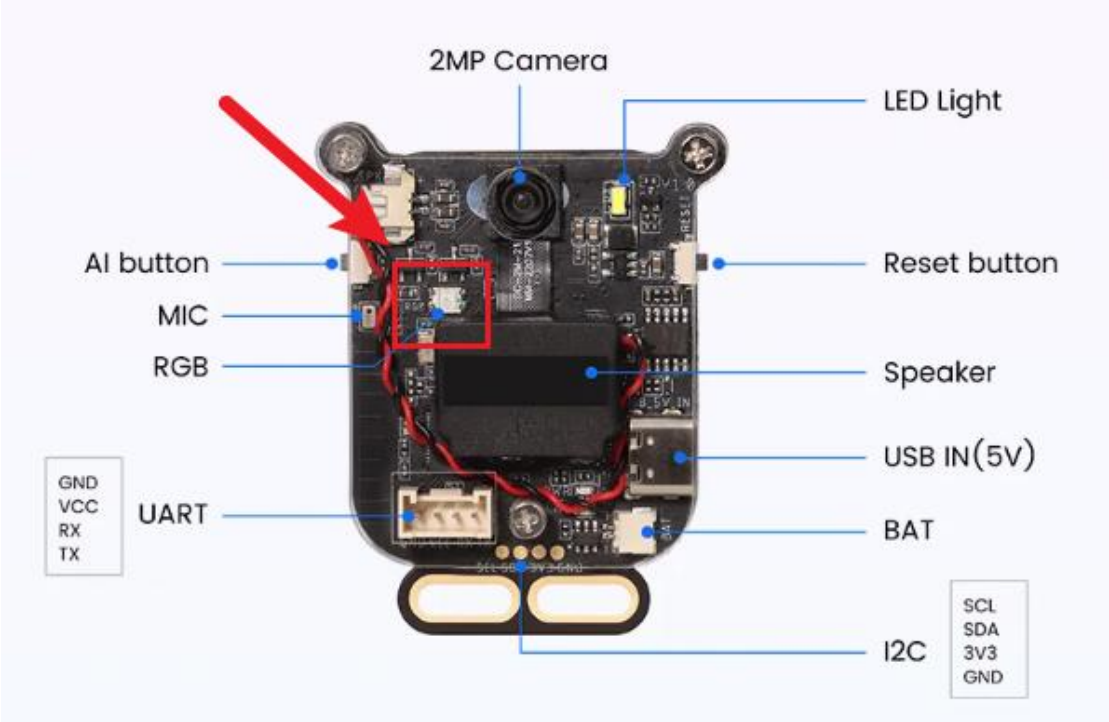
The MCU is responsible for calculating color data and animation logic;

The MCU sends RGB data to the AW9523 via I²C;

AW9523 drives the RGB LED to display the corresponding color according to the received data.

Required Modules

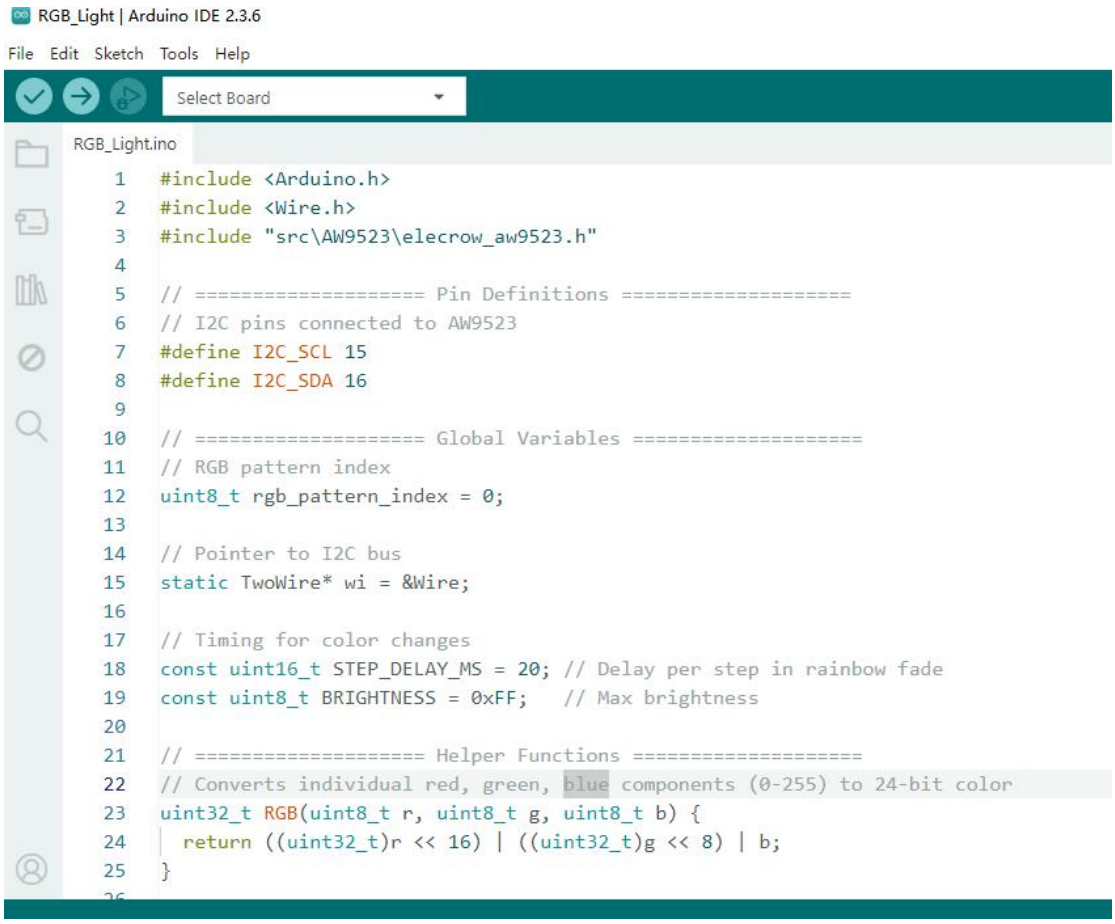
RGB light of Ai camera:



Example Explanation

Click the link to download the official example code: (GitHub link)

Open the course program with Arduino IDE:



Code Explanation

(1) First, include the necessary libraries.

```
#include <Arduino.h>
#include <Wire.h>
#include "src/AW9523/elecrow_aw9523.h"
```

Arduino.h: Core Arduino header file, providing basic functions (pinMode, digitalRead, millis(), etc.);

Wire.h: Arduino I²C communication library, used for communication between MCU and AW9523

elecrow_aw9523.h: AW9523 driver library, encapsulating chip initialization and LED control interfaces

(2) Define digital pins and variables for readability.

```
#define I2C_SCL 15
#define I2C_SDA 16
```

I2C_SCL 15 assigns a “name” to the pin number.
I2C_SCL represents digital pin 15, I2C_SDA represents digital pin 16, indicating that the I2C signal lines are connected to pins 15 and 16.
The advantage of writing this way is:
If you want to move the module or other hardware to different pins later, you only need to modify the pin numbers here, without changing the rest of the code, making the program clearer and easier to maintain.

```
uint8_t rgb_pattern_index = 0;
```

Used to record the current RGB light effect mode:

0 → Rainbow gradient mode;

1 → RGB smooth cycle mode.

```
static TwoWire* wi = &Wire;
```

Define an I²C bus pointer to conveniently specify SDA/SCL pins and initialize I²C later.

```
const uint16_t STEP_DELAY_MS = 20;
const uint8_t BRIGHTNESS = 0xFF;
```

STEP_DELAY_MS: Delay for each step of color change (20ms);

BRIGHTNESS: Maximum brightness (not directly used in the current example, reserved for extension).

(3) RGB Color Composition Function

```
static TwoWire* wi = &Wire;
uint32_t RGB(uint8_t r, uint8_t g, uint8_t b) {
   return ((uint32_t)r << 16) | ((uint32_t)g << 8) | b;
}
```

The function combines three 8-bit R/G/B values into a single 24-bit RGB color value.

For example:

RGB(255, 0, 0) → Red;

RGB(0, 255, 0) → Green;

RGB(0, 0, 255) → Blue.

(4) Color Fade Function

```
void fadeColor(uint32_t fromColor, uint32_t toColor, uint16_t steps, uint16_t delayMs)
```

The function performs a smooth transition between the starting color and the target color.

```
int r1 = (fromColor >> 16) & 0xFF;
int g1 = (fromColor >> 8) & 0xFF;
int b1 = fromColor & 0xFF;
```

Split the RGB components, decomposing the 24-bit color into R/G/B channels.

```
uint8_t r = r1 + (r2 - r1) * i / steps;
```

Interpolation calculation (core of fading), using mathematical interpolation to change colors gradually, avoiding sudden jumps.

```
aw9523.AW_set_RGB(RGB(r, g, b));
```

Send RGB data to AW9523 via I²C, letting AW9523 drive the LED display.

(5) Rainbow Color Generation Function

```
uint32_t rainbowColor(uint8_t step)
```

The function automatically generates continuously changing rainbow colors based on the step (0~255).

Principle:

0 ~ 85: Red → Green;

85 ~ 170: Green → Blue;

170 ~ 255: Blue → Red.

This is a common simplified HSV → RGB rainbow algorithm.

(6) Initialization Function setup()

```
void setup(void) {
  Serial.begin(115200);
  while (!Serial);
```

Open serial debugging with a baud rate of 115200.

(7) I²C and AW9523 Initialization

```
bool reading = digitalRead(BUTTON_PIN);
wi->setPins(I2C_SDA, I2C_SCL);
wi->begin();
delay(100);
```

```
aw9523.AW_init();
aw9523.AW_set_POWER(true);
```

Specify I²C pins and start the I²C bus. Initialize the AW9523 chip and turn on the LED power so the RGB LED can be controlled.

(8) Main Loop loop()

```
switch (rgb_pattern_index)
```

Use a switch structure to execute different lighting effects based on the current mode.

Mode 1: Rainbow Gradient

```
for (uint8_t i = 0; i < 255; i++) {
  aw9523.AW_set_RGB(rainbowColor(i));
  delay(STEP_DELAY_MS);
}
```

The RGB light shows a continuous flowing rainbow effect with smooth and natural color changes.

Mode 2: RGB Smooth Cycle Rainbow Gradient

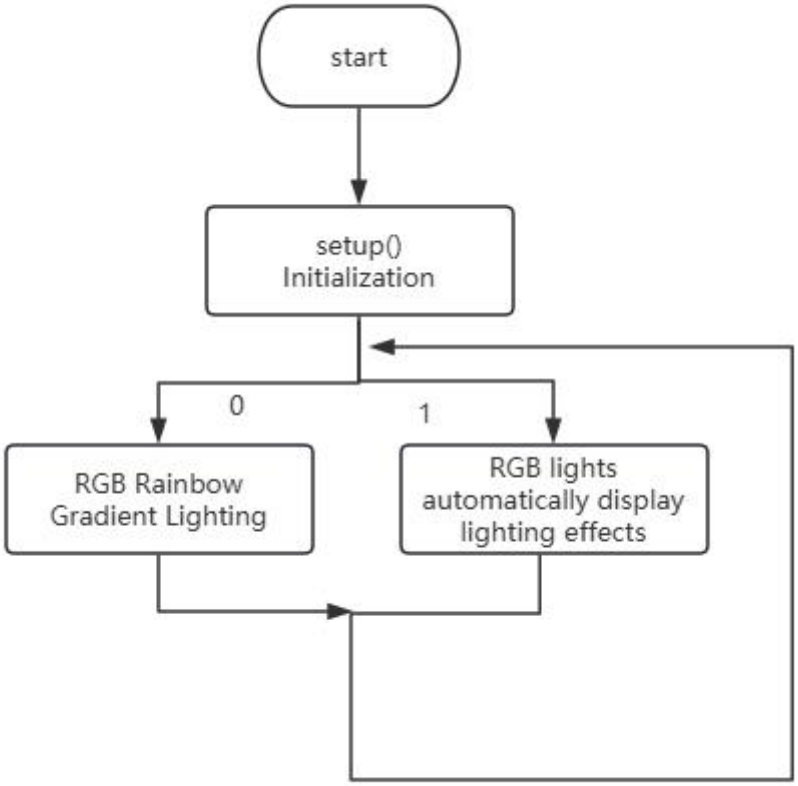
```
fadeColor(RGB(255, 0, 0), RGB(0, 255, 0), 50, STEP_DELAY_MS);
fadeColor(RGB(0, 255, 0), RGB(0, 0, 255), 50, STEP_DELAY_MS);
fadeColor(RGB(0, 0, 255), RGB(255, 0, 0), 50, STEP_DELAY_MS);
```

Displays a smooth gradient from Red → Green → Blue → Red without flickering.

```
rgb_pattern_index = (rgb_pattern_index + 1) % 2;
```

Switch the lighting mode, cycling between the two effects. % 2 ensures the mode value is always 0 or 1.

Program Flowchart:



Download and Run the Program, Observe the Effect

(1) Connect to the computer and perform basic download settings according to the Arduino IDE operation steps introduced in the preface.

Click Download:

RGB_Light | Arduino IDE 2.3.6

File Edit Sketch Tools Help

ESP32S3 Dev Module Upload

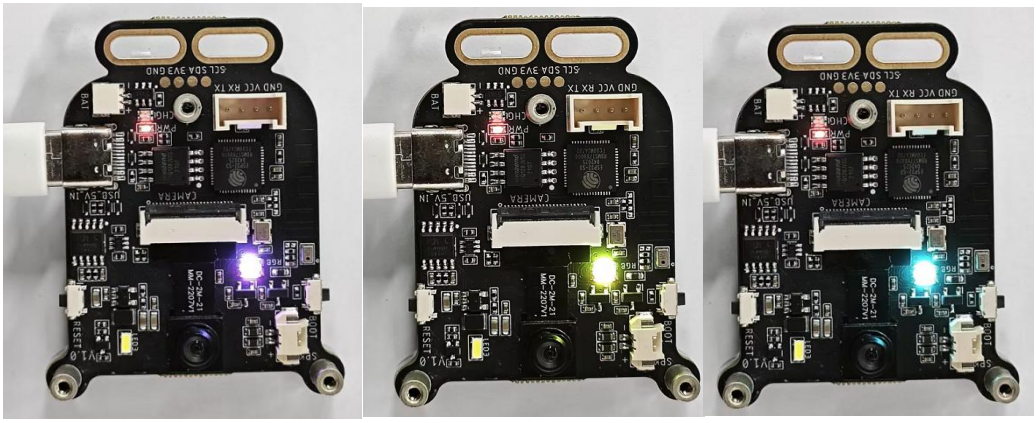
Upload

RGB_Light.ino

```
#include <Arduino.h>
#include <Wire.h>
#include "src\AW9523\elecrow_aw9523.h"

// ==================== Pin Definitions ====================
// I2C pins connected to AW9523
#define I2C_SCL 15
#define I2C_SDA 16

// ==================== Global Variables ====================
// RGB pattern index
uint8_t rgb_pattern_index = 0;

// Pointer to I2C bus
static TwoWire* wi = &Wire;

// Timing for color changes
const uint16_t STEP_DELAY_MS = 20; // Delay per step in rain
const uint8_t BRIGHTNESS = 0xFF;   // Max brightness

// ==================== Helper Functions ====================
// Converts individual red, green, blue components (0-255) t
uint32_t RGB(uint8_t r, uint8_t g, uint8_t b) {
```

Output

The RGB light automatically displays rainbow gradient effects, with the lighting modes cycling continuously.



Lesson 3 — SCREEN

Overview

This lesson focuses on the circular screen on the AI Camera development board and systematically explains the complete process from screen driving to graphical interface display. The course content includes understanding the basic working principles of the circular LCD screen, using SquareLine Studio to design a visual UI interface, porting the generated project to an ESP32 + LVGL environment, and using LVGL to display and switch interfaces. At the same time, this lesson will implement a simple second-level interface example: clicking a button on the main interface to enter the second-level interface and display the “Elecrow” information. Through this lesson, you will master the complete development process from UI design → screen driving → LVGL display → interface switching, laying a foundation for more complex graphical interface applications.

Learning Objectives

1. Understand the basic working principles of the circular LCD screen;
2. Learn how to use SquareLine Studio to design UI;
3. Master LVGL initialization and interface display;
4. Implement a second-level interface switching example;
5. Be able to understand and modify the example code.

Project Running Effect:

Click the screen to enter the second-level page.

I. Principle Explanation

1. Button principle and circular LCD screen working principle

The circular screen is essentially a TFT LCD display with an integrated display controller (such as ST7789). The MCU sends commands and pixel data to the screen via the SPI bus, and the controller refreshes the display according to the data content.

LVGL as the upper-level graphics library:

Responsible for managing widgets (buttons, text, etc.);

Drawing interface elements into the buffer;

Writing pixel data to the screen through display driver functions.

Overall structure:

LVGL → LovyanGFX → SPI → LCD Controller → Screen

SPI is the abbreviation of Serial Peripheral Interface, which is a commonly used high-speed synchronous serial communication protocol mainly used for data communication between microcontrollers (MCUs) and peripherals (such as displays, Flash, sensors, etc.).

2. Pin Explanation

A. LCD Screen (SPI)

Function	GPIO
SCLK	14

MOSI	13
DC	9
CS	10
RST	NC

Pin Description:

SCLK (Clock): SPI communication clock signal, used to synchronize data transmission.
MOSI (Data Line): The line through which the master controller sends display data to the screen.
DC (Data/Command Select): Used to distinguish whether the current transmission is a “command” or “display data”.
CS (Chip Select): Used to select the current SPI device, active low.
RST (Reset): Performs a hardware reset on the screen controller. In this example, software reset is used, so it is not connected.

B. Touch Chip CST816S (I2C)

Function	GPIO
SDA	16
SCL	15
INT	11
I2C address	0x15

Pin Description:
SDA (Data Line): I2C bus data transmission line.
SCL (Clock Line): I2C bus clock signal.
INT (Interrupt): Outputs an interrupt signal when touch is detected, used to notify the main controller to read touch coordinates.
I2C Address: Used to distinguish different devices on the bus. The address of this touch chip is 0x15.

II. Required Modules

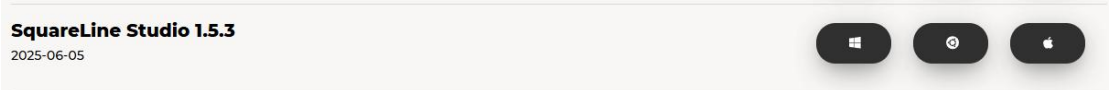
AI Camera display screen

III. Example Explanation

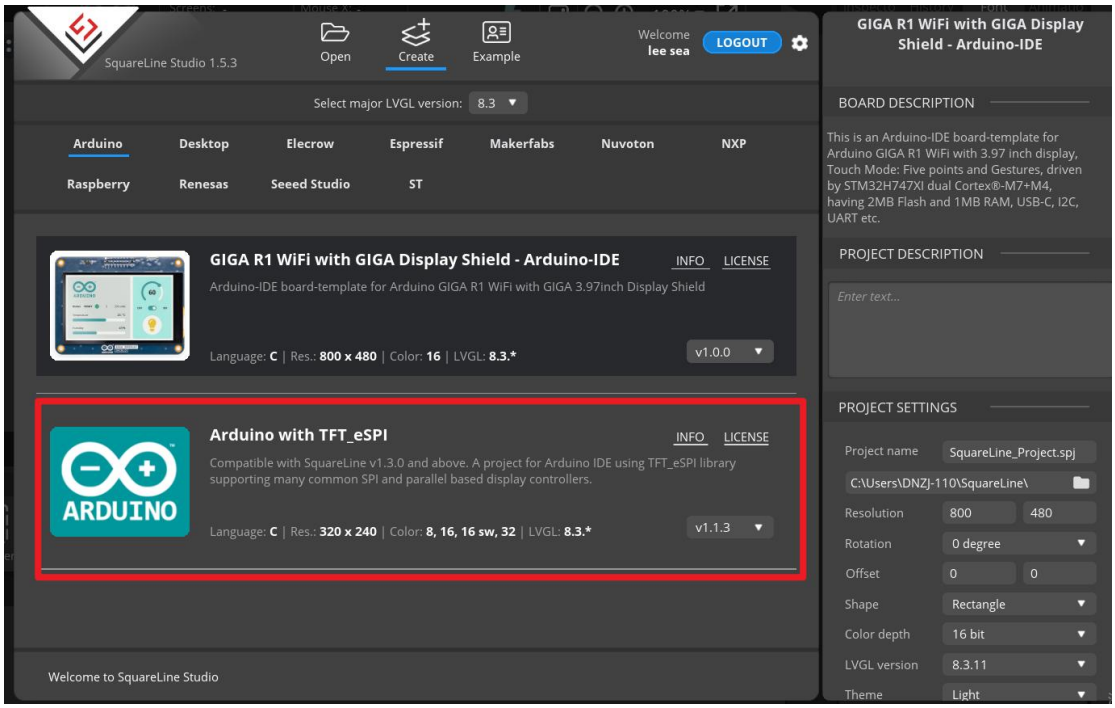
Click the link to download the officially provided example code and UI project files: (GitHub link)

A. Use SquareLine Studio to design UI

SquareLine Studio version 15.3 is used here for design. Please use the corresponding version. For installation guidance, refer to this link: https://www.elecrow.com/wiki/Get_Started_with_SquareLine_Studio.html



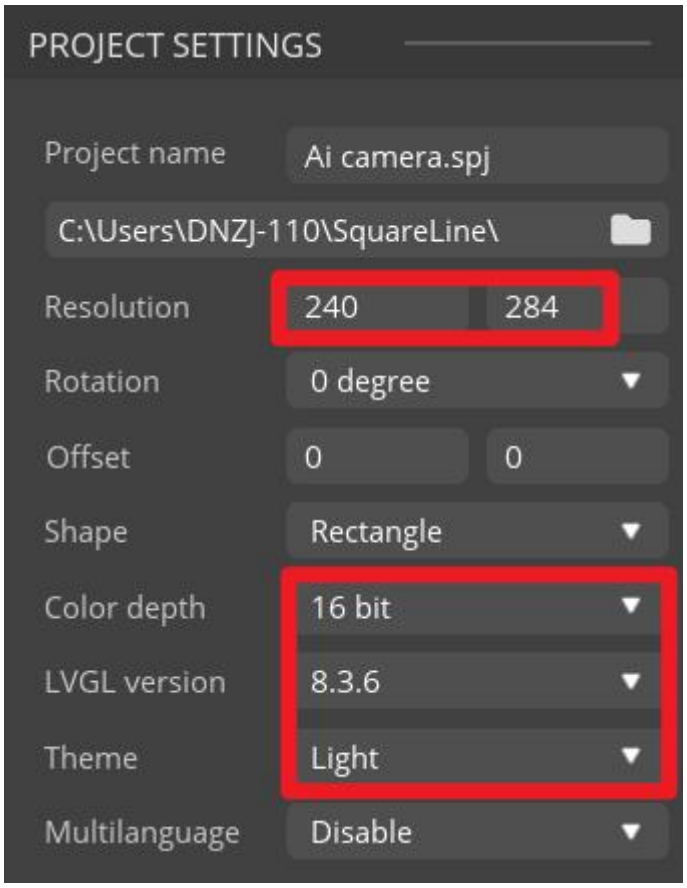
1. Open SquareLine Studio and create a project. Select “Arduino” → “Arduino with TFT_eSPI”.



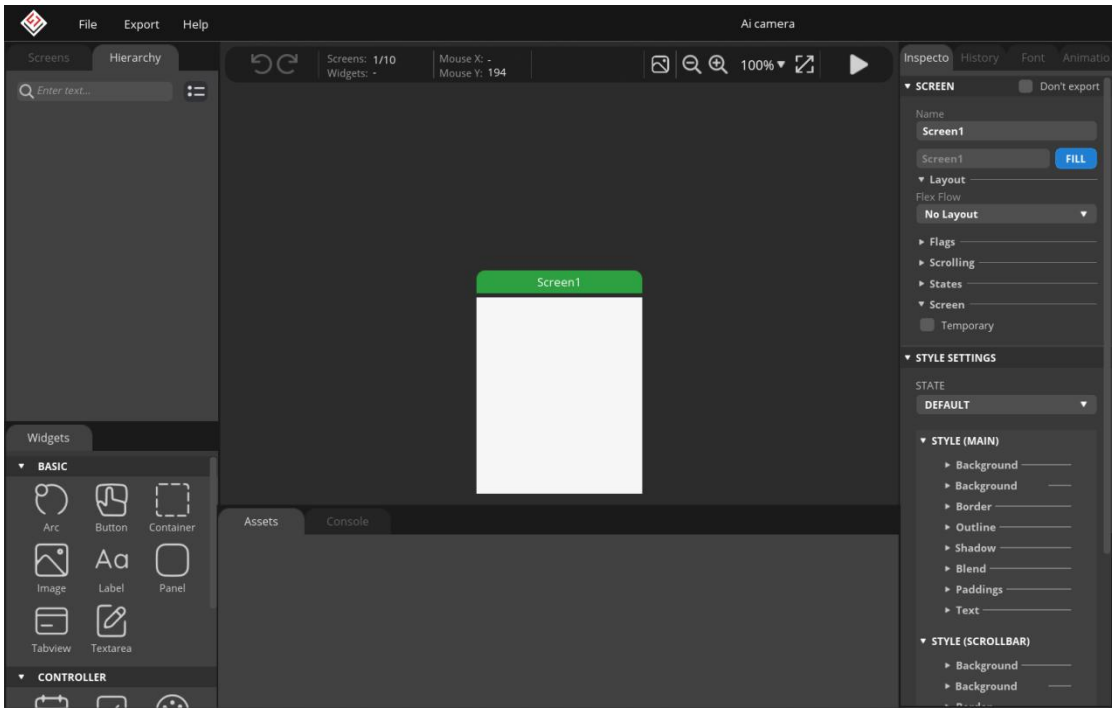
When selecting the Arduino framework, there is only one option, “Arduino with TFT_eSPI”. After selecting this option, SquareLine Studio will generate template code suitable for the TFT_eSPI library. However, SquareLine Studio not only supports the TFT_eSPI library, but also supports multiple libraries to adapt to different hardware and application requirements, such as the Adafruit_GFX library and LovyanGFX. After generating the UI code using SLS, we use different graphics libraries according to different hardware and modify the corresponding code to display the content you designed.

2. Set the project name, set the screen resolution to 240*284, set the color depth to 16-bit, and keep other default settings. After completing the settings, click the Create button to create the project.

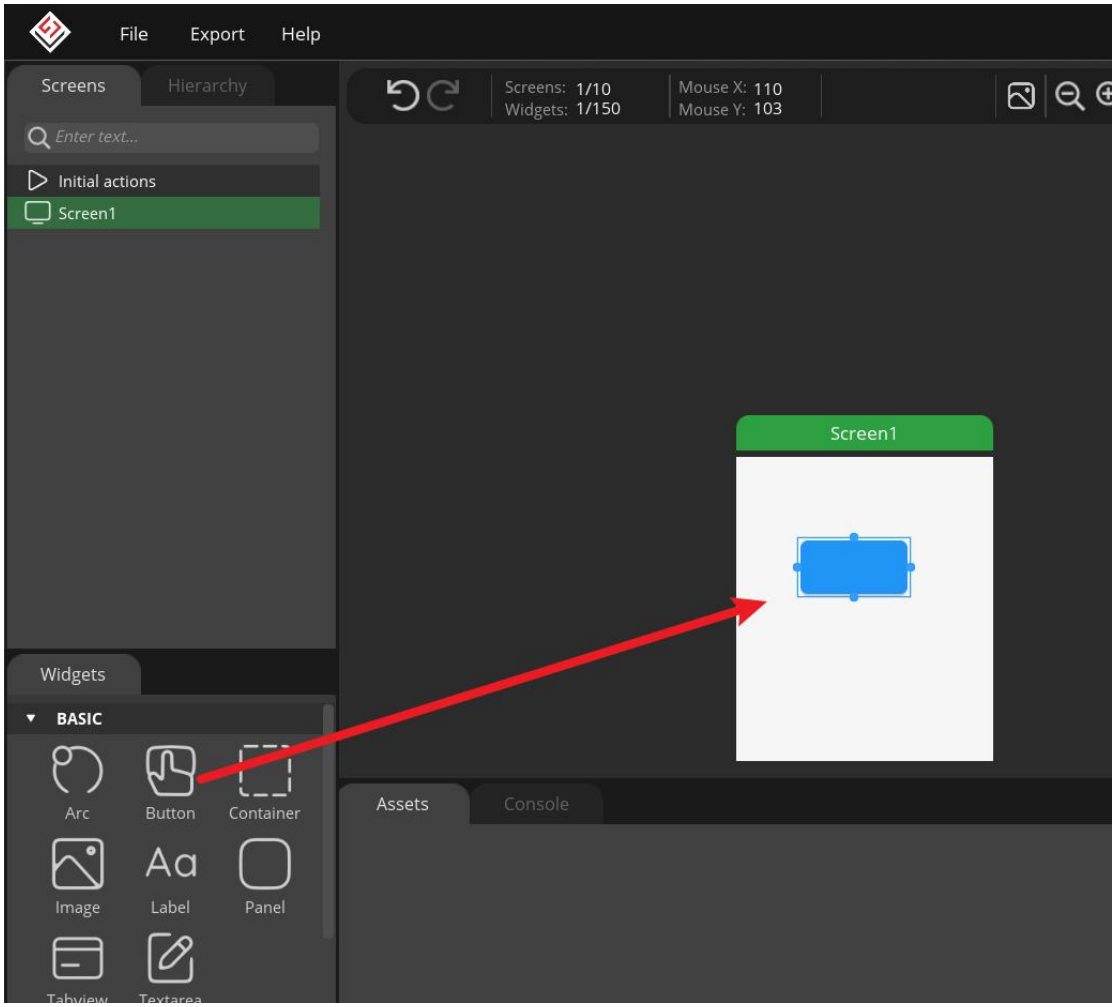
16-bit color depth: 65,536 colors can be represented using RGB 5:6:5 sub-pixel representation. That is, each RGB channel uses 5 bits and 6 bits respectively, for a total of 16 bits, to represent colors.



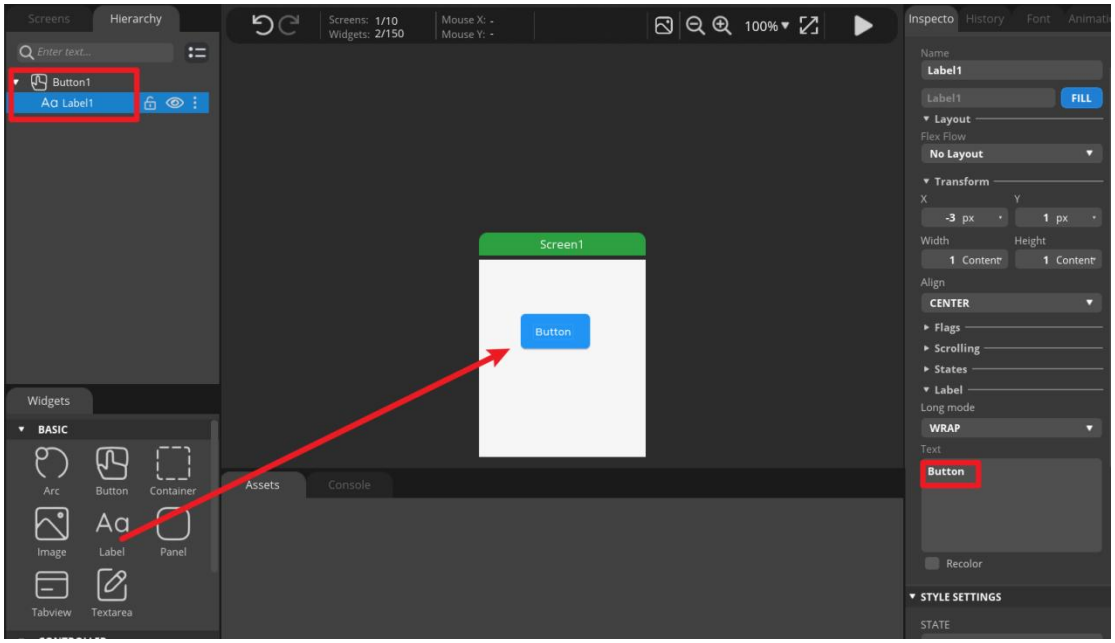
3. After creation, enter the following blank background interface.



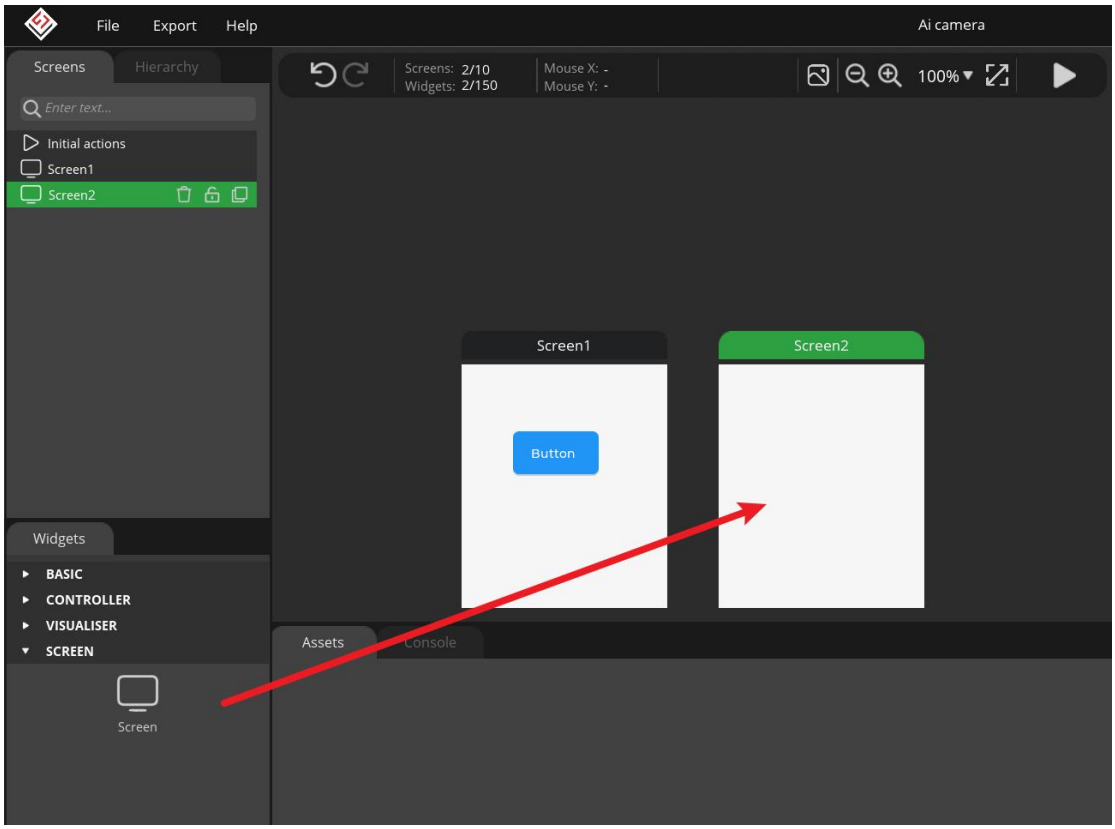
4. Click Button to add a button on Screen1.

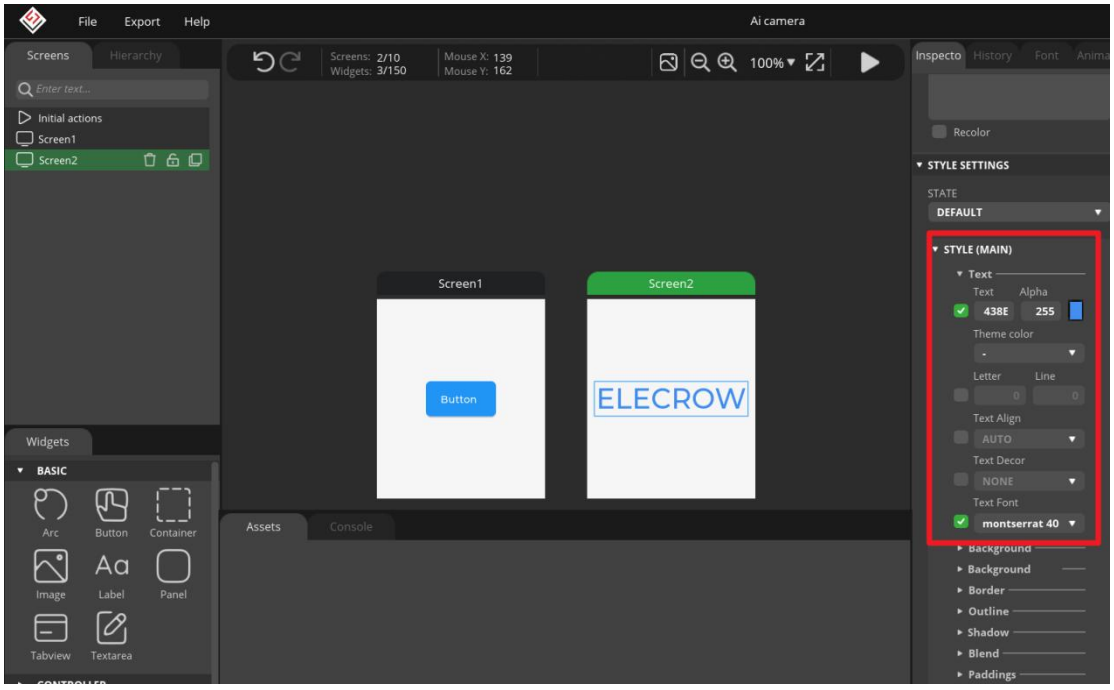


5. Add a label on top of the button and name it Button.

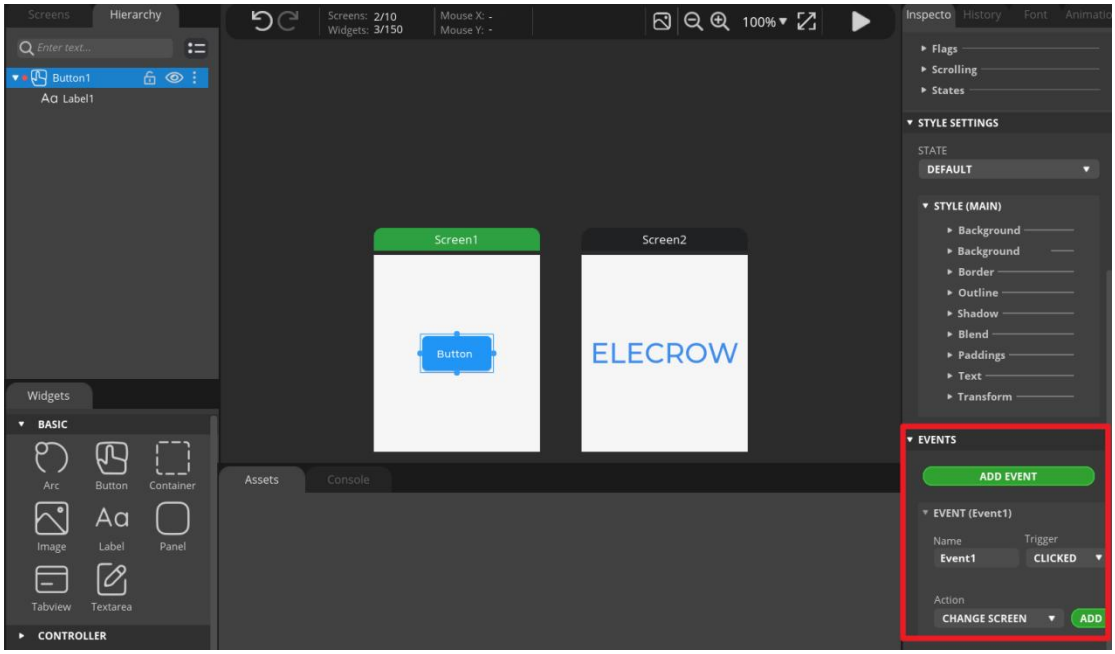


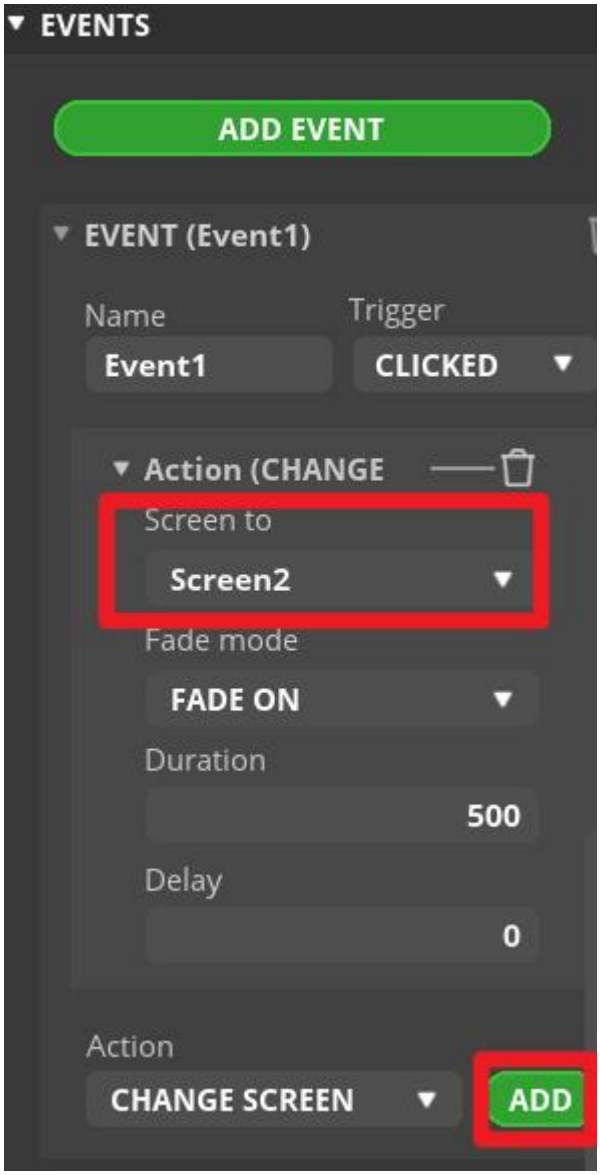
6. Add a new Page 2 and add some display content on Page 2.



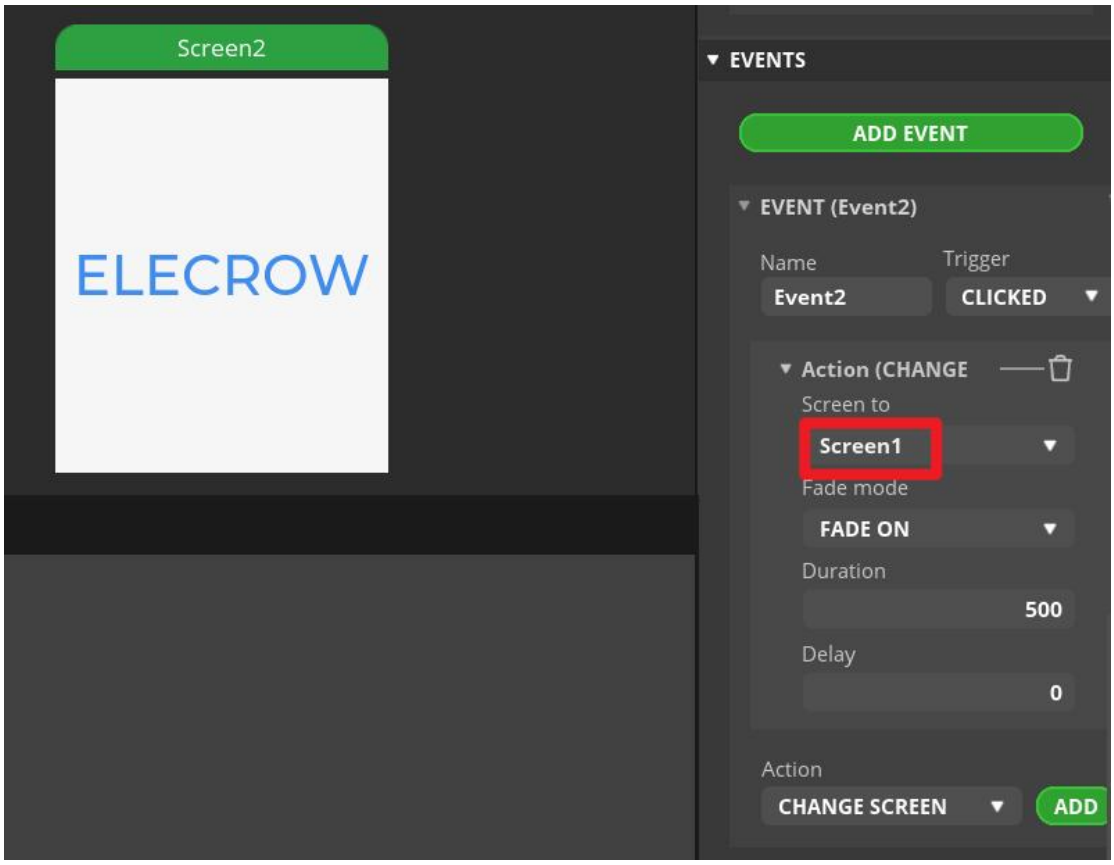


7. Set the button event so that when the button is pressed, it jumps to the second page.

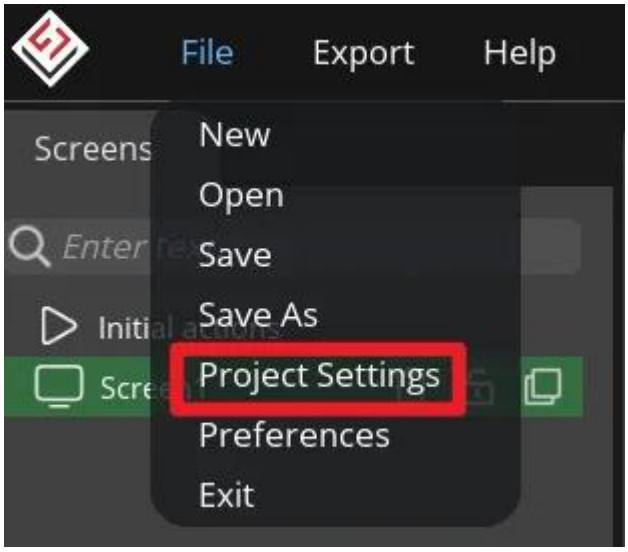

Screens
Hierarchy
Button1
Label1
Widgets
BASIC
Arc
Button
Container
Image
Label
Panel
Tabview
Textarea
CONTROLLER
Screens: 2/10
Widgets: 3/150
Mouse X: -
Mouse Y: -
100%
Screen1
Button
Screen2
ELECROW
Assets
Console
Inspecto
History
Font
Flags
Scrolling
States
STYLE SETTINGS
STATE
DEFAULT
STYLE (MAIN)
Background
Background
Border
Outline
Shadow
Blend
Paddings
Text
Transform
EVENTS
ADD EVENT
EVENT (Event1)
Name
Event1
Trigger
CLICKED
Action
CHANGE SCREEN
ADD


EVENTS
ADD EVENT
EVENT (Event1)
Name
Event1
Trigger
CLICKED
Action (CHANGE
Screen to
Screen2
Fade mode
FADE ON
Duration
500
Delay
0
Action
CHANGE SCREEN
ADD

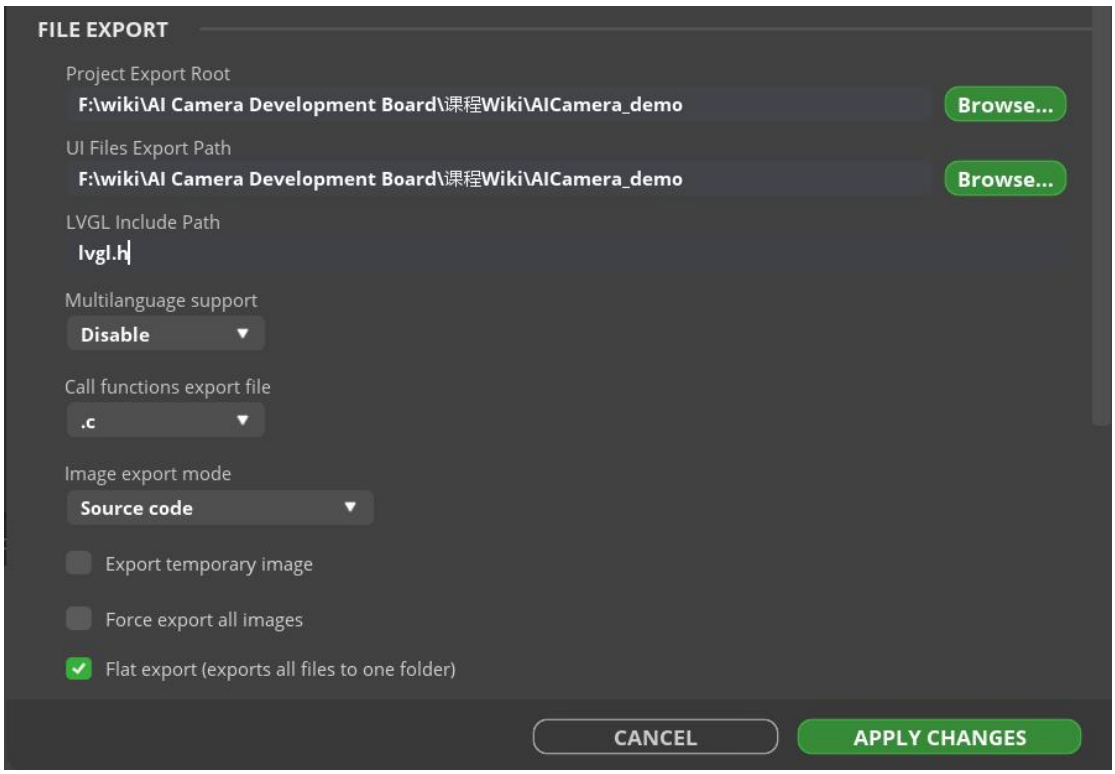
8. Also add a page jump on Page 2, and click it to return to Page 1.



Click “File” → “Project Settings” and configure the exported files.



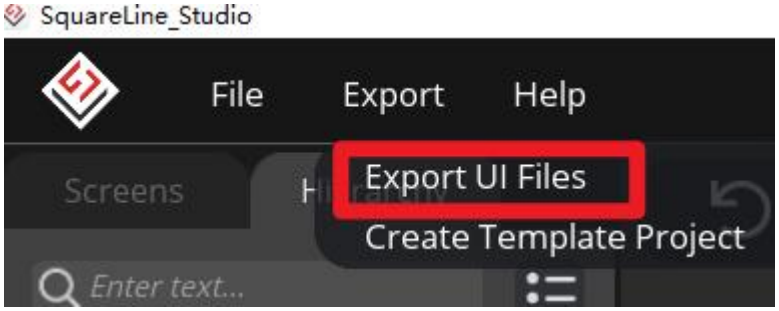
Set the export path of the files according to your own file configuration.

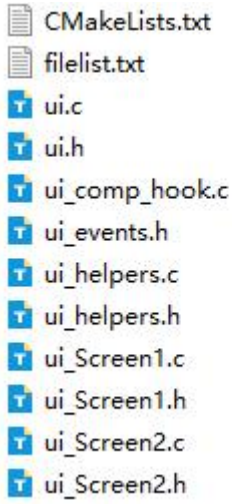


Enter lvgl.h in the LVGL Include Path. Check “Flat export (export all files to one folder)”.

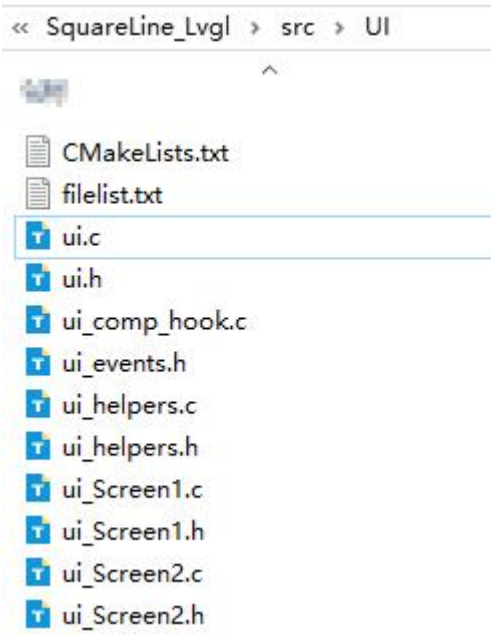
Note: After selecting flat export, the output files will be located in the same folder, so there is no need to modify paths in the program. If not selected, the output files will be categorized and placed in different folders, and the compiler may not recognize different paths, which can cause issues. In this case, manual modification is required, so it is recommended to export all files to the same folder.

Export the UI files. The exported files will be located in the path set earlier.

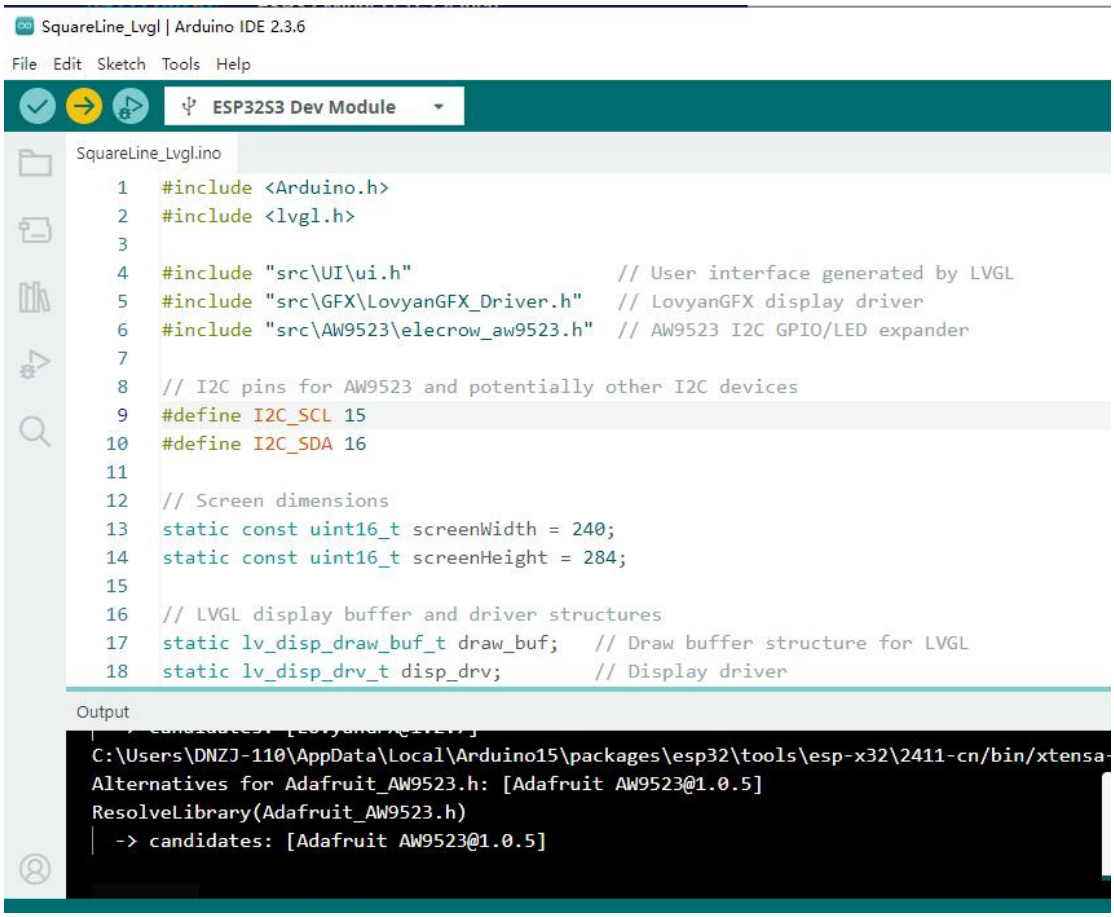




To use them together with the main program, all generated UI files need to be placed in the same folder as the main program. Here, they are placed in the project's src folder. The UI part is completed.



B. Use Arduino IDE to open the Lesson 2 program:



Code Explanation

(1) Header files and macro definitions, for easier reading.

```
#include <Arduino.h>
#include <lvgl.h>
#include "src\UI\ui.h"                    // User interface generated by LVGL
#include "src\GFX\LovyanGFX_Driver.h"     // LovyanGFX display driver
#include "src\AW9523\elecrow_aw9523.h"    // AW9523 I2C GPIO/LED expander
```

The required header files and libraries are included here. Their functions are as follows:

Arduino.h: Arduino core support;

lvgl.h: LVGL graphics library;

ui.h: Interface generated by SquareLine;

LovyanGFX_Driver.h: Screen driver;

elecrow_aw9523.h: Backlight and IO expansion control.

```
#define I2C_SCL 15
```

```
#define I2C_SDA 16
```

Define the pins used by I2C.

```
static const uint16_t screenWidth = 240;
static const uint16_t screenHeight = 284;
// I2C object
static TwoWire* wi = &Wire;
```

Define the screen resolution.

(2) LVGL display buffer objects

```
static lv_disp_draw_buf_t draw_buf;
static lv_disp_drv_t disp_drv;
static lv_indev_drv_t indev_drv;
```

draw_buf: stores the LVGL drawing buffer;

disp_drv: display driver structure;

indev_drv: input device structure.

(3) LVGL refresh callback function

```
void my_disp_flush(lv_disp_drv_t *disp,
const lv_area_t *area,
lv_color_t *color_p)
```

This function is automatically called when LVGL needs to update the screen.

```
if (gfx.getStartCount() > 0) {
gfx.endWrite();
}
```

Ensure that the previous SPI transmission has completed.

```
gfx.pushImageDMA(
area->x1,
area->y1,
area->x2 - area->x1 + 1,
area->y2 - area->y1 + 1,
(lgfx::rgb565_t *)&color_p->full
);
```

Send the pixel data generated by LVGL to the screen through DMA.

```
lv_disp_flush_ready(disp);
```

Notify LVGL that this refresh is complete.

(4) Touch reading callback function

```
void my_touchpad_read(lv_indev_drv_t *indev_driver,
lv_indev_data_t *data)
```

Function: Periodically called by LVGL to read the touch state.

```
data->state = LV_INDEV_STATE_REL;
```

No touch by default.

```
if (gfx.getTouch(&touchX, &touchY))
```

Read coordinates from the touch chip.

```
data->state = LV_INDEV_STATE_PR;
data->point.x = 240 - touchX;
data->point.y = touchY + 10;
```

Set the touch pressed state and correct the coordinate direction.

(5) setup() initialization process

Initialize I2C and AW9523.

```
wi->setPins(I2C_SDA, I2C_SCL);
wi->begin();
delay(100);
aw9523.AW_init();
aw9523.AW_set_POWER(true);
aw9523.AW_set_lcd_blight(100);
```

Start the I2C bus, initialize the chip, enable power, and turn on the backlight.

Initialize the screen.

```
gfx.init();
gfx.initDMA();
gfx.startWrite();
```

```
gfx.fillScreen(TFT_BLACK);
```

Complete the screen hardware initialization and clear the screen.

Initialize LVGL.

```
lv_init();
```

Start the LVGL core.

Allocate display buffer.

```
  size_t buffer_size = sizeof(lv_color_t) * screenWidth * screenHeight;
  static lv_color_t *buf = (lv_color_t *)heap_caps_malloc(buffer_size,
MALLOC_CAP_SPIRAM);
  static lv_color_t *buf1 = (lv_color_t *)heap_caps_malloc(buffer_size,
MALLOC_CAP_SPIRAM);
```

Allocate a large memory buffer in PSRAM.

Register display driver.

```
  lv_disp_drv_init(&disp_drv);
  disp_drv.hor_res = screenWidth;          // Horizontal resolution
  disp_drv.ver_res = screenHeight;        // Vertical resolution
  disp_drv.flush_cb = my_disp_flush;      // Set flush callback
  disp_drv.draw_buf = &draw_buf;           // Assign draw buffer
  lv_disp_drv_register(&disp_drv);
```

Bind the refresh function to LVGL.

Register touch driver.

```
lv_indev_drv_init(&indev_drv);
indev_drv.read_cb = my_touchpad_read;
indev_drv.type = LV_INDEV_TYPE_POINTER;
lv_indev_drv_register(&indev_drv);
```

Bind the touch function to LVGL.

```
ui_init();
```

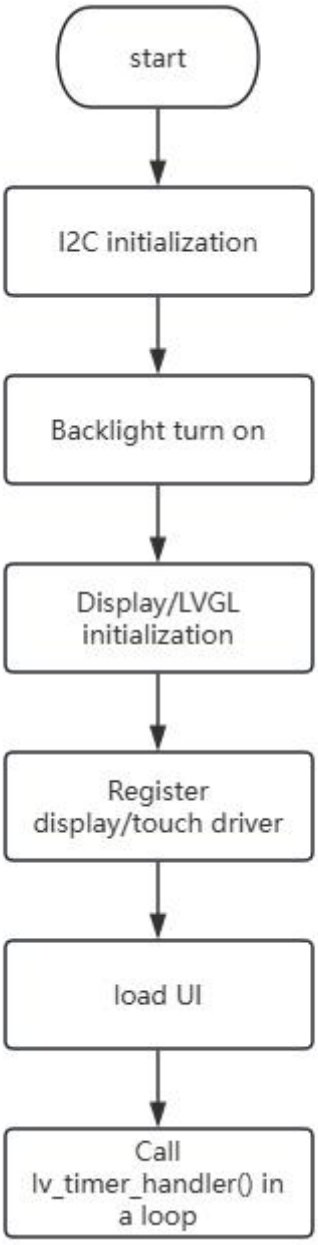
Initialize the user interface (generated by LVGL)

loop() main loop

```
lv_timer_handler();
```

Let LVGL handle animations, touch input, and interface refresh.

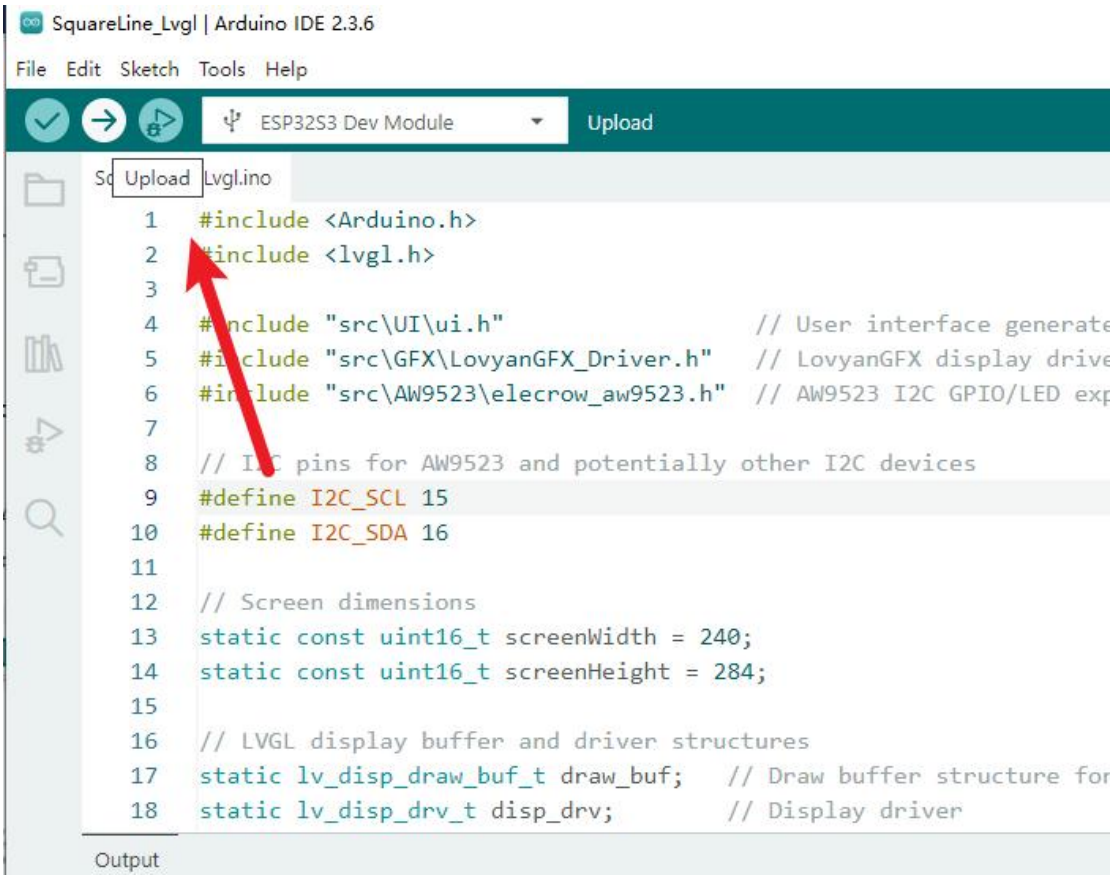
Program flowchart:



IV. Download and run the program to observe the effect

(1) According to the Arduino IDE operation process introduced in the preface, connect the computer and perform basic download configuration.

Click Download:



When the button is clicked, the page jumps to the second page and displays ELECROW. Touching the second page at this time returns to the first page.

Lesson 4 — Mic_Speaker

Overview

In this lesson, we will learn how to use the microphone (MIC) and speaker modules on the AI Camera development board, and implement the complete process of capturing audio data with the microphone, reading microphone signals with the MCU, and outputting audio to the speaker in real time via the I2S interface through an example program. In this course, you will understand the basic working principle of digital microphones, the role of the I2S audio interface in audio transmission, and the complete audio signal path of Microphone → MCU → Speaker. At the same time, you will learn how to use the MIC and Audio driver libraries provided by Elecrow to complete audio input and output control. Through this lesson, you will master the basic usage of audio input and output, laying a solid foundation for subsequent applications such as voice interaction, speech recognition, and sound effect playback.

Learning Objectives

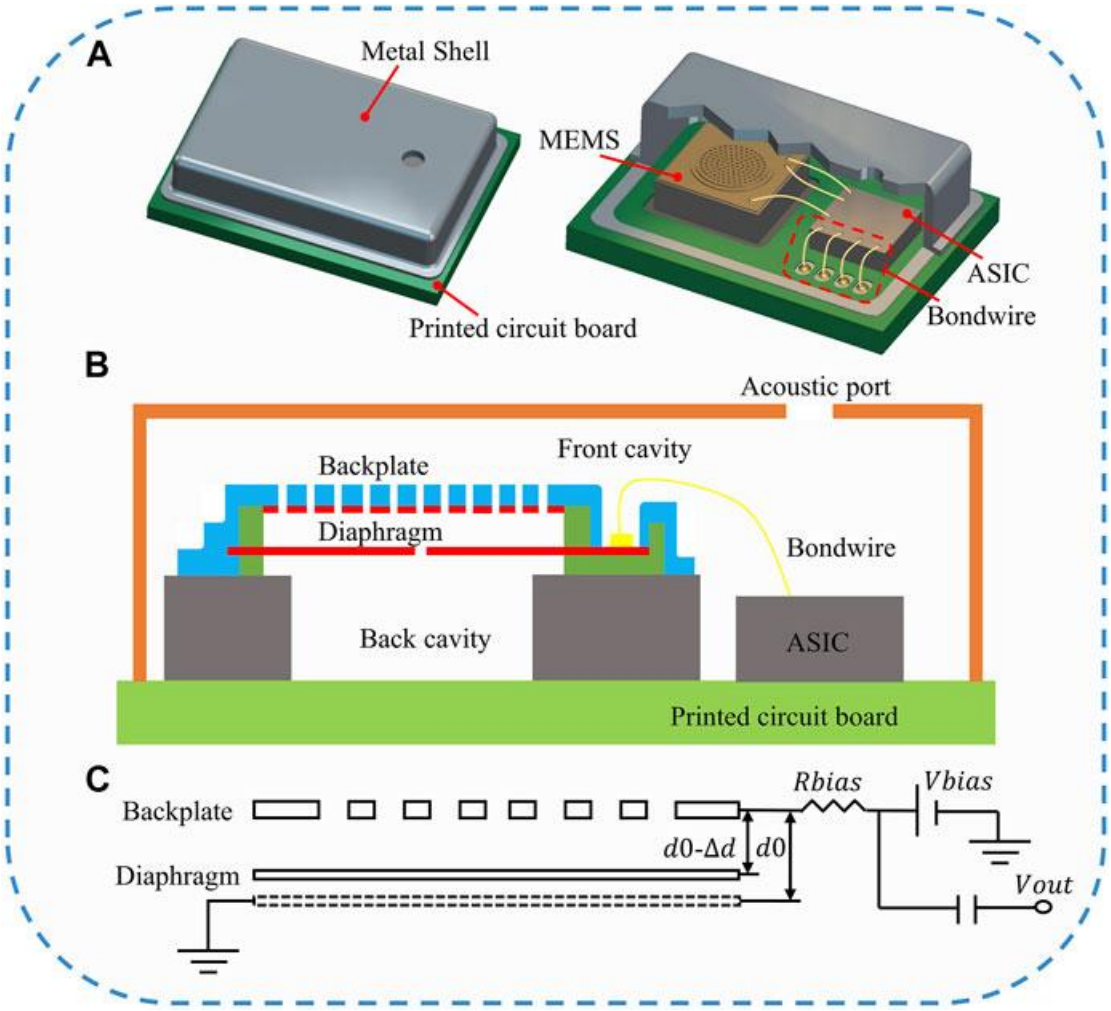
1. Understand how digital microphones work;
2. Become familiar with the I2S audio communication interface;
3. Master the pin connection methods for MIC and Speaker;
4. Learn how to initialize the MIC and Audio modules;
5. Implement the basic process of “capture → process → output” for audio.

Project Running Effect:

After startup, speak “hello” to the AI Camera. The AI Camera records the audio and then plays back the spoken “hello”.

I. Principle Explanation

1. Working Principle of Digital Microphone (MIC)



A MEMS microphone consists of a MEMS chip, an ASIC chip, a printed circuit board, and a metal housing with acoustic ports. The MEMS chip mainly includes a rigid backplate and a flexible diaphragm, usually made of polysilicon material. The silicon backplate contains many air holes that allow air to pass through to reduce air damping. The diaphragm also requires ventilation holes to balance the pressure between the front and rear cavities. The microphone on the development board is a digital microphone, which internally completes analog signal acquisition, signal amplification, and analog-to-digital conversion (ADC), and outputs digital audio data.

Common signal lines:

CLK (clock);

DOUT (data output);

The MCU reads the data stream output by the microphone synchronously through the clock signal.

2. Introduction to the I2S Audio Interface

I2S (Inter-IC Sound) is a serial communication interface standard specifically designed for digital audio transmission, and is commonly used for data communication between microcontrollers, audio codecs, digital microphones, and audio amplifiers.

Unlike general serial interfaces, I2S is designed specifically for audio data and can transmit continuous audio data streams in a stable and high-speed manner.

I2S typically uses the following three core signal lines:

BCLK (Bit Clock): controls the timing of each bit of data transmission

LRC / WS (Left Right Clock / Word Select): used to distinguish between left and right audio channels

DOUT (Data Out): transmits audio data

During data transmission:

BCLK provides the timing clock;

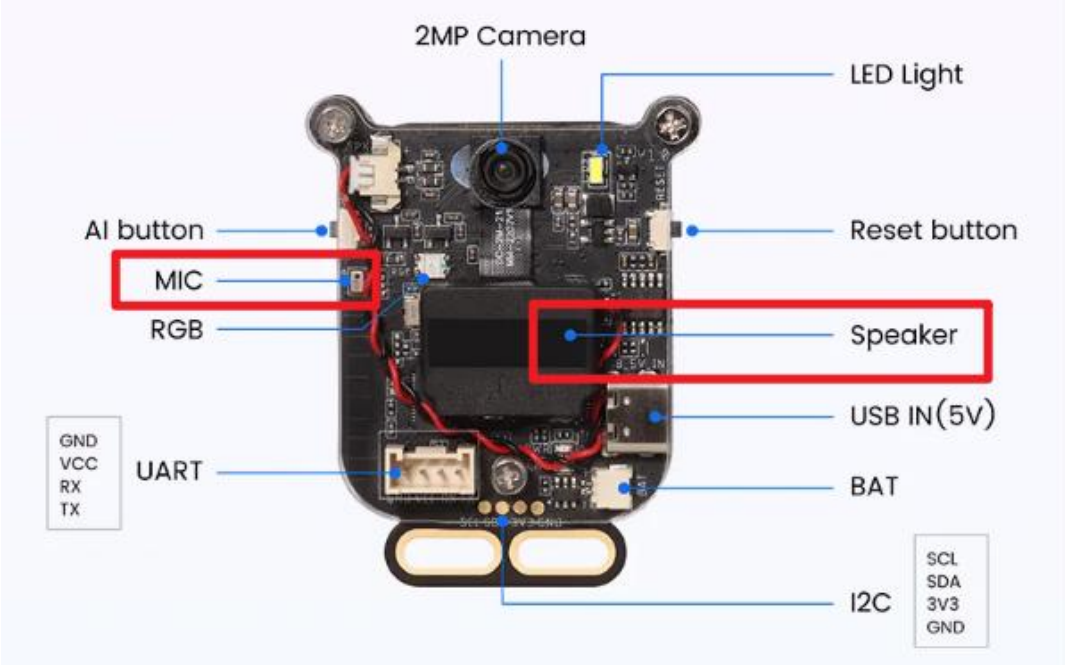
LRC indicates whether the current data belongs to the left channel or the right channel;

DOUT sends audio sample data sequentially.

In this lesson, the MCU receives digital audio data from the microphone via I2S, and then sends the audio data to the speaker via I2S for playback.

II. Required Modules

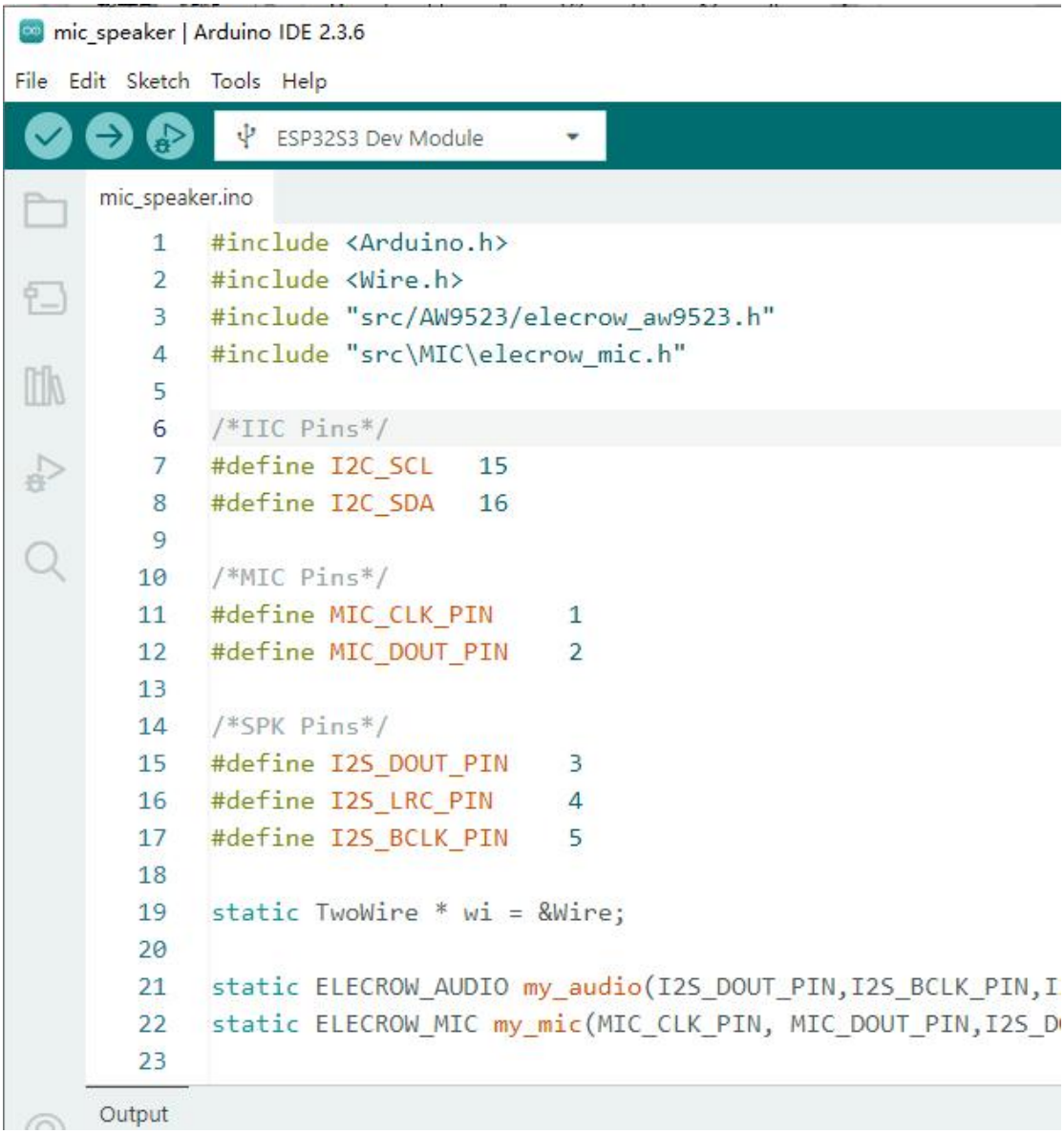
MIC and speaker of the AI Camera:



III. Example Explanation

Click the link to download the official example code provided: (GitHub link)

Open the course program using Arduino IDE:



Code Explanation

(1) First, add the required header files.

```
#include <Arduino.h>
#include <Wire.h>
#include "src/AW9523/elecrow_aw9523.h"
#include "src\MIC\elecrow_mic.h"
```

Arduino.h is the Arduino core header file, providing basic functions such as pinMode, delay, and Serial.

Wire.h is Arduino's I2C communication library, used for communication between the MCU and the AW9523 chip.

elecrow_aw9523.h is the driver library for the AW9523 I/O expander and power control chip.

elecrow_mic.h is the microphone and audio driver library provided by Elecrow, which encapsulates microphone initialization and audio output related functions.

These header files together provide the basic hardware control and communication support for the program.

(2) Define digital pins and variables for readability.

```
#define I2C_SCL 15
#define I2C_SDA 16
```

This section defines the pins used by the I2C bus:

I2C_SCL represents the clock line, connected to GPIO15;

I2C_SDA represents the data line, connected to GPIO16;

Using macro definitions makes the code clearer, and when pin changes are needed, only this section needs to be modified.

(3) Define microphone pins

```
#define MIC_CLK_PIN        1
#define MIC_DOUT_PIN       2
```

MIC_CLK_PIN: microphone clock input pin;

MIC_DOUT_PIN: microphone data output pin;

The MCU drives the microphone operation through the clock signal and reads audio data from the data pin.

(4) Define speaker (I2S) pins

```
#define I2S_DOUT_PIN       3
#define I2S_LRC_PIN        4
#define I2S_BCLK_PIN       5
```

I2S_DOUT_PIN: audio data output pin;

I2S_LRC_PIN: left/right channel selection signal;

I2S_BCLK_PIN: bit clock signal;

These three pins together form the I2S audio output interface used to send digital audio data to the speaker.

(5) Create an I2C bus object

```
static TwoWire * wi = &Wire;
```

This statement creates an I2C bus pointer and points it to the system default Wire object, making it convenient to configure I2C pins and communication later.

(6) Create audio and microphone objects

```
static ELECROW_AUDIO my_audio(I2S_DOUT_PIN,
                              I2S_BCLK_PIN,
                              I2S_LRC_PIN);
```

This object is used to initialize the speaker audio output interface and specify the I2S data, bit clock, and left/right channel pins.

static ELECROW_MIC my_mic(MIC_CLK_PIN,

```
static ELECROW_MIC my_mic(MIC_CLK_PIN,
                          MIC_DOUT_PIN,
                          I2S_DOUT_PIN,
                          I2S_BCLK_PIN,
                          I2S_LRC_PIN);
```

This object is used to initialize the microphone module and associate microphone input with the I2S output interface, enabling direct output after capture.

(7) setup() Function — Initialization Stage

```
Serial.begin(115200);
while(!Serial);
```

Start serial communication and wait for the serial connection to be established for subsequent debugging.

```
wi->setPins(I2C_SDA, I2C_SCL);
wi->begin();
```

Initialize I2C by assigning SDA and SCL pins and starting I2C bus communication.

```
aw9523.AW_init();
aw9523.AW_set_POWER(true);
```

This step completes the initialization of the AW9523 chip and turns on peripheral power to supply operating voltage for audio-related circuits.

```
aw9523.AW_set_MUTE(true);
delay(100);
aw9523.AW_set_MUTE(false);
delay(100);
aw9523.AW_set_MUTE(true);
```

This process toggles the mute state multiple times to stabilize the speaker and prevent popping or noise during power-up.

(8) Main Loop loop()

```
my_mic.MIC_init();
```

Initialize the microphone module to put it into working state.

```
my_mic.MIC_ReadandOut();
```

This function completes two main tasks:

Read audio data from the microphone;

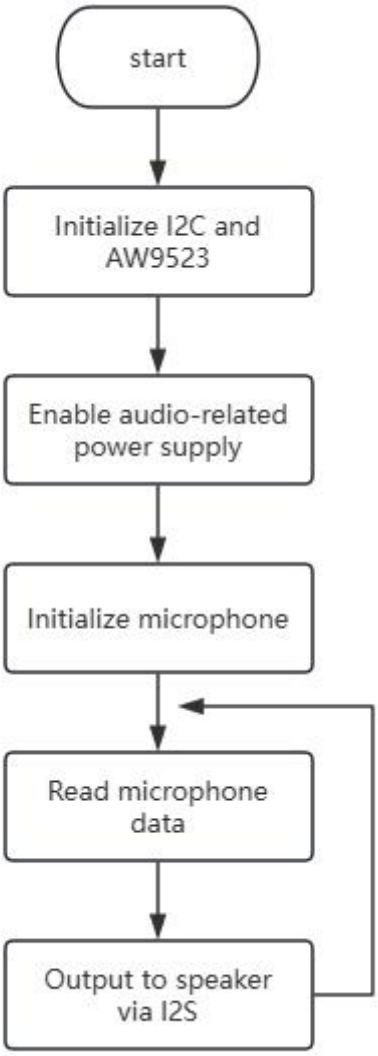
Output the data to the speaker via the I2S interface;

Achieve real-time audio transmission.

```
delay(1000);
```

Delay for 1 second to control the program execution rhythm.

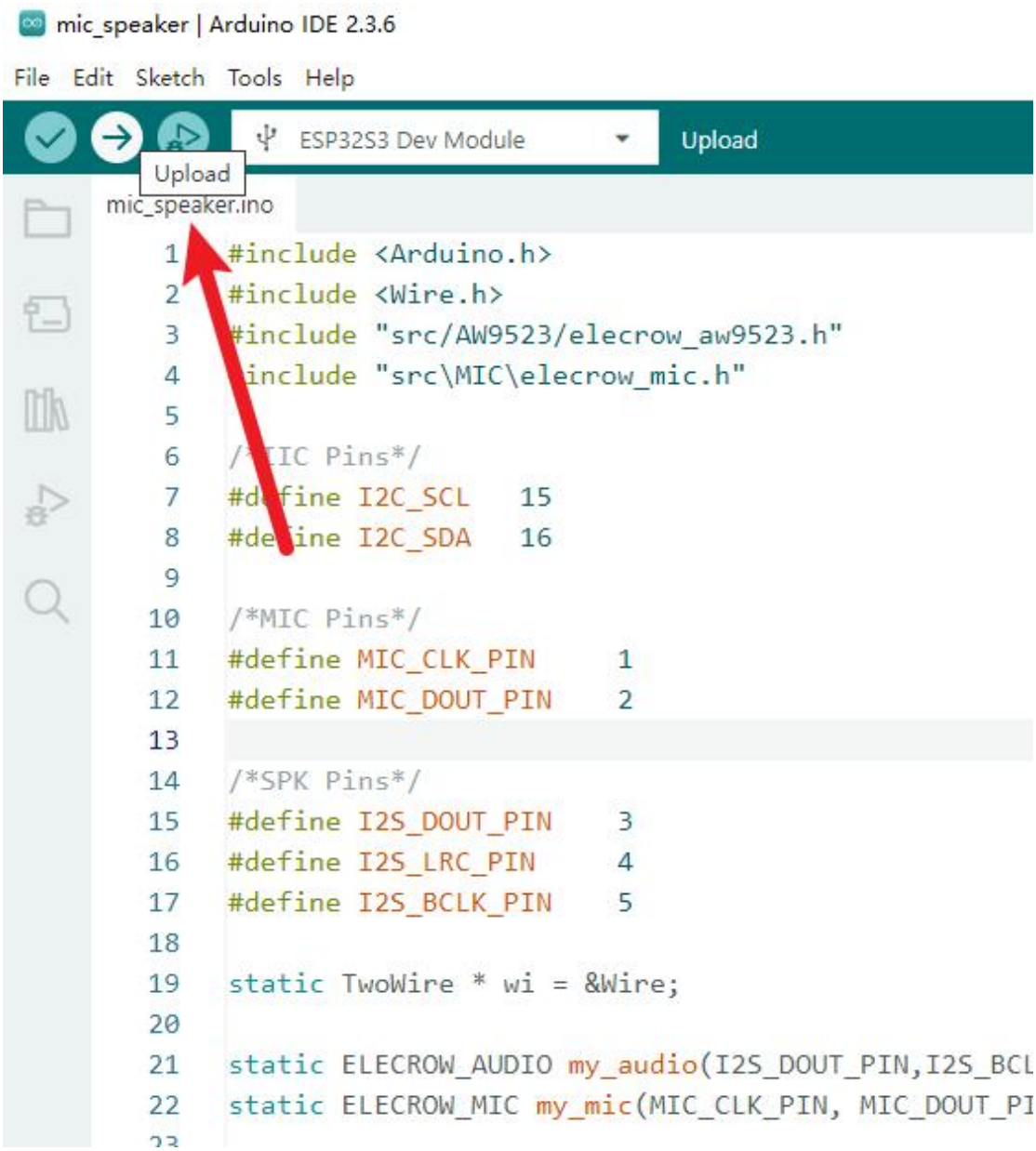
Program Flowchart:



IV. Download the Program and Observe the Effect

(1) According to the Arduino IDE operation process introduced in the preface, connect the computer and perform basic download configuration.

Click Download:



After startup, speak “hello” to the AI Camera. The AI Camera records the audio and then plays back the spoken “hello”.


Hello!
Hello!

Lesson 5 — Camera_TFCard

Overview

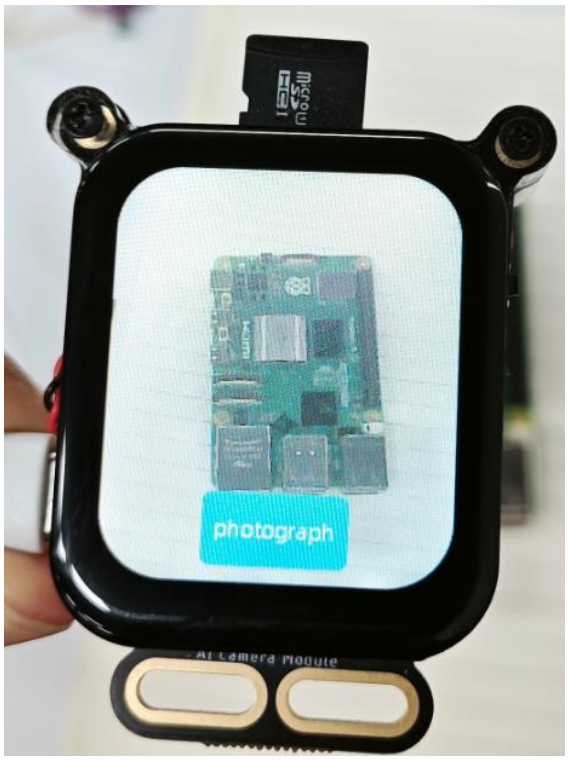
In this lesson, we will learn how to use the GC2145 camera module and TF (SD) card module on the AI Camera development board, and implement the complete functional process including camera initialization, real-time image display, touch button photo capture, and saving captured photos to the SD card in BMP format through an example program. During the learning process, you will gradually understand the basic working principle of the camera, the hardware connection method between the ESP32-S3 and the camera, the organization format of RGB image data, and the basic read/write operations of the SD card file system. Through the practice in this lesson, you will master the complete implementation process from image acquisition by the camera → MCU processing → screen display → saving to the SD card, laying a solid foundation for subsequent image storage, image recognition, and AI vision-related application development.

Learning Objectives

1. Understand the basic working principle of the GC2145 camera
2. Understand the parallel data interface between the camera and the MCU
3. Master the camera pin connection method
4. Learn to initialize the camera and SD card module
5. Implement the basic process of “Capture → Display → Save to SD Card”

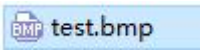
Project Operation Effect:

After power-on, the screen displays the real-time camera image. Click the photograph button at the bottom of the screen to capture the current frame and save it to the SD card.



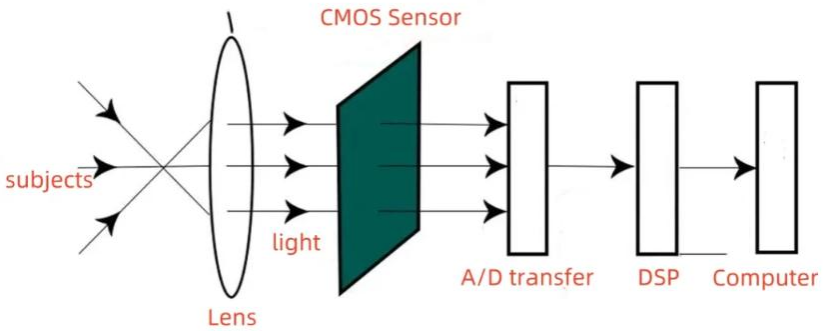
File generated in the SD card:

/test.bmp



I. Principle Explanation

1. Working Principle of the GC2145 Camera



The GC2145 is a CMOS image sensor that converts light signals into digital image data.

The basic process is as follows:

The lens captures light

The CMOS sensor converts light into electrical signals

The sensor completes analog-to-digital conversion internally

RGB data is output through the parallel data bus

The data output to the MCU includes:

Pixel data (D0~D7)

Horizontal synchronization signal (HREF)

Vertical synchronization signal (VSYNC)

Pixel clock (PCLK)

The MCU reads pixel data sequentially according to the clock and synchronization signals and forms a complete image.

2. RGB565 Pixel Format

Used in this example:

PIXFORMAT_RGB565

Each pixel occupies 16 bits:

R: 5 bits

G: 6 bits

B: 5 bits

Advantages:

Smaller data size, suitable for real-time display, and easy to save as BMP.

3. SD Card Storage Principle

The SD card communicates with the MCU through the SPI bus

The ESP32 mounts the FAT file system

Image data is written through the file path

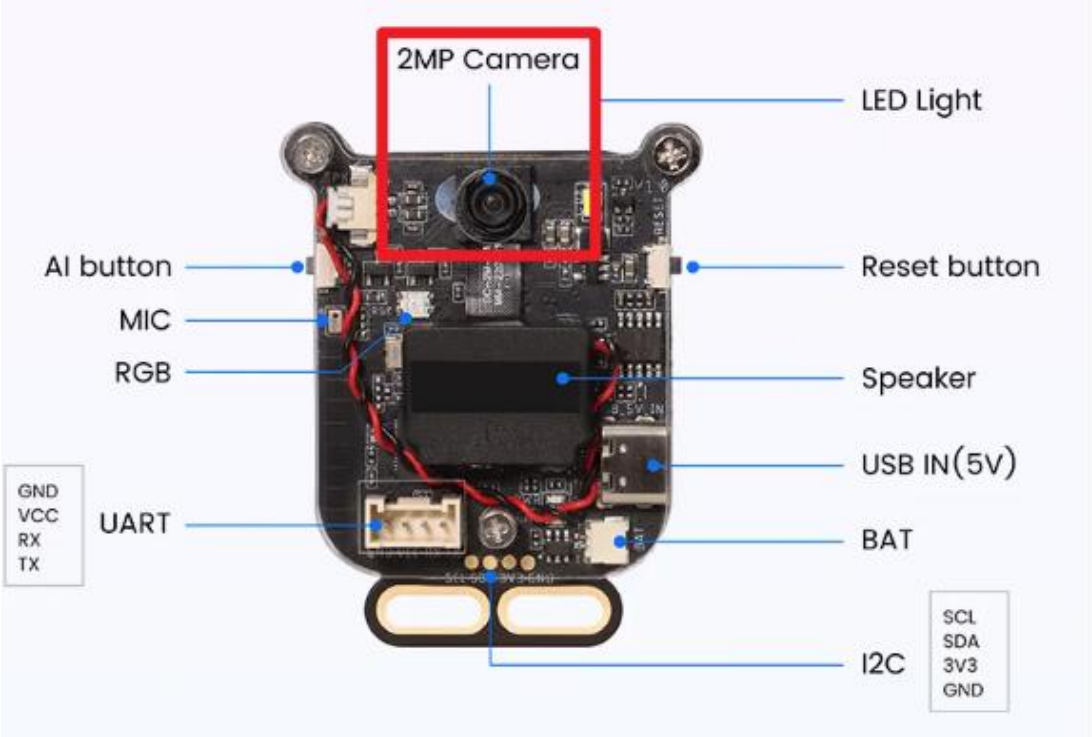
This example uses the encapsulated function:

save_rgb565_to_bmp()

Converts the RGB565 image in memory into a BMP file and writes it to the SD card.

II. Required Modules

AI Camera GC2145 camera:

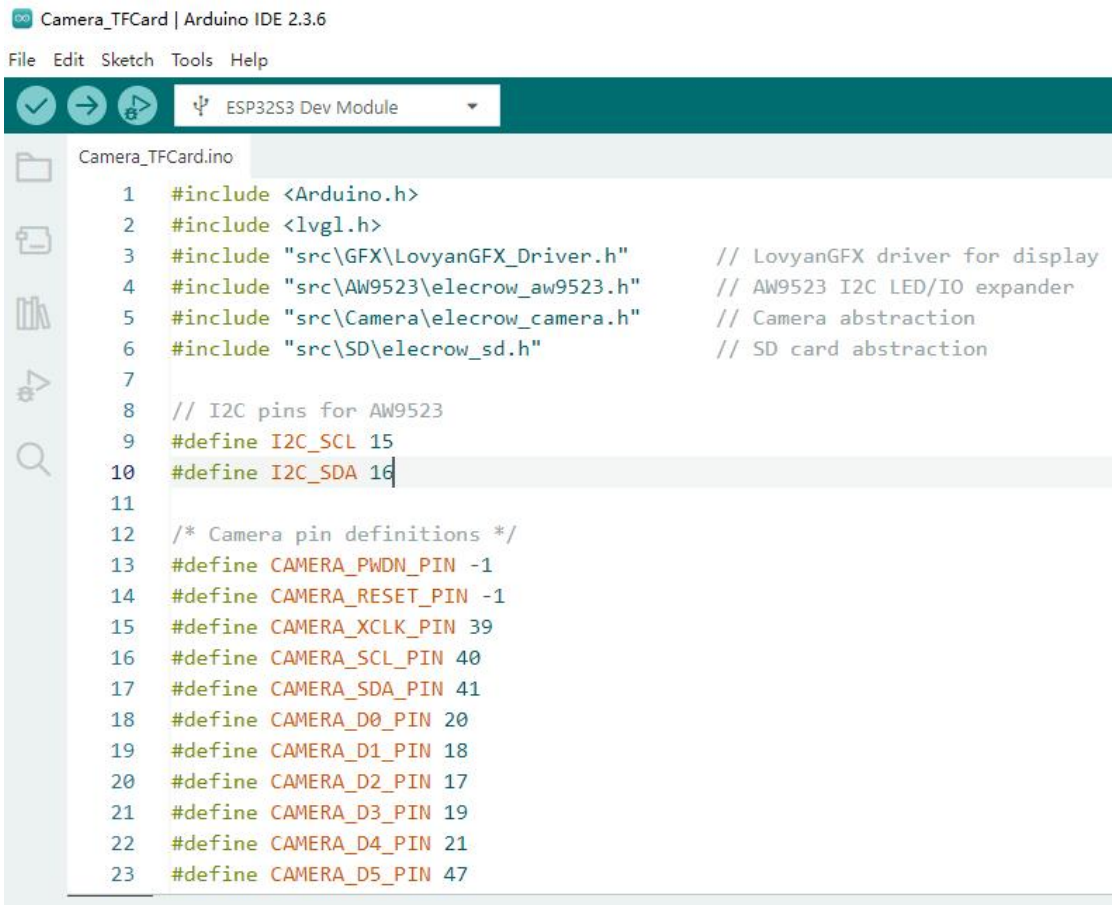


TF (Micro SD) card

III. Example Explanation

Click the link to download the official example code: (GitHub link)

Use Arduino IDE to open the course program:



Code Explanation

(1) First, add the necessary header files.

```
#include <Arduino.h>
#include <lvgl.h>
#include "src\GFX\LovyanGFX_Driver.h"          // LovyanGFX driver for display
#include "src\AW9523\elecrow_aw9523.h"        // AW9523 I2C LED/IO expander
#include "src\Camera\elecrow_camera.h"        // Camera abstraction
#include "src\SD\elecrow_sd.h"                // SD card abstraction
```

Arduino.h is the Arduino core header file, providing basic functions such as pinMode, delay, and Serial.

lvgl.h is a lightweight graphics library used to create and manage UI interfaces.

LovyanGFX_Driver.h is the display driver library responsible for screen initialization and drawing.

elecrow_aw9523.h is the driver library for the AW9523 I/O expander and power control chip.

elecrow_camera.h is the camera driver library that encapsulates image acquisition and control interfaces.

elecrow_sd.h is the SD card driver library used for file reading/writing and storage management.

(2) Initialize I2C / AW9523

```
#define I2C_SCL 15
#define I2C_SDA 16
static TwoWire* wi = &Wire;
```

Define the pins used by the I2C bus, where GPIO16 serves as SDA and GPIO15 serves as SCL.

Create an I2C bus object pointer wi, pointing to the default Arduino Wire object. All subsequent I2C operations are completed through this object.

```
#define I2C_SCL 15
#define I2C_SDA 16
static TwoWire* wi = &Wire;
```

Assign SDA and SCL pins for the I2C bus and start I2C communication.

(3) AW9523 Initialization

```
aw9523.AW_init();
aw9523.AW_set_POWER(true);
aw9523.AW_set_lcd_blight(100);
```

AW_init(): Initialize the AW9523 chip

AW_set_POWER(true): Enable peripheral power supply

AW_set_lcd_blight(100): Set LCD backlight brightness to 100

At this point, the power supply and control functions of the development board peripherals are ready.

(4) Initialize the LCD display

```
gfx.init();
gfx.initDMA();
gfx.fillScreen(TFT_BLACK);
```

gfx.init(): Initialize display hardware

gfx.initDMA(): Enable DMA accelerated transfer

gfx.fillScreen(TFT_BLACK): Clear the screen and fill it with black

The display initialization is complete.

(5) Initialize the camera

Pin definitions:

```
#define CAMERA_XCLK_PIN 39
#define CAMERA_SCL_PIN 40
#define CAMERA_SDA_PIN 41
#define CAMERA_D0_PIN 20
#define CAMERA_D1_PIN 18
#define CAMERA_D2_PIN 17
#define CAMERA_D3_PIN 19
#define CAMERA_D4_PIN 21
#define CAMERA_D5_PIN 47
#define CAMERA_D6_PIN 38
#define CAMERA_D7_PIN 46
#define CAMERA_VSYNC_PIN 42
#define CAMERA_HREF_PIN 45
#define CAMERA_PCLK_PIN 48
```

These macro definitions clearly specify the GPIO pins connected to the GC2145 camera's clock lines, control lines, and 8-bit parallel data lines.

Create the camera object:

```
static ELECROW_CAMERA my_camera;
```

Camera parameter configuration:

```
camera_config_t config = {
    .pin_pwdn = CAMERA_PWDN_PIN,
    .pin_reset = CAMERA_RESET_PIN,
    .pin_xclk = CAMERA_XCLK_PIN,
    .pin_sccb_sda = CAMERA_SDA_PIN,
    .pin_sscb_scl = CAMERA_SCL_PIN,
```

Specify camera control and I2C communication pins.

```
    .pin_d7 = CAMERA_D7_PIN,
    .pin_d6 = CAMERA_D6_PIN,
    .pin_d5 = CAMERA_D5_PIN,
```

```
    .pin_d4 = CAMERA_D4_PIN,
    .pin_d3 = CAMERA_D3_PIN,
    .pin_d2 = CAMERA_D2_PIN,
    .pin_d1 = CAMERA_D1_PIN,
    .pin_d0 = CAMERA_D0_PIN,
```

Specify the 8-bit data bus pins of the camera.

```
    .pin_vsync = CAMERA_VSYNC_PIN,
    .pin_href = CAMERA_HREF_PIN,
    .pin_pclk = CAMERA_PCLK_PIN,
```

Specify synchronization signals and pixel clock.

```
    .xclk_freq_hz = 10000000,
    .pixel_format = PIXFORMAT_RGB565,
    .frame_size = FRAMESIZE_CIF,
```

Set:

Camera main clock 10MHz;

Output format RGB565;

Resolution CIF.

```
    .fb_count = 2,
    .fb_location = CAMERA_FB_IN_PSRAM,
    .grab_mode = CAMERA_GRAB_LATEST,
};
```

Use double buffering and store frame buffers in PSRAM.

Start the camera:

```
my_camera.camera_init(&config);
my_camera.camera_sensor_init();
```

Complete camera driver initialization and sensor initialization.

(6) Initialize the SD card

```
static ELECROW_SD my_sd(SPI_SCK_PIN, SPI_MISO_PIN, SPI_MOSI_PIN, SPI_NSS_PIN);
```

Create the SD card object and specify SPI pins.

```
my_sd.SD_init();
```

Initialize the SD card and mount the file system.

(7) Initialize LVGL

```
lv_init();
```

Initialize the LVGL graphics library.

```
size_t buffer_size = sizeof(lv_color_t) * LCD_W * LCD_H;
static lv_color_t *buf1 = (lv_color_t *)heap_caps_malloc(buffer_size, MALLOC_CAP_SPIRAM);
static lv_color_t *buf2 = (lv_color_t *)heap_caps_malloc(buffer_size, MALLOC_CAP_SPIRAM);
lv_disp_draw_buf_init(&draw_buf, buf1, buf2, LCD_W * LCD_H);
```

Create double buffers in PSRAM for LVGL display.

Register display driver:

```
disp_drv.hor_res = LCD_W;
disp_drv.ver_res = LCD_H;
disp_drv.flush_cb = my_disp_flush;
disp_drv.draw_buf = &draw_buf;
lv_disp_drv_register(&disp_drv);
```

Bind screen resolution and refresh function.

Register touch input:

```
indev_drv.type = LV_INDEV_TYPE_POINTER;
indev_drv.read_cb = my_touchpad_read;
lv_indev_drv_register(&indev_drv);
```

Register the touch function as an LVGL input device.

(8) Create camera display object

```
camera_output = lv_img_create(lv_scr_act());
lv_obj_set_width(camera_output, 240);
lv_obj_set_height(camera_output, 284);
lv_obj_set_align(camera_output, LV_ALIGN_CENTER);
```

Create an LVGL image object for displaying the camera image.

(9) Create a timer to update the image in real time

```
camera_timer = lv_timer_create(lvgl_camera_update_cb, 30, NULL);
```

Call the image refresh function every 30 ms.

(10) Create capture button.

```
create_take_photo_button();
```

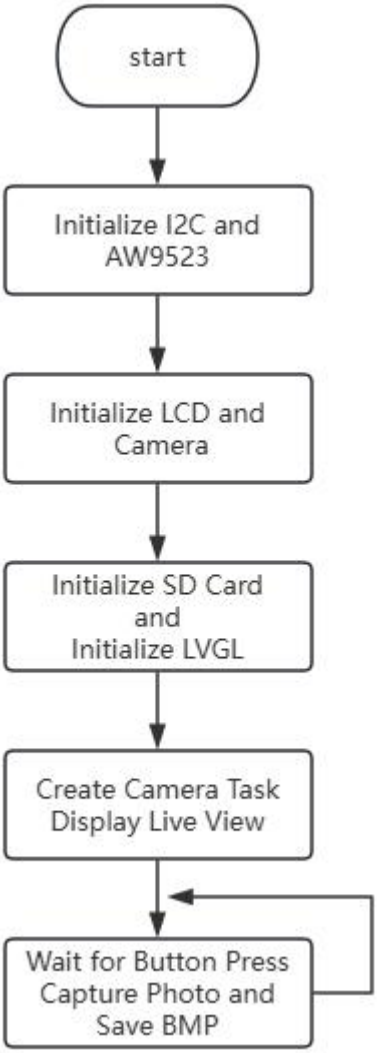
Create the bottom capture button and bind the click event.

(11) Create capture button.

```
xTaskCreatePinnedToCore(camera_task, "camera_task", 4096, NULL, 5, &camera_task_handle, 1);
```

Create the bottom capture button and bind the click event.

Program flowchart:

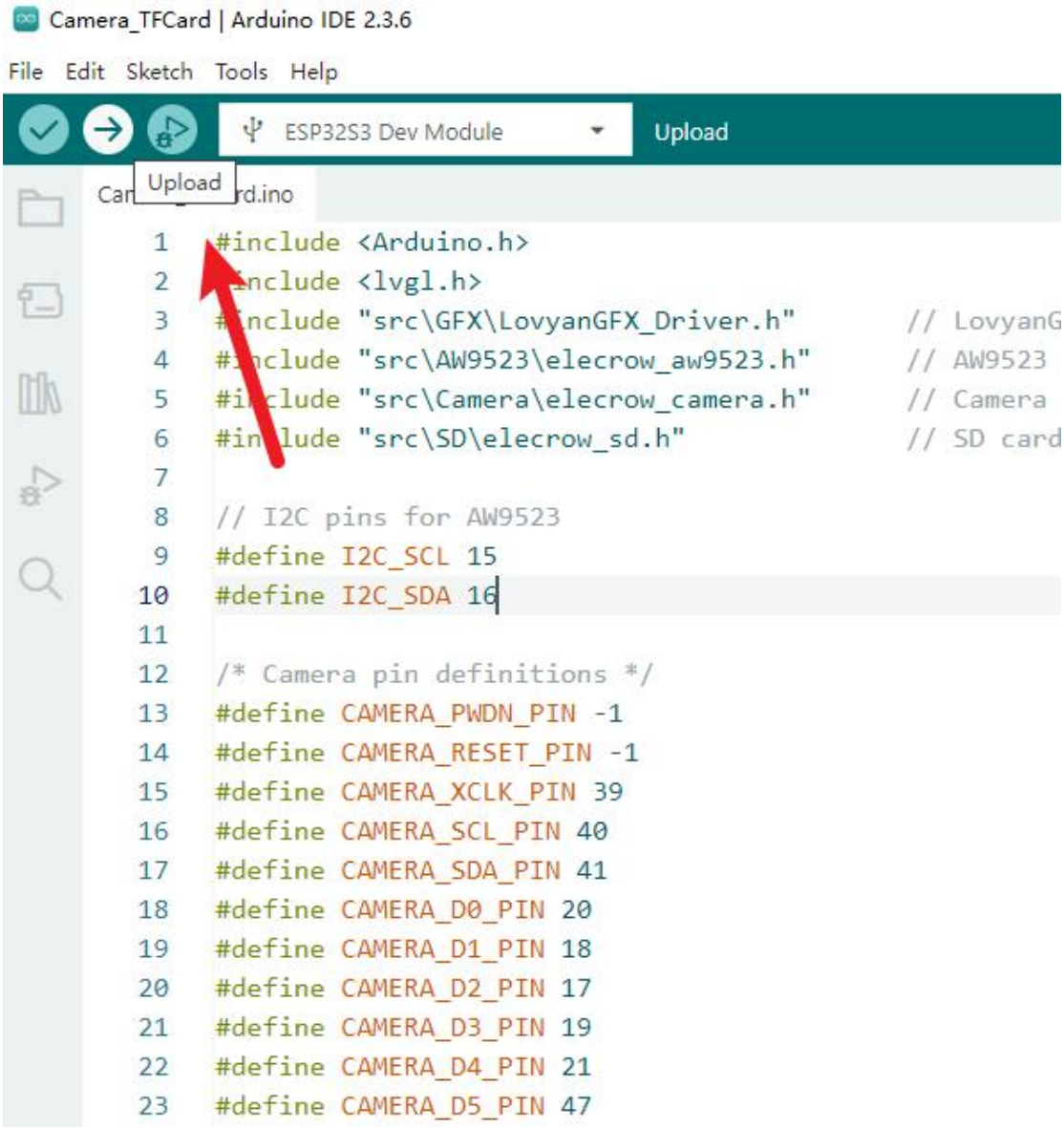


IV. Download the program and observe the effect

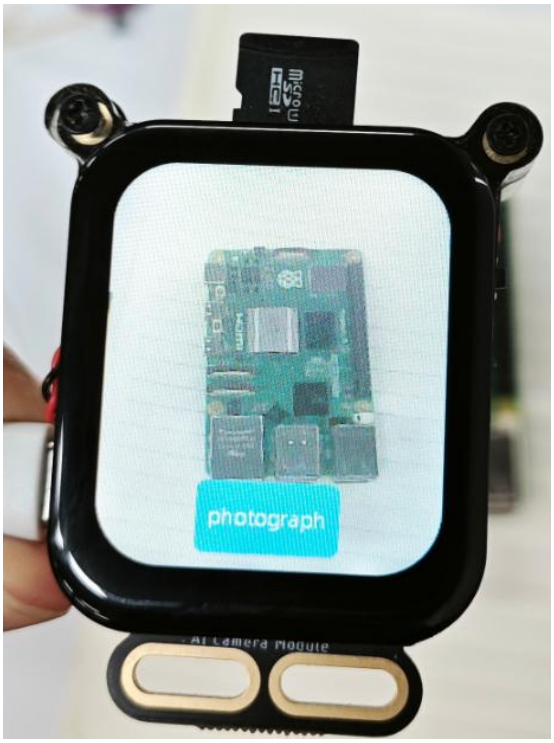
(1) Follow the Arduino IDE operation process introduced in the preface to connect

the computer and complete the basic download settings.

Click Download:



The screen displays real-time video. Click the photograph button to save the current image to the SD card.



test.bmp

Lesson 6 — Image recognition

Overview

In this lesson, we will use the AI Camera's WiFi function to read image data from the SD card, send the image to a server via network request for AI recognition, and then print the recognition results through the serial port. This completes the full intelligent recognition process of image acquisition, network transmission, cloud recognition, and result feedback.

Learning objectives

1. Master the WiFi usage of the AI Camera
2. Become familiar with Elecrow server registration and API call methods
3. Complete a cloud-server-based image recognition project example

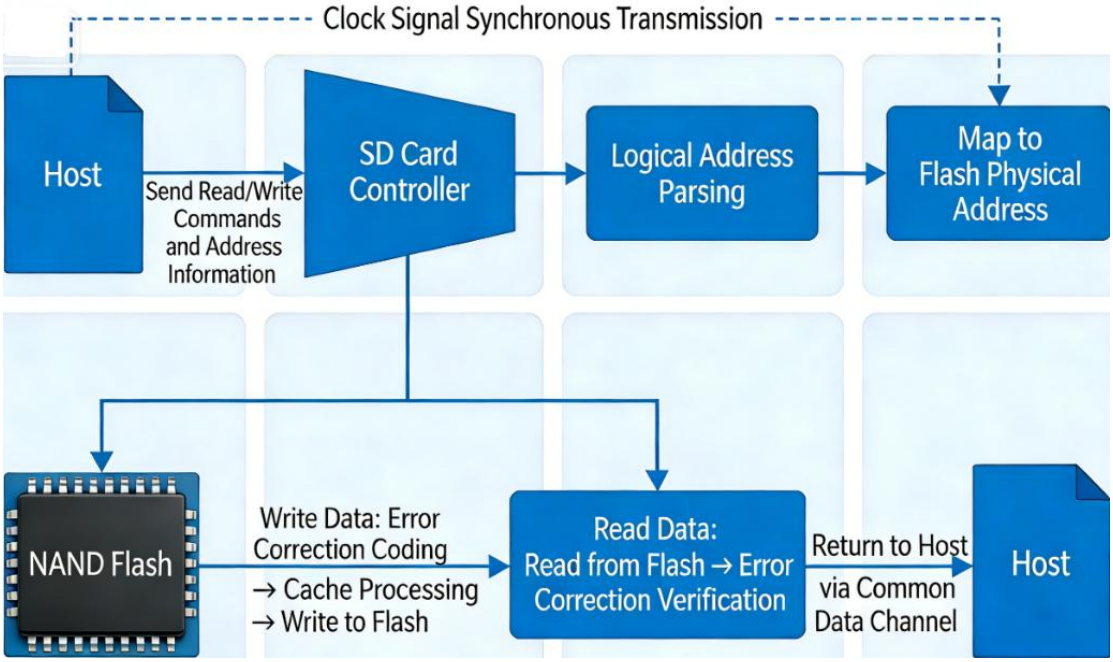
Project running effect:

Store a photo in the SD card, insert the SD card into the AI Camera, and after running the program, the content returned by the AI image recognition will be printed on the serial port.

三、Principle explanation

1. Working principle of SD card data transfer

The SD card transfers data with the host device through a standard digital communication interface. The host first sends read/write commands and address information to the SD card. After receiving the commands, the SD card's internal controller parses the logical address and maps it to the actual flash memory physical address. When writing data, the controller performs error correction coding and caching before writing data to NAND flash; when reading data, it reads data from flash and performs error checking to ensure accuracy, and finally returns the data to the host via the data bus. During this process, the clock signal synchronizes the data transfer to achieve high-speed and stable data exchange.



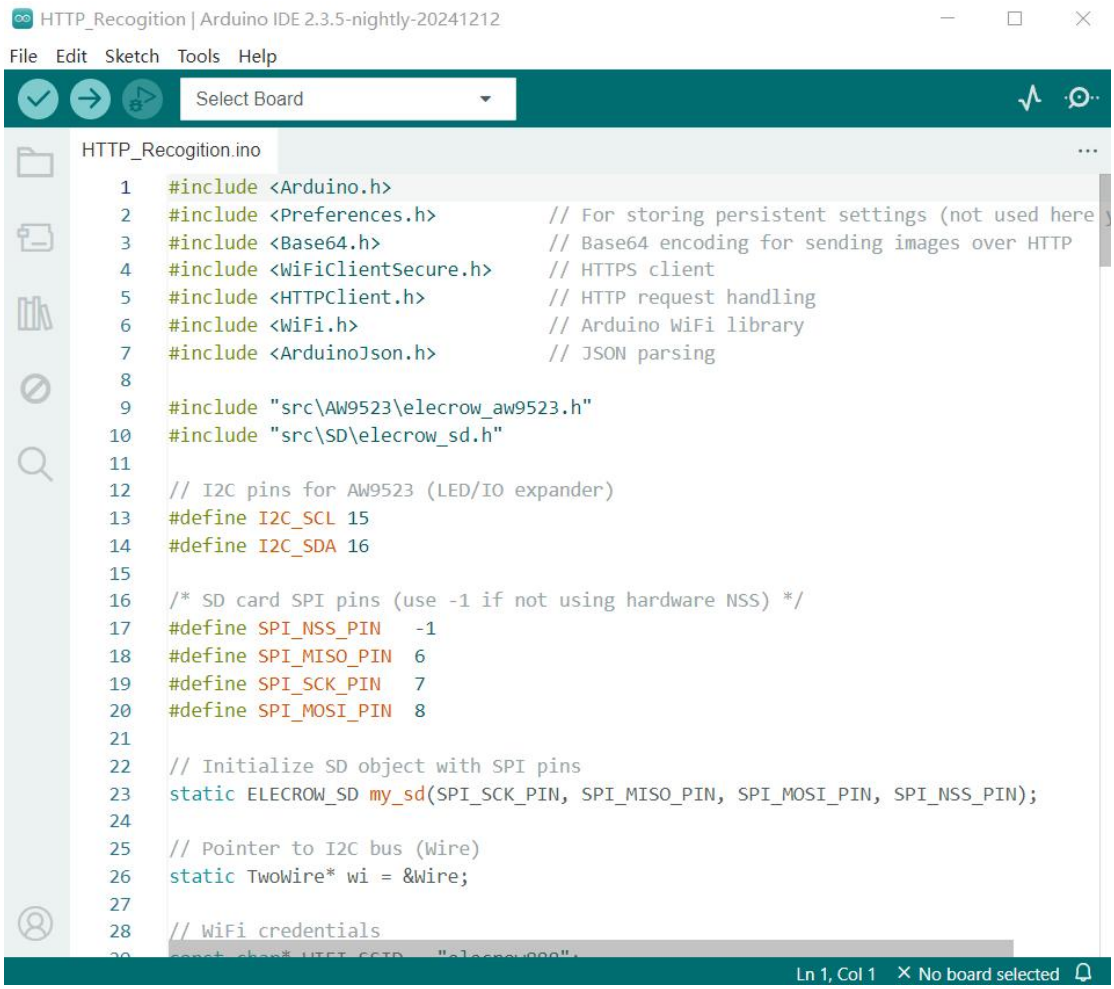
Required modules

AI Camera's SD card:

Example explanation

Click the link to download the official example code: (GitHub link)

Open the course program using Arduino IDE:



Code explanation

(1) First, include the necessary libraries.

```
#include <Arduino.h>
#include <Preferences.h>            // For storing persistent settings (not used here yet)
#include <Base64.h>                 // Base64 encoding for sending images over HTTP
#include <WiFiClientSecure.h>       // HTTPS client
#include <HTTPClient.h>             // HTTP request handling
#include <WiFi.h>                   // Arduino WiFi library
#include <ArduinoJson.h>            // JSON parsing

#include "src\AW9523\elecrow_aw9523.h"
#include "src\SD\elecrow_sd.h"
```

Arduino.h: Arduino core header file, necessary for using Arduino syntax

Preferences.h: Used to store data that is not lost on power off

Base64.h: Used to convert images to Base64 string

WiFiClientSecure.h: For HTTPS secure connection

HTTPClient.h: Used to send HTTP requests

WiFi.h: ESP32 WiFi control library

ArduinoJson.h: JSON parsing and generation, used for communication with the server

src\AW9523\elecrow_aw9523.h: IO expansion chip

src\SD\elecrow_sd.h: SD card read/write module

(2) Define digital pins and variables for readability.

```
// I2C pins for AW9523 (LED/IO expander)
#define I2C_SCL 15
#define I2C_SDA 16

/* SD card SPI pins (use -1 if not using hardware NSS) */
#define SPI_NSS_PIN    -1
#define SPI_MISO_PIN   6
#define SPI_SCK_PIN    7
#define SPI_MOSI_PIN   8
```

Define the I2C bus for the AW9523 expansion chip. The SPI interface pins are for the SD card.

(3) Initialize the SD card and set WiFi credentials

```
// Initialize SD object with SPI pins
static ELECROW_SD my_sd(SPI_SCK_PIN, SPI_MISO_PIN, SPI_MOSI_PIN, SPI_NSS_PIN);
```

Initialize the SD card. This step is essential; otherwise, the SD card will not work.

```
// WiFi credentials
const char* WIFI_SSID = "elecrow888";
const char* WIFI_PASS = "elecrow2014";
```

Set the WiFi name and password. You need to write the correct credentials according to the WiFi around you.

```
// Initialize WiFi in STA mode and connect
void wifi_init() {
    WiFi.mode(WIFI_STA);              // Set ESP32 as WiFi station
    WiFi.begin(WIFI_SSID, WIFI_PASS);

    Serial.print("Connecting WiFi\n");
    // Wait until WiFi is connected
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }

    Serial.println("WiFi connected");
    Serial.println(WiFi.localIP()); // Print local IP
}
```

Set WiFi to STA mode, which is the standard network connection mode. Then start connecting to WiFi using a blocking wait until the connection succeeds. After connecting, print the WiFi information.

(4) Device activation process explanation

```
bool GetActivationCode()
```

This is the core logic for device registration and authentication, mainly to let the server know that the device is authorized to connect.

```
String deviceId = GetMacAddress();
```

Obtain the device's MAC address, which acts as its ID.

```
https://api.thinknode.cc/ota/
```

Elecrow cloud server address. If you want to connect to another server, modify this URL and follow the request rules of that server.

```
{
   "application":{},
   "board":{"type":"AICamera"}
}
```

Construct the request body according to Elecrow server rules to tell the server that this is an AI Camera device.

```
int httpCode = http.POST(requestBody);
```

Send a POST request to the Elecrow server.

```
deserializeJson(jsonDoc, response);
```

Parse the JSON response.

```
activationCode = jsonDoc["activation"]["code"];
challengeCode    = jsonDoc["activation"]["challenge"];
```

Obtain the activation information issued by the server. We can print it to the serial port.

(5) AI recognition core function explanation

```
void AI_Recognition_func()
```

This is the soul function of the entire program. It sends the image to the server for recognition and prints the recognition result to the serial port.

```
my_sd.read_jpg_from_fs(SD, "/test.jpg", &jpg_buf, &jpg_size)
```

Read the “/test.jpg” image from the SD card, obtaining the image data pointer and image size.

```
String imageBase64 = base64::encode(jpg_buf, jpg_size);
```

Convert the image to Base64 format because HTTP can only transmit text, not images directly.

```
"image": ""
```

Construct the JSON request, placing the entire image into the JSON.

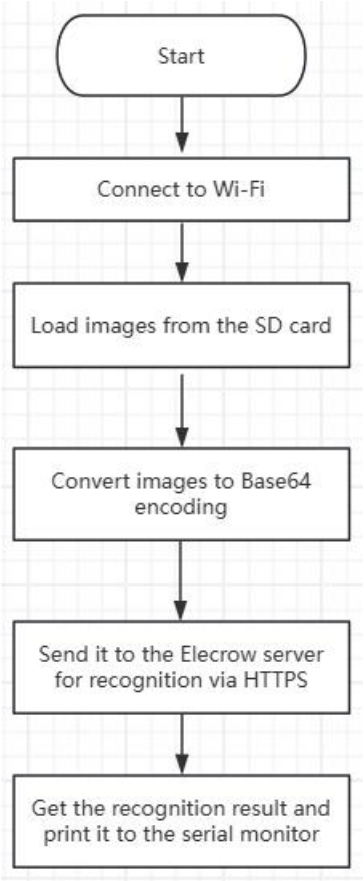
```
http.POST(jsonRequest);
```

Send a POST request to the AI server.

```
doc["data"]["response"]
```

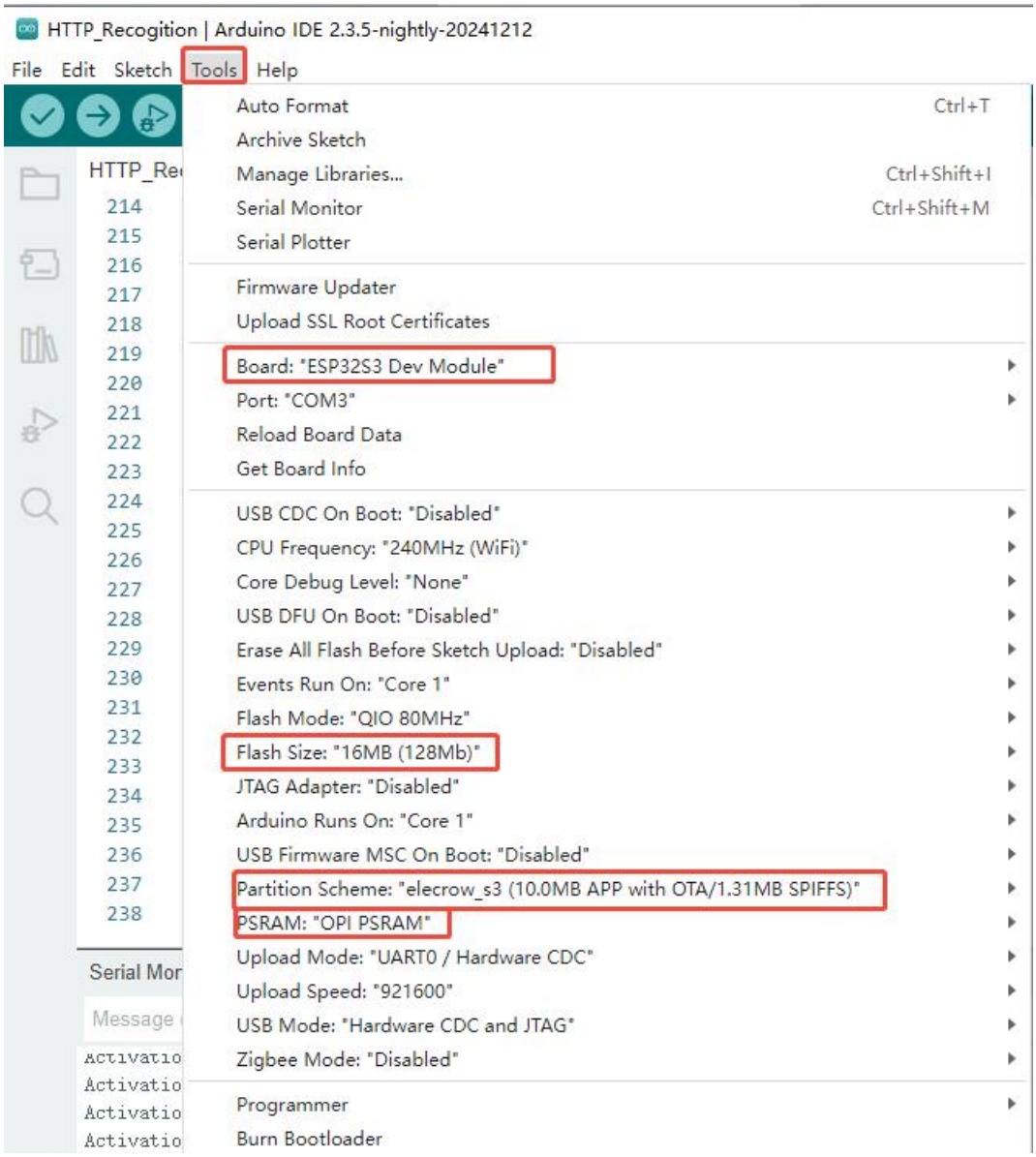
Parse the AI response and print it to the serial port.

Program flowchart:

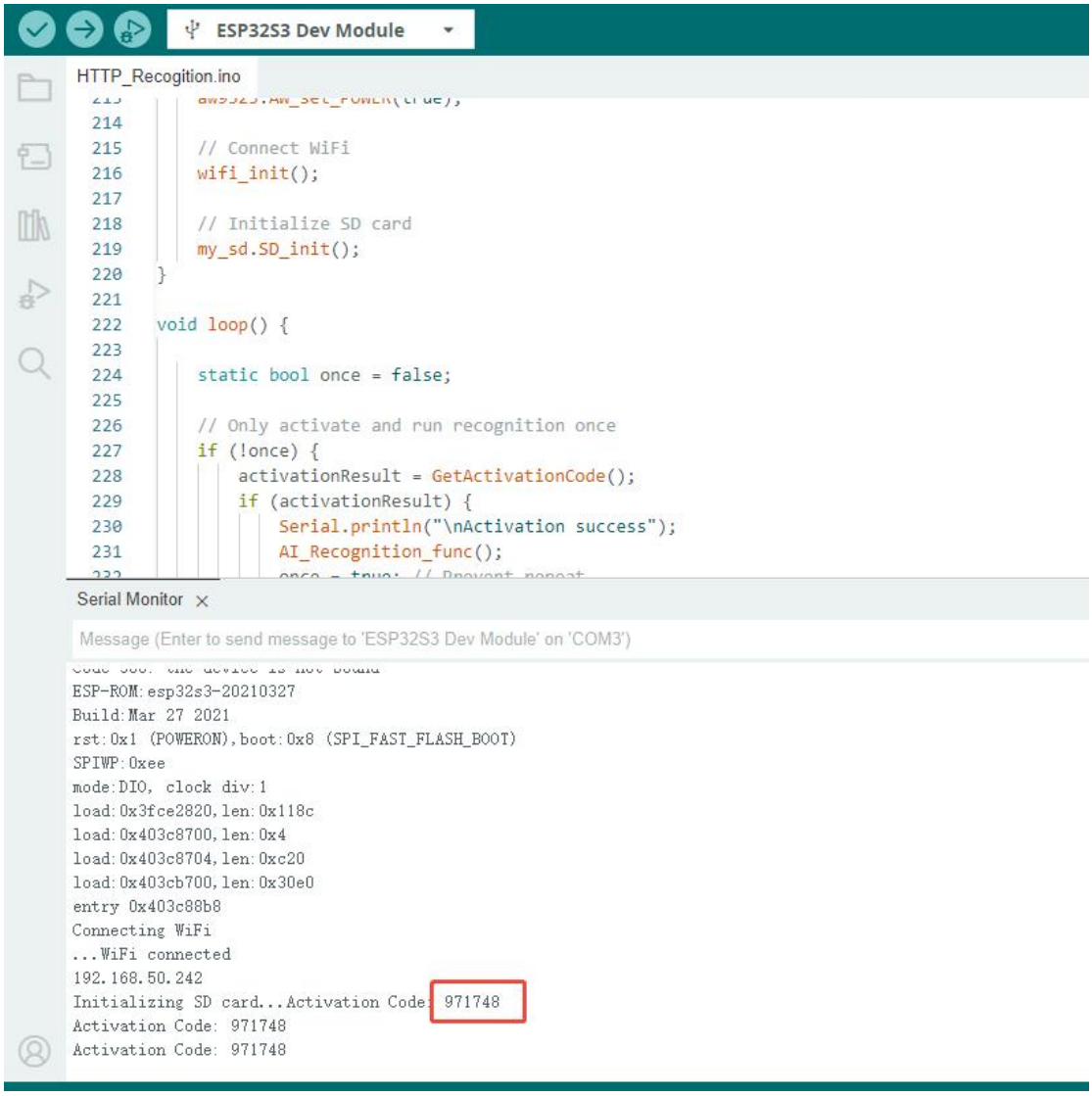


Download and run the program

(1) Connect to the computer and perform basic download setup according to the Arduino IDE operation process described in the preface.



After flashing the program, we can see on the serial port that after the WiFi connection is successful, an activation code will appear:

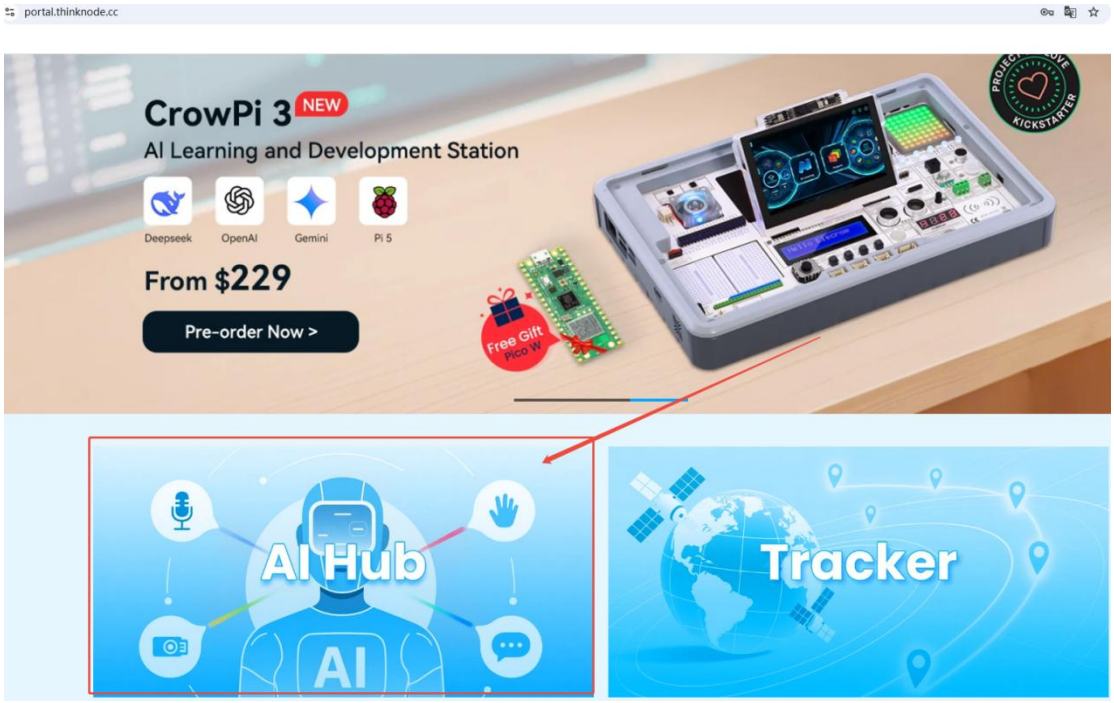


Log in to the Elecrow server website:

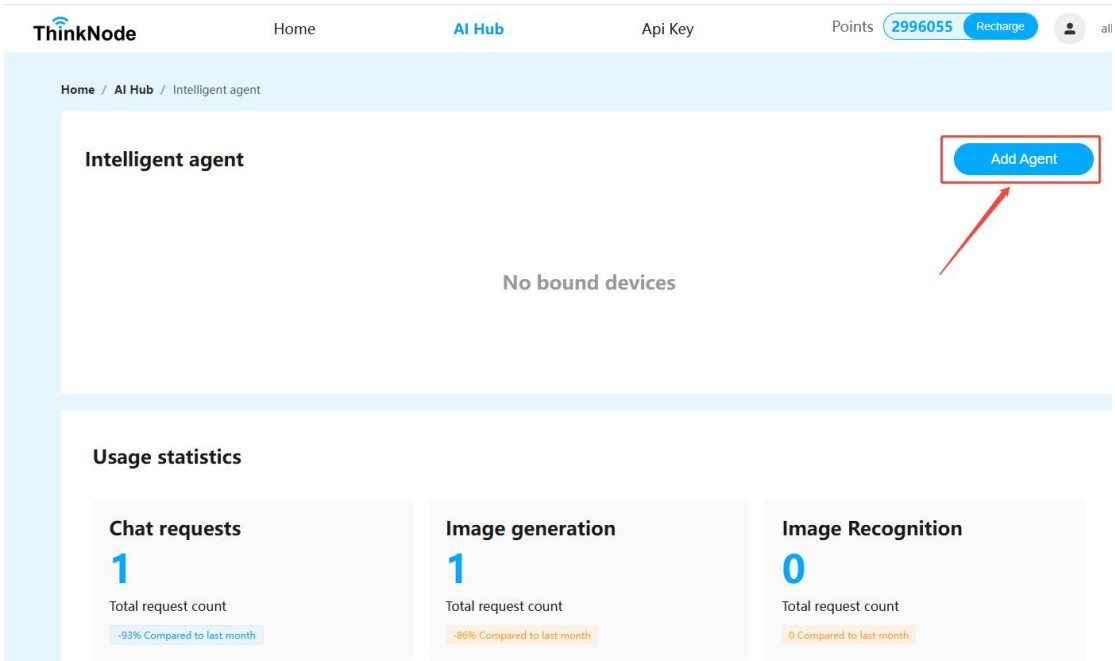
https://portal.thinknode.cc/

If you do not have an account, you can register one and get 5000 points for free use.

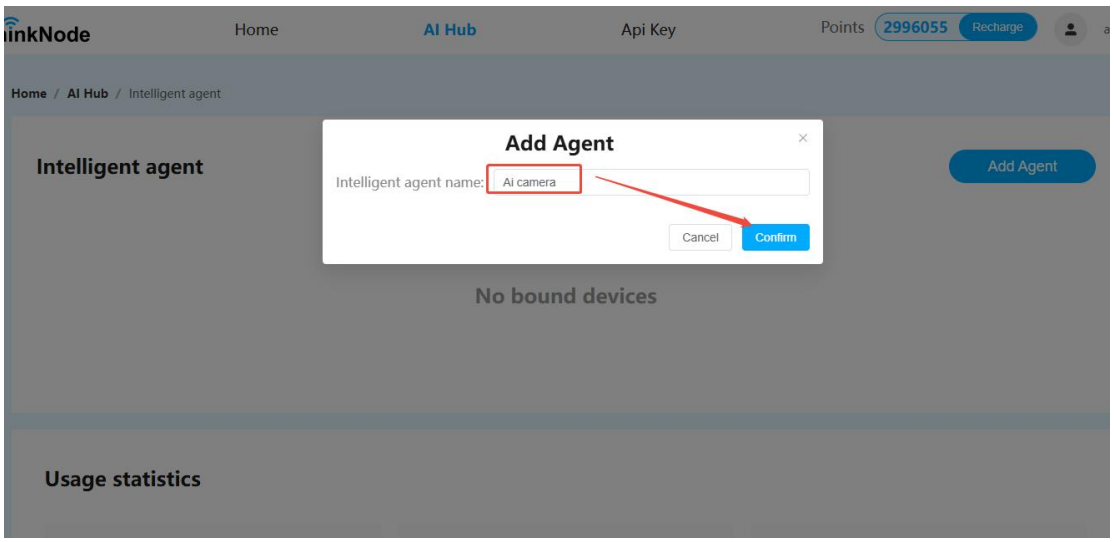
After entering, click AI Hub to enter the AI service window.



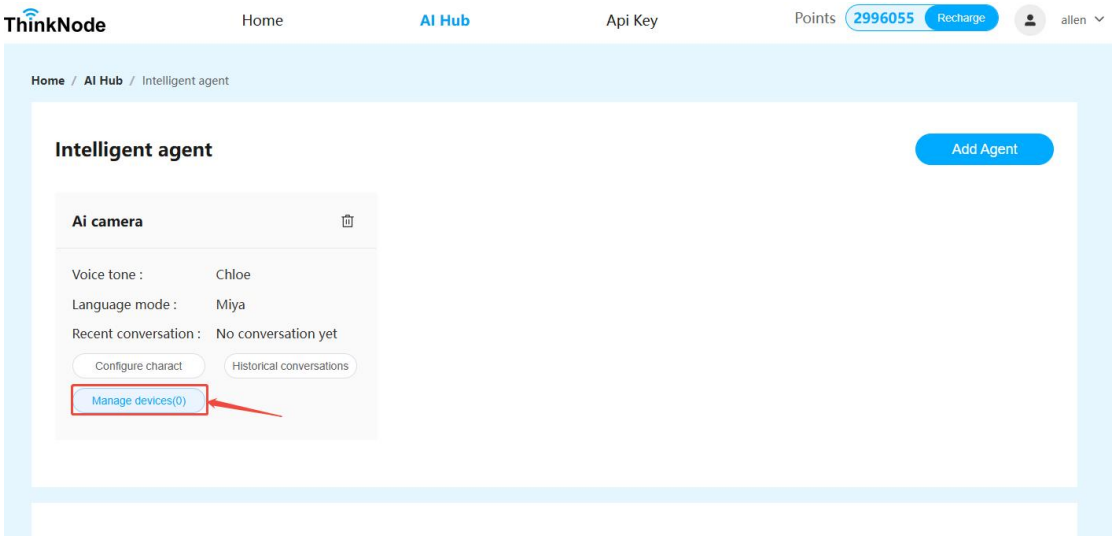
Click "Add Agent"



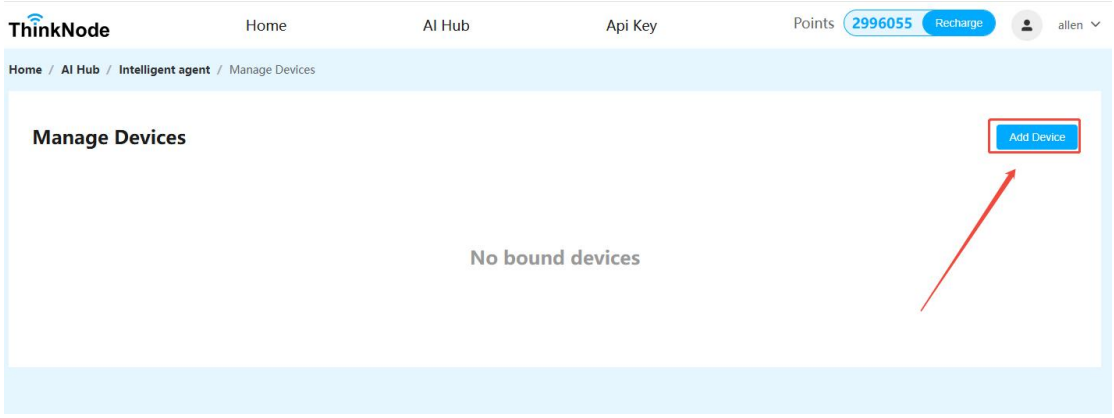
Give your Agent a name



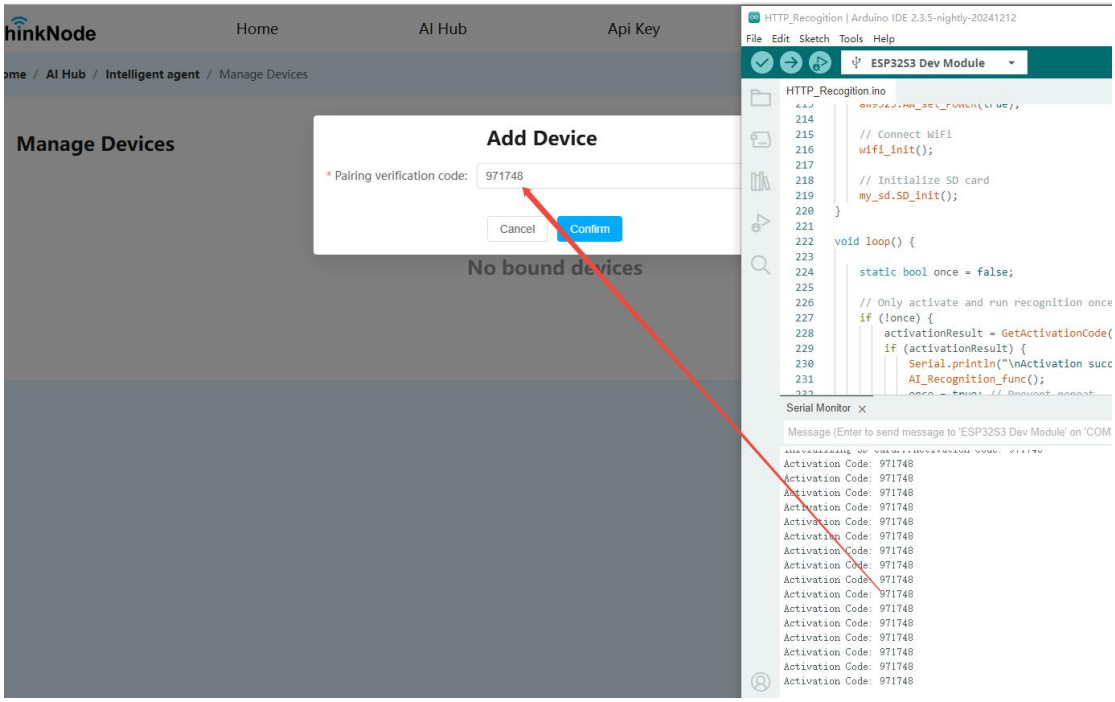
Click "Manage devices" to enter the device management interface



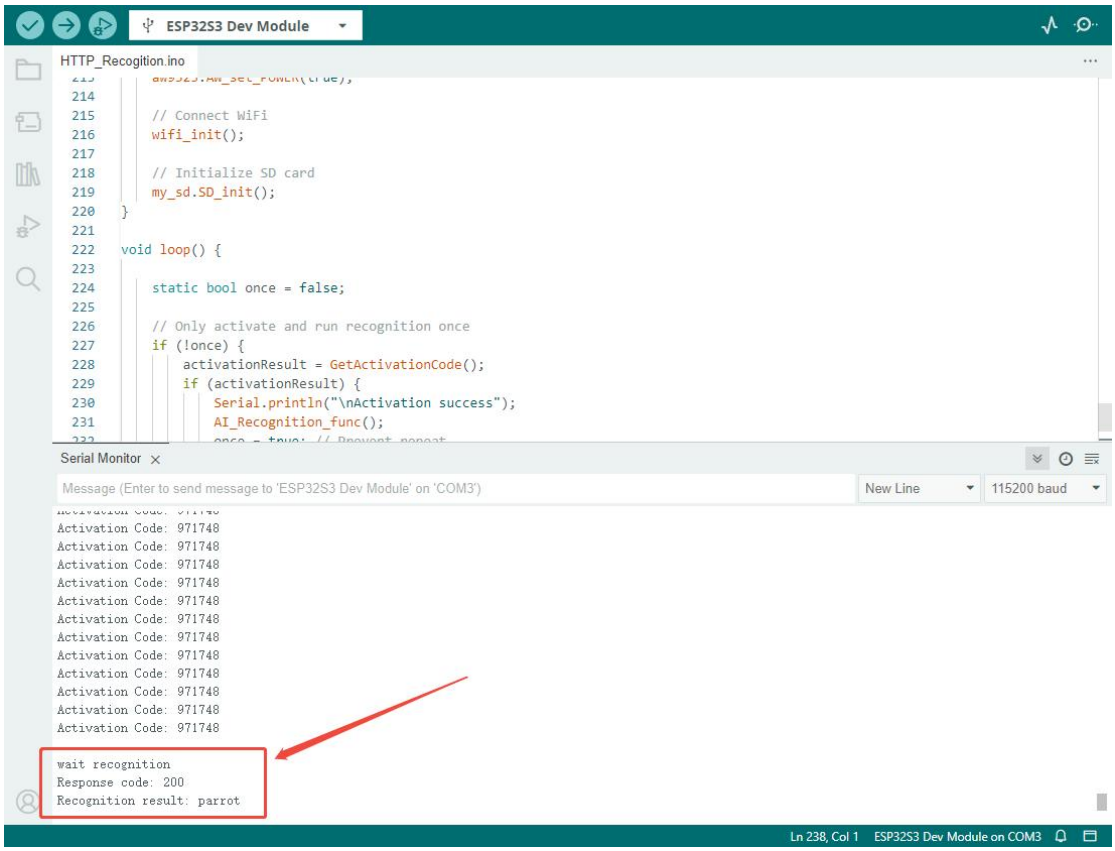
Click "Add Device" to add a device



Enter the activation code seen on the serial port



After clicking "Confirm," recognition is successful, and the result will be printed to the serial port



Lesson 7 — AI Chatbot

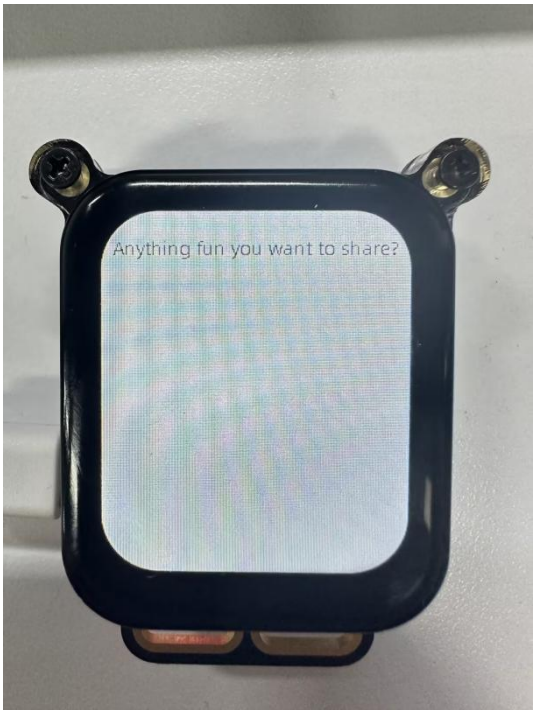
Overview

In this lesson, we will use the microphone and speaker functions of the AI Camera to complete a simple AI chatbot case. You will learn how to connect to the Elecrow server to implement voice acquisition, cloud-based speech recognition, and intelligent dialogue processing, and play back the returned results through the speaker, thereby building a complete AI voice interaction application workflow.

Learning Objectives

1. Master how to use the microphone of the AI camera
2. Master how to use the speaker of the AI camera
3. Complete a simple AI voice interaction application case

Project Running Effect:

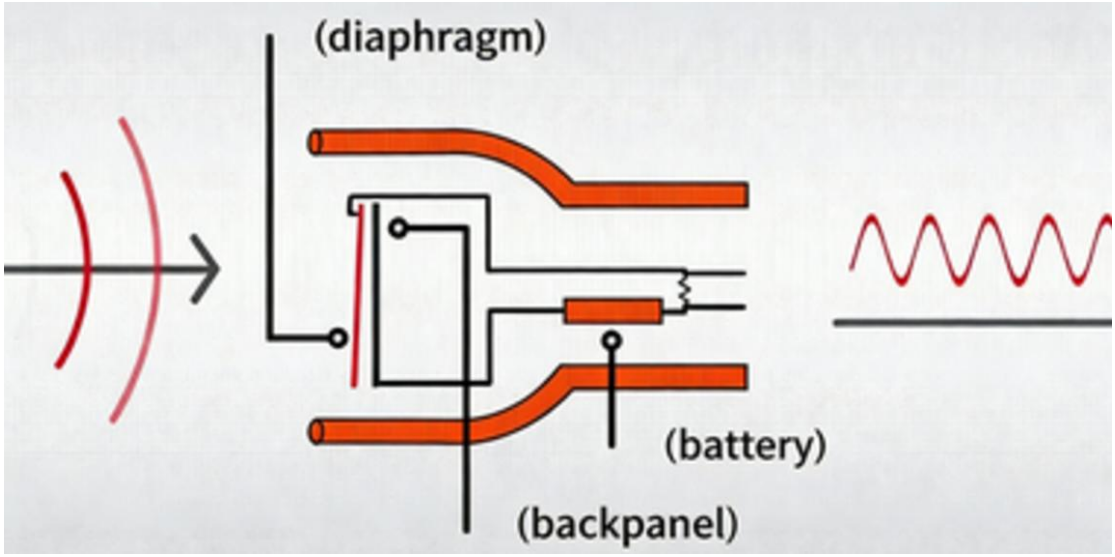


```
Message (Enter to send message to 'ESP32S3 Dev Module' on 'COM3')

role: assistant, content: So happy to see you
role: assistant, content: What's up?
role: assistant, content: Anything fun you want to share?
state changed from 5 to 4
Listening...
role: user, content: To go
state changed from 4 to 5
Speaking...
emotion: embarrassed
role: assistant, content: Oh
role: assistant, content: Are you heading out somewhere fun?
role: assistant, content: That sounds amazing
role: assistant, content: Where to?
state changed from 5 to 4
Listening...
```

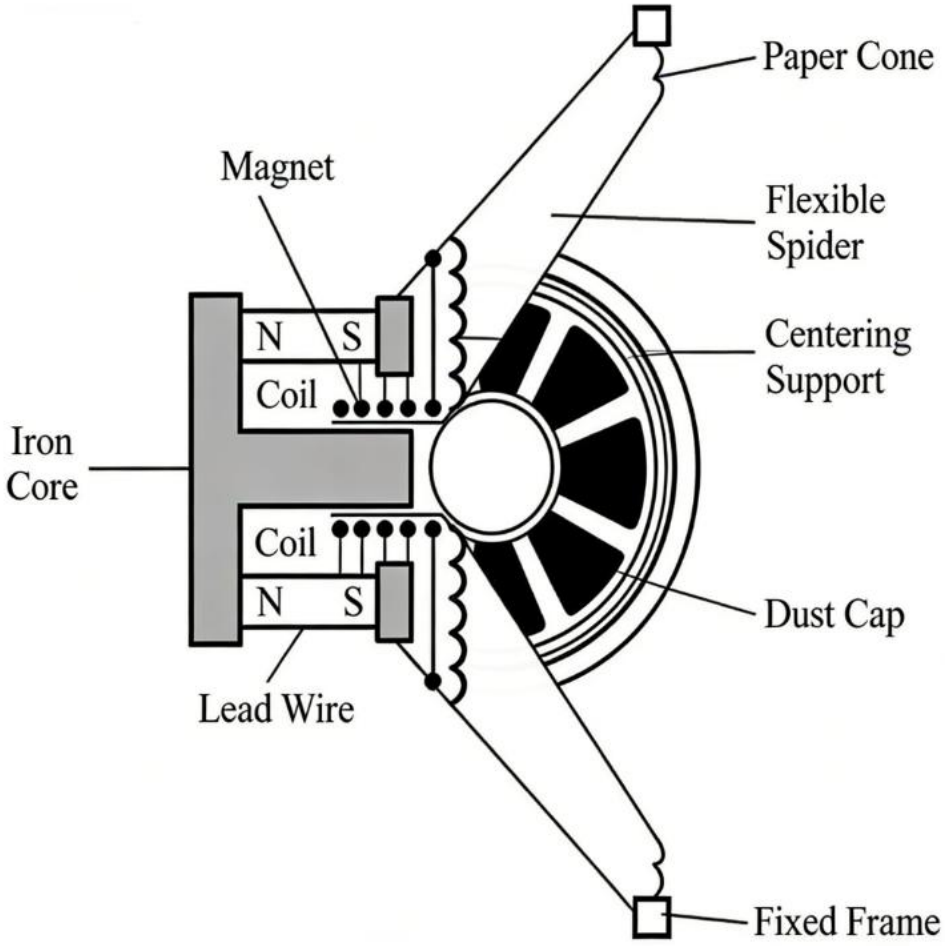
I. Principle Explanation

1. Microphone/Speaker Working Principle

When external sound waves arrive, they push the extremely thin diaphragm of the condenser microphone to vibrate synchronously with the sound waves. This action changes the distance between the diaphragm and the fixed backplate, causing fluctuations in the capacitance formed by the two. The built-in battery of the microphone provides a fixed polarization voltage for this capacitive structure. When the capacitance changes with the sound wave vibration, a weak alternating current consistent with the original sound wave pattern is generated in the circuit. After amplification by subsequent circuits, it is finally output as an audio electrical signal that can be recognized by the device.

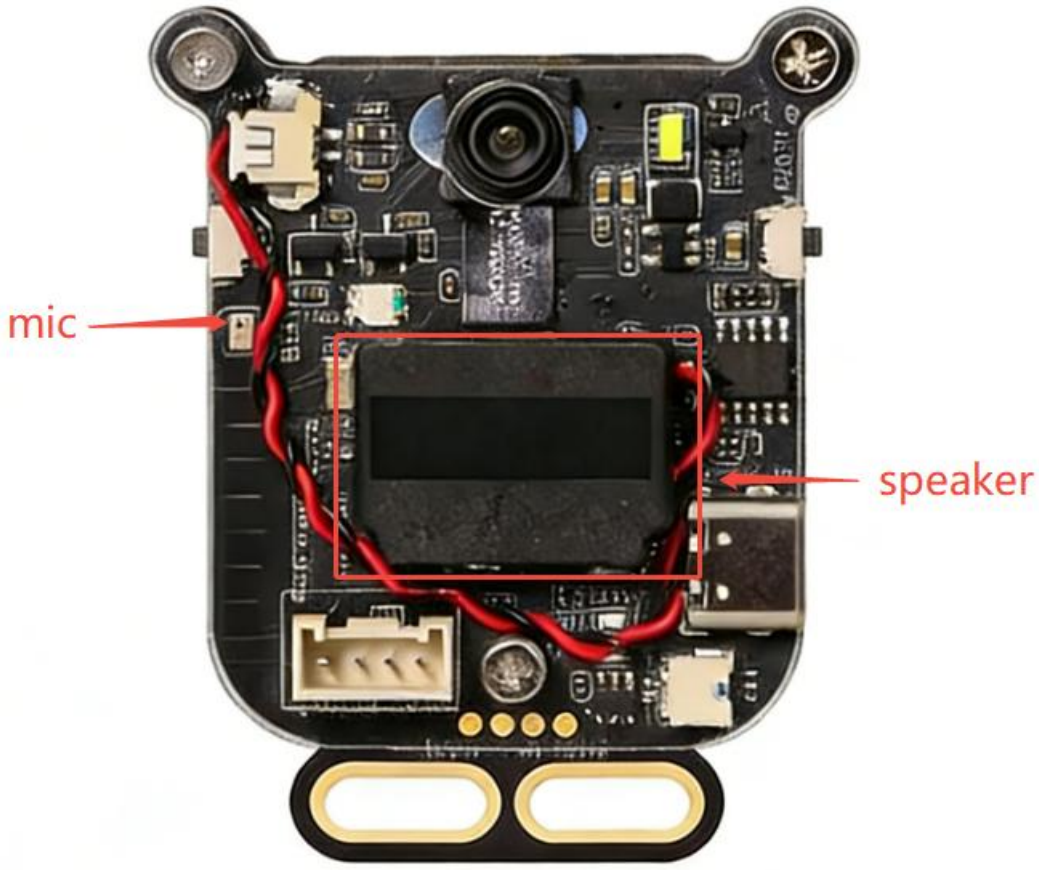


The working principle of the speaker is as follows: the energized coil is placed in the magnetic field formed by the magnet and iron core. When audio current flows through the coil, changes in the current cause the coil to be subjected to an Ampere force that continuously changes in magnitude and direction, thereby driving the diaphragm, centering support, and other connected components to vibrate synchronously. This vibration pushes the surrounding air to produce compressions and rarefactions, ultimately converting electrical signals into sound that can be perceived by the human ear. The centering support and elastic spider structure maintain the stable position of the coil to ensure accurate transmission of vibration.



II. Required Modules

AI Camera MIC and Speaker:



III. Example Explanation

Click the link to download the official example code: (GitHub link)

Use Arduino IDE to open the course program:

ai_voice_display_lcd | Arduino IDE 2.3.5-nightly-20241212

File Edit Sketch Tools Help

ESP32S3 Dev Module

ai_voice_display_lcd.ino

```
#include <Arduino.h>
#include <lvgl.h>
#include <WiFi.h>
#include "src\GFX\LovyanGFX_Driver.h"       // LovyanGFX driver for display
#include "src\AW9523\elecrow_aw9523.h"      // AW9523 I2C LED/IO expander

#include "ai_vox_engine.h"
#include "ai_vox_observer.h"

#include "i2s_pdm_audio_input_device.h"
#include "i2s_std_audio_output_device.h"


// I2C pins for AW9523
#define I2C_SCL 15
#define I2C_SDA 16


// Display resolution
#define LCD_H   284
#define LCD_W   240

// LVGL buffers and drivers
static lv_disp_draw_buf_t draw_buf;
static lv_disp_drv_t disp_drv;
static lv_indev_drv_t indev_drv;

static TwoWire* wi = &Wire;

lv_obj_t * text = NULL;

static std::string g_pending_text;
static volatile bool g_text_ready = false;
bool mute = true;

#ifndef WIFI_SSID
#define WIFI_SSID "yanfa_software"
#endif

#ifndef WIFI_PASSWORD
#define WIFI_PASSWORD "yanfa-123456"
```

Code Explanation

(1) Key Header File Module Explanation

```
#include <lvgl.h>

#include "src\GFX\LovyanGFX_Driver.h"
```

lvgl.h: Advanced UI graphics library

LovyanGFX.h: Library used to drive the LCD to display images

```
#include "ai_vox_engine.h"
#include "ai_vox_observer.h"
```

Core AI voice engine library, mainly responsible for voice acquisition, cloud communication, speech recognition, speech synthesis, and event callbacks.

```
#include "i2s_pdm_audio_input_device.h"
#include "i2s_std_audio_output_device.h"
```

The PDM microphone is mainly used for voice acquisition, and the I2S speaker is mainly used for voice playback.

(2) Define I2C & LCD Parameters

```
#define I2C_SCL 15
#define I2C_SDA 16
```

Define the I2C bus pins. AW9523 communicates via I2C.

```
#define LCD_H     284
#define LCD_W     240
```

Define the screen resolution as 240*284.

(3) LVGL Core Objects

```
static lv_disp_draw_buf_t draw_buf;
static lv_disp_drv_t disp_drv;
static lv_indev_drv_t indev_drv;
```

The three core LVGL structures:

draw_buf: display buffer

disp_drv: display driver

indev_drv: input device driver

```
lv_obj_t * text = NULL;
```

UI text control pointer, used later to display what the AI says.

(4) Thread Communication Variables

```
static std::string g_pending_text;
```

```
static volatile bool g_text_ready = false;
```

Cross-thread communication. The AI thread stores the string, and the UI thread reads and displays it.

```
bool mute = true;
```

Whether it is currently muted. Initialized as muted and enabled when needed.

(5) WiFi Macro Definitions

```
#ifndef WIFI_SSID
#define WIFI_SSID "yanfa_software"
#endif

#ifndef WIFI_PASSWORD
#define WIFI_PASSWORD "yanfa-123456"
#endif
```

Define the WiFi to connect to. Modify according to the available WiFi.

(6) Display Refresh Callback (Most Critical)

```
void my_disp_flush(...)
```

This is the core interface for LVGL display. After LVGL finishes drawing a region, it calls this function, and you are responsible for refreshing the screen.

```
if (gfx.getStartCount() > 0) gfx.endWrite();
```

DMA protection to prevent conflicts.

```
gfx.pushImageDMA(...)
```

Use DMA to push pixel data into the LCD.

```
lv_disp_flush_ready(disp);
```

Notify LVGL: the refresh is complete, you can continue drawing.

(7) Touch Read Callback

```
void my_touchpad_read(...)
```

This function determines whether the screen is currently being touched.

```
data->state = LV_INDEV_STATE_REL;
```

Default state is not pressed.

```
if (gfx.getTouch(&touchX, &touchY))
```

If a touch is detected

```
data->state = LV_INDEV_STATE_PR;
```

Then mark it as pressed.

```
data->point.x = 240 - touchX;
data->point.y = touchY;
```

Since the screen orientation is opposite to the touch coordinates, we use coordinate mapping to resolve the orientation issue.

(8) AI Hardware Pin Definitions

```
constexpr gpio_num_t kMicPdmClk = GPIO_NUM_1;
constexpr gpio_num_t kMicPdmDin = GPIO_NUM_2;
```

PDM microphone clock and data.

```
constexpr gpio_num_t kSpeakerPinBclk = GPIO_NUM_5;
constexpr gpio_num_t kSpeakerPinWs   = GPIO_NUM_4;
constexpr gpio_num_t kSpeakerPinDout = GPIO_NUM_3;
```

I2S speaker three wires.

```
constexpr gpio_num_t kTriggerPin = GPIO_NUM_0;
```

Voice trigger button, the boot button on the side of the AI camera.

(9) AI Object Creation

```
auto g_observer = std::make_shared<ai_vox::Observer>();
```

Create AI event receiver object.

```
auto g_audio_output_device = ...
```

Create speaker output object.

```
auto audio_input_device = ...
```

Create microphone acquisition object.

(10) UI Thread Task Function (Key)

```
void lvgl_task(void *pvParameters)
```

Dedicated independent thread responsible for UI.

```
if (g_text_ready)
```

If the AI thread sends new text

```
lv_label_set_text(text, g_pending_text.c_str());
```

Refresh the UI display content.

```
lv_timer_handler();
```

Core LVGL scheduling function.

```
vTaskDelay(pdMS_TO_TICKS(5));
```

Yield the CPU to prevent freezing.

(11) Initialization Function (Most Important)

```
Serial.begin(115200);
WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
while(WiFi.status() != WL_CONNECTED)
```

Initialize serial debugging, start connecting to WiFi, and wait until connected.

```
wi->setPins(I2C_SDA, I2C_SCL);
wi->begin();
aw9523.AW_init();
aw9523.AW_set_POWER(true);
```

Initialize the I2C bus, initialize the IO expander chip, and power it on.

```
aw9523.AW_set_MUTE(true);
```

```
gfx.init();
gfx.initDMA();
lv_init();
```

Enable mute, initialize LCD and DMA, and initialize the LVGL system.

```
heap_caps_malloc(... MALLOC_CAP_SPIRAM);
lv_disp_draw_buf_init(...)
lv_disp_drv_register(&disp_drv);
text = lv_label_create(lv_scr_act());
lv_label_set_text(text, "Idle");
```

Use PSRAM as video memory, initialize the LVGL display buffer, register the display driver, and create the UI text control. Initialize the display to "Idle".

(12) AI Engine Initialization

```
auto& ai_vox_engine = ai_vox::Engine::GetInstance();
```

Obtain the global AI engine instance.

```
ai_vox_engine.SetObserver(g_observer);
```

Register the event receiver.

```
ai_vox_engine.SetTrigger(kTriggerPin);
```

Bind the voice wake-up button.

```
ai_vox_engine.SetOtaUrl(...)
```

Firmware upgrade server address.

```
ai_vox_engine.ConfigWebsocket(...)
```

Configure the real-time AI channel.

```
ai_vox_engine.Start(...)
```

Start the complete AI voice system.

(13) Create UI Thread

```
xTaskCreatePinnedToCore(lvgl_task,...,1);
```

Pin it to CPU1 to prevent resource contention with AI.

(14) Loop Breakdown Line by Line (Core of Event System)

```
const auto events = g_observer->PopEvents();
```

Retrieve the AI event queue.

```
ActivationEvent
```

Device first activation.

```
StateChangedEvent
```

Voice system state machine.

```
EmotionEvent
```

AI recognizes emotion.

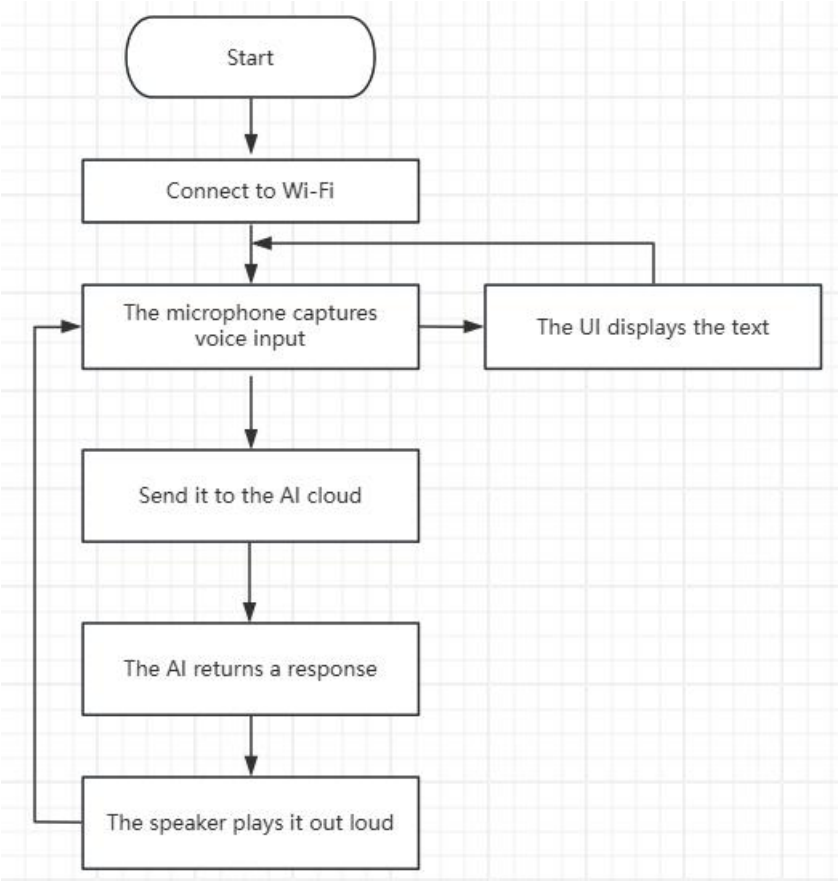
```
ChatMessageEvent
```

Display AI or user dialogue.

```
IotMessageEvent
```

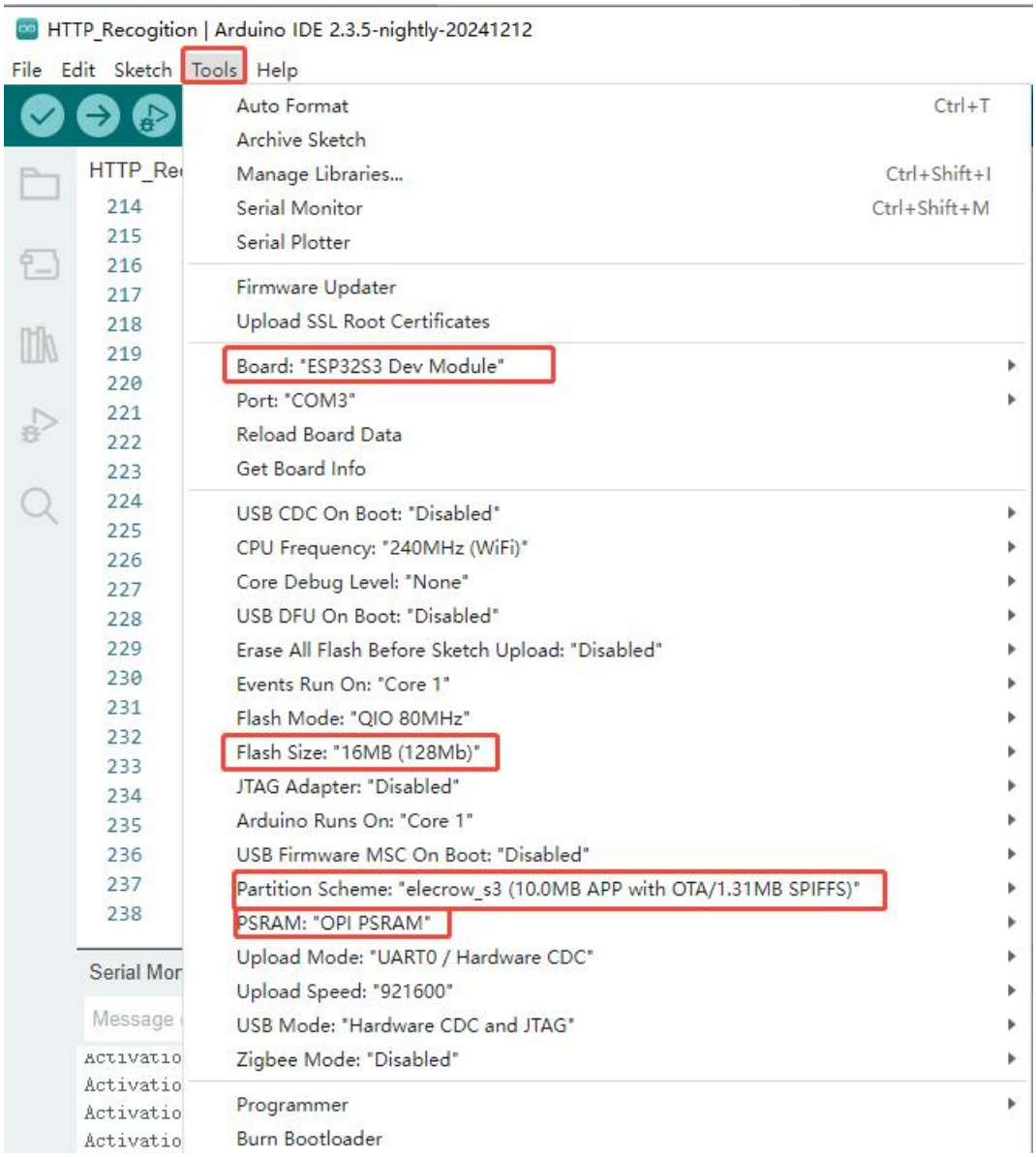
AI controls real hardware.

Program Flowchart:

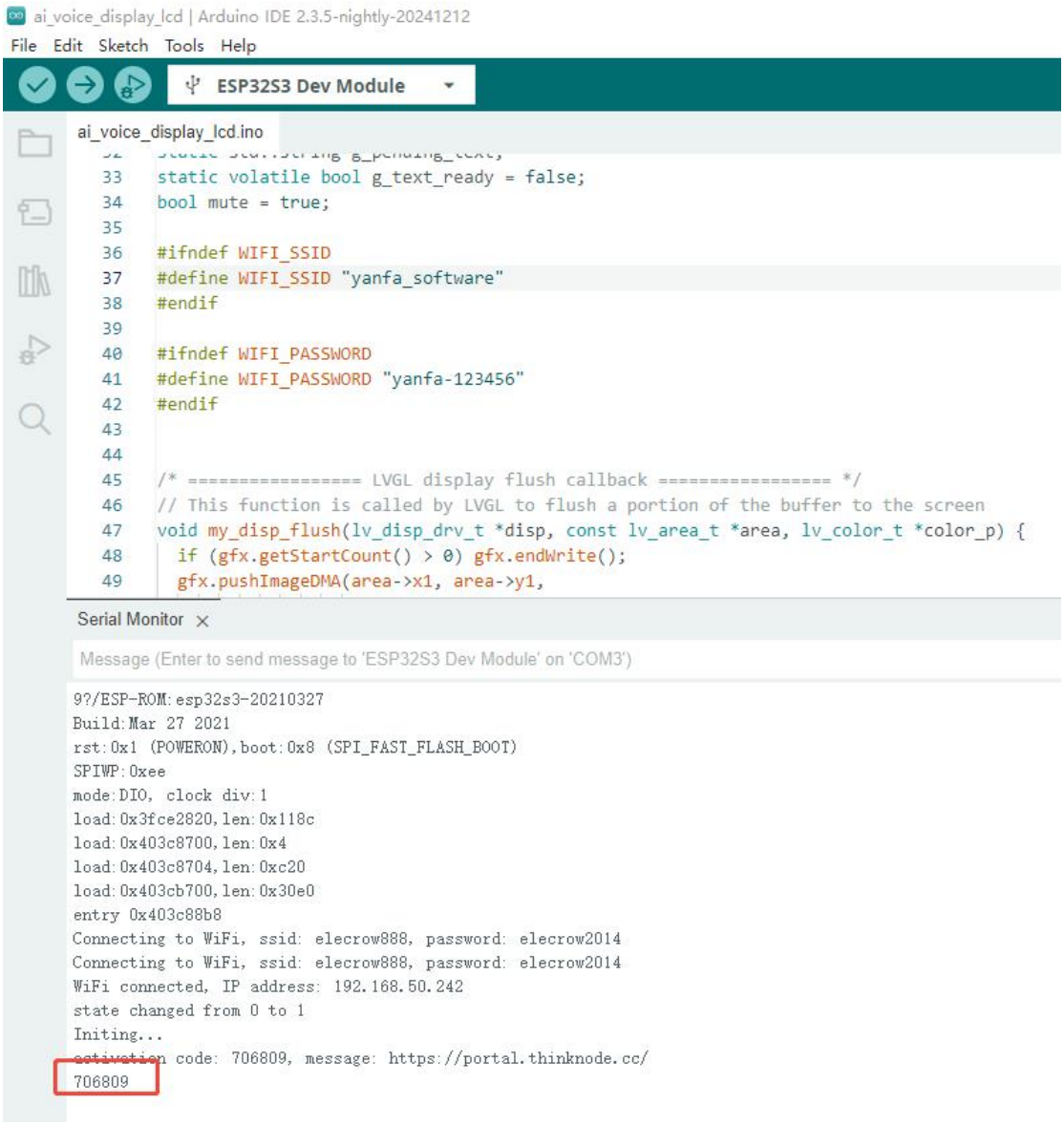


IV. Download and Run the Program

(1) Connect to the computer according to the Arduino IDE operation process introduced in the preface and perform the basic download settings.



After flashing the program, we can see an activation code appear on the serial monitor once the WiFi connection is successful:

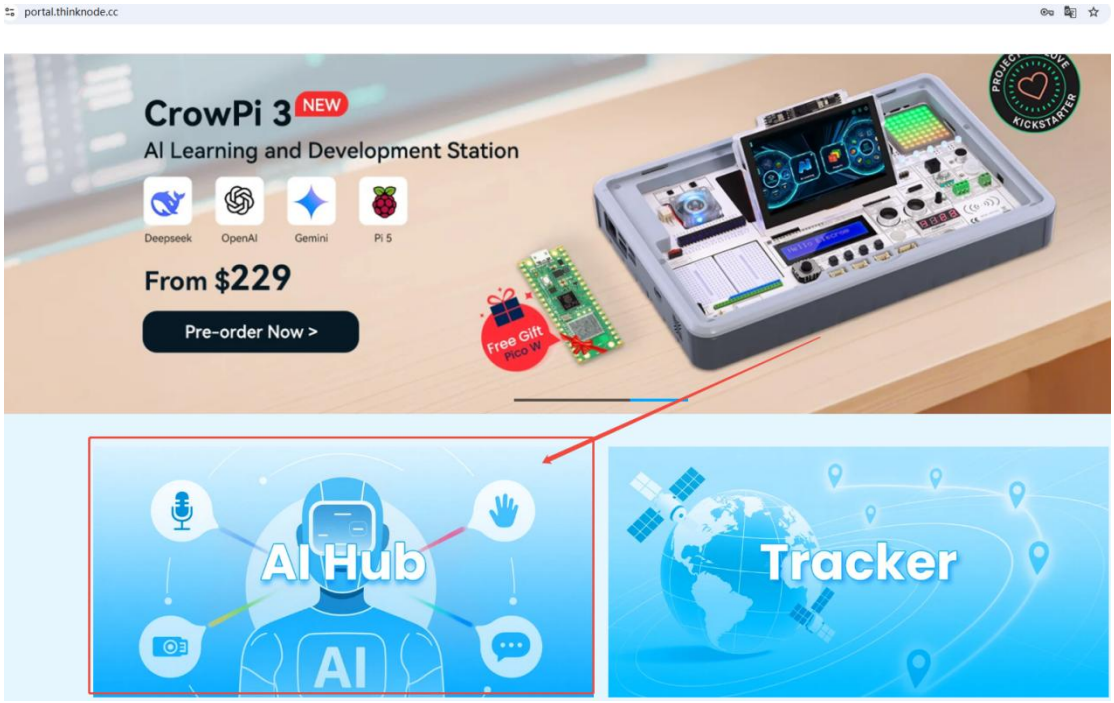


Log in to the Elecrow server website:

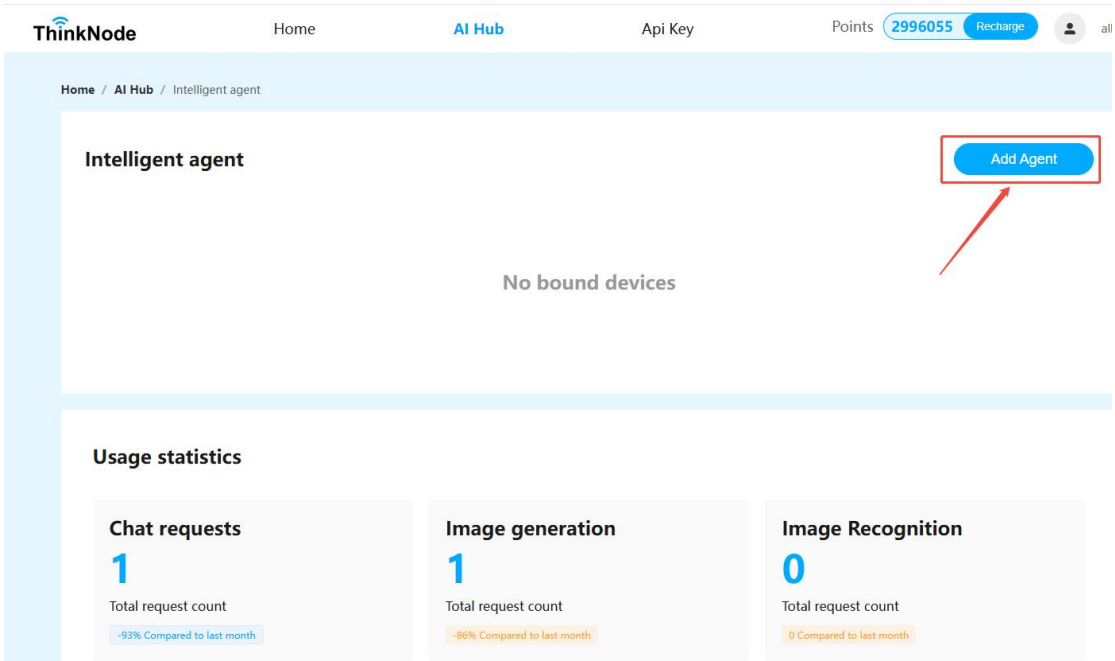
https://portal.thinknode.cc/

If you do not have an account, you can register one, and you will receive 5000 points for free use.

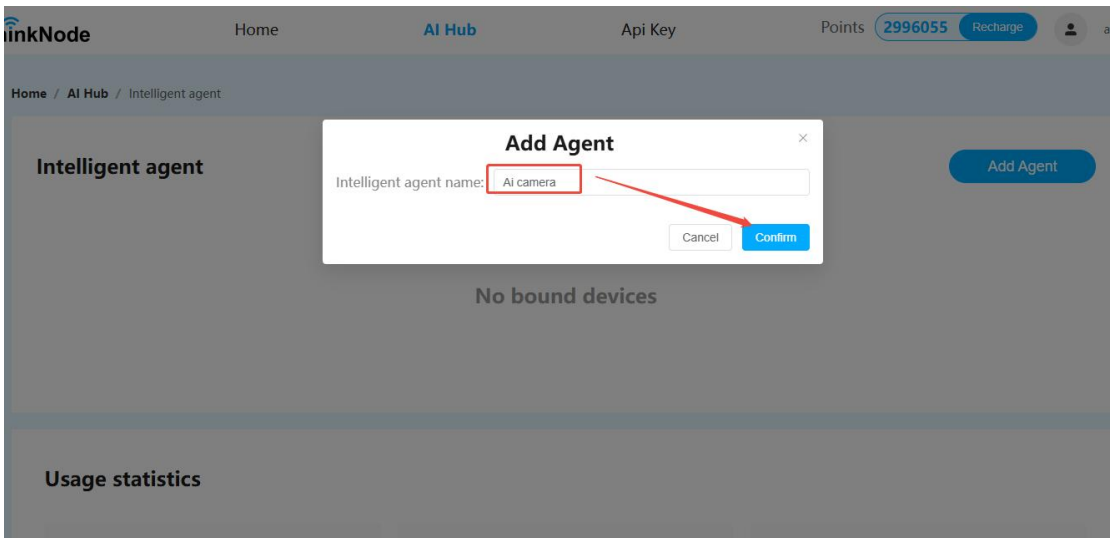
After entering, click AI Hub to access the AI service interface.



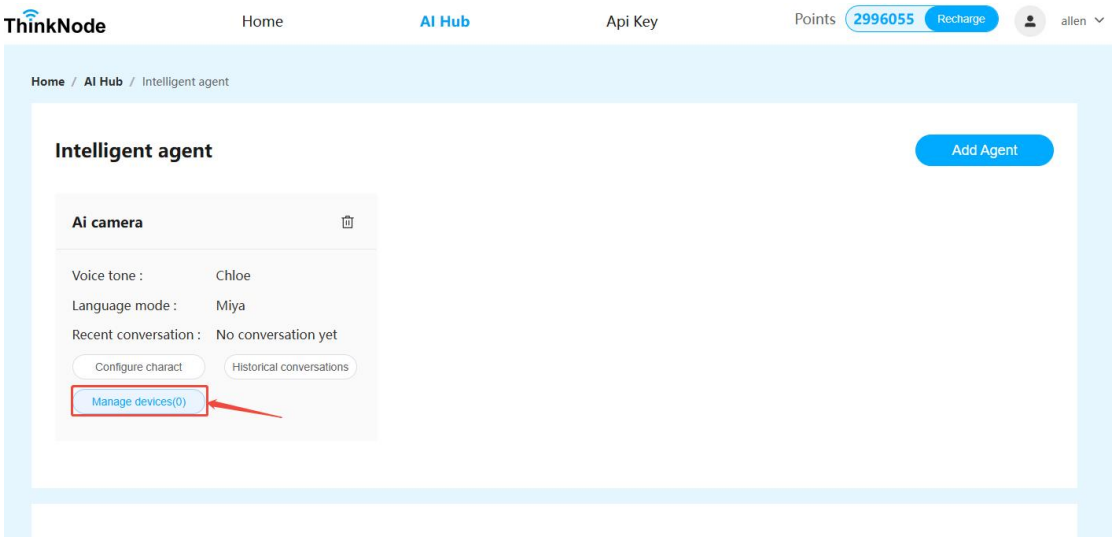
Click "Add Agent".



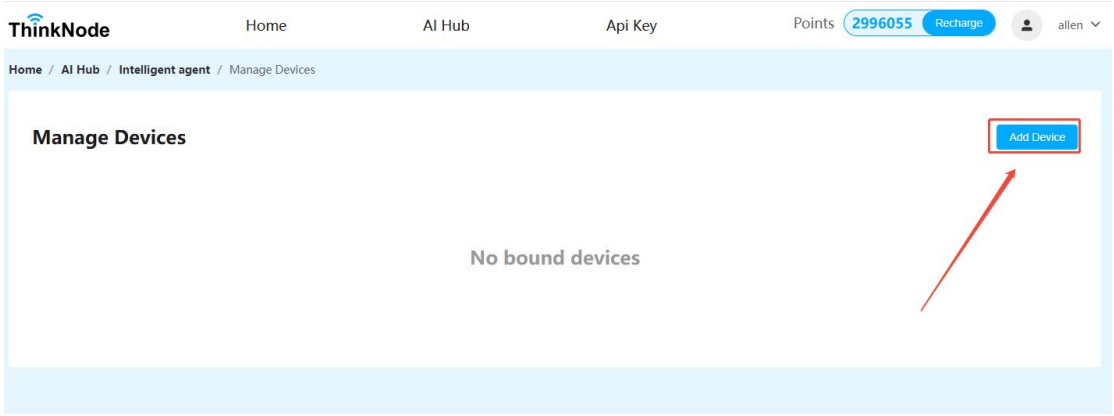
Give your Agent a name.



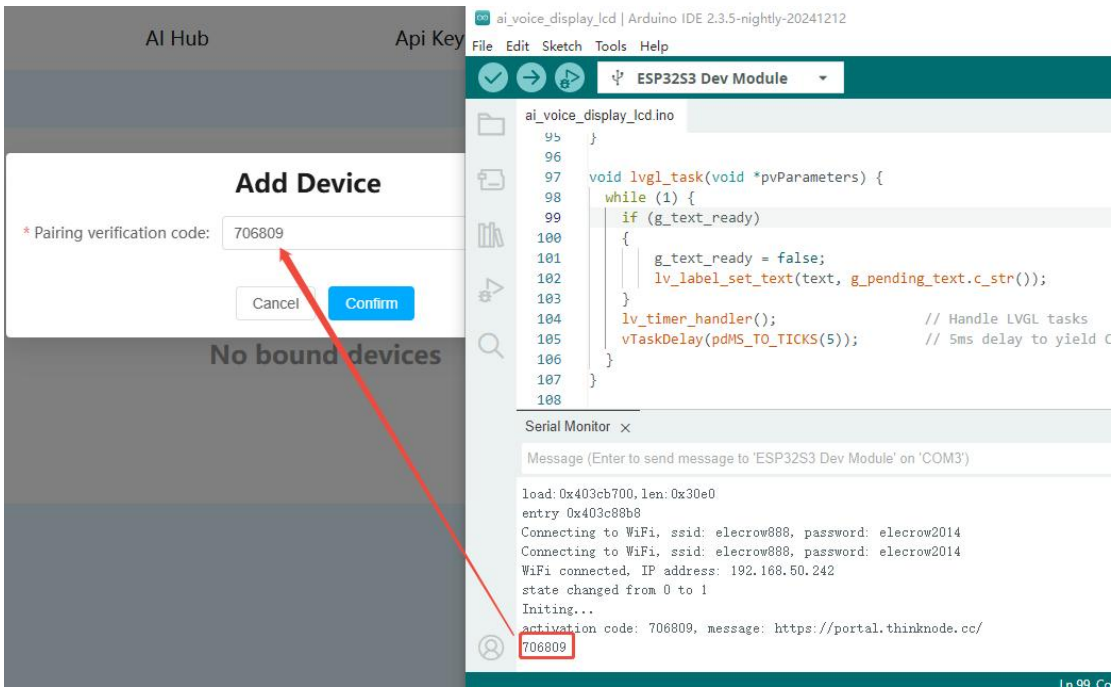
Click "Manage devices" to enter the device management interface.



Click "Add Device" to add a device.



Enter the activation code displayed on the serial monitor.



After clicking "Confirm", press the "boot" button on the side of the AI camera to start the device, and you can begin the conversation.