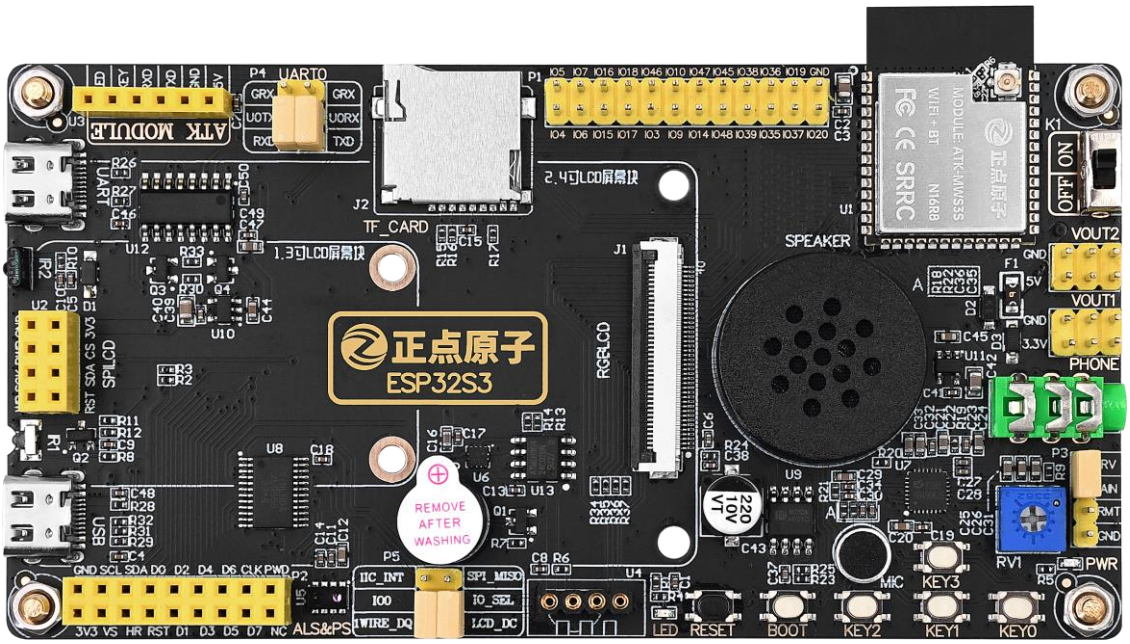
Espressif_IDE 使用说明

V1.0



-正点原子 DNESP32S3 开发板教程

注：本教程仅适用于 DNESP32S3 开发板

修订历史:

版本	日期	修改内容
V1.0	2024/3/22	第一次发布

正点原子公司名称 ：广州市星翼电子科技有限公司
原子哥在线教学平台：www.yuanzige.com
开源电子网 / 论坛 ：www.openedv.com
正点原子官方网站 ：www.alientek.com
正点原子淘宝店铺 ：https://openedv.taobao.com
正点原子 B 站视频 ：https://space.bilibili.com/394620890

电话：020-38271790 传真：020-36773971

请下载原子哥 APP，数千讲视频免费学习，更快更流畅。
请关注正点原子公众号，资料发布更新我们会通知。

扫码下载“原子哥”APP

扫码关注正点原子公众号

前言

Espressif_IDE 软件是一款基于 Eclipse CDT 的集成开发环境（IDE），专为乐鑫物联网开发框架 ESP-IDF 打造。它支持用户使用 ESP-IDF 实现端到端物联网应用开发，并附带了 IDF Eclipse 插件、重要的 Eclipse CDT 插件以及 Eclipse 平台上的其他第三方插件，以支持构建 ESP-IDF 应用程序。

Espressif_IDE 的主要特性包括易于使用的界面、专为 ESP-IDF 应用程序开发而打造的定制功能、自动配置编译环境变量、提供新建项目向导以及 ESP-IDF 快速入门模板等。此外，它还具备领先的编辑、编译以及语法着色功能，支持预建的函数头和函数定义导航，以及安装和配置全新的或现有的 ESP-IDF。

在使用 Espressif_IDE 进行开发时，用户需要先下载并安装软件包。安装完成后，IDE 会自动配置所需的环境变量和工具路径，无需手动设置。用户可以通过查看环境变量是否存在来确保安装成功，并可以进一步了解 IDE 的图标功能和其他使用细节。

Espressif_IDE 的使用对于新手来说是非常友好的，它简化了开发环境的搭建过程，使得用户能够更专注于物联网应用的开发。同时，官方也提供了丰富的资料和文档，帮助用户更好地理解和使用这款强大的开发工具。

第一章 Espressif_IDE 软件介绍

关于 ESP 系列芯片的开发，存在多种途径，如 ESP_IDF、Arduino 和 MicroPython 等。这些开发手段的核心实现均基于乐鑫公司官方推出的物联网开发框架——IDF。在本教程中，我们推荐并采用 VS Code 结合 ESP_IDF 插件的方式进行开发工作。此外，乐鑫官方也提供了 Espressif IDE 开发环境，用户只需安装该软件，即可开始对 ESP 系列芯片进行开发。本教程将重点阐述如何使用 Espressif IDE 进行开发的详细步骤和注意事项。

本章节将分为以下几个小节：

1.1 Espressif_IDE 软件介绍
1.2 Espressif_IDE 软件下载及安装
1.3 Espressif_IDE 界面介绍

1.1 Espressif_IDE 软件介绍

Espressif_IDE 是一个基于 Eclipse CDT 的集成开发环境（IDE），专为 ESP-IDF 框架设计的物联网应用程序开发而打造。它作为一个独立且定制的 IDE，集成了 IDF Eclipse 插件、关键的 Eclipse CDT 插件以及其他第三方插件，旨在提供全面支持以构建 ESP-IDF 应用程序。Espressif_IDE 的简图如下所示。

图 1.1.1 Espressif IDE 软件简图

其特点如下：

- 基于 Eclipse CDT 环境构建的易于使用的 IDE
- 专门为 ESP-IDF 应用程序开发而构建
- 自动配置环境变量
- 集成工具链配置
- 新的项目向导和 ESP-IDF 开始示例
- 具有语法着色功能的高级编辑、编译和编辑
- 预构建的函数头和函数定义导航
- 直接从 IDE 安装和配置 ESP-IDF 和 ESP-IDF 工具
- 用于项目特定设置的 SDK 配置编辑器
- 用于编辑 CMakeLists.txt 等 CMake 文件的集成 CMake 编辑器插件
- 基于 CMake 的构建支持
- 支持 UART 和 JTAG 烧写
- 使用预构建的配置和设置的自定义 ESP-IDF OpenOCD 调试
- GDB 硬件调试
- 集成 ESP-IDF 串口监视器
- 带有预配置 ESP-IDF 构建环境的 ESP-IDF 终端
- 应用程序大小分析编辑器，用于分析应用程序的静态内存占用

- 支持堆分析，用于内存分析和查找内存泄漏
- 支持 GDB Stub 调试和应用程序级跟踪
- 支持 esp32、esp32s2、esp32s3 和 esp32c3 芯片
- IDE 的英文和中文支持
- 用于 Eclipse 生态系统中其他第三方插件的可扩展 IDE
- 支持的主机操作系统：Windows、macOS 和 Linux

1.2 Espressif_IDE 软件下载及安装

1.2.1 Espressif_IDE 软件下载

Espressif_IDE 软件是与 ESP-IDF 软件包捆绑下载的，因此，我们需要访问 ESP-IDF 软件包的下载网址来获取它。对于 Windows 版本，开发者可以通过以下链接下载 ESP-IDF 软件包：https://dl.espressif.cn/dl/esp-idf/。其下载界面如图所示：。

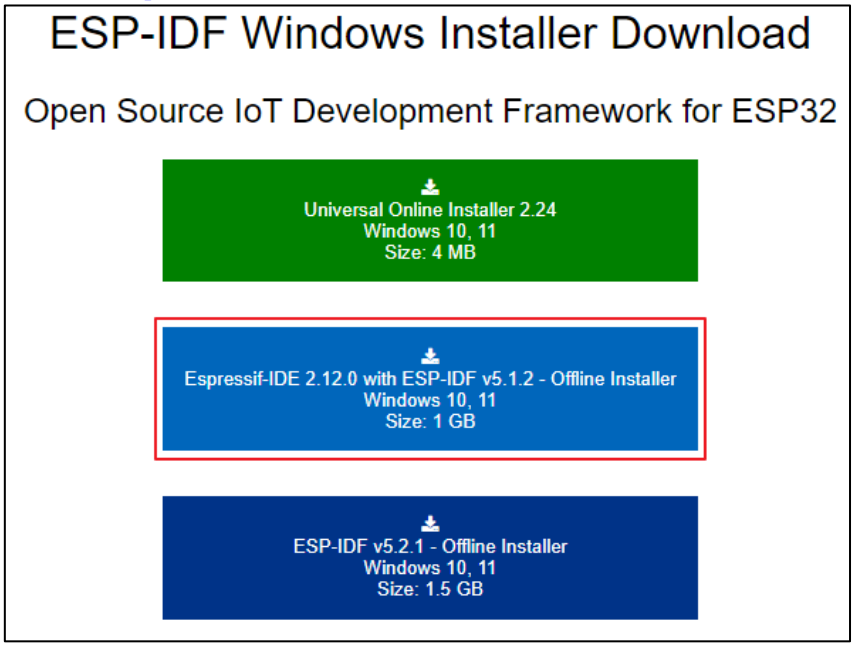


图 1.2.1.1 ESP-IDF 安装包

在上图中，只有红框标注的版本“Espressif-IDE 2.12.0 with ESP-IDF v5.1.2 – Offline Installer”包含 Espressif-IDE 安装包。如果开发者希望下载其他版本的 Espressif-IDE，可以访问以下网址：https://github.com/espressif/idf-installer?tab=readme-ov-file#espressif-ide-offline-installer。在该网址下，开发者可以找到 Espressif_IDE 软件包的下载选项，其界面展示如下。

Version	Content	Size
Espressif-IDE v2.12.0 and ESP-IDF 5.1.2 download/mirror	Espressif-IDE 2.12.0 + JDK 17 + ESP-IDF v5.1.2	1 GB
Espressif-IDE v2.11.1 and ESP-IDF 5.1.2 download/mirror	Espressif-IDE 2.11.1 + JDK 17 + ESP-IDF v5.1.2	1 GB
Espressif-IDE v2.11.0 and ESP-IDF 5.1.1 download/mirror	Espressif-IDE 2.11.0 + JDK 17 + ESP-IDF v5.0.2	1 GB
Espressif-IDE v2.11.0 and ESP-IDF 5.0.1 download/mirror	Espressif-IDE 2.11.0 + JDK 17 + ESP-IDF v5.0.2	1 GB
Espressif-IDE v2.10.0 and ESP-IDF 5.0.2 download/mirror	Espressif-IDE 2.10.0 + JDK 17 + ESP-IDF v5.0.2	1 GB
Espressif-IDE v2.10.0 and ESP-IDF 5.0.1 download/mirror	Espressif-IDE 2.10.0 + JDK 17 + ESP-IDF v5.0.1	1 GB
Espressif-IDE v2.9.1 and ESP-IDF 5.0.1 download/mirror	Espressif-IDE 2.9.1 + JDK 17 + ESP-IDF v5.0.1	1 GB
Espressif-IDE v2.9.0 and ESP-IDF 5.0.1 download/mirror	Espressif-IDE 2.9.0 + JDK 17 + ESP-IDF v5.0.1	1 GB
Espressif-IDE v2.8.1 and ESP-IDF 5.0 download/mirror	Espressif-IDE + JDK + ESP-IDF v5.0	1 GB
Espressif-IDE v2.8.1 and ESP-IDF 4.4.4 download/mirror	Espressif-IDE + JDK + ESP-IDF v5.0	1 GB
Espressif-IDE v2.8.0 and ESP-IDF 5.0 download/mirror	Espressif-IDE + JDK + ESP-IDF v5.0	1 GB
Espressif-IDE v2.8.0 and ESP-IDF 4.4.4 download/mirror	Espressif-IDE + JDK + ESP-IDF v5.0	1 GB
Espressif-IDE v2.7.0 and ESP-IDF 5.0 download/mirror	Espressif-IDE + JDK + ESP-IDF v5.0	1 GB
Espressif-IDE v2.7.0 and ESP-IDF 4.4.3 download/mirror	Espressif-IDE + JDK + ESP-IDF v4.4.3	1 GB
Espressif-IDE v2.6.0 download/mirror	Espressif-IDE + JDK + ESP-IDF v4.4.2	1 GB
Espressif-IDE v2.5.0 download/mirror	Espressif-IDE + JDK + ESP-IDF v4.4	1 GB
Espressif-IDE v2.4.2 download/mirror	Espressif-IDE + JDK + ESP-IDF v4.4	1 GB
Espressif-IDE v2.4.1 download/mirror	Espressif-IDE + JDK + ESP-IDF v4.4	1 GB
Espressif-IDE v2.4.0 download/mirror	Espressif-IDE + JDK + ESP-IDF v4.4	1 GB

图 1.2.1.2 Espressif_IDE 软件包下载列表

通过上图的 Content 部分，您可以清晰地了解到 Espressif_IDE 不同版本所包含的具体内容。例如，“Espressif-IDE v2.12.0 and ESP-IDF 5.1.2”这个版本中，就包含了 Espressif_IDE 2.12.0、JDK17 以及 ESP-IDF v5.1.2 等组件。这样的信息展示有助于开发者快速选择适合自己需求的版本。采用 VS Code 与 ESP-IDF 插件进行开发时，您需要先下载 ESP-IDF 软件包，接着安装 VS Code，最后在 VS Code 中安装 ESP-IDF 插件。相比之下，本教程所介绍的 Espressif_IDE 开发方式则更为便捷，仅需安装软件包即可立即开始开发。

Espressif_IDE 软件的下载步骤已在图中详细展示，供您参考。

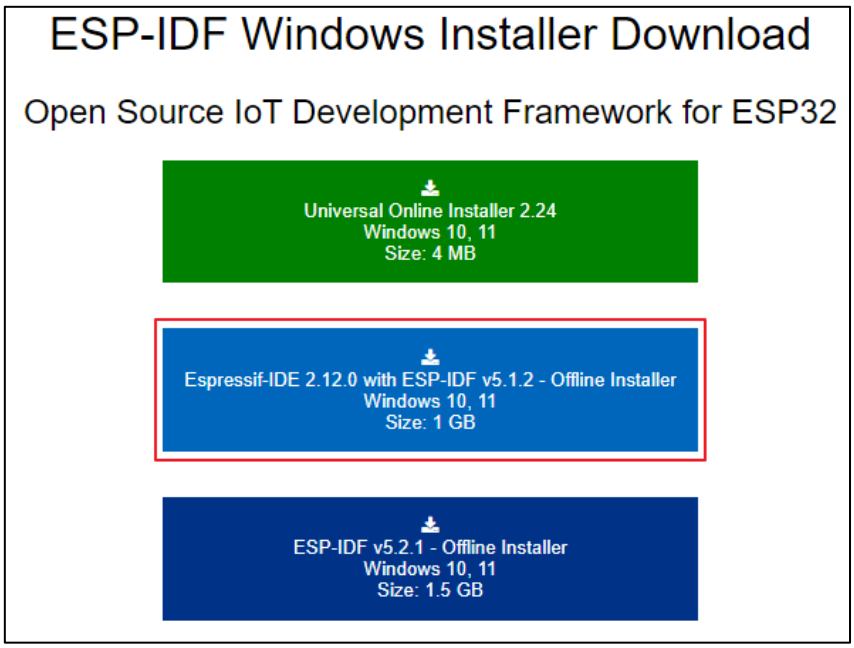


图 1.2.1.3 Espressif_IDE 软件下载步骤

首先，我们需要点击选择带有 Espressif-IDE 的软件包进行下载，并将其保存到预先设定的目录下。另外，若开发者已拥有 ESP32-S3 开发板的资料盘，则可直接在其中找到该下载包，无需再次下载。该下载包的路径为：“A 盘→6，软件资料→1，IDF 开发工具→05-Espressif_IDE Offline Installer”，具体路径请参照图示。

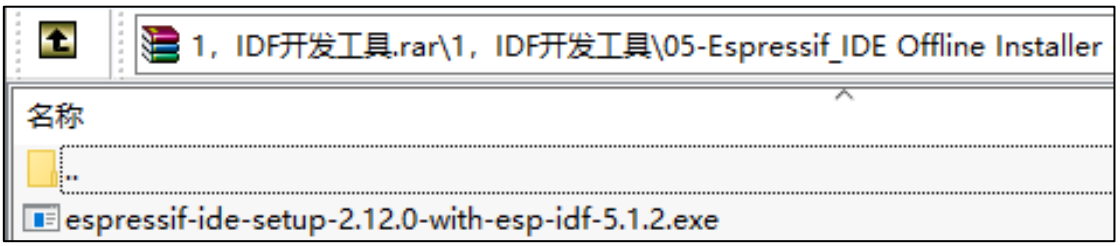


图 1.2.1.4 开发板资料盘下的 Espressif_IDE 软件包

1.2.2 Espressif_IDE 软件安装

Espressif_IDE 安装程序是一款便捷的离线安装工具，它集成了 ESP-IDF 应用开发所需的全部核心组件。这些组件包括嵌入式 Python、交叉编译器、开放 OCD、CMake 和 Ninja 构建工具、ESP-IDF 框架、Espressif_IDE 本身以及 Amazon Corretto OpenJDK。通过这一安装程序，您可以一次性获取所有必要的组件和工具，轻松完成安装，进而顺利开展方案开发。在 IDE 启动时，它会自动配置所有必需的构建环境变量和工具路径，无需您手动进行任何设置，极大地提升了开发效率。

下面是 Espressif_IDE 软件的安装流程。

1，以管理员身份运行 espressif-ide-setup-2.12.0-with-esp-idf-5.1.2.exe 文件。

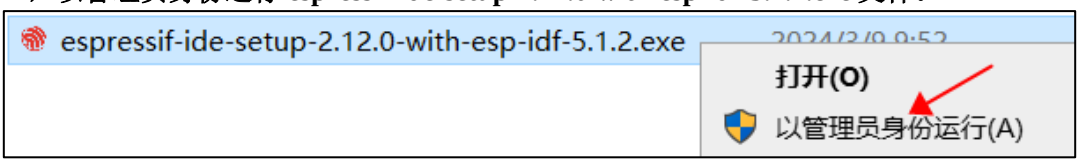


图 1.2.2.1 以管理员身份运行安装包

2，选择安装语言界面

打开安装程序后选择简体中文安装，如下图所示：

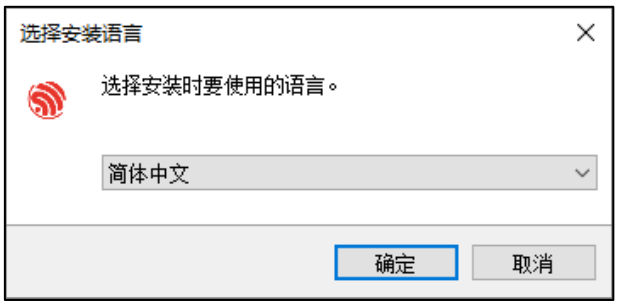


图 1.2.2.2 选择安装语言

在这里，直接选择简体中文即可，然后点击确定。

2，同意许可协议

进入许可协议界面，如下图所示。

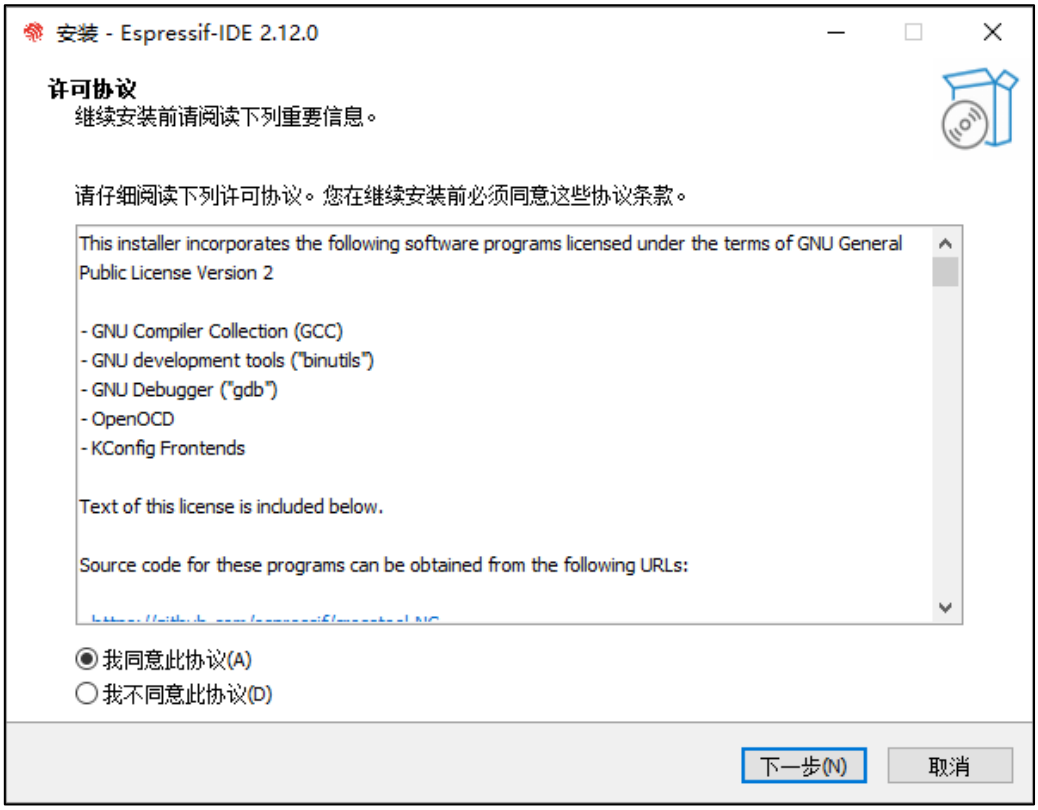


图 1.2.2.3 许可协议确认

点击“我同意此协议”，后点击下一步。

3，安装前系统检查

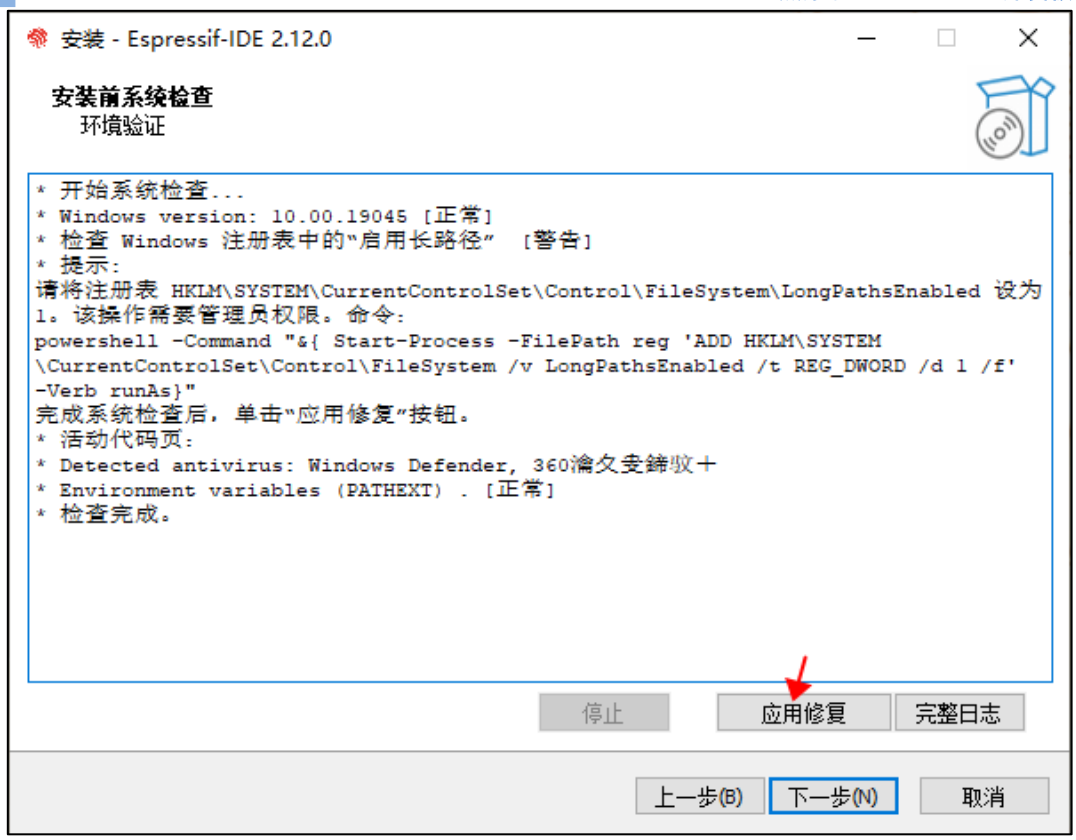


图 1.2.2.4 安装界面

安装程序会检查你当前系统有没有打开"长路径支持"，因为 GNU 编译器产生的编译文件会有非常深的目录结构，如果不支持长路径，编译可能出现文件不存在，目录不存在等奇怪的错误。这里单击**应用修复**按钮，可以修复这个问题。在弹出的确认对话框中，选择是，开始修复。

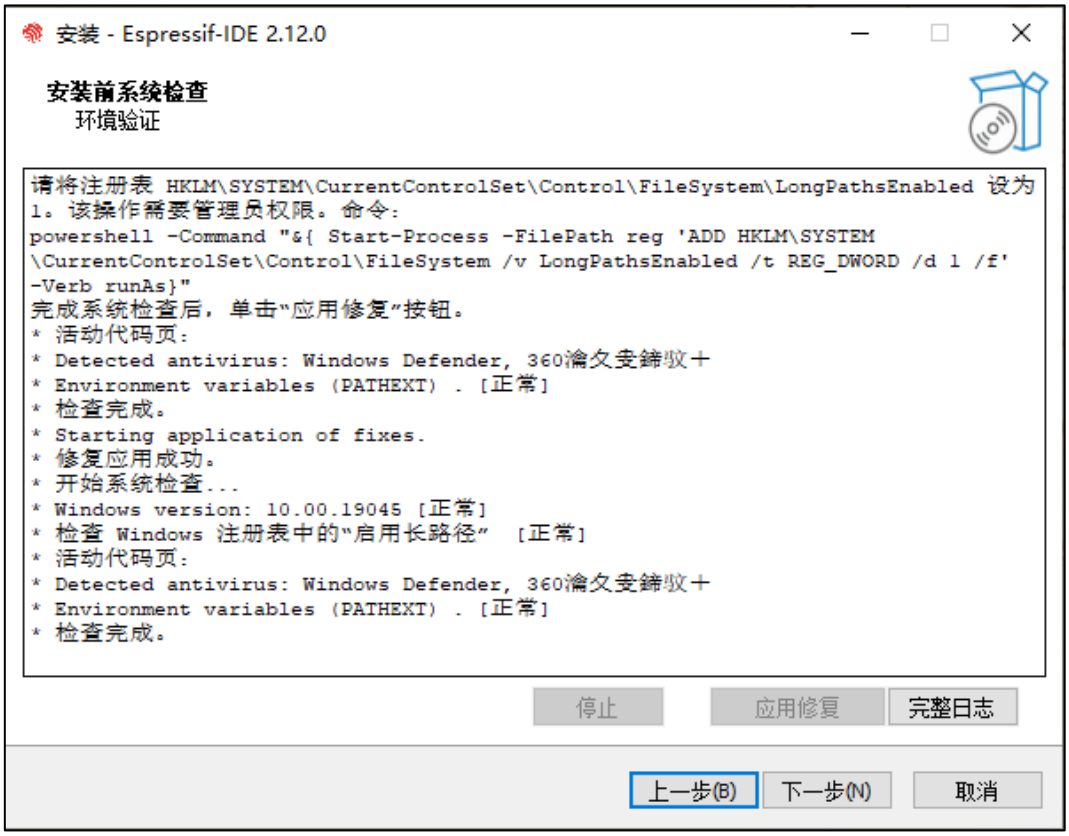


图 1.2.2.5 在注册表中启用长路径

如果修复不成功，一般情况是安装软件打开时没有使用管理员权限打开，可以手动修改注册表来支持长路径：打开注册表 HKLM\SYSTEM\CurrentControlSet\Control\FileSystem\LongPathsEnabled 设置为 1。如下图所示：

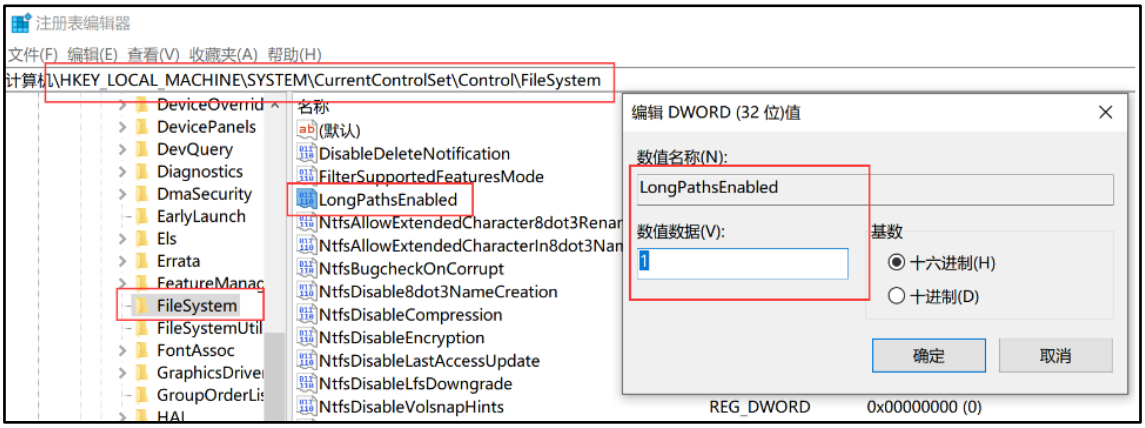


图 1.2.2.6 手动启用长路径

图 1.2.2.5 提示修复完成后，点击下一步

4，配置安装路径

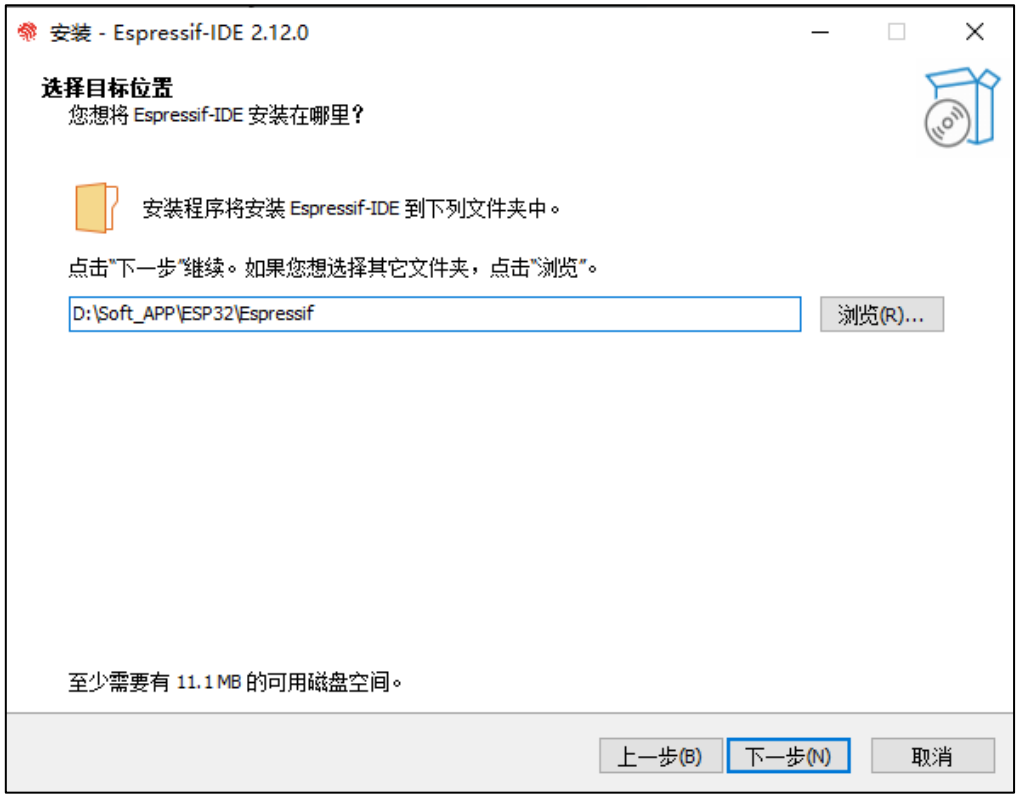


图 1.2.2.7 配置安装路径

安装程序默认的安装位置为 C:\Espressif，但这里我是安装在 D 盘，如果全部源码编译后可能产生几十 G 的大小占用，我们在 D 盘下创建 Soft_APP_ESP32\Espressif 文件夹来保存 ESP32-IDF 库安装过程中生成的文件。注意：这个安装路径非常重要，因为 VS Code 软件的 IDF 插件需要此路径来获取相关文件，所以开发者务必牢记该路径。

5，选择组件安装

设置安装路径后点击 “下一步”选项，进入确认安装组件界面，这里全部打勾。然后单击下一步。

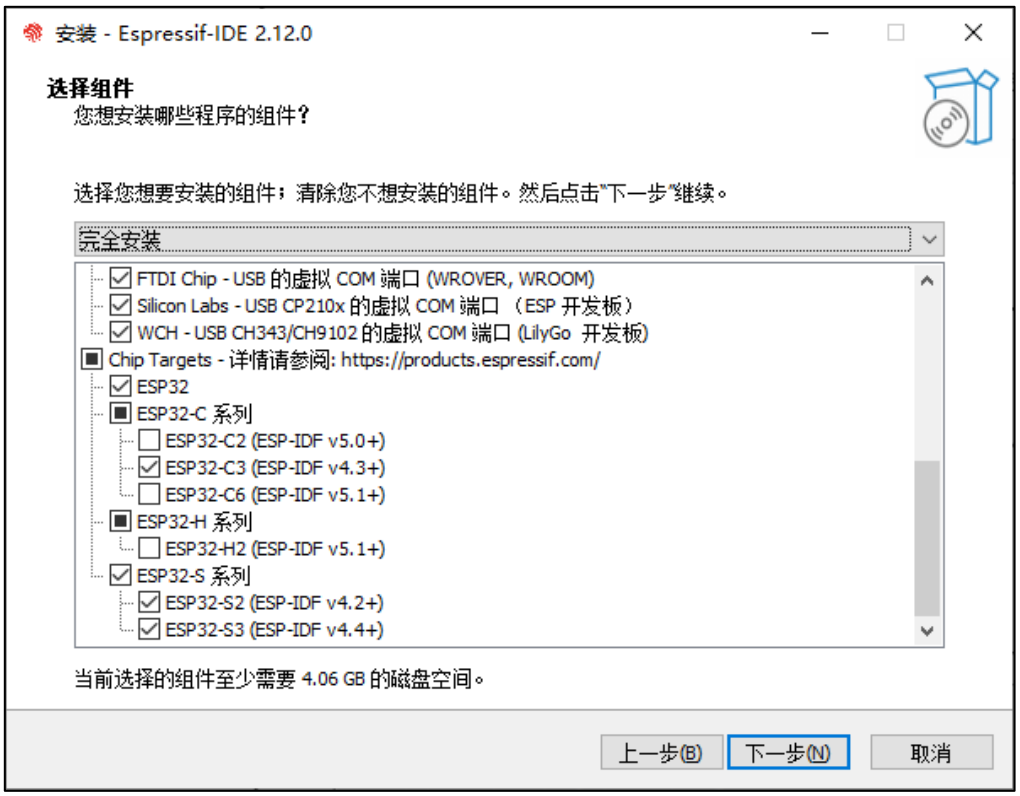


图 1.2.2.8 选择安装组件

我们选择全部安装。点击下一步再次确认安装目录信息，然后点击下一步，进入到准备安装界面，如下图所示。

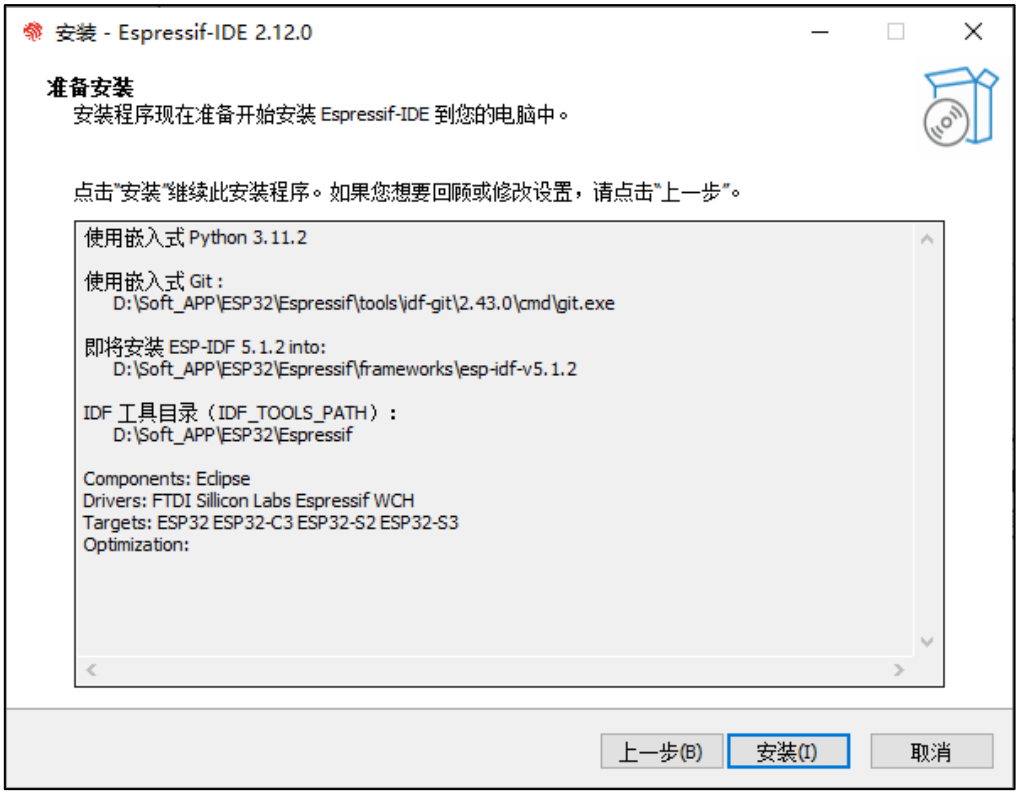


图 1.2.2.9 准备安装

在安装之前，提供了一个预览页面，点击安装即可，进入到软件安装过程。安装完成，三个全部勾选，1、2 用于测试环境安装是否成功，3 是将 ESP-IDF 工具链加入杀毒工具排除项，以加快编译速度，如下图所示：

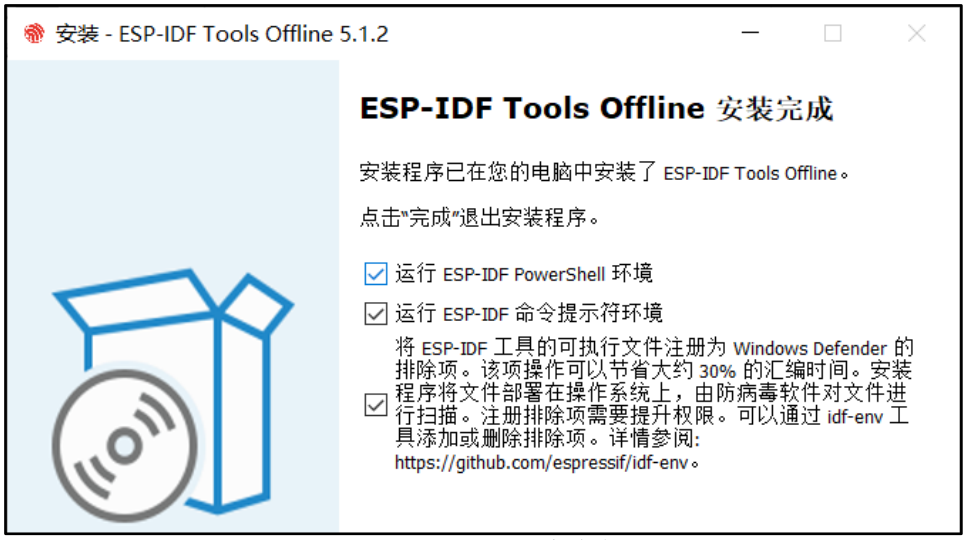


图 1.2.2.9 ESP32-IDF 库安装完成

到这里软件已经安装完成了，由于勾选了“运行 ESP-IDF PowerShell 环境”和“运行 ESP-IDF 命令提示符环境”，所以会弹出这两个窗口，如下图所示。

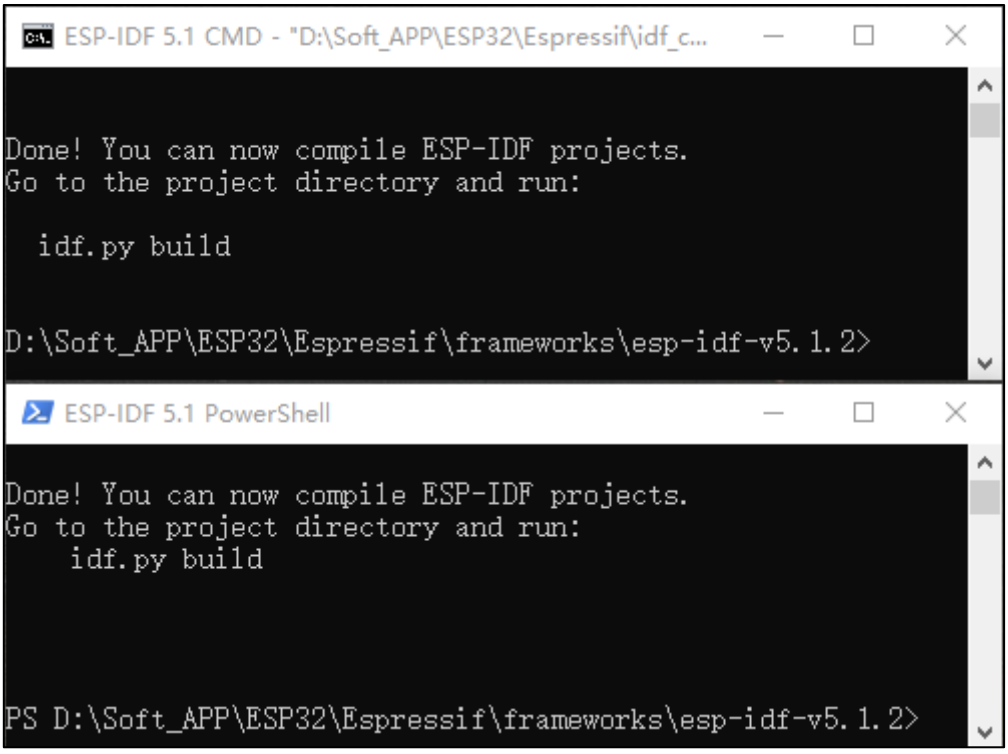


图 1.2.2.10 运行 ESP-IDF 环境

等待命令行出现“idf.py build”字样，后面就可以关闭窗口。这时候，就可以发现在桌面有三个新图标，如下图所示。

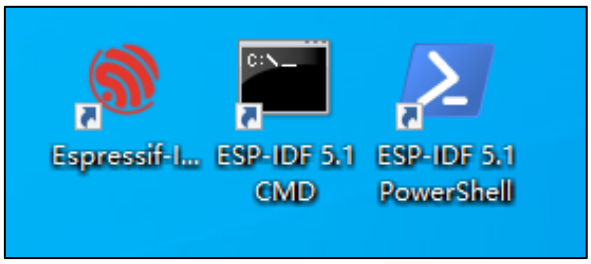


图 1.2.2.11 软件图标

上图中，Espressif-IDE 为我们安装的开发 ESP32 系列芯片的集成开发环境。双击“Espressif-IDE”图标，准备启动软件，如下图所示。

图 1.2.2.12 软件启动

稍等片刻，需要对 Espressif-IDE 工作区进行设置，如下图所示。

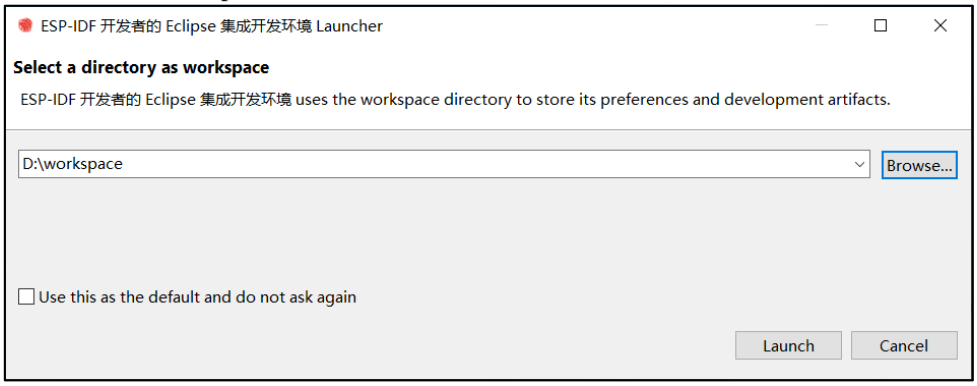


图 1.2.2.13 设置 IDE 工作区

使用工作区来存储首选项和开发过程中生成的文件，在这里根据自己实际情况进行选择即可。点击 Launch，即可进入到软件主界面，如下图所示。

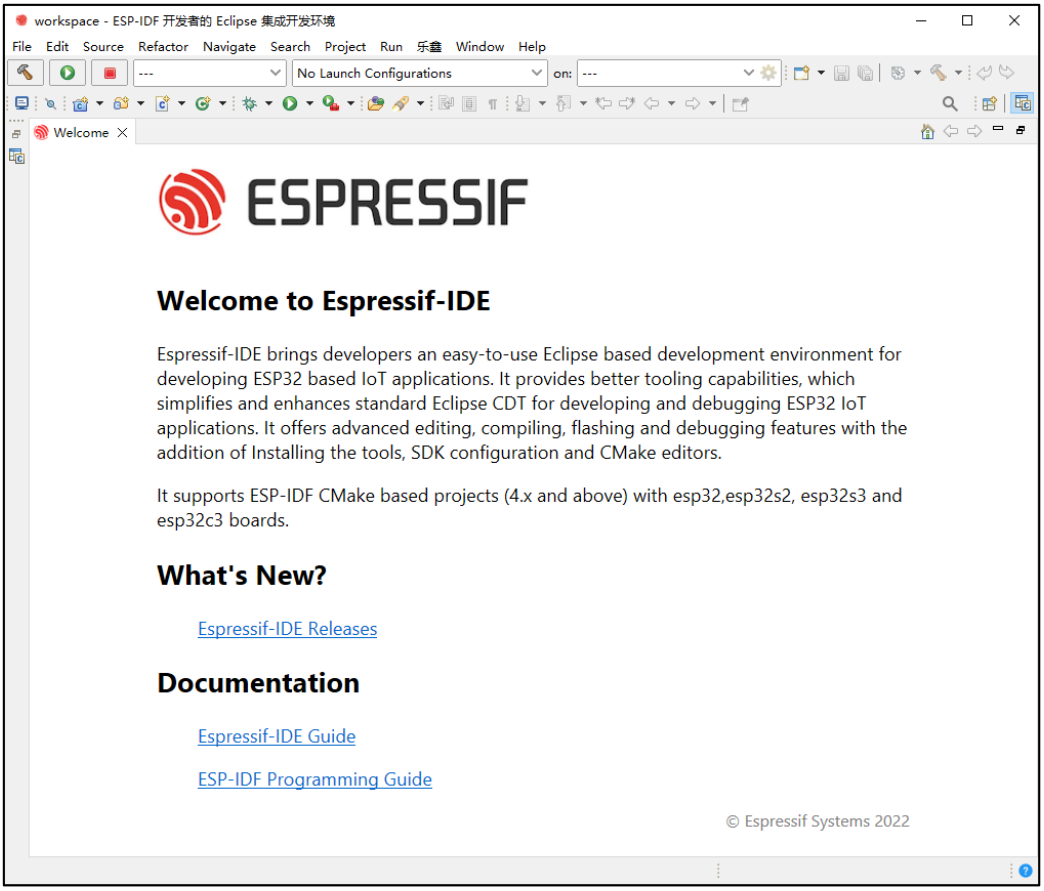


图 1.2.2.14 软件主界面

启动 Espressif-IDE 后，它会自动配置所需的环境变量并随即展示欢迎页面。为确保环境变量已正确配置，您可以进入“Window→Preferences→C/C++→Build→Environment”进行检查，相关显示页面如图所示。这样，您可以更加安心地进行后续的开发工作。

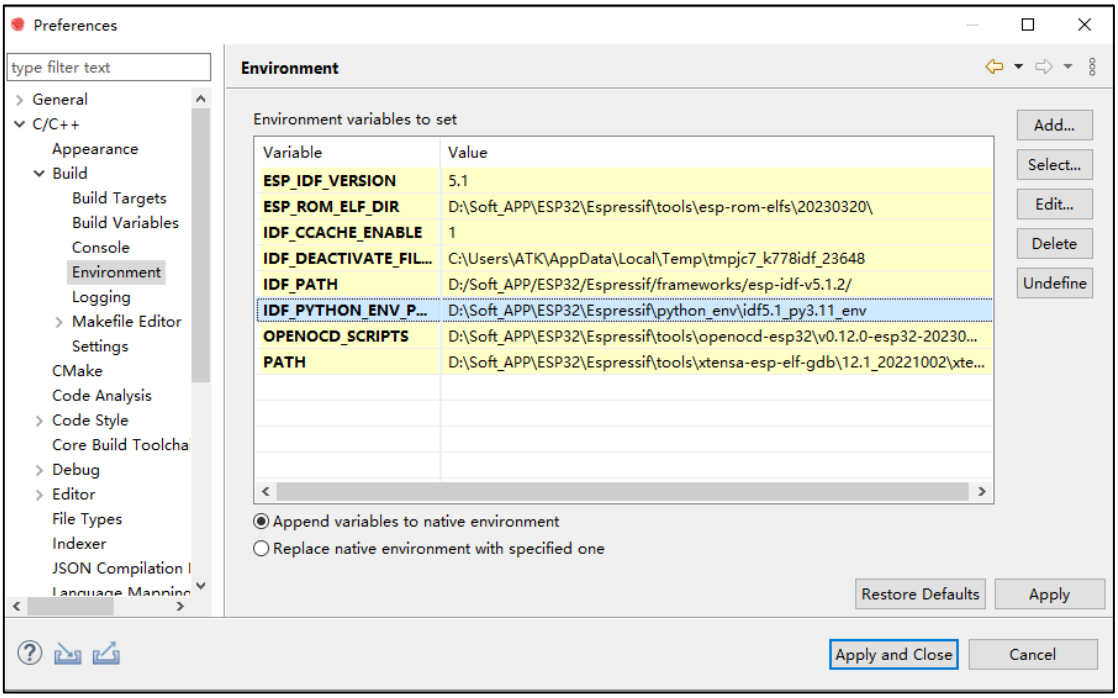


图 1.2.2.15 环境变量

安装好软件后，建议大家检查一下是否已正确设置所需的环境变量。如果环境变量存在，那么后续的新建工程和编译工程过程通常不会遇到问题。如果发现这些环境变量不存在，请检查系统的环境变量设置，以排除可能的冲突情况。相关操作可参照图示进行。

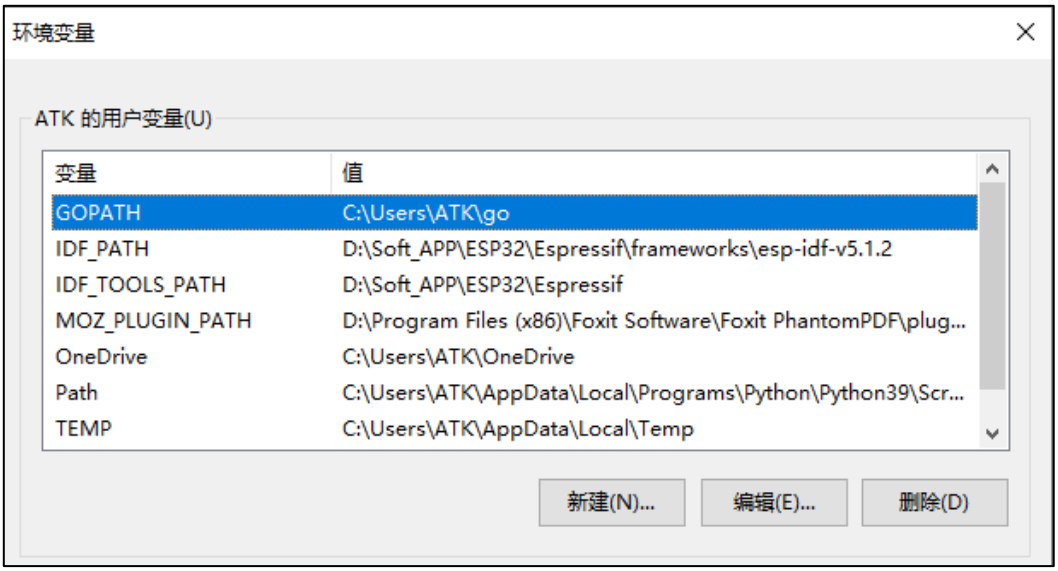


图 1.2.2.16 系统环境变量

主要是检查 IDF_PATH 和 IDF_TOOLS_PATH 这两个环境变量是否存在冲突或不一致的问题。若确实存在此类问题，建议修改这两个环境变量的设置，或者选择删除后重新安装 Espressif_IDE 软件，以解决潜在的问题。

1.3 Espressif_IDE 界面介绍

软件的主界面如下图所示。

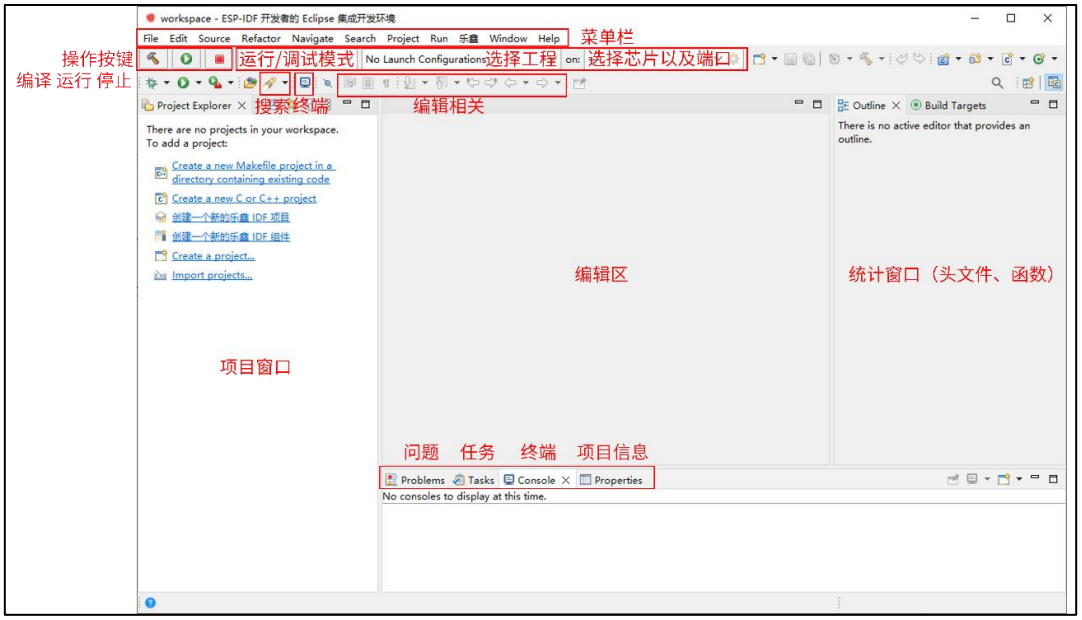


图 1.3.1 Espressif_IDE 软件主界面

上图中已对 Espressif_IDE 的部分图标功能进行了详细说明，后续的使用中，您可能会进一步涉及这些功能的操作。若在使用过程中遇到任何问题，建议您查阅官方资料以获取更详细的指导，具体链接为：https://github.com/espressif/idf-eclipse-plugin/blob/master/README_CN.md#Installation。

第二章 Espressif_IDE 软件使用

Espressif_IDE 软件本质上是在 Eclipse 开发环境中安装了 ESP-IDF 插件，集成了编辑、编译、烧录和调试等基础功能，同时附加了安装工具、SDK 配置和 CMake 编辑器等实用工具，极大地简化了开发人员使用标准 Eclipse CDT 开发和调试 ESP32 IoT 应用程序的流程，提升了整体开发体验。本章节将重点介绍如何便捷地使用 Espressif_IDE 对 ESP32S3 进行开发，如需了解更多高级功能，建议查阅该软件的详细介绍资料。

本章节将分为以下几个小节：

2.1 Espressif_IDE 新建项目工程
2.2 Espressif_IDE 添加项目组件
2.3 Espressif_IDE 调试项目工程

2.1 Espressif_IDE 新建项目工程

在 Espressif_IDE 软件中，新建项目工程是一个相对简单的流程。接下来，我们将详细讲解这一步骤。

第一步，打开 Project Explorer 窗口，找到“创建一个新的乐鑫 IDF 项目”选项，并按照下方图示指引执行操作。

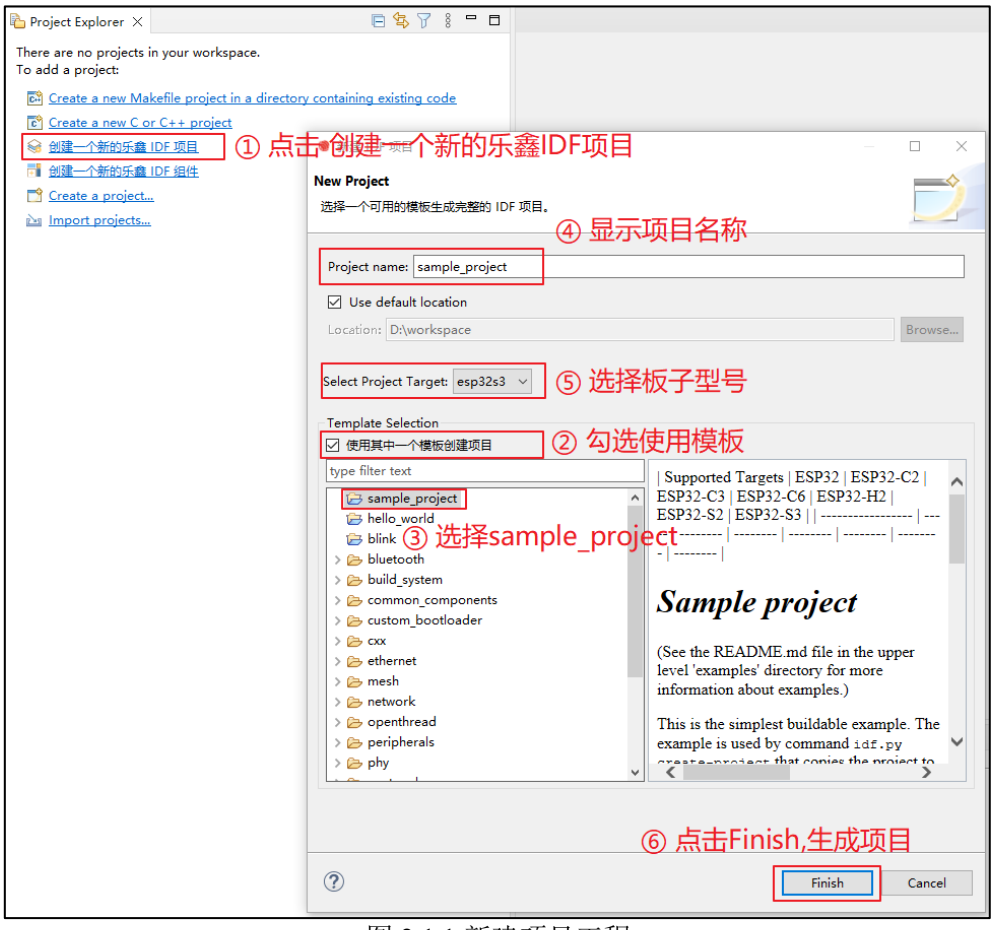


图 2.1.1 新建项目工程

在新建工程时，您可以选择直接使用预设的模板，也可以不使用模板从零开始。在上图中，我们直接采用了 sample_protect 模板项目工程作为示例。

第二步，点击“sample_project”项目名以展开整个工程。接着，在 main.c 文件的 app_main 函数中编写您的代码。具体的编写过程，可参照下图所示进行。

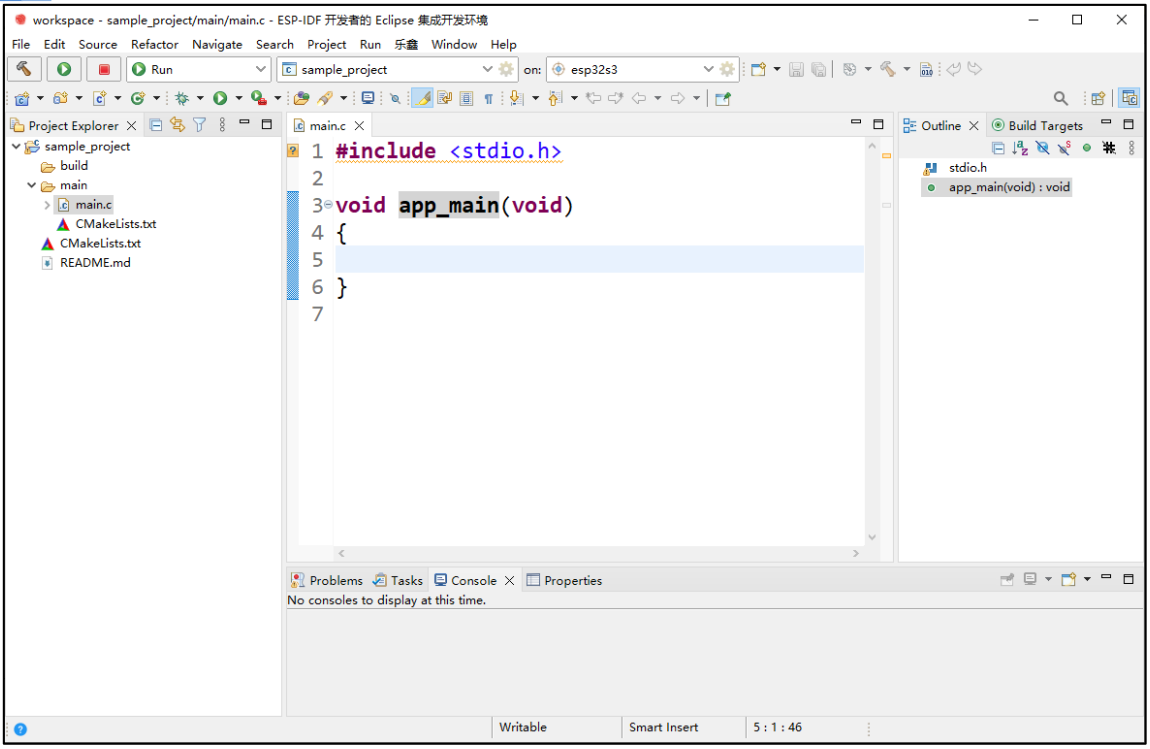


图 2.1.2 sample_project 项目工程

默认情况下，软件中的字体可能显得较小，为提升阅读体验，您可以在软件中调整字体大小。具体操作步骤为：选择“Window”菜单，进入“Preferences”，然后点击“General”下的“Appearance”，最后选择“Colors and Fonts”进行字体设置。

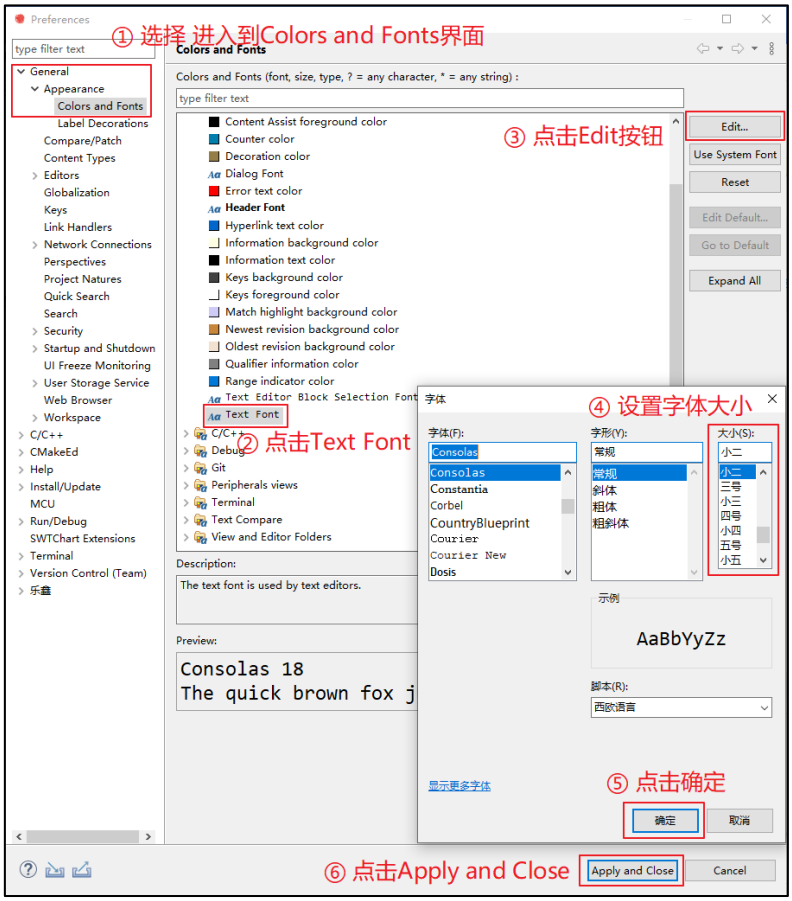


图 2.1.3 设置字体大小

第三步，您需要在 app_main 函数中编写所需的一行代码。完成编写后，点击编译按钮，即可对工程进行编译。具体的操作过程，可参照下图所示进行。

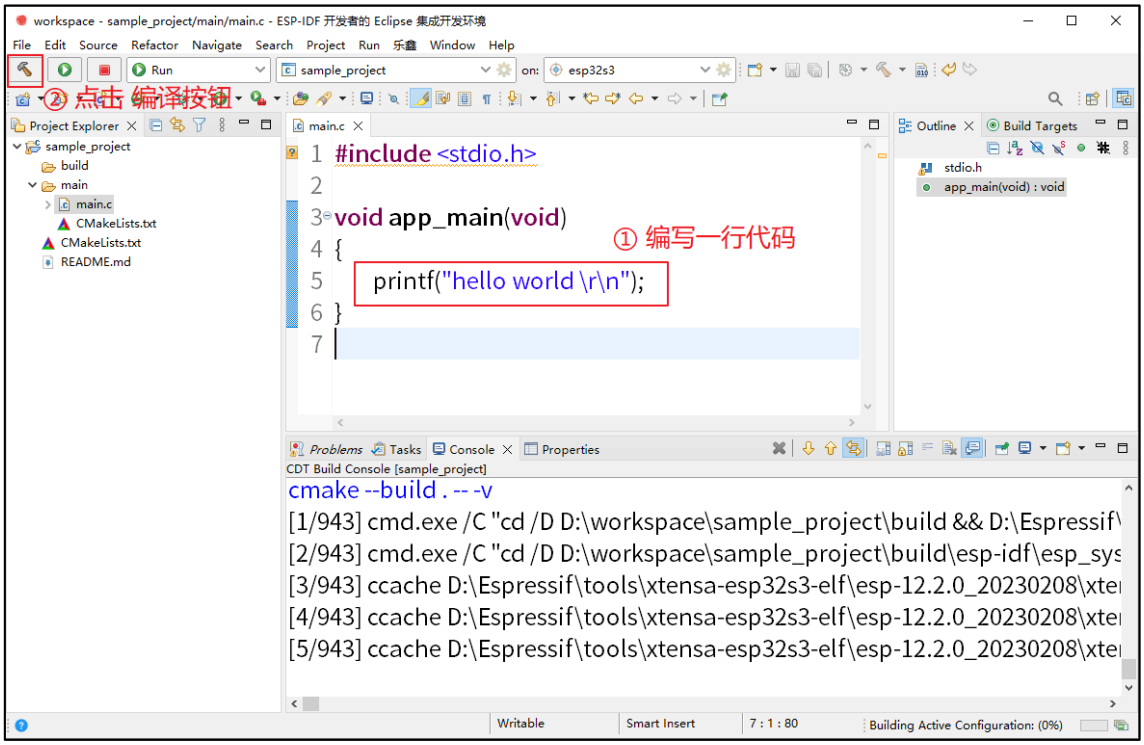


图 2.1.4 编译工程

第四步，请将开发板的 UART 口通过 TYPE-C 线连接到电脑。接着，配置启动目标，确保一切准备就绪后，点击下载按钮，执行程序的烧录工作。具体的操作步骤，可参照下图所示进行。

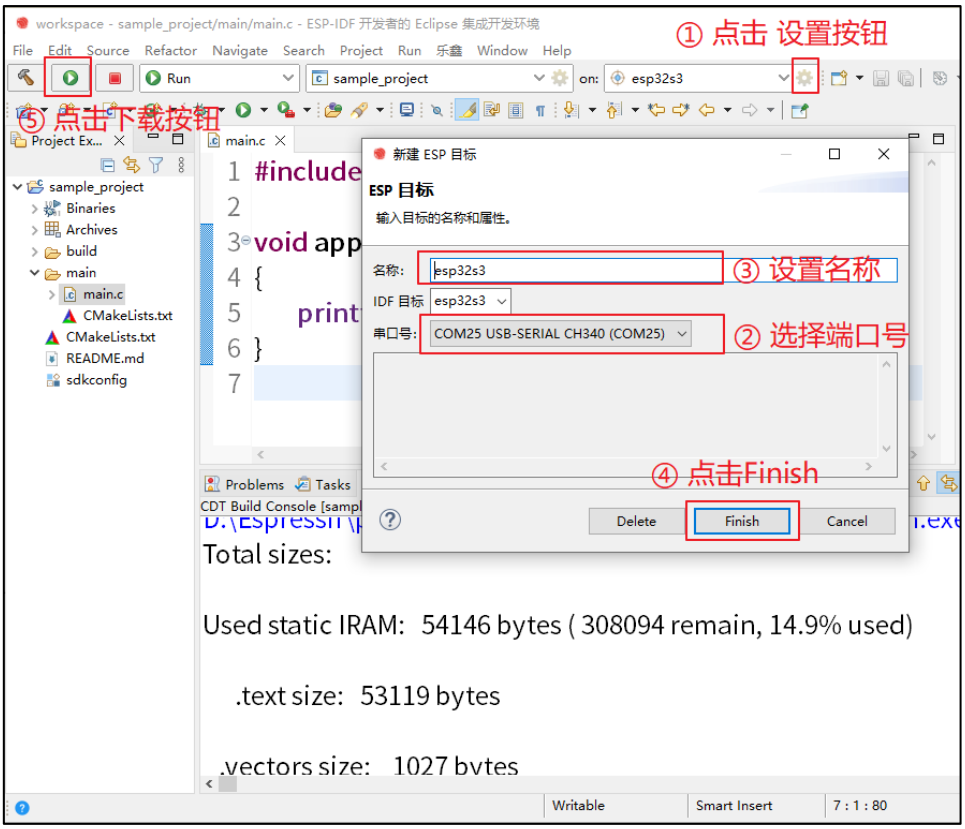


图 2.1.5 程序烧录

当然，配置启动目标这一步骤，开发者可以选择在新建工程完成后立即进行，也就是在第二步时便完成相关设置。这样做能够确保在后续的开发过程中，您能够顺利地启动并运行目标程序。

第五步，验证程序是否正确执行。由于我们在程序中使用了 printf 函数来打印“hello world”，因此需要先打开串口助手来查看输出结果。具体操作如下。

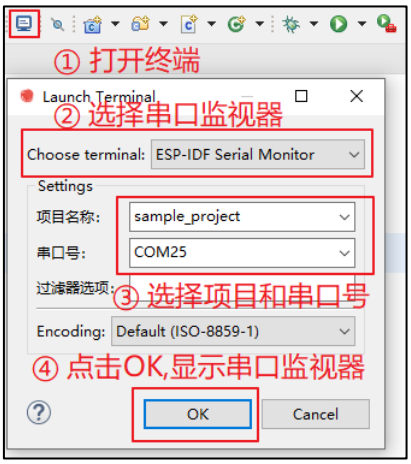


图 2.1.6 打开串口助手

按下板子的复位键，串口助手显示的内容如下：

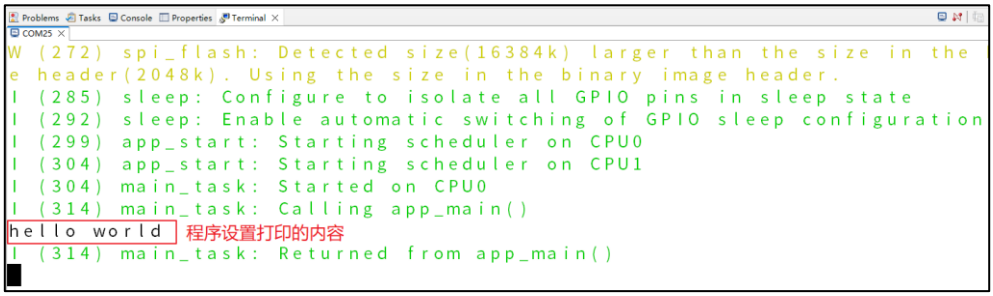


图 2.1.7 程序执行效果

到这里，就证明 Espressif_IDE 已经可以正常使用。

2.2 Espressif_IDE 添加项目组件

使用 Espressif_IDE 软件新建项目工程时，虽然模板中仅包含一个 main.c 文件用于代码编写，但这显然无法满足复杂项目的需求。因此，ESP_IDF 采用了组件（components）的方式来进行扩展。这些组件通常用于存放第三方驱动库以及开发者自行编写的驱动库，从而实现功能的模块化与复用，提高开发效率。

基于 2.1 创建的 sample_project 工程，进行添加组件操作，如下图所示。

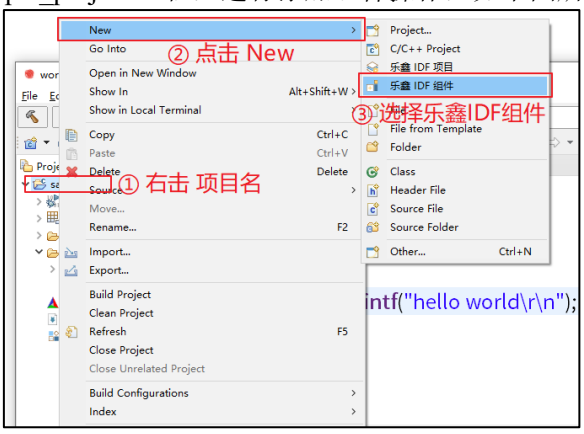


图 2.2.1 添加组件操作 1

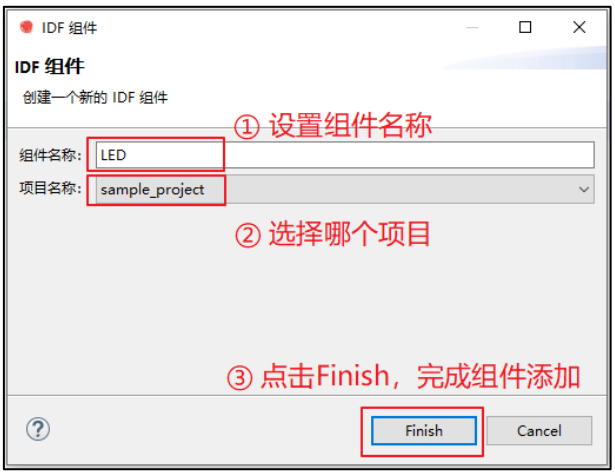


图 2.2.2 添加组件操作 2

接下来，您会在工程目录下看到一个名为“component”的文件夹，其中包含一个“LED”文件夹。这与图 2.2.2 中的展示完全对应，表明您的项目结构已经按照预期设置完成。

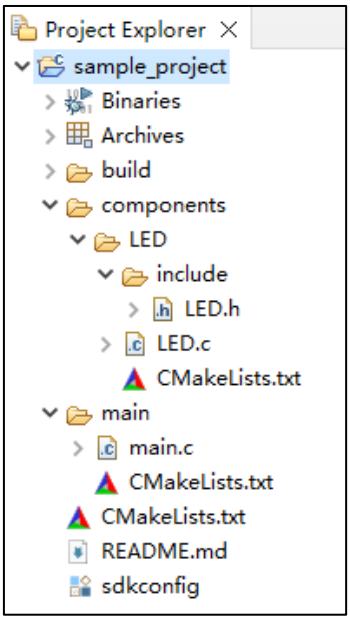
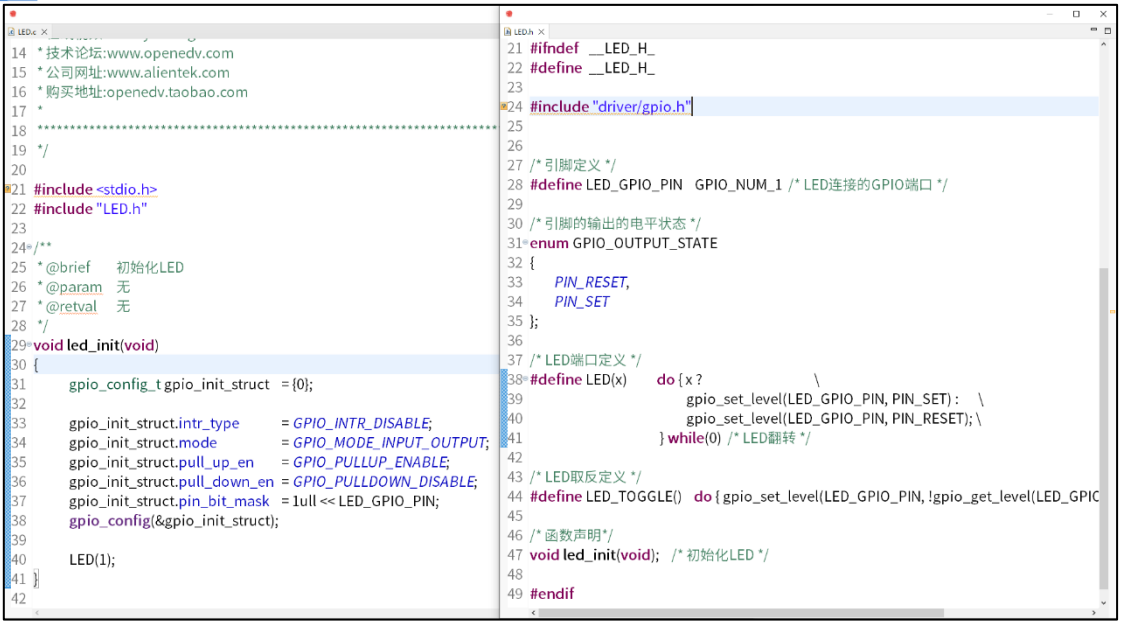


图 2.2.3 添加组件后的工程目录

LED 组件的作用就是要驱动 ESP32S3 板载的 LED 灯，所以关于 led 的代码需要编写在 LED.c 和 LED.h 两个文件中。

这里，我们直接拷贝 IDF 版本 01_led 的代码过来，如下图所示。



```
 * 技术论坛:www.openedv.com
 * 公司网址:www.alientek.com
 * 购买地址:openedv.taobao.com
 *
 ****************************************************************
 */

#include <stdio.h>
#include "LED.h"

/**
 * @brief     初始化LED
 * @param     无
 * @retval    无
 */
void led_init(void)
{
    gpio_config_t gpio_init_struct = {0};

    gpio_init_struct.intr_type    = GPIO_INTR_DISABLE;
    gpio_init_struct.mode         = GPIO_MODE_INPUT_OUTPUT;
    gpio_init_struct.pull_up_en   = GPIO_PULLUP_ENABLE;
    gpio_init_struct.pull_down_en = GPIO_PULLDOWN_DISABLE;
    gpio_init_struct.pin_bit_mask = 1ull << LED_GPIO_PIN;
    gpio_config(&gpio_init_struct);

    LED(1);
}
```

```
#ifndef __LED_H_
#define __LED_H_

#include "driver/gpio.h"


/* 引脚定义 */
#define LED_GPIO_PIN  GPIO_NUM_1 /* LED连接的GPIO端口 */

/* 引脚的输出的电平状态 */
enum GPIO_OUTPUT_STATE
{
    PIN_RESET,
    PIN_SET
};

/* LED端口定义 */
#define LED(x)      do { x ?                    \
                         gpio_set_level(LED_GPIO_PIN, PIN_SET) :    \
                         gpio_set_level(LED_GPIO_PIN, PIN_RESET); \
                    } while(0) /* LED翻转 */

/* LED取反定义 */
#define LED_TOGGLE()  do { gpio_set_level(LED_GPIO_PIN, !gpio_get_level(LED_GPIC

/* 函数声明*/
void led_init(void);  /* 初始化LED */

#endif
```

图 2.2.4 LED 代码

由于 LED 组件需要调用官方提供的 driver 中的 gpio 接口，因此需要在 LED 组件的 CMakeLists.txt 文件中设置相应的依赖库。具体来说，需要添加“REQUIRES driver”来指明对 driver 库的依赖关系。修改后的 CMakeLists.txt 文件如下图所示，确保正确配置了组件间的依赖关系。

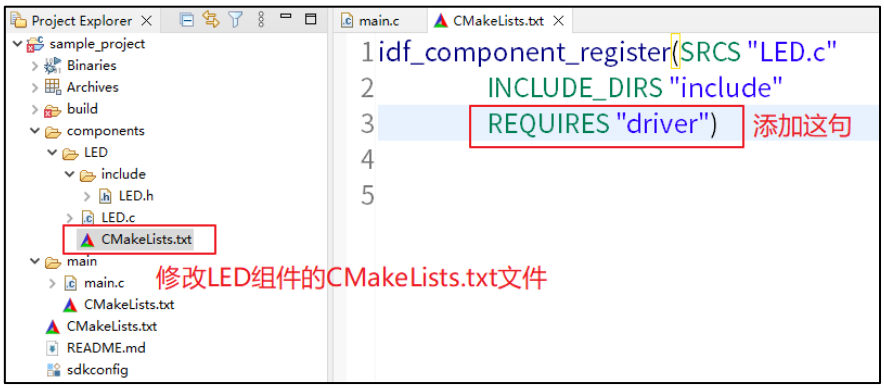


图 2.2.5 CMakeLists.txt 文件

随后，便在 main.c 文件中的 app_main 函数调用 led 相关代码，如下图所示。

```
#include <stdio.h>
#include "led.h"
#include "FreeRTOS/freertos.h"
#include "freertos/task.h"

void app_main(void)
{
    led_init();
    printf("hello world\r\n");

    while (1)
    {
        LED_TOGGLE();
        vTaskDelay(1000);
    }
}
```

图 2.2.6 main.c 文件内容

编写完程序后，我们依次点击编译和下载，期望能够观察到 LED 灯以 1 秒的间隔闪烁。然而，实际情况却是灯的闪烁间隔非常长，远非预期的 1 秒。这究竟是何原因呢？

回顾整个操作过程，我们似乎都按照步骤顺利进行，但似乎遗漏了一个关键点——idf 的 menuconfig。这个工具允许我们根据芯片的实际情况进行项目配置。考虑到遇到的延时问题，很可能是由于 configTICK_RATE_HZ 参数设置不当所致。

为了解决这一问题，我们需要打开 menuconfig 进行配置。在 Espressif_IDE 软件中，具体操作如下：

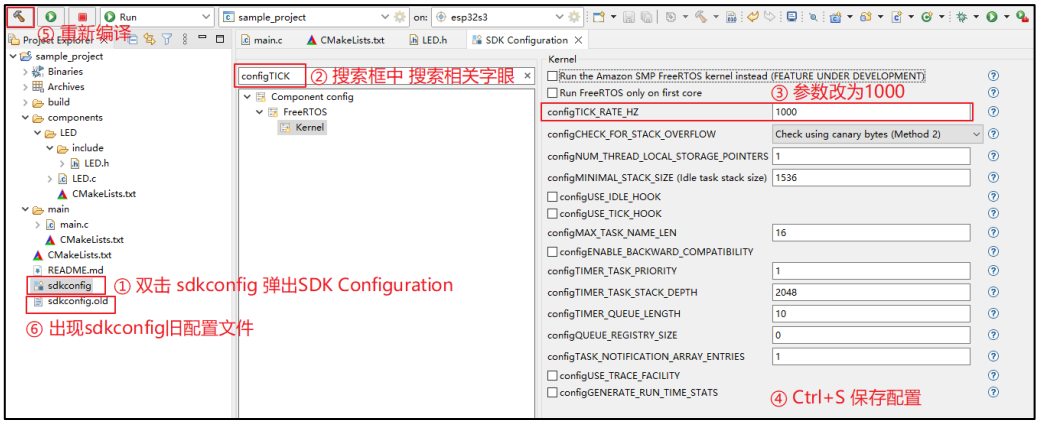


图 2.2.7 SDK Configuration 配置过程

为了优化定时精度和操作的便捷性，我们推荐将 configTICK_RATE_HZ 参数修改为 1000。该参数代表系统节拍时钟（tick clock）的频率。默认情况下，configTICK_RATE_HZ 的值为 100，意味着节拍时钟的周期为 10ms。因此，在调用 vTaskDelay(1000)时，将会导致延时长达 10 秒。为了提高定时精度，我们建议将该值设置为 1000，这样节拍时钟的周期就缩短至 1ms，从而使得 vTaskDelay(1000)能够准确地代表延时 1 秒。

修改相关参数后，请重新编译并下载程序至开发板，此时程序应能按照代码逻辑实现预期效果。此外，对于正点原子 ESP32-S3 开发板，除了调整上述参数，还需对 CPU 时钟频率、FLASH 接口模式/速度/大小、分区表以及 PSRAM 接口/速度等参数进行适当设置，以确保硬件资源的正确配置。具体设置方法可参照附图进行操作。

1，设置 CPU 主频

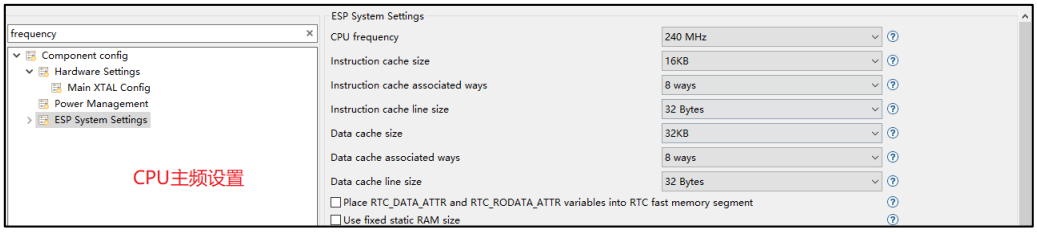


图 2.2.8 CPU 主频参数设置

2，Flash 配置

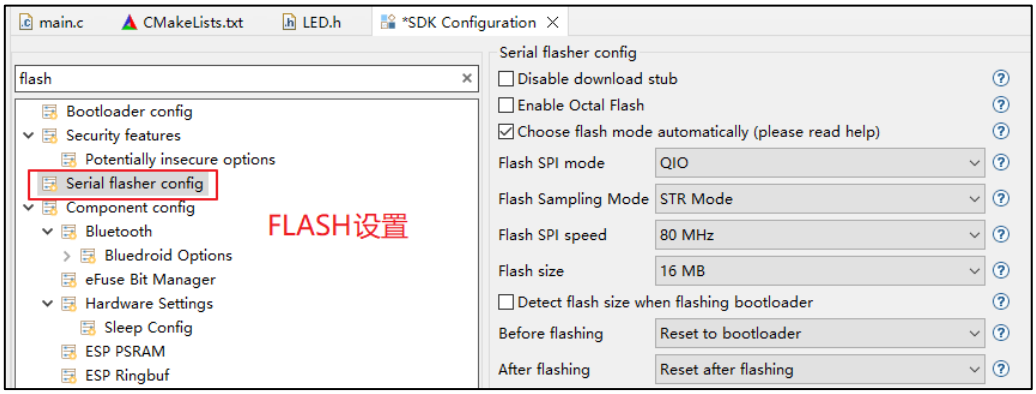


图 2.2.9 FLASH 相关设置

上图中的“Flash SPI mode”支持四种不同的 SPI flash 访问模式，它们分别为 DIO、DOUT、QIO 和 QOUT。下面我们来看一下这几种模式到底有哪些区别，这些模式的对比如下表所示：

可选项	模式名称	引脚	速度
QIO	Quad I/O	地址和数据 4pins	最快
QOUT	Quad Output	数据 4pins	约比 qio 模式下慢 15%
DIO	Dual I/O	地址和数据 2pins	约比 qio 模式下慢 45%
DOUT	Dual Output	数据 2pins	约比 qio 模式下慢 50%

表 2.2.1 四种 SPI 模式的对比

从上表可知，QIO 模式的速率为最快，所以我们把基础工程的 Flash SPI mode 设置为 QIO。

上图中的“Flash SPI speed”提供了 120、80、40 和 20MHz 的配置选项。在选择具体速度时，我们需要考虑 Flash 和 PSRAM 的 SPI 接口共享情况。为了优化模组性能，我们最好将 flash 和 PSRAM 的 SPI 速率设置为一致，这样分时访问这两个存储设备时，就不必切换时钟频率了。鉴于 PSRAM 的 SPI 速率最高可设置为 80MHz，因此我们将 flash 的 SPI 速率也设置为 80MHz，以确保最佳性能。

上图的“Flash size”是根据模组挂载的 flash 来确定的，这里我们选择 16MB 大小，是毫无争议的。

3，分区表配置

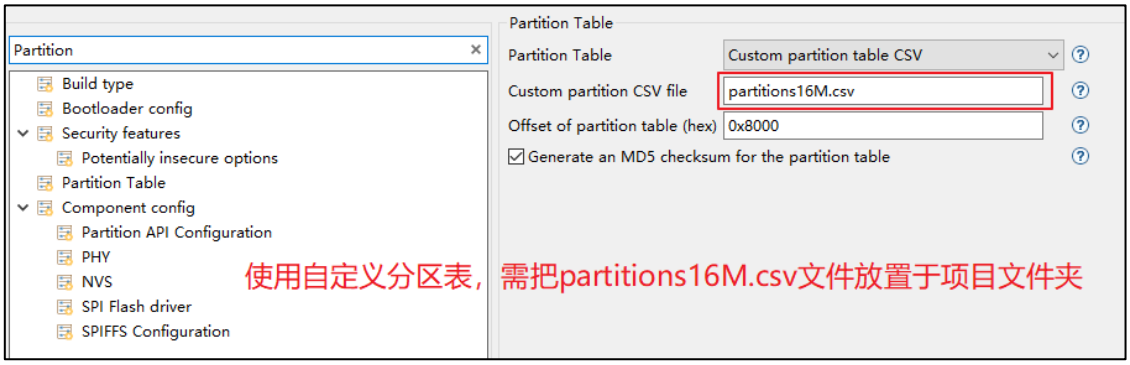


图 2.2.10 分区表设置

（注意：partitions16M.csv 文件需从 IDF 版本源码中提取，放置于 sample_project 文件夹内. 也可以自己设置分区表的内容）。

4，PSRAM 配置

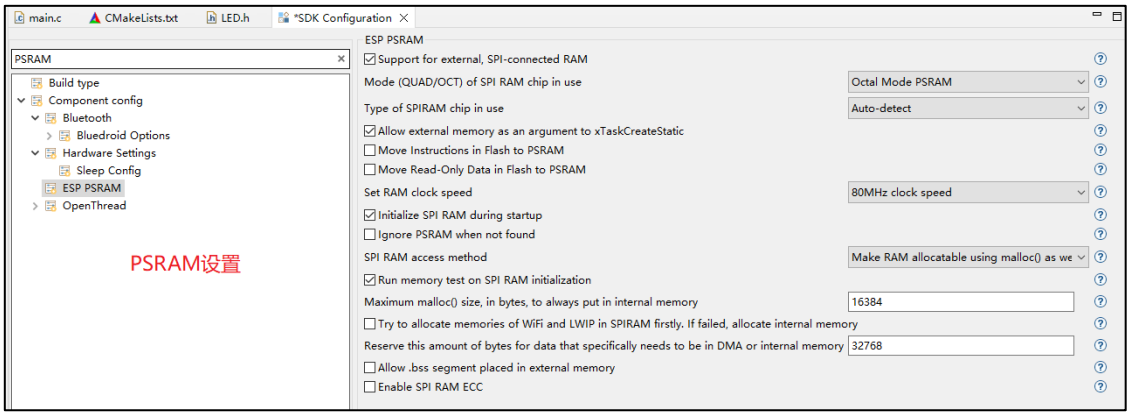


图 2.2.11 PSRAM 设置

上图的“Mode (QUAD/OCT of SPI RAM chip in use)”选项是基于模组内部芯片的选择来确定的。为了正确配置，读者可以查阅《esp32-s3-wroom-1_wroom-1u_datasheet_cn》数据手册的第三页。在该页中，我们可以看到 ESP32-S3-WROOM-1-N16R8 模组所挂载的 PSRAM 使用的是 Octal SPI 模式。因此，在配置过程中，我们应该选择“Octal Mode PSRAM”这一选项。

上图的“Set RAM clocj speed”选项选择最该的速率即可，并且与 Flash SPI 速率一致。

5，分区表条目设置

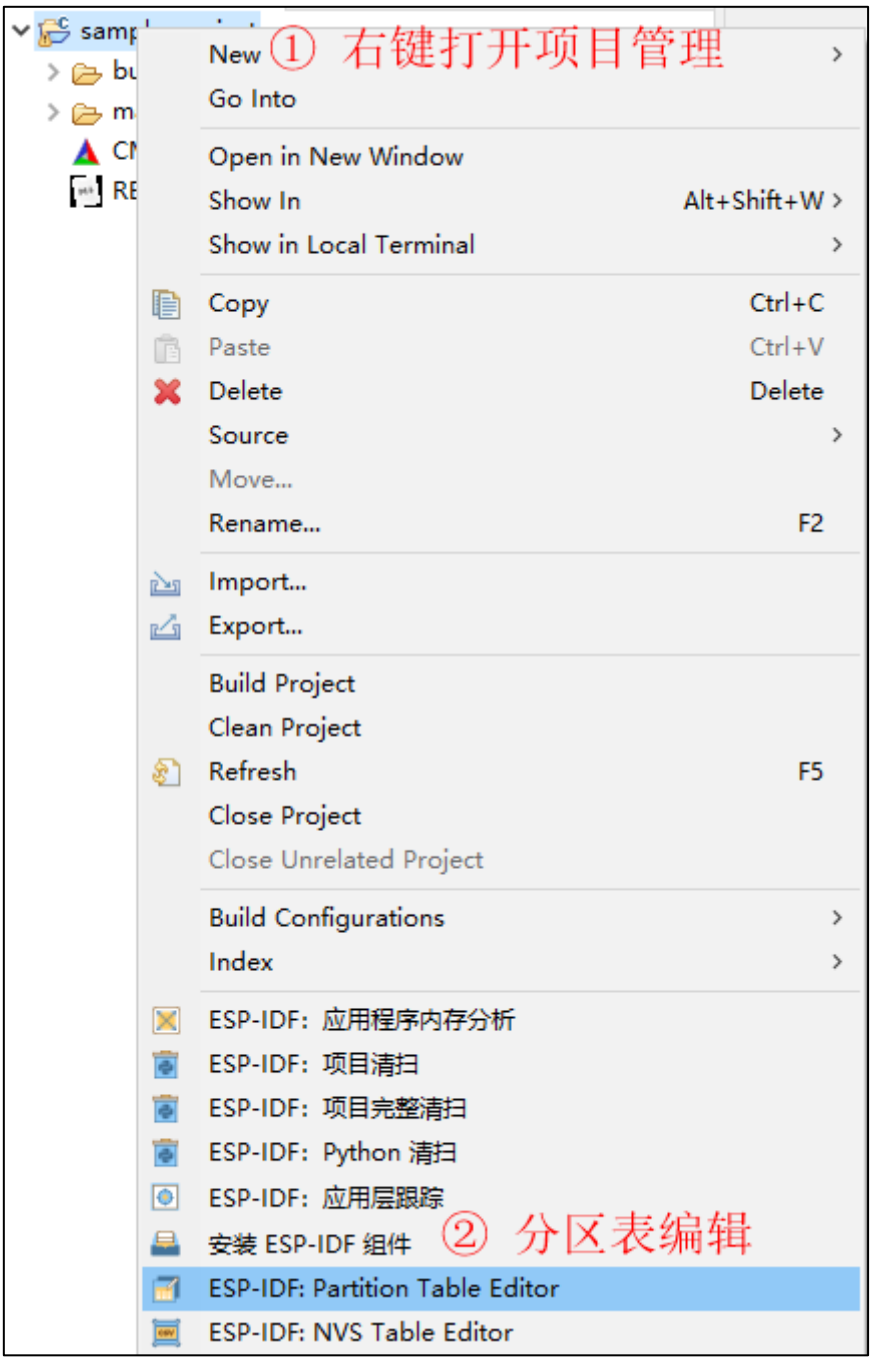


图 2.2.12 打开分区表编辑器

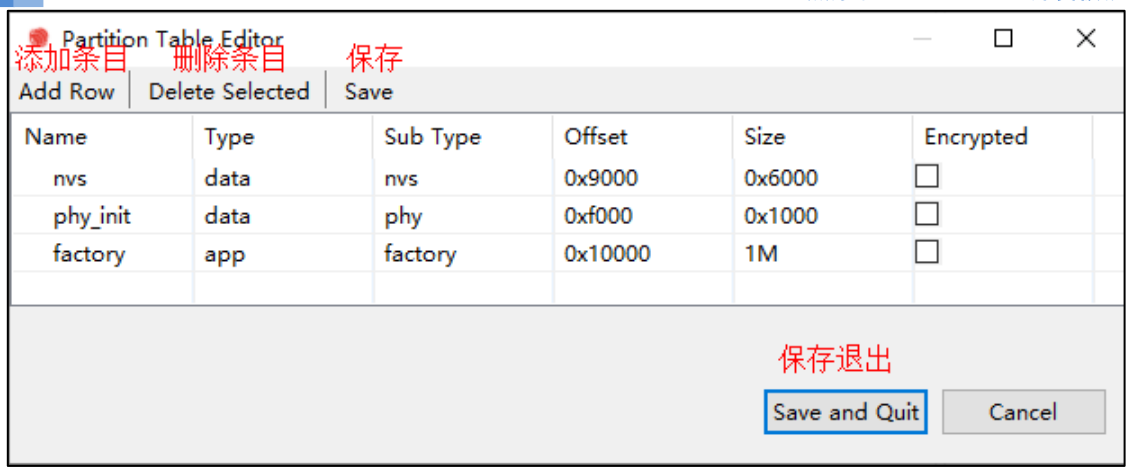


图 2.2.13 分区表编辑操作

我们根据上方的按键来添加条目、删除条目和保存分区表即可完成分区表编辑。

拓展：添加 ESP-IDF 组件方法，如下图所示。

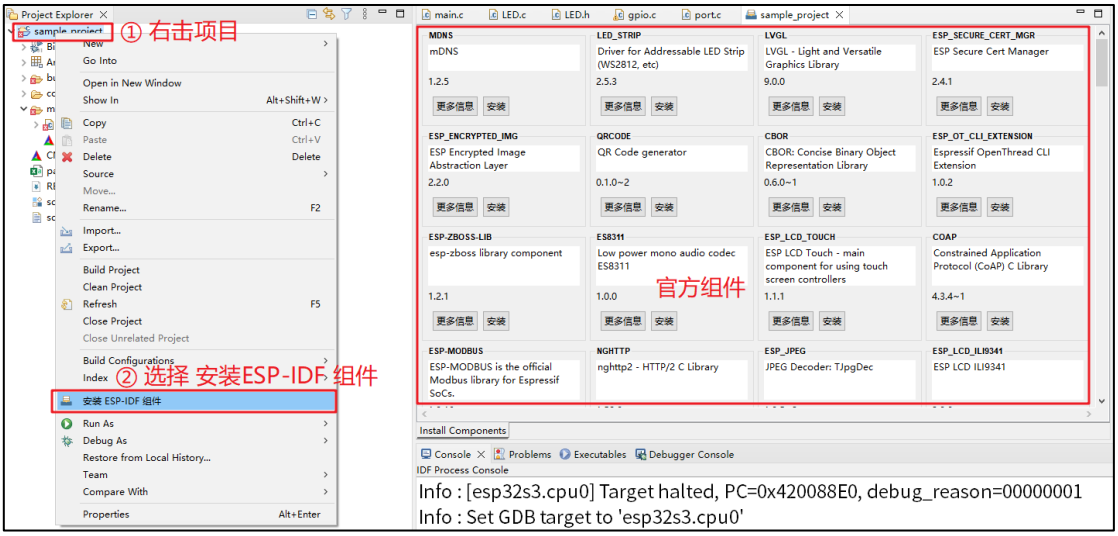


图 2.2.14 添加 ESP-IDF 组件

2.3 Espressif_IDE 调试项目工程

调试程序是软件开发中至关重要的一个环节，若软件缺乏调试功能，我们只能依赖传统的 printf 方法进行逐步调试，这无疑增加了开发难度。幸运的是，Espressif_IDE 软件提供了强大的调试支持，极大地简化了调试过程。

ESP32-S3 内置 JTAG 电路，无需额外硬件即可实现调试功能，这极大地提升了开发的便捷性。开发人员只需将 USB 线连接到 ESP32-S3 的 D+/D-引脚，即可轻松完成程序的下载与调试。此外，借助 JTAG 接口，开发人员还可以使用开源工具 OpenOCD 对 ESP32-S3 进行高效的调试操作，进一步提升了开发效率与准确性。

接下来，简单说明一下，在 Espressif_IDE 如何进行程序的简单调试。

第一步：将 TypeC 连接到 ESP32-S3 开发板的 USB 口。

右击项目名，依次选择“Debug As” > “Debug Configurations”，将弹出一个调试配置窗口。在此窗口中，选择“ESP-IDF GDB OpenOCD Debugging”作为调试方式，并新建一个调试配置。通常情况下，无需对默认配置进行修改，直接点击 Debug 按钮即可进入调试模式。具体操作步骤如下：

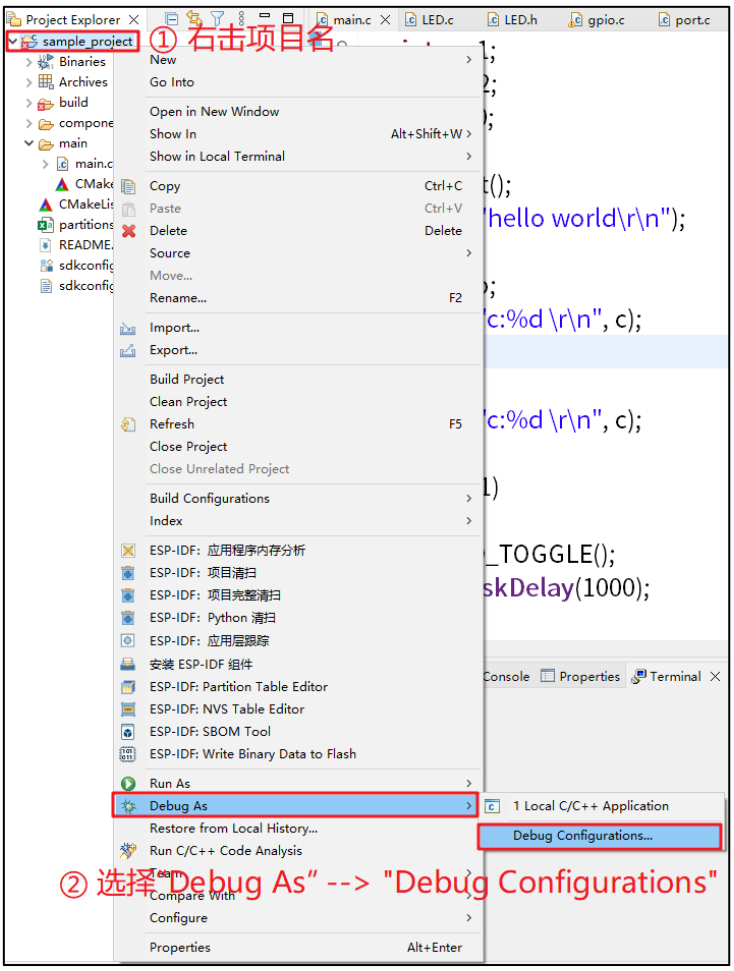


图 2.3.1 进入 Debug 配置界面

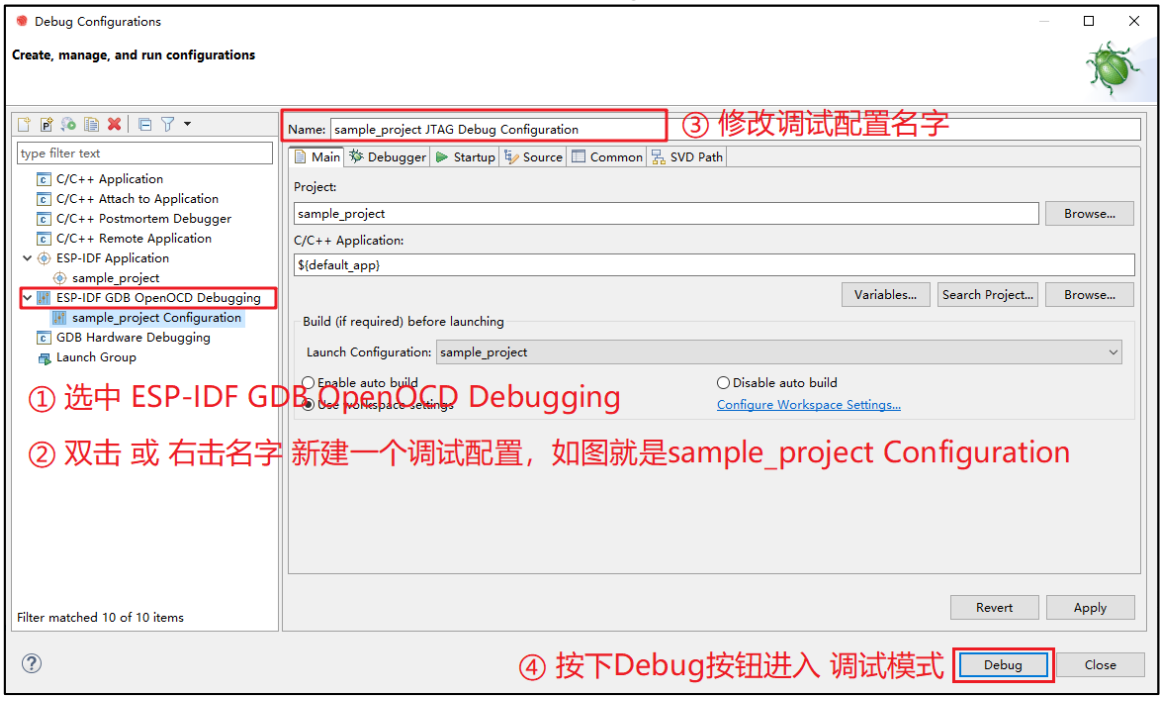


图 2.3.2 新建 Debug 配置并进入调试模式

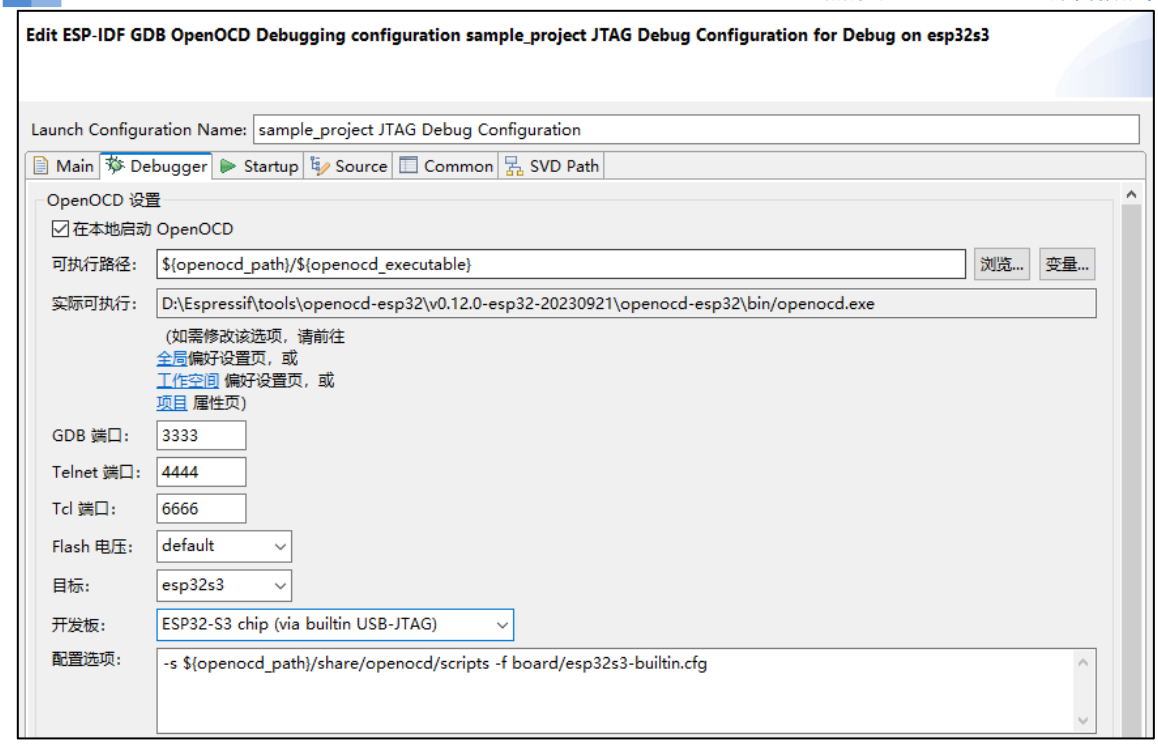


图 2.3.3 Debug 的 Debugger 参数配置

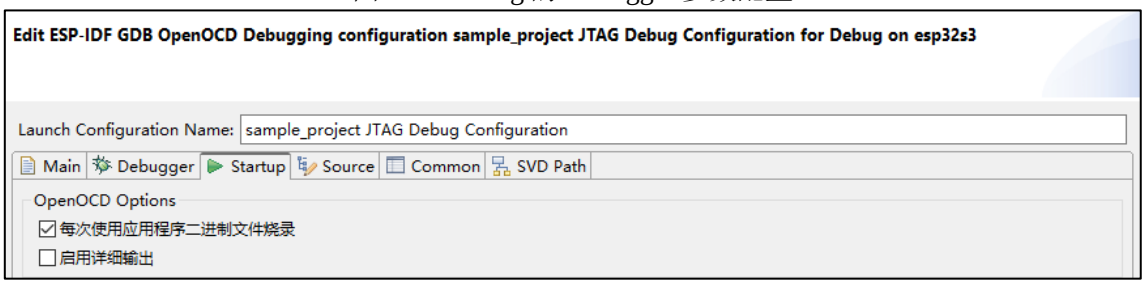


图 2.3.4 Debug 的 Startup 参数配置

关于 Debug 配置，通常按照默认生成即可。然而，在“Debugger 参数”中，有两个关键参数需要特别注意，即“目标”和“开发板”。确保这两个参数正确设置，以确保调试过程的顺利进行。至于“Startup 参数”，由于勾选了“每次使用应用程序二进制文件烧录”选项，因此在每次进行调试时，都会自动烧录程序到芯片中，这有助于确保调试时使用的是最新的程序版本。

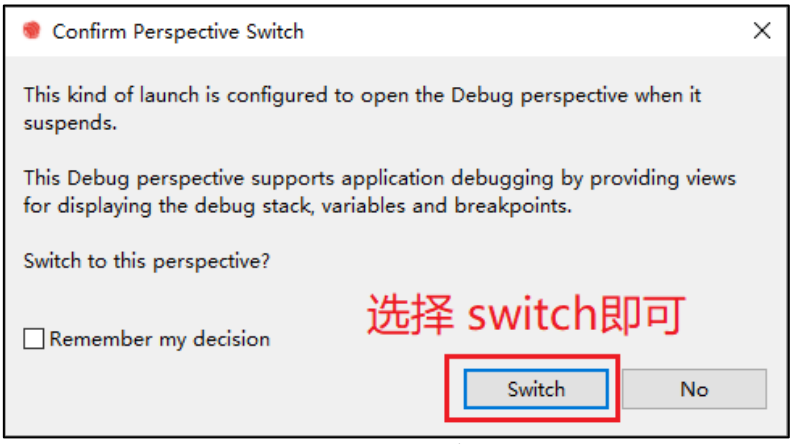


图 2.3.5 选择调试窗口视图

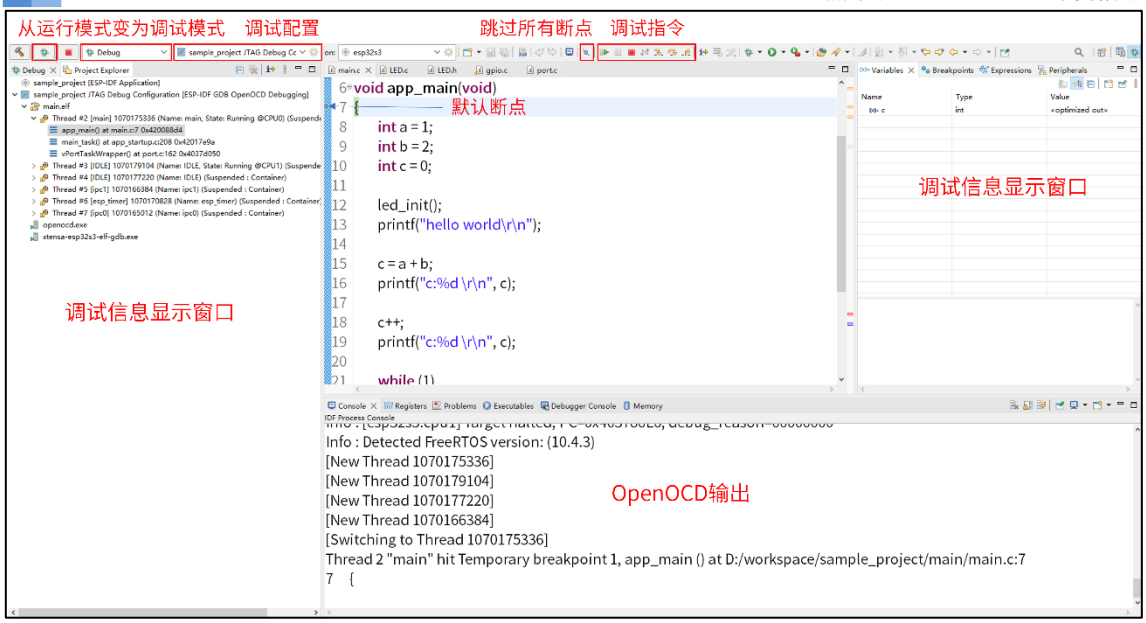


图 2.3.6 进入 Debug 状态

在 Debug 模式下，IDE 提供了多个调试信息窗口，以便开发者能够清晰地查看当前程序的执行情况。其中，Variables 窗口尤为实用，它会实时显示程序中定义的变量，使开发者能够轻松追踪变量的变化过程。

在使用调试功能时，掌握基本的调试指令是关键。从左到右，这些指令依次是“继续执行”、“暂停执行”、“停止调试”、“单步执行”（进入函数内部）和“逐行执行”（不进入函数内部）。另外，还有一个“单步跳出”指令，用于在执行完当前函数或循环后暂停。建议开发者亲自实践这些功能，以更好地掌握其用法。