Viện Đào tạo công nghệ thông tin, chuyển đổi số - TDMU

# BÁO CÁO THỰC HÀNH

Môn học: Chuyên đề Internet of Things

Buổi báo cáo: Lab 02

Tên chủ đề: Lập trình cơ bản với ESP32 (ESP-IDF)

*GVHD: Phạm Nguyễn Hữu Thiện*

*Ngày báo cáo: 06/09/2025*

## THÔNG TIN CHUNG:

|  |  |  |  |
| --- | --- | --- | --- |
| **STT** | **Lớp** | **MSSV** | **Họ và tên** |
| **1** | D22CNTT06 | 2224802010872 | Hồ Tuấn Phước |
| **2** | D22CNTT06 | 2224802010871 | Phan Phước Hồng Phúc |
| **3** | D22CNTT06 | 2224802010934 | Nguyễn MInh Nghi |
| **4** | D22CNTT06 | 2224802010867 | Nguyễn Trung Phong |
| **5** | D22CNTT03 | 2224802010371 | Đào Minh Sơn |
| **6** | D22CNTT02 | 2224802010910 | Nguyễn Quốc Việt |

## ĐÁNH GIÁ KHÁC:

|  |  |
| --- | --- |
| **Nội dung** | **Kết quả** |
| **Tổng thời gian thực hiện bài thực hành trung bình** | 1 ngày |
| **Link thực hiện**  **(nếu có)** | https://youtu.be/dCfkmmcZhNY |
| **Ý kiến (nếu có)**  **+ Khó khăn**  **+ Đề xuất …** |  |
| **Điểm tự đánh giá** | 10 |

# **BÁO CÁO CHI TIẾT**

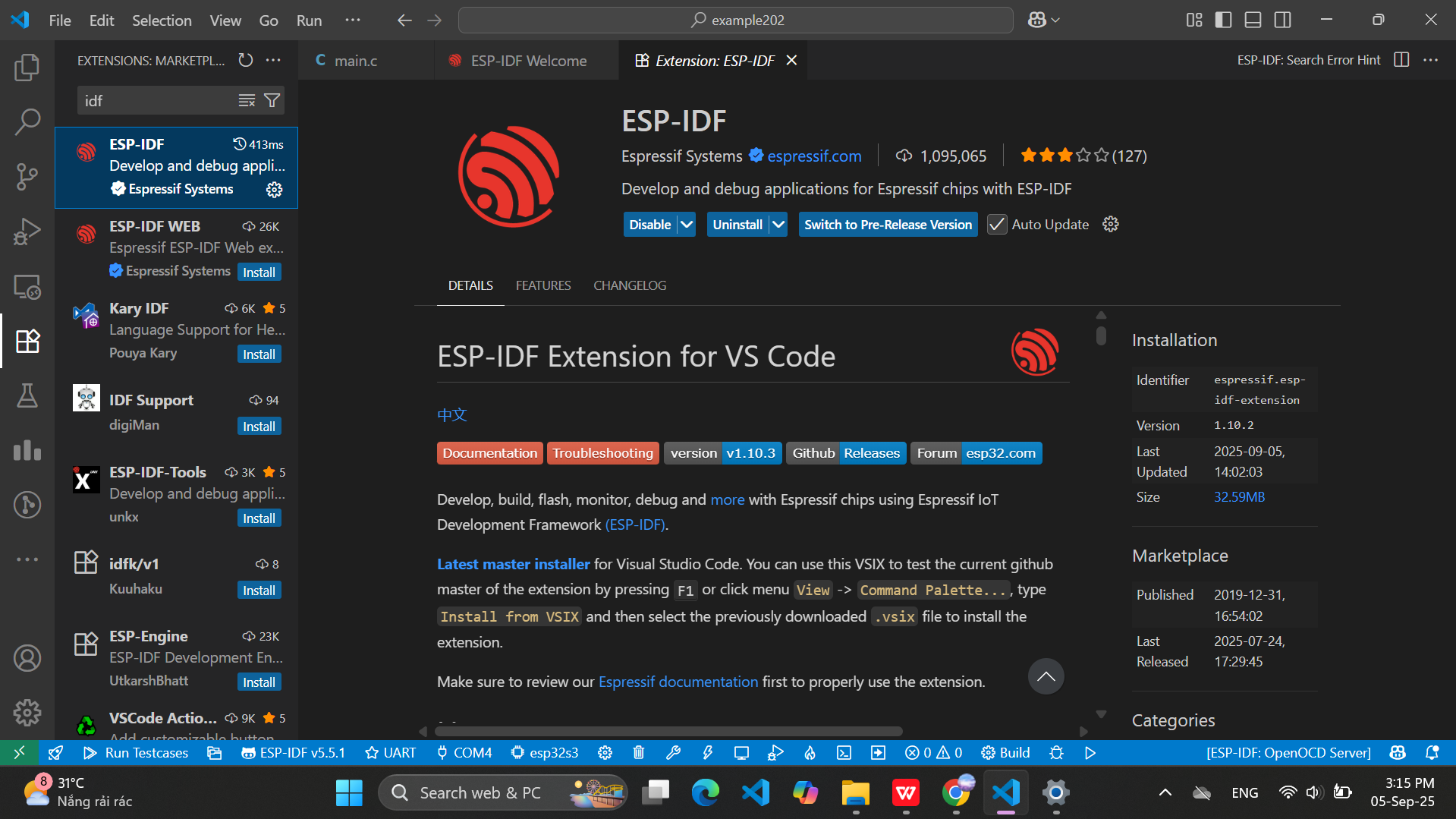
1. **Cài đặt môi trường ESP-IDF trên VS Code:**
   1. **Cài đặt VS Code**

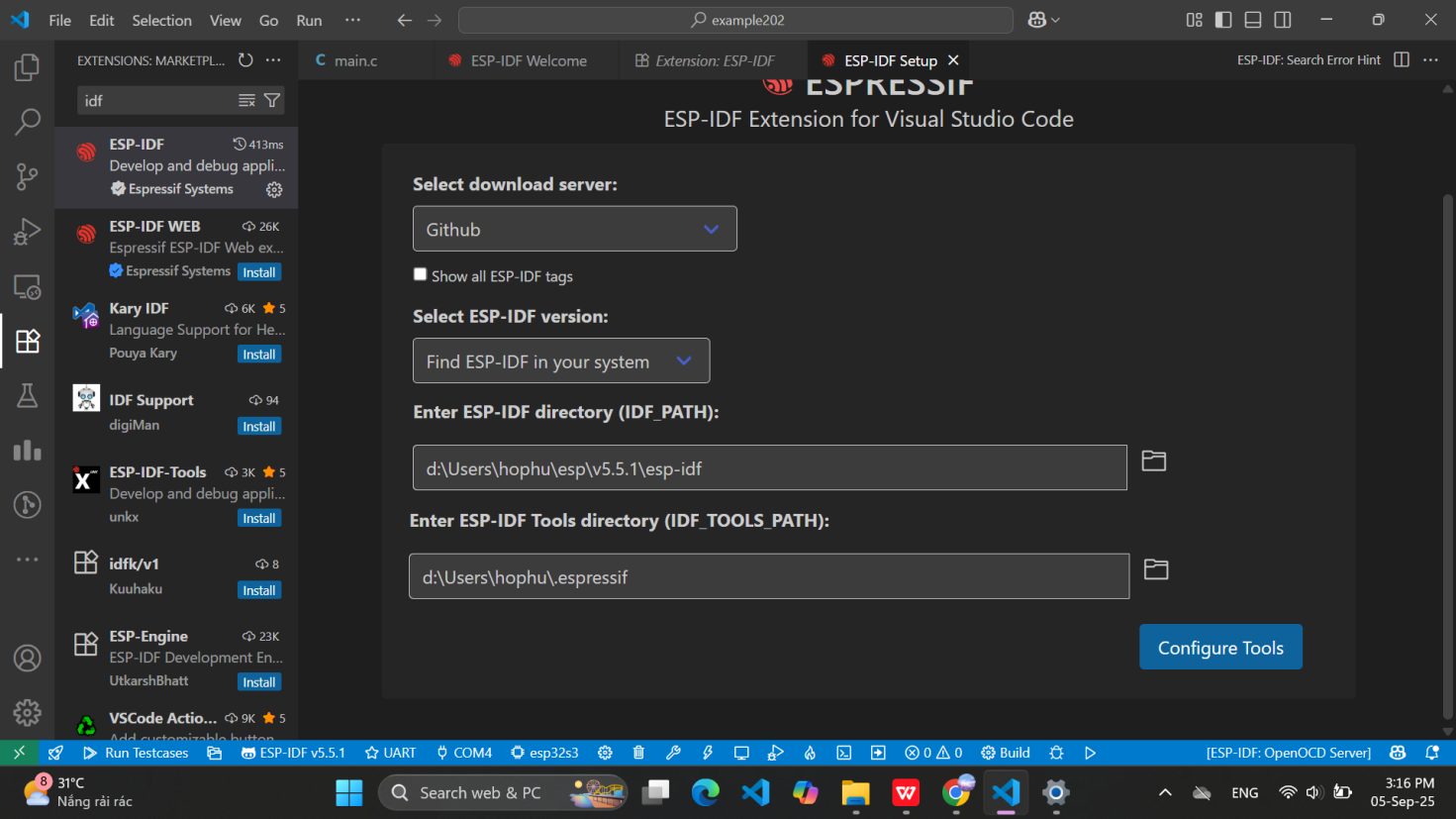
Truy cập trang <https://code.visualstudio.com/> và chọn phiên bản vs code cần cài đặt

* 1. **Cài đặt và cấu hình Espressif IDF**

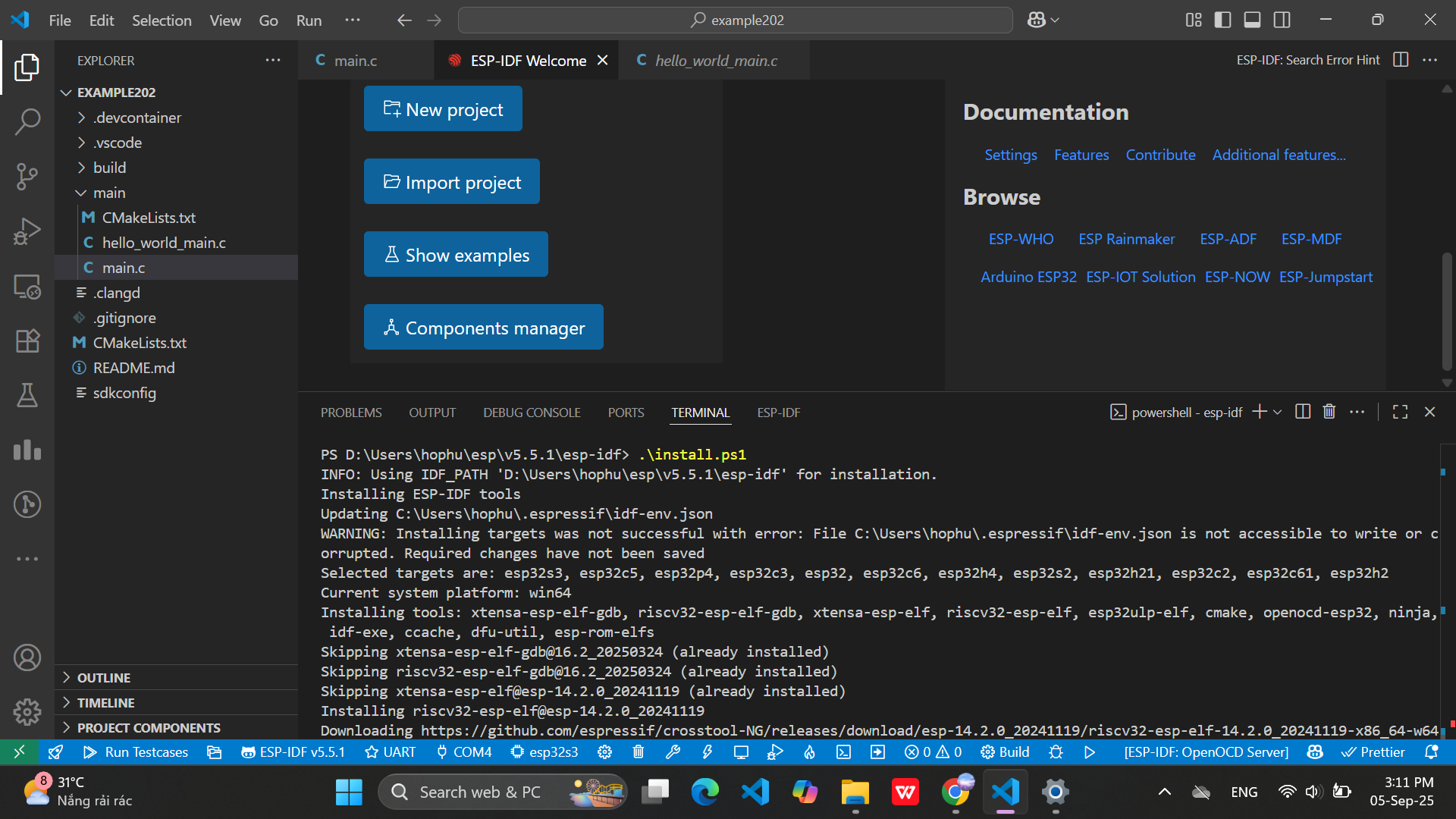
Espressif IoT Development Framework (ESP-IDF) là bộ công cụ phát triển phần mềm chính thức do Espressif cung cấp cho các vi điều khiển dòng ESP32. ESP-IDF cung cấp đầy đủ thư viện, driver, và các công cụ hỗ trợ giúp lập trình viên dễ dàng xây dựng, biên dịch và nạp ứng dụng cho các thiết bị IoT dựa trên ESP32.

ESP-IDF hỗ trợ nhiều tính năng quan trọng như quản lý bộ nhớ, kết nối Wi-Fi, Bluetooth, giao tiếp GPIO, và đa nhiệm với FreeRTOS, giúp phát triển các ứng dụng nhúng mạnh mẽ và ổn định.

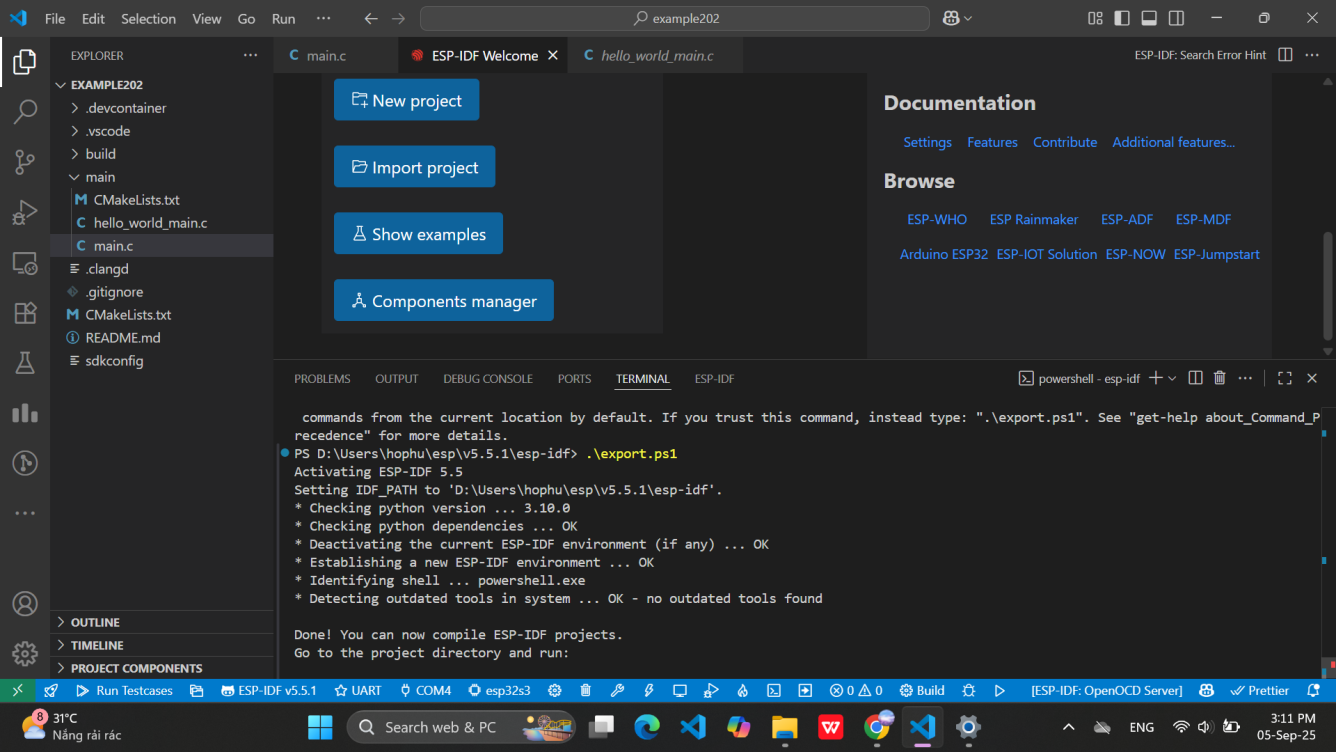
Cấu hình cho ESP-IDF: 



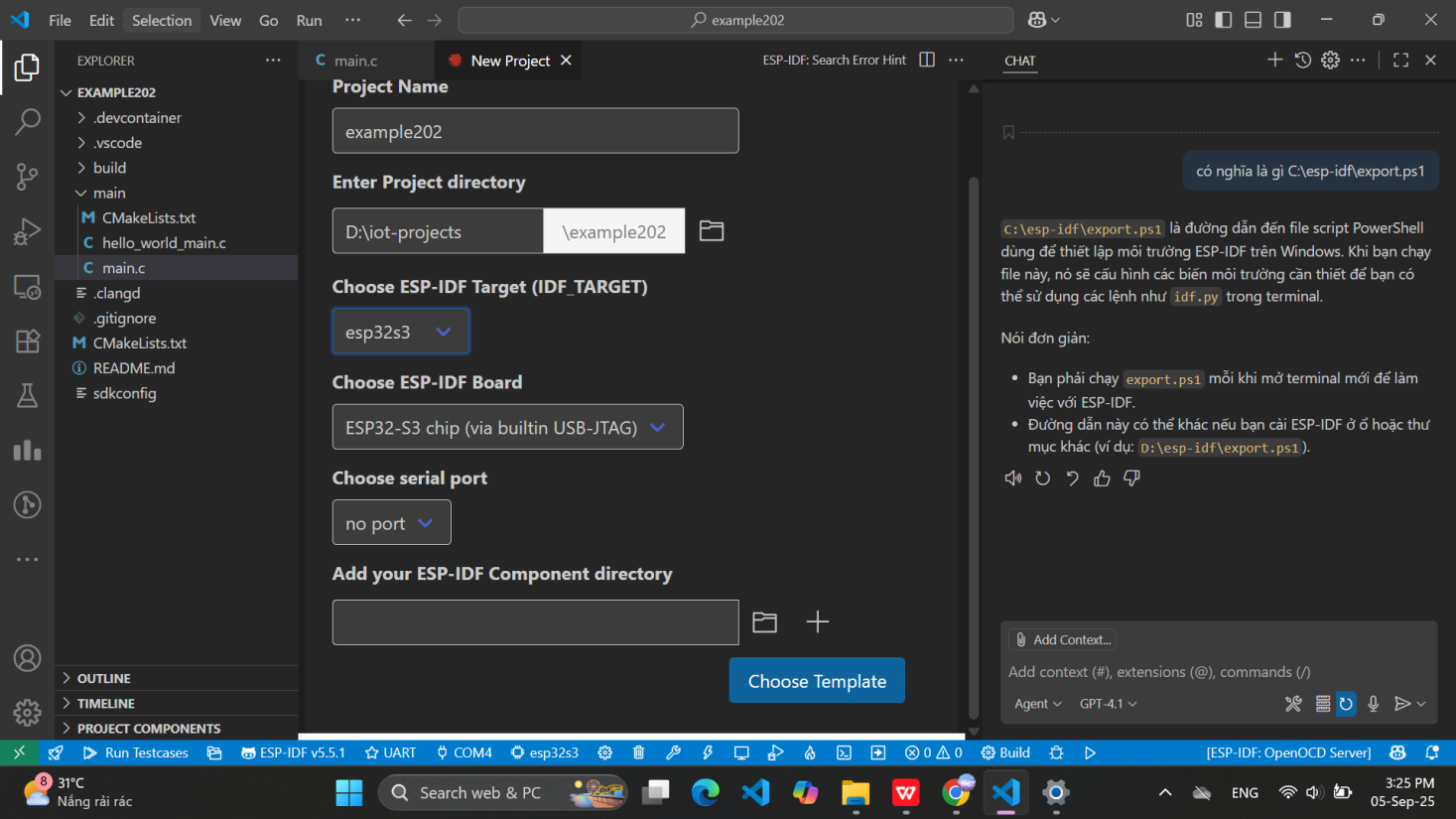
Cài đặt môi trường ảo python cho ESP-IDF:



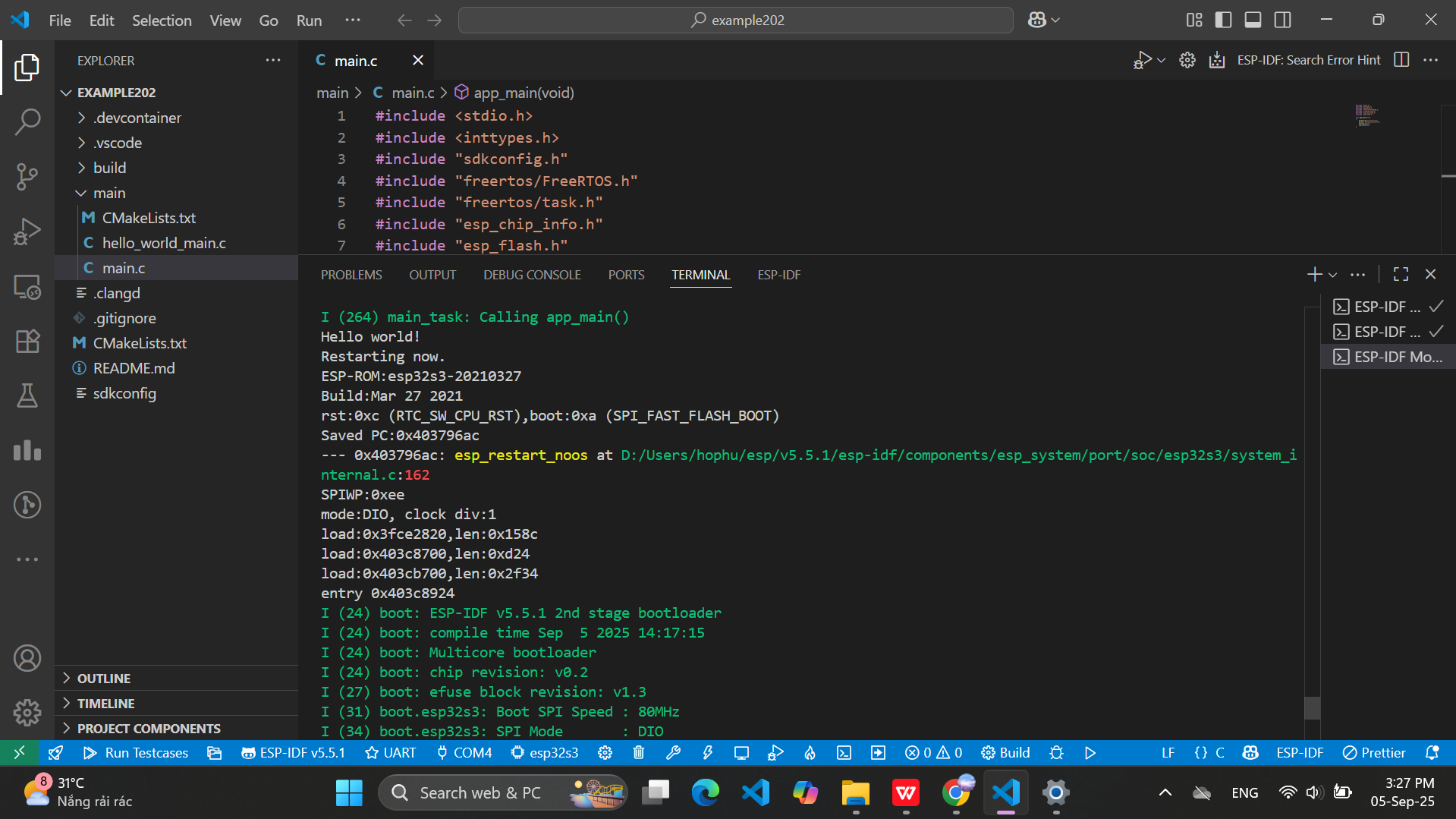
Thiết lập môi trường: Môi trường này giúp cài đặt và quản lý các thư viện Python cần thiết một cách độc lập, tránh xung đột với các môi trường Python khác trong hệ thống.



1. Ứng dụng Hello World**:**

**Tạo project mới**

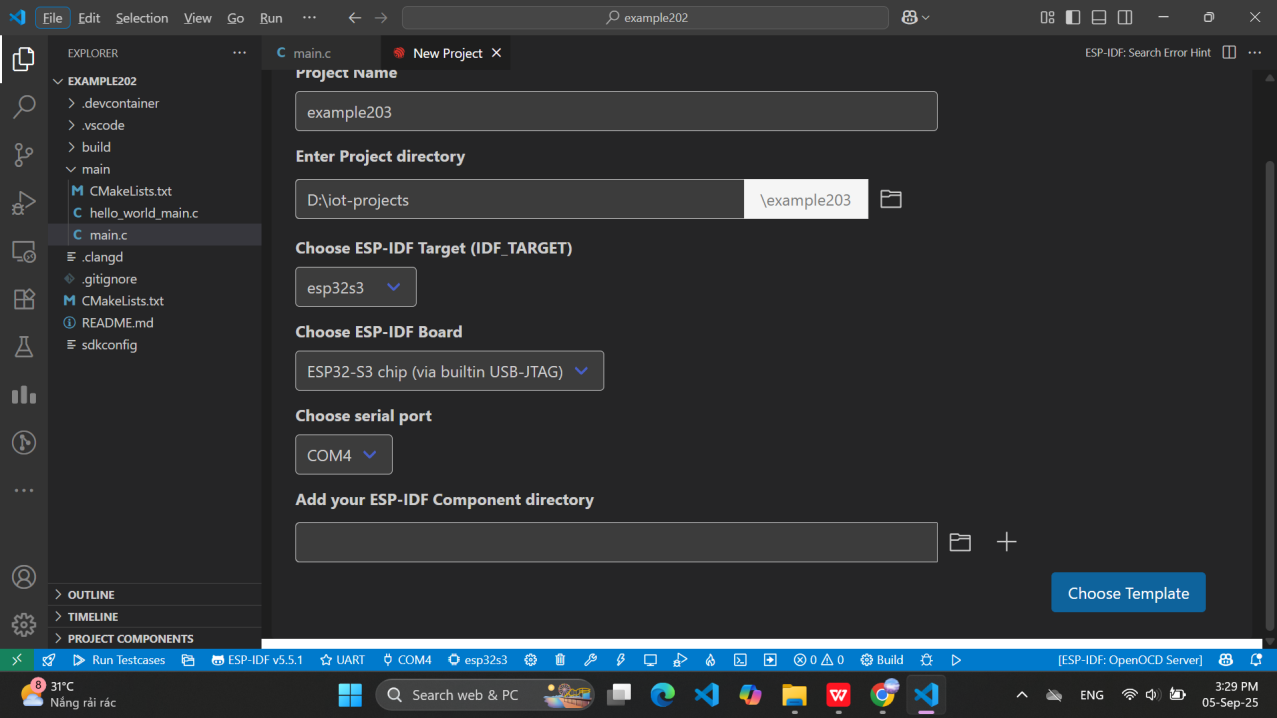
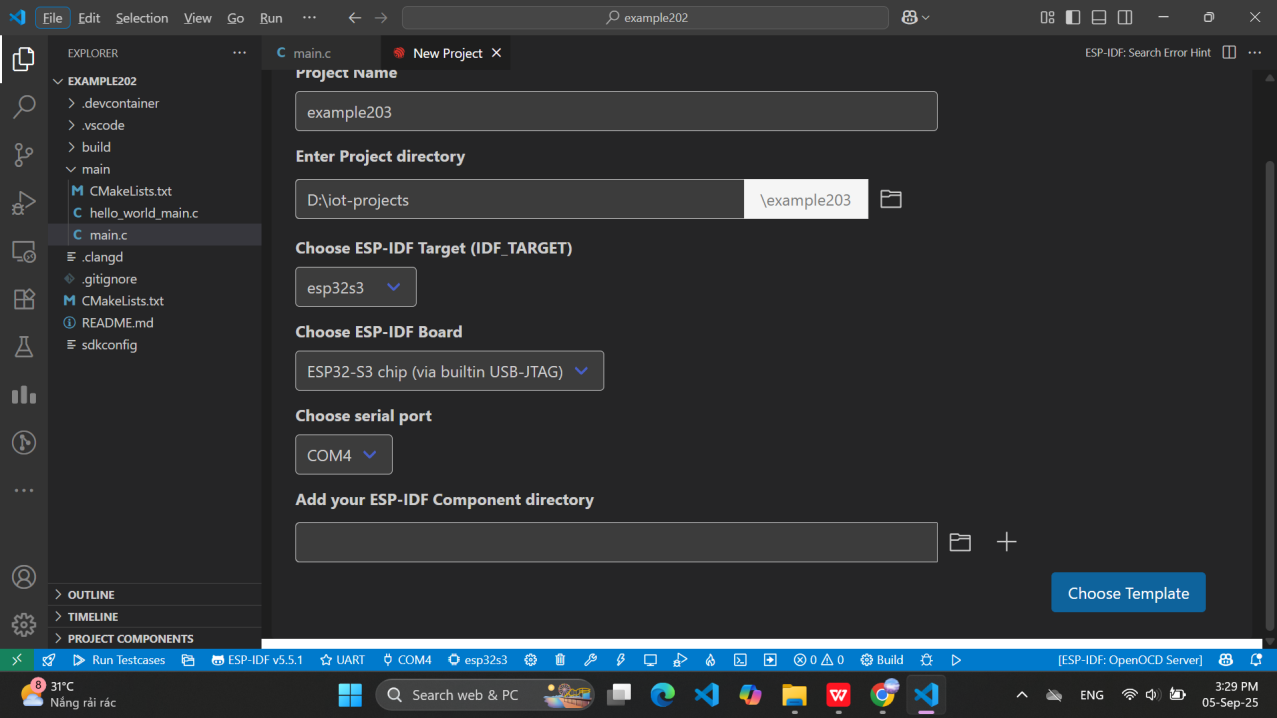
**Build project và nạp chương trình vào vi điều khiển**



**Giải thích code:**

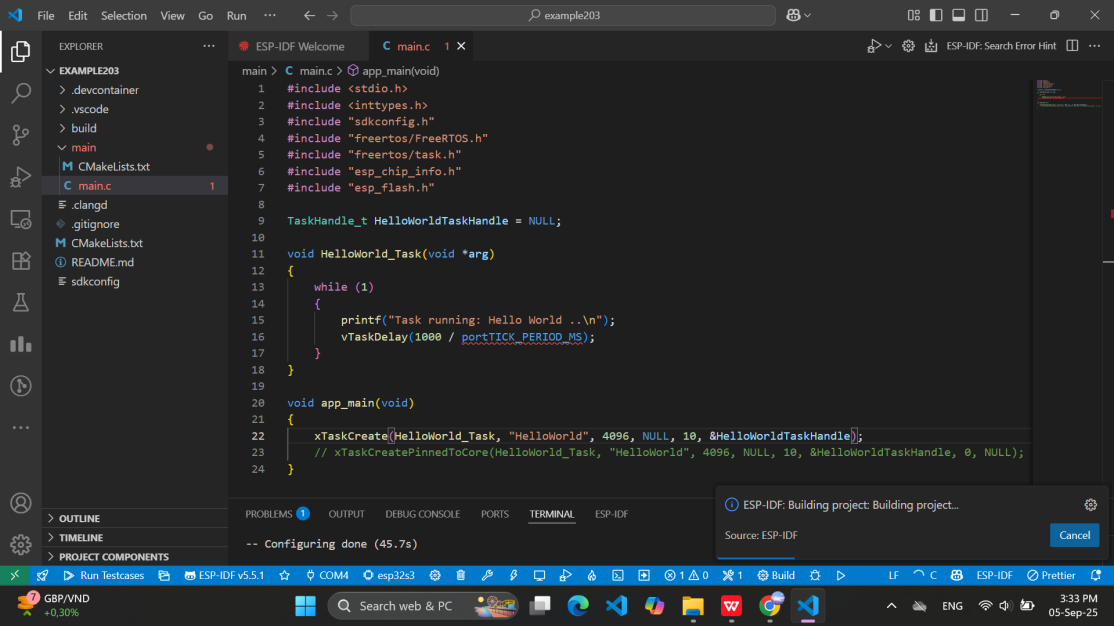
* #include <stdio.h>: thư viện để sử dụng hàm printf().
* #include <inttypes.h>: hỗ trợ định dạng in kiểu int8\_t, uint\_32…
* #include "sdkconfig.h": cấu hình hệ thống tự sinh ra khi build dự án (chỉ tồn tại khi chạy idf.py build).
* #include "freertos/FreeRTOS.h": thư viện của FreeRTOS hệ điều hành thời gian thực tích hợp sẵn trong ESP-IDF.
* #include "esp\_chip\_info.h" và #include "esp\_flash.h": Các hàm để truy xuất thông tin về chip ESP và bộ nhớ Flash.
* Hàm chính void app\_main(void): đây là hàm bắt buộc phải có trong chương trình ESP-IDF.Nó giống như main() trong C bình thường. ESP-IDF sẽ gọi app\_main() khi thiết bị khởi động.
* “fflush(stdout);” : Đảm bảo rằng dữ liệu được in ra ngay lập tức, không bị giữ lại trong bộ đệm.
* “esp\_restart();” : Hàm này sẽ reset lại vi điều khiển ESP32, tương đương với việc bạn nhấn nút "RESET" trên board.
* Kết quả chạy chương trình: Terminal sẽ hiển thị “**Hello word! Restarting now.**”, sau đó thiết bị sẽ tự động reset, và lại in ra y như vậy – vòng lặp vô tận nếu bạn không thay đổi gì.

1. **Sử dụng FreeRTOS tạo task in Hello World.**

**Tạo project mới**

Ở giao diện New Project chúng ta điền các thông tin như sau:

* Project Name: **example203**
* Enter Project directory: **D:\iot-projects**
* Choose ESP-IDF Board: **ESP32-C3 chip(via ESP USB Bridge)**
* Choose serial port: COMx
* Choose Template: **template-app**

**Thư mục dự án**

* **include khai báo thư viện:**
* **Stdio.h:** hỗ trợ hàm printf để in ra console.
* **Inttypes.h:** định nghĩa kiểu số nguyên chuẩn (int32\_t, uint64\_t, …).
* **sdkconfig.h:** file cấu hình được tạo khi build ESP-IDF project (chứa macro config).
* **FreeRTOS.h & task.h:** để dùng FreeRTOS API (tạo task, delay, quản lý đa luồng).
* **esp\_chip\_info.h và esp\_flash.h:** để lấy thông tin chip, flash (không dùng trong code này nhưng có thể cần mở rộng).
* **Khai báo task handle: TaskHandle\_t HelloWorldTaskHandle = NULL;**
* **TaskHandle\_t:** là kiểu dữ liệu để lưu handle của task (giúp quản lý task sau này: xóa, tạm dừng…).
* Gắn NULL vì chưa có task nào được tạo.
* **Định nghĩa task HelloWorld:**

void HelloWorld\_Task(void \*arg)

{

while (1)

{

printf("Task running: Hello World ..\n");

vTaskDelay(1000 / portTICK\_PERIOD\_MS);

}

}

* Đây là một task FreeRTOS chạy song song với các task khác.
* while(1) là vòng lặp vô hạn (task sẽ luôn chạy cho đến khi bị xóa).
* printf(“Task running: Hello World ..\n”); sẽ in ra thông báo liên tục.
* vTaskDelay(1000 / portTICK\_PERIOD\_MS);
  + Delay 1000ms ( 1 giây )
  + portTICK\_PERIOD\_MS = số ms trên mỗi tick ( thường = 1ms nếu config tick = 1000Hz)
* Kết quả: mỗi 1 giấy, task in ra “Task running: Hello World ..”
* **Hàm app\_main:**

void HelloWorld\_Task(void \*arg)

{

while (1)

{

printf("Task running: Hello World ..\n");

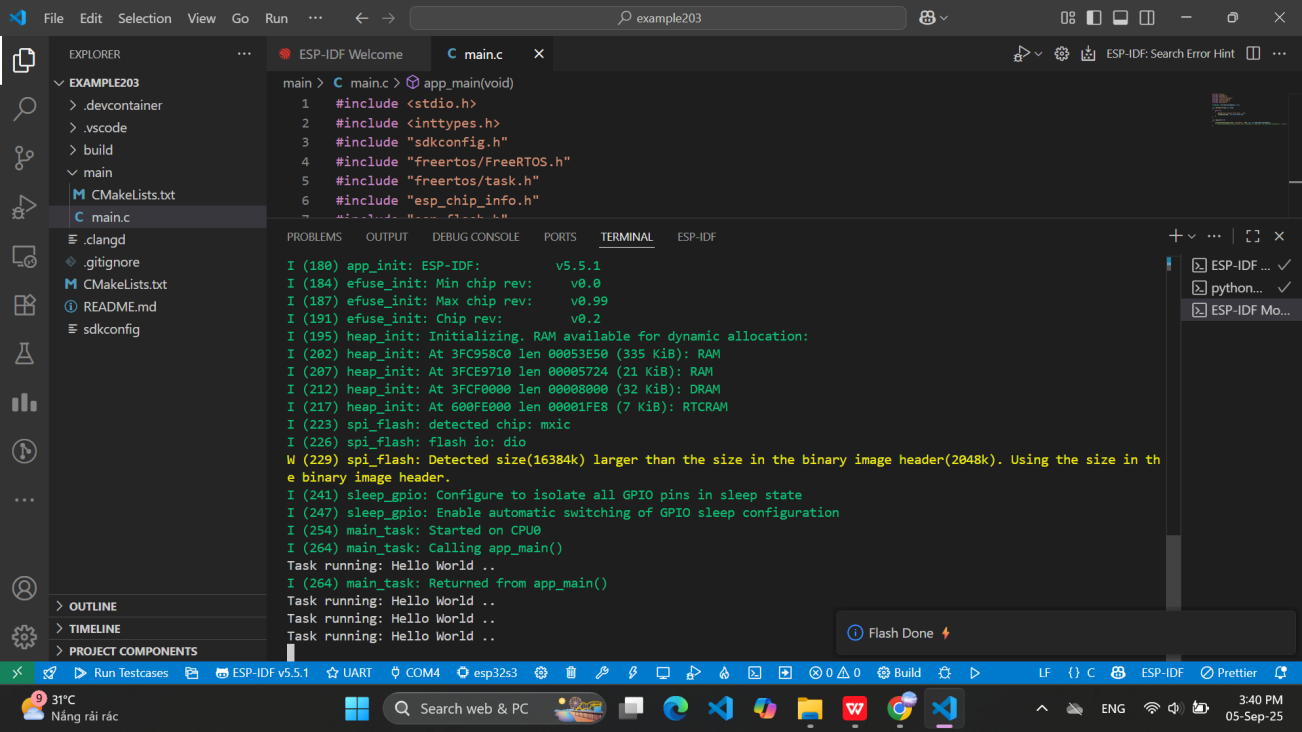
vTaskDelay(1000 / portTICK\_PERIOD\_MS);

}

}

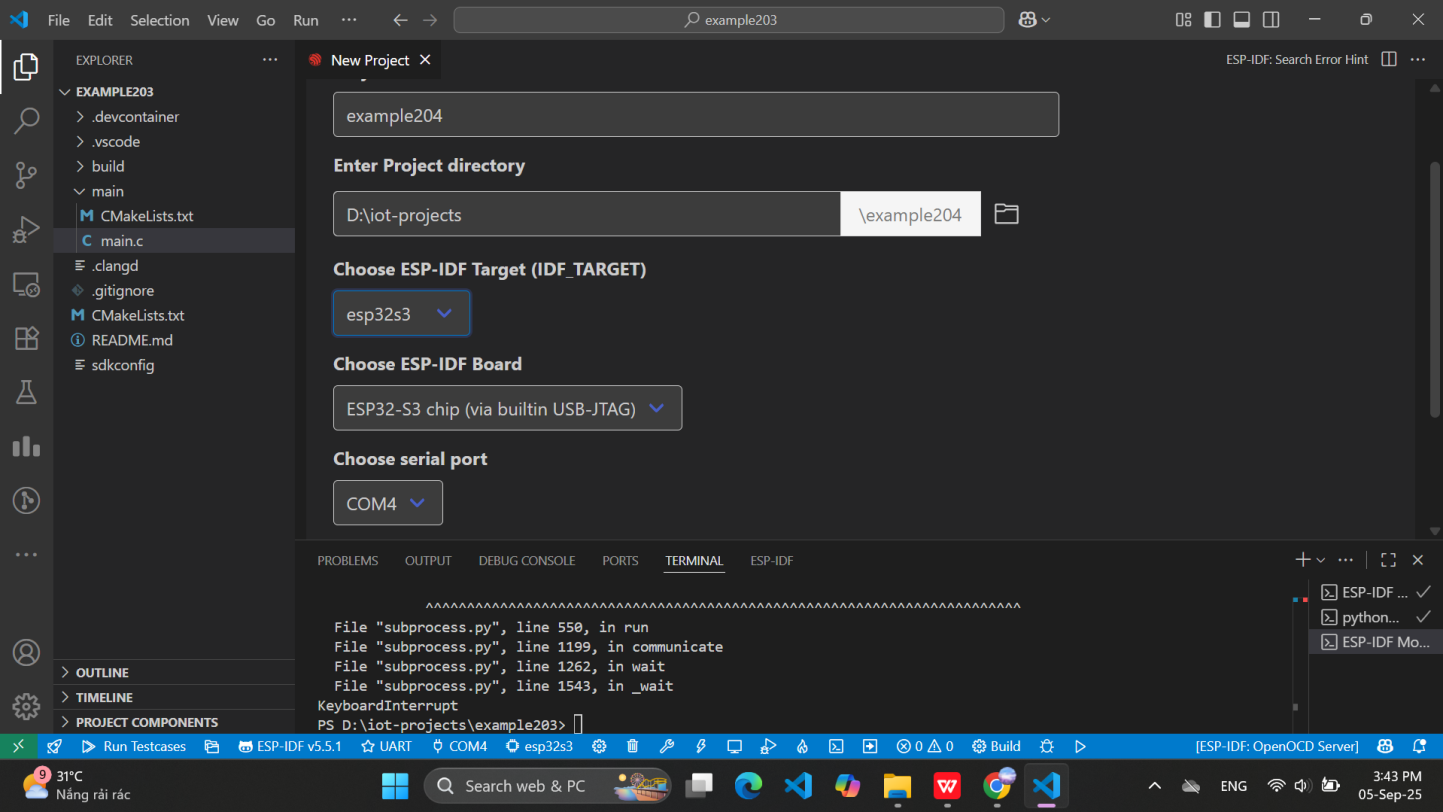
* App\_main là hàm entry point trong ESP-IDF
* xTaskCreate(): tạo một task FreeRTOS.
  + HelloWorld\_Task: hàm thực thi task.
  + “HelloWorld”: tên task ( dùng để debug )
  + 4069: kích thước stack
  + NULL: tham số truyền vào task
  + 10: mức ưu tiên của task
  + &HelloWorldTaskHandle: nơi lưu lại handle task.

**Build project và nạp chương trình vào vi điều khiển**

* SP32 khởi động thành công, nhận diện flash, RAM.
* Có cảnh báo flash thực tế 16 MB nhưng cấu hình project là 2 MB
* Task FreeRTOS chạy ổn định, in ra “Task running” Hello World ..” mỗi giây.

**4. Điều khiển LED với GPIO.**

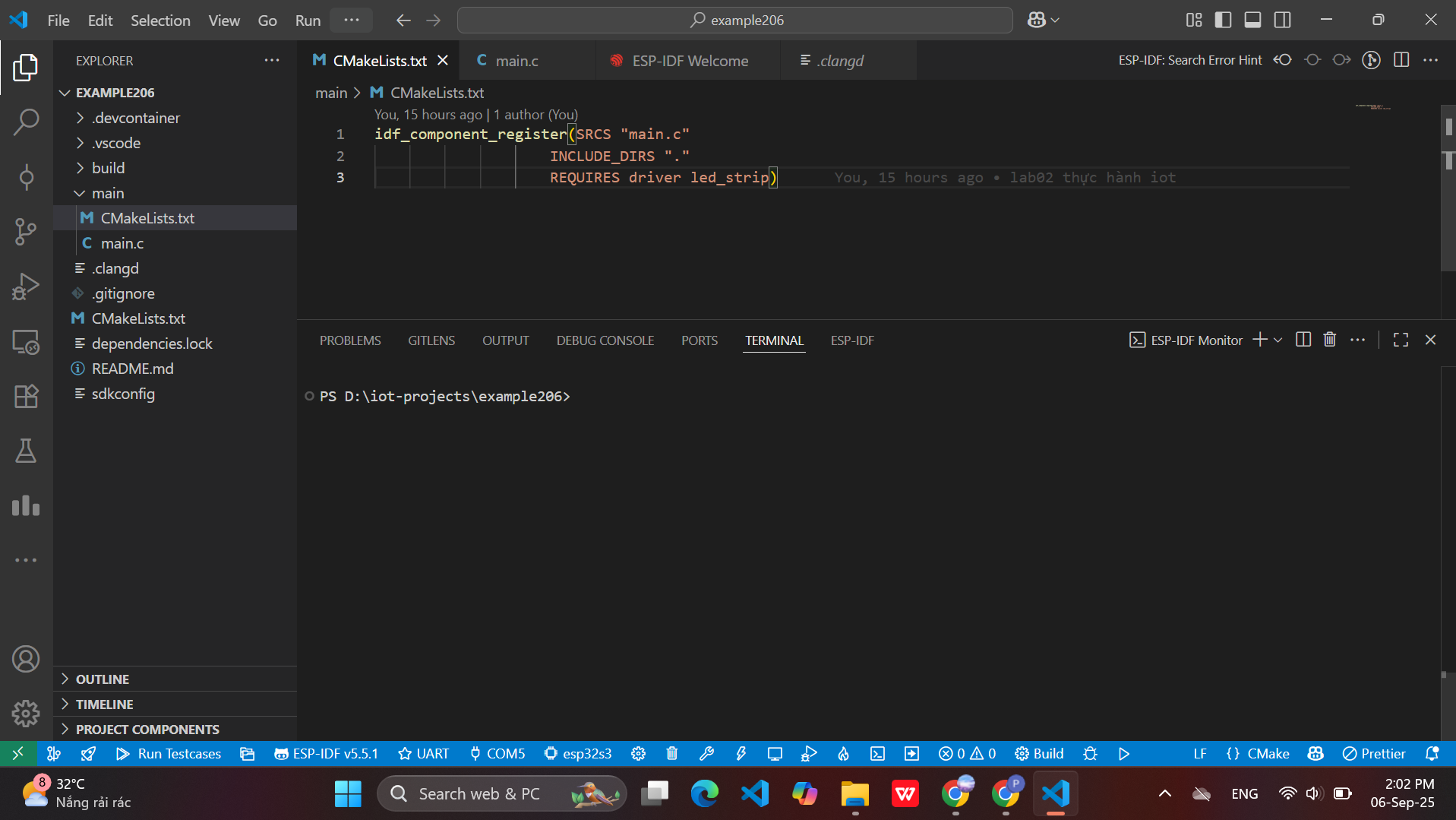
Tạo project



Đầu tiên tạo một project mới có tên là example204 và example205 vì cả hai làm chung với config như trên ảnh :

* Project Name: example204 hoặc example205
* Enter Project directory: D:\iot-projects
* Choose ESP-IDF Board: ESP32-C3 chip(via ESP USB Bridge)
* Choose serial port: COMx
* Choose Template: template-app

Do nhóm em dùng esp32-s3 nên nhóm có thêm thư viện Thêm thư viện **led\_strip** (https://components.espressif.com/components/espressif/led\_strip/versions/3.0.1~1/readme) trong CMakeLists.txt để điều khiển led



**- Trong example204**

* blink\_main.c

#include "freertos/FreeRTOS.h"

#include "freertos/task.h"

#include "esp\_log.h"

#include "led\_strip.h"

// Định nghĩa chân GPIO kết nối với LED WS2812

#define LED\_GPIO GPIO\_NUM\_48

void app\_main(void)

{

led\_strip\_config\_t strip\_config = {

.strip\_gpio\_num = LED\_GPIO,

.max\_leds = 1,

};

led\_strip\_rmt\_config\_t rmt\_config = {

.resolution\_hz = 10 \* 1000 \* 1000,

};

// Tạo handle cho dải LED

led\_strip\_handle\_t led\_strip;

led\_strip\_new\_rmt\_device(&strip\_config, &rmt\_config, &led\_strip);

// Xóa đèn LED về trạng thái tắt trước khi bắt đầu

led\_strip\_clear(led\_strip);

while (1)

{

// Bật LED màu đỏ

printf("Turning ON RED\n");

// Hàm set\_pixel vẫn theo thứ tự (R, G, B)

led\_strip\_set\_pixel(led\_strip, 0, 50, 0, 0);

led\_strip\_refresh(led\_strip);

vTaskDelay(1000 / portTICK\_PERIOD\_MS);

// Tắt LED (bằng cách xóa)

printf("Turning LED OFF\n");

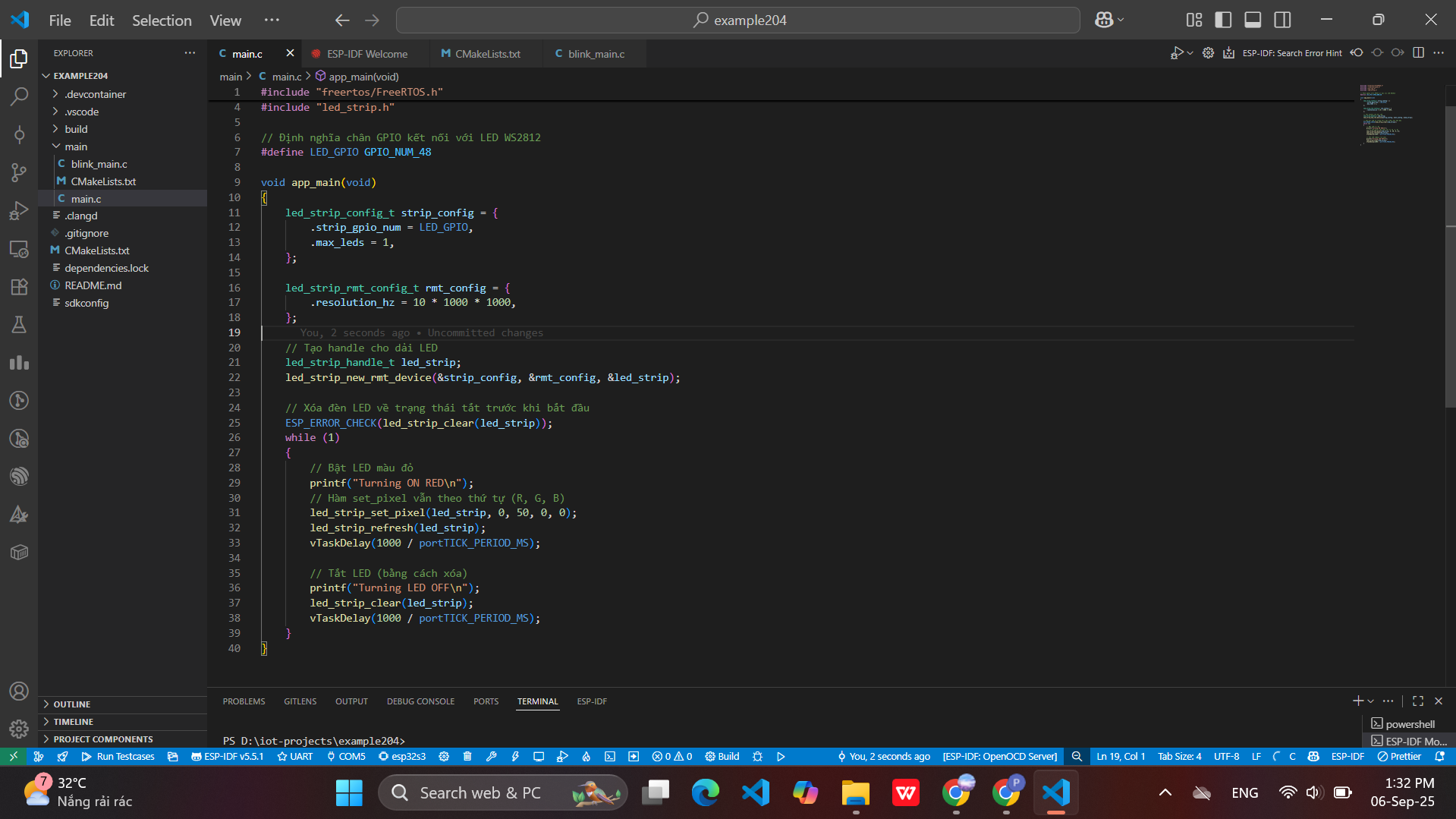
led\_strip\_clear(led\_strip);

vTaskDelay(1000 / portTICK\_PERIOD\_MS);

}

}

Do em dùng esp32-s3 dùng LED WS2812 nên em đổi qua GPIO48 với cũng đổi code ạ



Kết quả

* Giải thích code:
* **freertos/FreeRTOS.h** dùng để thao tác với task của hệ điều hành FreeRTOS
* **esp\_log.h** chứa hàm log của ESP-IDF
* **led\_strip.h** là thư viện điều khiển dải led WS2812 sử dụng RMT
* **#define LED\_GPIO GPIO\_NUM\_48** Đây là chân GPIO mà bạn kết nối chân data của WS2812. GPIO\_NUM\_48 là enum của ESP-IDF.
* **void app\_main(void)** hàm chính chạy trên lõi mặc định (FreeRTOS task khởi tạo bởi IDF). Bắt đầu khởi tạo cấu hình và thiết bị LED.
* **led\_strip\_config\_t strip\_config = { .strip\_gpio\_num = LED\_GPIO, .max\_leds = 1, }**; strip\_gpio\_num: số GPIO dùng để xuất tín hiệu data tới WS2812. max\_leds = 1: bạn khai báo dải LED có 1 LED
* **led\_strip\_rmt\_config\_t rmt\_config = { .resolution\_hz = 10 \* 1000 \* 1000, }**; Cấu hình cho driver RMT resolution\_hz = 10MHz nghĩa là bộ đếm RMT chạy ở tần số 10 MHz → 1 tick = 1/10MHz = 100 ns.
* Tạo handle cho dải LED hàm này khởi tạo device handle sử dụng RMT theo cấu hình đã cho và gán led\_strip để sử dụng sau này.
* **ESP\_ERROR\_CHECK(led\_strip\_clear(led\_strip));** led\_strip\_clear xóa trạng thái (đặt tất cả pixel = 0), ESP\_ERROR\_CHECK sẽ dừng chương trình nếu hàm trả lỗi
* Vòng lặp chính:
* **led\_strip\_set\_pixel(led\_strip, 0, 50, 0, 0);** thiết lập pixel thứ 0 (vị trí đầu) thành màu (R=50, G=0, B=0). led\_strip\_refresh(led\_strip);
* **led\_strip\_refresh(led\_strip);** gửi dữ liệu xuống dải LED qua RMT bắt buộc gọi sau khi thay đổi pixel để cập nhật trạng thái đèn.
* **vTaskDelay(1000 / portTICK\_PERIOD\_MS)**;dừng task hiện tại 1000 ms. portTICK\_PERIOD\_MS là số ms mỗi tick
* **led\_strip\_clear(led\_strip);** tắt LED sau đó bạn delay 1s.

**- Trong example205**

* main.c

#include "freertos/FreeRTOS.h"

#include "freertos/task.h"

#include "led\_strip.h"

// Chân GPIO kết nối với LED WS2812

#define LED\_GPIO GPIO\_NUM\_48

// Handle cho task

TaskHandle\_t BlinkyTaskHandle = NULL;

// Task nhấp nháy LED

void Blinky\_Task(void \*arg)

{

// Cấu hình LED strip

led\_strip\_config\_t strip\_config = {

.strip\_gpio\_num = LED\_GPIO,

.max\_leds = 1,

};

// Cấu hình RMT cho LED WS2812

led\_strip\_rmt\_config\_t rmt\_config = {

.resolution\_hz = 10 \* 1000 \* 1000, // 10 MHz

};

// Tạo handle cho LED strip

led\_strip\_handle\_t led\_strip;

led\_strip\_new\_rmt\_device(&strip\_config, &rmt\_config, &led\_strip);

// Xóa LED về trạng thái tắt ban đầu

led\_strip\_clear(led\_strip);

while (1)

{

// Bật LED màu đỏ

printf("LED ON\n");

led\_strip\_set\_pixel(led\_strip, 0, 50, 0, 0); // (R, G, B)

led\_strip\_refresh(led\_strip);

vTaskDelay(1000 / portTICK\_PERIOD\_MS);

// Tắt LED

printf("LED OFF\n");

led\_strip\_clear(led\_strip);

vTaskDelay(1000 / portTICK\_PERIOD\_MS);

}

}

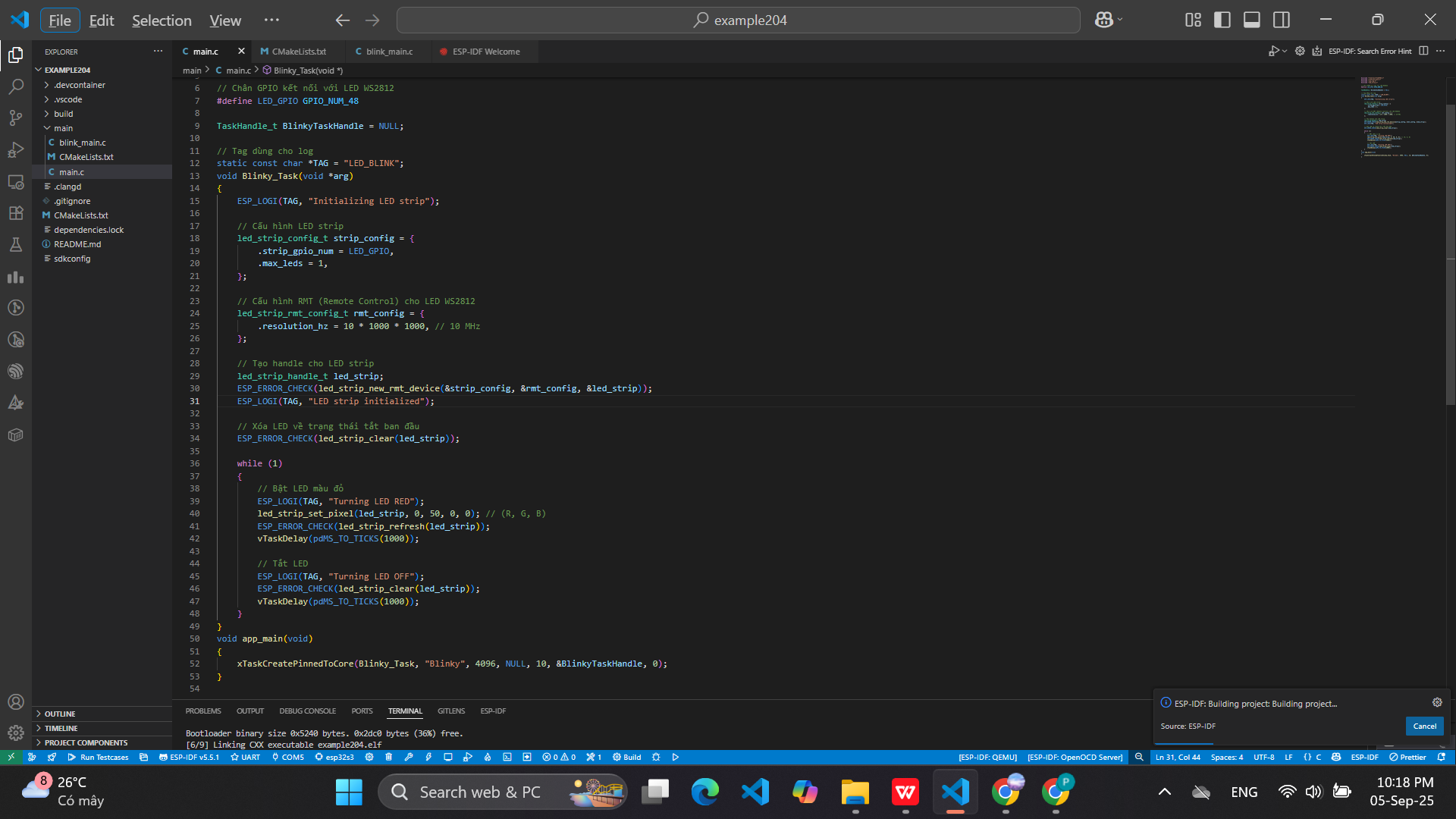
// Hàm main

void app\_main(void)

{

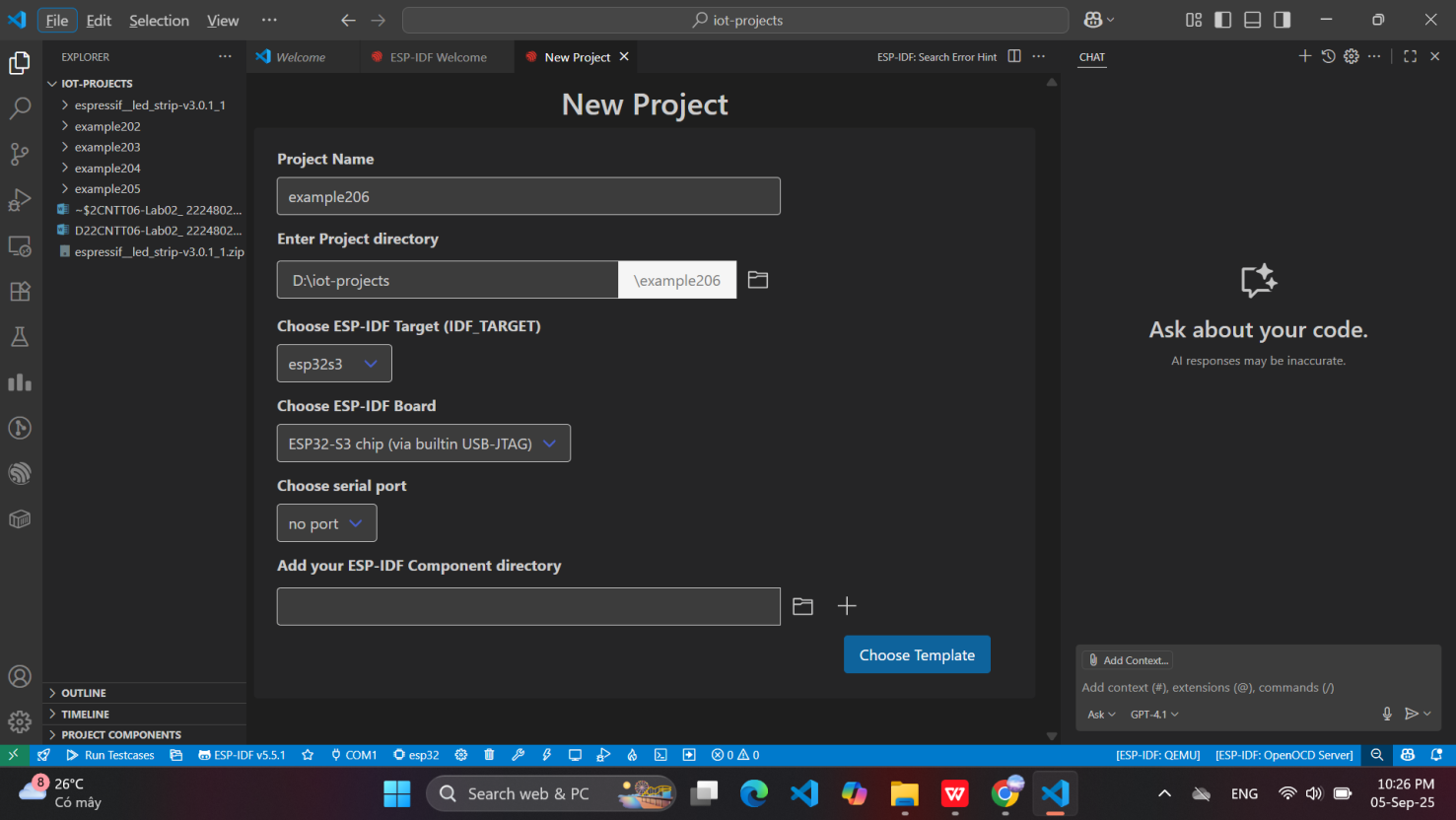
xTaskCreatePinnedToCore(Blinky\_Task, "Blinky", 4096, NULL, 10, &BlinkyTaskHandle, 0);

}

- Giải thích code :

* Phần include thư viện , ta sẽ import FreeRTOS.h, task.h thì được dùng để thao tác với task của hệ điều hành FreeRTOS (ESP32 luôn chạy FreeRTOS) . Thư viện esp\_log.h dùng để ghi log và led\_strip.h dùng để hổ trợ điều khiển LED WS2812 thông qua RMT( giao thức điều khiển độ chính xác cao).
* Define LED\_GPIO GPIO\_NUM\_48 : sẽ định nghĩa GPIO để điều khiển LED, sử dụng chân GPIO 48 để xuất tín hiệu điều khiển LED WS2812 .
* TaskHandle\_t BlinkyTaskHandle = NULL, sẽ tạo biến để lưu trữ handle của task , giúp quản lý task sau này nếu muốn dừng và sửa task.
* Void Blinky\_Task(void \*arg) : đây là một FreeRTOS task, chạy song song trong hệ thống, chứa logic để điều khiển LED.
* Được gọi một lần trong app\_main() và chạy liên tục trong vòng lặp .
* Cấu hình dải LED WS2812 : strip\_gpio\_num: GPIO để xuất tín hiệu và max\_leds: Số lượng LED trong dải — ở đây chỉ có 1.
* Khởi tạo LED Strip : led\_strip là handle để thao tác sau này với LED.
* Ở vòng lặp while :
* led\_strip\_set\_pixel(): Thiết lập màu sắc cho pixel (ở đây là LED số 0).
* led\_strip\_refresh(): Áp dụng thay đổi (gửi dữ liệu ra LED).
* vTaskDelay(): Dừng task trong khoảng thời gian (ở đây là 1000ms = 1 giây).
* led\_strip\_clear(): Tắt LED (gửi giá trị 0,0,0).

1. **Kết hợp FreeRTOS + GPIO (quản lý task).**

Tạo project

Sủa code trong main.c

#include <stdio.h>

#include "freertos/FreeRTOS.h"

#include "freertos/task.h"

#include "driver/gpio.h"

#include "esp\_log.h"

#include "sdkconfig.h"

#include "led\_strip.h"

#define LED\_GPIO GPIO\_NUM\_48

TaskHandle\_t HelloWorldTaskHandle = NULL;

TaskHandle\_t BlinkyTaskHandle = NULL;

void HelloWorld\_Task(void \* arg) {

while (1) {

printf("Task running: Hello World ..\n");

vTaskDelay(1000 / portTICK\_PERIOD\_MS);

}

}

void Blinky\_Task(void \* arg) {

// Cấu hình LED strip

led\_strip\_config\_t strip\_config = {

.strip\_gpio\_num = LED\_GPIO,

.max\_leds = 1,

};

// Cấu hình RMT (Remote Control) cho LED WS2812

led\_strip\_rmt\_config\_t rmt\_config = {

.resolution\_hz = 10 \* 1000 \* 1000, // 10 MHz

};

// Tạo handle cho LED strip

led\_strip\_handle\_t led\_strip;

ESP\_ERROR\_CHECK(led\_strip\_new\_rmt\_device( & strip\_config, & rmt\_config, & led\_strip));

int count\_second = 0;

while (1) {

count\_second += 1;

switch (count\_second) {

case 10:

vTaskSuspend(HelloWorldTaskHandle);

printf("HelloWorld task suspended .. \n");

break;

case 14:

vTaskResume(HelloWorldTaskHandle);

printf("HelloWorld task resumed .. \n");

break;

case 20:

vTaskDelete(HelloWorldTaskHandle);

printf("HelloWorld task deleted .. \n");

break;

default:

break;

}

led\_strip\_set\_pixel(led\_strip, 0, 50, 0, 0); // (R, G, B)

ESP\_ERROR\_CHECK(led\_strip\_refresh(led\_strip));

vTaskDelay(1000 / portTICK\_PERIOD\_MS);

// Tắt LED

ESP\_ERROR\_CHECK(led\_strip\_clear(led\_strip));

vTaskDelay(1000 / portTICK\_PERIOD\_MS);

}

}

void app\_main(void) {

xTaskCreatePinnedToCore(Blinky\_Task, "Blinky", 4096, NULL, 10, & BlinkyTaskHandle, 0);

xTaskCreatePinnedToCore(HelloWorld\_Task, "HelloWorld", 4096, NULL, 10, & HelloWorldTaskHandle, 1);

}

* **Thêm thư viện:**
  + #include <stdio.h> // Thư viện chuẩn cho các nhập xuất dữ liệu, ở đây dùng cho hàm printf().
  + #include "freertos/FreeRTOS.h" // Thư viện lõi của FreeRTOS.
  + #include "freertos/task.h" // Thư viện quản lý các Task (tạo, xóa, delay...).
  + #include "driver/gpio.h" // Thư viện điều khiển các chân GPIO (General Purpose Input/Output).
  + #include "sdkconfig.h" // Chứa các cấu hình của project.
  + #include "led\_strip.h" // Thư viện chuyên dụng để điều khiển các loại LED
* #define LED\_GPIO GPIO\_NUM\_48
  + Định nghĩa biến toàn cục ở đây là chân 48, do nhóm em dùng esp32-s3 nên dùng chân này để điều khiển

TaskHandle\_t HelloWorldTaskHandle = NULL;

TaskHandle\_t BlinkyTaskHandle = NULL;

* + Khai báo 2 biến để điều khiển công việc
* Hàm **HelloWorld\_Task**

void HelloWorld\_Task(void \* arg) {

while (1) {

printf("Task running: Hello World ..\n");

vTaskDelay(1000 / portTICK\_PERIOD\_MS);

}

}

* + Vòng lặp vô hạn để in log Hello World sau mỗi 1 giây

Hàm **Blinky\_Task**

**// Cấu hình LED strip**

**led\_strip\_config\_t strip\_config = {**

**.strip\_gpio\_num = LED\_GPIO,**

**.max\_leds = 1,**

**};**

**// Cấu hình RMT (Remote Control) cho LED WS2812**

**led\_strip\_rmt\_config\_t rmt\_config = {**

**.resolution\_hz = 10 \* 1000 \* 1000, // 10 MHz**

**};**

**// Tạo handle cho LED strip**

**led\_strip\_handle\_t led\_strip;**

**led\_strip\_new\_rmt\_device( & strip\_config, & rmt\_config, & led\_strip);**

* Cấu hình led và khai báo **led\_strip** đẻ điều khiển led

int count\_second = 0;

while (1) {

count\_second += 1;

switch (count\_second) {

case 10:

vTaskSuspend(HelloWorldTaskHandle);

printf("HelloWorld task suspended .. \n");

break;

case 14:

vTaskResume(HelloWorldTaskHandle);

printf("HelloWorld task resumed .. \n");

break;

case 20:

vTaskDelete(HelloWorldTaskHandle);

printf("HelloWorld task deleted .. \n");

break;

default:

break;

}

led\_strip\_set\_pixel(led\_strip, 0, 50, 0, 0); // (R, G, B)

ESP\_ERROR\_CHECK(led\_strip\_refresh(led\_strip));

vTaskDelay(1000 / portTICK\_PERIOD\_MS);

// Tắt LED

ESP\_ERROR\_CHECK(led\_strip\_clear(led\_strip));

vTaskDelay(1000 / portTICK\_PERIOD\_MS);

}

* count\_second đếm số giây trôi qua.
* Sau 10 giây: tạm dừng HelloWorld\_Task.
* Sau 14 giây: tiếp tục HelloWorld\_Task.
* Sau 20 giây: xóa hẳn HelloWorld\_Task.

led\_strip\_set\_pixel(led\_strip, 0, 50, 0, 0); // (R, G, B)

ESP\_ERROR\_CHECK(led\_strip\_refresh(led\_strip));

vTaskDelay(1000 / portTICK\_PERIOD\_MS);

// Tắt LED

ESP\_ERROR\_CHECK(led\_strip\_clear(led\_strip));

vTaskDelay(1000 / portTICK\_PERIOD\_MS);

Đây là logic điều khiển LED nhấp nháy:

* led\_strip\_set\_pixel(): Đặt màu cho LED
* led\_strip\_refresh(): Gửi dữ liệu màu ra cho LED để nó hiển thị.
* vTaskDelay(): Chờ 1 giây.
* led\_strip\_clear(): Tắt LED.
* vTaskDelay(): Chờ 1 giây.

void app\_main(void) {

xTaskCreatePinnedToCore(Blinky\_Task, "Blinky", 4096, NULL, 10, & BlinkyTaskHandle, 0);

xTaskCreatePinnedToCore(HelloWorld\_Task, "HelloWorld", 4096, NULL, 10, & HelloWorldTaskHandle, 1);

}

Tạo 2 task chạy song song:

* + Blinky\_Task chạy trên core 0.
  + HelloWorld\_Task chạy trên core 1.

Trong đó hàm xTaskCreatePinnedToCore() gồm các tham số lần lượt là: tham số đầu tiên là tên hàm cần thực hiện, tên task, kích thước vùng nhớ, tham số cần truyền vào hàm, đô ưu tiên, con trỏ task, id của core

**YÊU CẦU CHUNG**

1. **Đánh giá**

* Chuẩn bị tốt các yêu cầu đặt ra trong bài thực hành.
* Sinh viên hiểu và tự thực hiện được bài thực hành, trả lời đầy đủ các yêu cầu đặt ra.
* Nộp báo cáo kết quả chi tiết những đã thực hiện, quan sát thấy và kèm ảnh chụp màn hình kết quả (nếu có); giải thích cho quan sát (nếu có).
* Sinh viên báo cáo kết quả thực hiện và nộp bài.

1. **Báo cáo**

* File .PDF hoặc .docx. Tập trung vào nội dung, giải thích.
* Nội dung trình bày bằng Font chữ Cambria hoặc Times New Roman (tuy nhiên, phải chuyển đổi hết báo cáo này sang 1 font chữ thống nhất) – cỡ chữ 13. Canh đều (Justify) cho văn bản. Canh giữa (Center) cho ảnh chụp.
* Đặt tên theo định dạng: Mã lớp-LabX\_MSSVTrưởngNhóm\_HọVàTênTrưởngNhóm. (trong đó X là Thứ tự buổi Thực hành).

Ví dụ: D21KTPM02-Lab04\_ 2124801030059\_ HoVaTen

* Nếu báo cáo có nhiều file, nén tất cả file vào file .ZIP với cùng tên file báo cáo.
* Không đặt tên đúng định dạng – yêu cầu, sẽ KHÔNG chấm điểm bài thực hành.
* Nộp file báo cáo trên theo thời gian đã thống nhất.

**Bài sao chép, trễ, ... sẽ được xử lý tùy mức độ vi phạm.**

**HẾT**