

Operating System Lab

Lab #9

Real-Time OS

Goal:

We will learn the FreeRTOS via a ESP32 simulator.

Preparation:

- This lab requires VSCode for development. If you do not have it, download and install it.
 - This lab is unlike previous labs so it does not need to use Docker Container environment.
 - You might see many files unresolved in the beginning. This is due to uninstalled extensions.

Part I: Environment Setup for VSCode

In this lab, we will use FreeRTOS on a ESP32 board simulator to learn how FreeRTOS controls task with its scheduler. The ESP32 uses a [customed version](#) of FreeRTOS, called [ESP32-IDF](#) (the development framework for Espressif SoCs). Because a ESP32 hardware board is not used, there is no deployment process (i.e., flashing to a ESP32 process) after build. The ESP32 simulator we use is called “[Wokwi](#)” for development.

- **ESP32 Simulator Wokwi Installation**

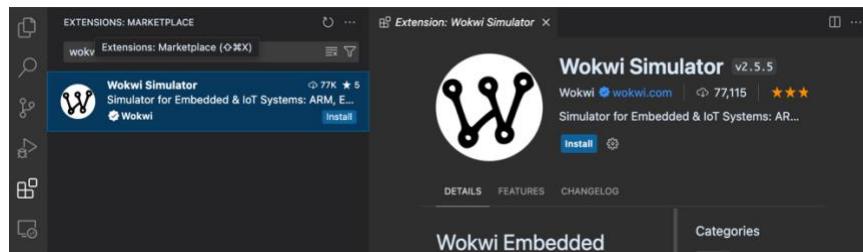


Figure 1 The Wokwi Extension

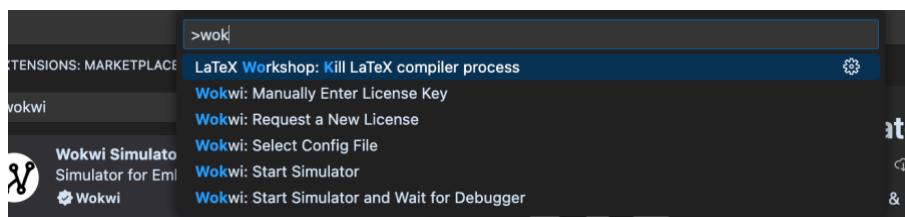


Figure 2 Request a New License Key

- The Step-by-Step Wokwi extension installation in VSCode
 1. Find the Wokwi extension in VSCode (see Fig. 1)
 2. Press F1 and select "Wokwi: Request a new License" (see Fig. 2). VS Code will ask you confirm opening the Wokwi website in your browser. Confirm by clicking "Open".
 3. Then click on the button that says "GET YOUR LICENSE" (see Fig. 3) . You may be asked to sign in to your Wokwi account. If you don't have an account, you can create one for free.

- The browser will ask for a confirmation to send the license to VS Code (see Fig. 4). Confirm (you may have to confirm twice, once in the browser, and once in VS Code). You'll see a message in VS Code that says "License activated for [your name]".

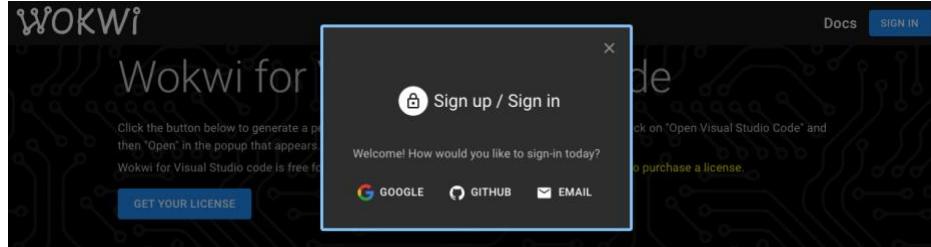


Figure 3 Website for Getting a License and Sign Up

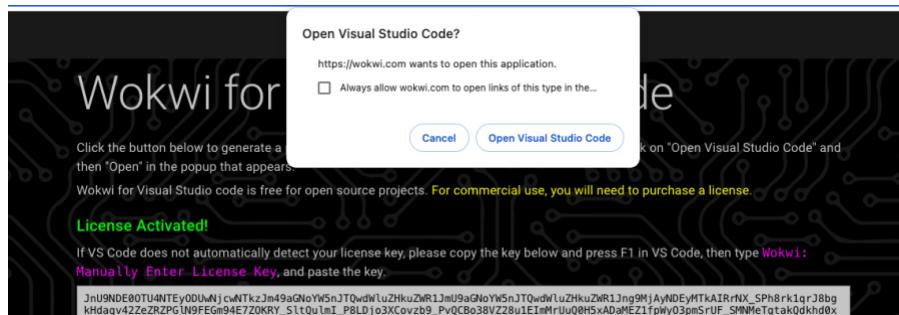


Figure 4 License Activation and Delivery

Two important files in a Wokwi simulator are “wokwi.tomi” and “diagram.json”. The “wokwi.tomi” defines which elf and bin files you will load into simulator. In our lab, we will later build our project into a “[embedded_os_lab.bin](#)” file inside the “[build](#)” folder. Regarding “diagram.json”, it defines which ESP32 board will be used for simulating. If you have “Wokwi” extension installed, it will show a diagram instead of a JSON file. If you use a text editor to open the “diagram.json”, you can see the board connections, sensors,...etc. are defined.

- ESP32 IDF Extension Installation in VSCode**

ESP32 IDF is the customized version of FreeRTOS. The extension helps the developers to deal with implementation, building, deployment, etc.

Our step-by-step VSCode extension installation guide is a simplified version of the detailed online guide and it is for Windows version. If you are using a MacOS, the VSCode part should be similar but the path will be different.

If you want to look at the detailed version or your OS is not Windows, you should check the URL: <https://github.com/espressif/vscode-esp-idf-extension/blob/master/docs/tutorial/install.md>

- Open the Extensions view by clicking on the Extension icon in the Activity Bar on the side of Visual Studio Code or the View: Extensions command (shortcut: ⌘ X or Ctrl+Shift+X).
- Search the extension with any related keyword like espressif, esp-idf, esp32, esp32s2, etc.

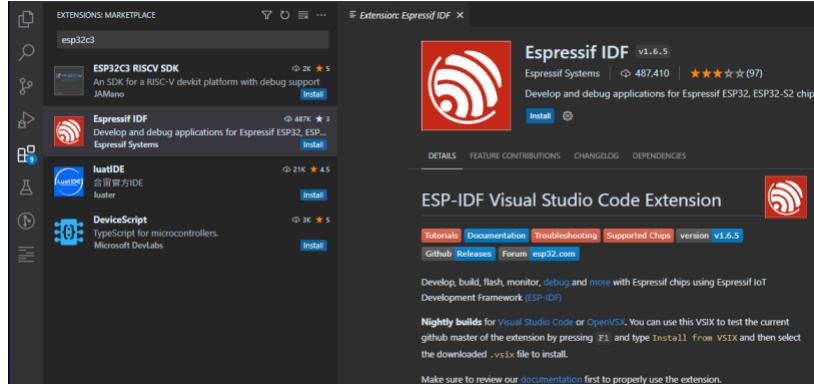


Figure 5 Extension of ESP-IDF

- In Visual Studio Code, select menu "View" and "Command Palette" and type [configure esp-idf extension]. After, choose the **ESP-IDF: Configure ESP-IDF extension option**. You can also choose where to save settings in the setup wizard.

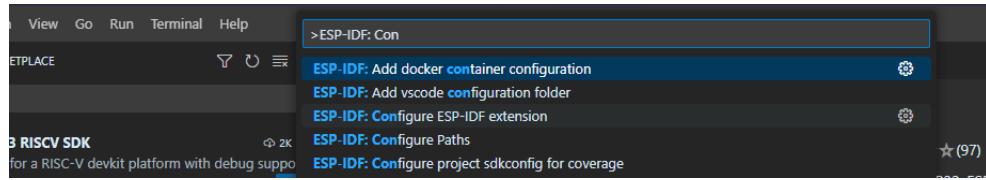


Figure 6 Command Palette: for ESP-IDF

- Now the setup wizard window will be shown with several setup options: Express, Advanced or Use existing setup.

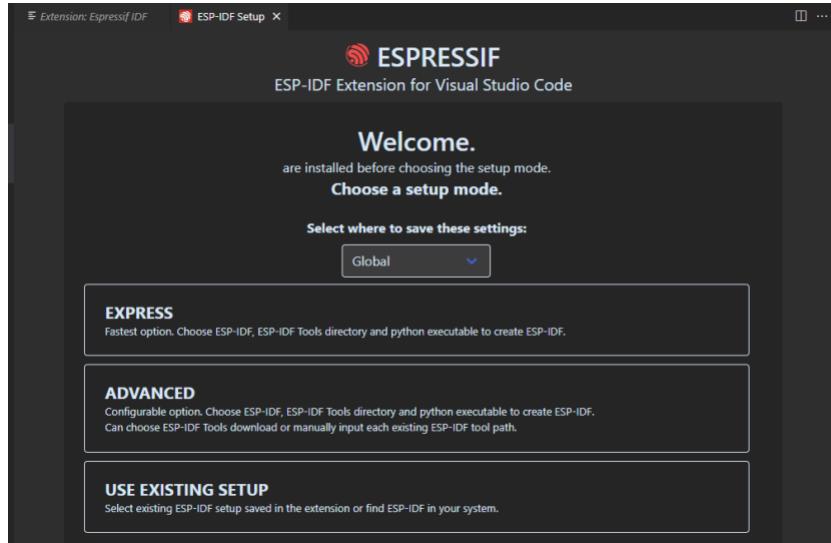


Figure 7 ESP-IDF Setup Mode

- Choose **Express** for the fastest option (or **Use Existing Setup** if ESP-IDF is already installed)
- If you choose **Express** setup mode:
 - Pick an ESP-IDF version to download (or find ESP-IDF in your system) and the python executable to create the virtual environment.
 - Choose the location for ESP-IDF Tools and python virtual environment (also known as `IDF_TOOLS_PATH`) which is `$HOME\.espressif` on MacOS/Linux and `%USERPROFILE%\espressif` on Windows by default.

- NOTE: Windows users don't need to select a python executable since it is part of the setup.
- NOTE: Make sure that IDF_PATH and IDF_TOOLS_PATH doesn't have any spaces to avoid any build issues.

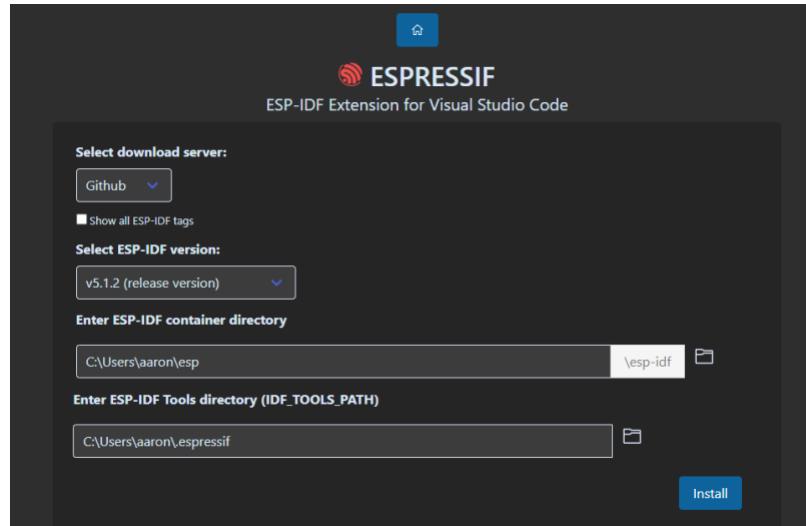


Figure 8 Configure the ESP-IDF

- The user will see a page showing the setup progress status showing ESP-IDF download progress, ESP-IDF Tools download and install progress as well as the creation of a python virtual environment.

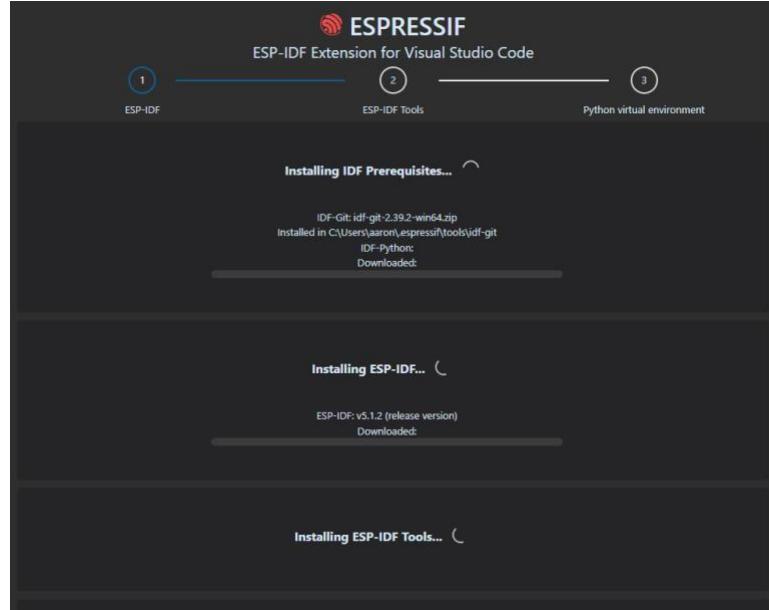
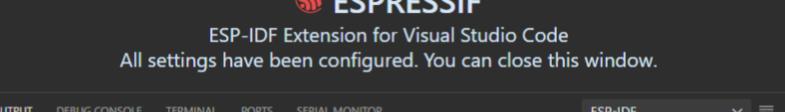


Figure 9 Installation Status for ESP-IDF

- If everything is installed correctly, the user will see a message that all settings have been configured. You can start using the extension.



The screenshot shows the Visual Studio Code interface with the ESP-IDF Extension installed. The title bar says "Extension: Espressif IDF" and "ESP-IDF Setup X". The main area has a dark background with the "ESPRESSIF" logo and the text "ESP-IDF Extension for Visual Studio Code". A message below it says "All settings have been configured. You can close this window." Below this are several tabs: PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, PORTS, and SERIAL MONITOR. The OUTPUT tab is active and shows the following log output:

```
Python packages installed successfully.
5\esp_debug_adapter\requirements.txt (line 6) (1.26.18)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\Aaron\.espressif\python_env\idf5.1_py3.
11_env\lib\site-packages (from requests>=2.21.0->-r c:\users\Aaron\.vscode\extensions\espressif.esp-idf-extension-1.6.
5\esp_debug_adapter\requirements.txt (line 6)) (2023.11.17)
Downloading esp_debug_backend-1.0.3-py3-none-any.whl (23 kB)
Installing collected packages: pywin32, esp-debug-backend
Successfully installed esp-debug-backend-1.0.3 pywin32-306
```

At the bottom, a message says "Python requirements has been installed".

Figure 10 Installation Output

- Installation Verification via the First Program - HelloWorld

1. This check-out program contains the HelloWorld program. Inside the “main” folder, the “[hello_world_main.c](#)” is your first FreeRTOS program.
 2. Now build the project. Use the command [ESP-IDF: Build your project](#). You will see a new terminal show the building process. After a successful build, you should be able to see the “[embedded_os_lab.bin](#)” in your project’s “build” folder. (The build will be created when you build the application.)

```
18 #include "freertos/FreeRTOS.h"
19 #include "freertos/task.h"
20 #include "esp_chip_info.h"

PROBLEMS 5 OUTPUT DEBUG CONSOLE TERMINAL PORTS SERIAL MONITOR

Merged 1 ELF section
Successfully created esp32c3 image.
Generated C:/Users/aaron/G_My_Drive/VM_Share_Drive/dev_workspace/course_project\swen310\lab1\lab9_2023_hello_world\build\bootloader\bootloader.elf
[96/96] cmd.exe /C "cd /D C:/Users/aaron/G_My_Drive/VM_Share_Drive/dev_workspace/course_project\swen310\lab1\lab9_2023_hello_world\build\bootloader & python .\espflash.py && C:/Users/aaron/.espressif/python_env/ldfs.1_py3.11_env\Scripts\python.exe C:/Users/aaron/esp\__idf_driver\driver.dir/gpio/gpio.c.obj"
Building Project
① Building Project
000 bootloader 0x0 C:/Users/aaron/G_My_Drive/VM_Share_Drive/dev_workspace/course_project\swen310\lab1\lab9_2023_hello_world\build\bootloader\bootloader.elf
Bootloader binary size 0x5070 bytes. 0x2f90 bytes (37%) free.
[672/870] Building C object esp-idf\driver\CMakeFiles\__idf_driver.dir/gpio/gpio.c.obj
[865/870] Generating ld\sections.ld
Source: Espressif IDF (Extension)
```

Figure 11 Building the Project

3. You can monitor your application execution status by checking the terminal tab.
 4. Click the file “`diagram.json`”. You will see an image like the Fig. 12 below.



Figure 12 Simulator and Terminal

5. Click the green “start” button. You will see the log in the terminal. If you see both working, your installation is done.

Part1 To-do:

1. Please answer the following questions from your hello world example in the “*answer.txt*”.
 - a) How many CPU cores does the program use? (put the answer under item **Part1-1a**)
 - b) What is the SPI speed? (put the answer under item **Part1-1b**)
 - c) What is the CPU frequency? (put the answer under item **Part1-1c**)
 - d) What is the size of the external flash? (put the answer under item **Part1-1d**)
 - e) What is the minimum free heap size? (put the answer under item **Part1-1e**)
2. Modify the “*hello_world_main.c*” file to print out the following information. Ensure the restarting message actually wait for 5 seconds. Your “hello world” print out should include **your name (not “Aaron”)**. Take a screenshot called “*output1.png*” or “*output2.jpg*” and save it in the answer folder. Also, save or move your “*hello_world_main.c*” file in answer folder for submission. (Later on, we will use a different filename for doing part 2’s to-do”.

```
I (273) sleep: Configure to isolate all GPIO pins in sleep state
I (280) sleep: Enable automatic switching of GPIO sleep configuration
I (287) app_start: Starting scheduler on CPU0
I (292) main_task: Started on CPU0
I (292) main_task: Calling app_main()
Hello world! This is Aaron's first application.
This is esp32c3 chip with 1 CPU core(s), WiFi/BLE, silicon revision v0.4, 2MB external flash
Minimum free heap size: 330660 bytes
Restarting in 15 seconds...
Restarting in 10 seconds...
Restarting in 5 seconds...
Restarting in 0 seconds...
|
```

Figure 13 Your Program Output

Part2: Programming on ESP-IDF FreeRTOS

Above program “HelloWorld” shows some important concepts about the program.

- (1) `app_main(void)` is the entry function for the program.
- (2) “freertos/FreeRTOS.h” and “freertos/task.h” are two files for using FreeRTOS.
- (3) The “main” folder contains the “*hello_world_main.c*” program file. The “make” file (for compilation) is the “*CMakeList.txt*”

The HelloWorld example program does not use the concept “task” which is the critical concepts in FreeRTOS.

Tasks are the key concept in FreeRTOS. Tasks will be created by calling the `xTaskCreate()`. The `xTaskCreate()` will pass a “`TaskFunction()`” which task content will be put in. In other words, the sequence of the development will be creating the `TaskFunction()` first and then associated it in a Task (by calling the `xTaskCreate()`).

Once the task is created, the task will be put in the ready state. Then, the scheduler will run the task according to the tasks’ priorities. The typical ESP-IDF FreeRTOS program structure looks as Fig. 14.

```

void vTaskFunction ( void *pvParameters ) {
    /* As per most tasks,
       this task is implemented in an infinite loop. */
    for( ; ; ) {

        /* Doing something. */

        /* Task can call vTaskDelete( NULL );
           to ensure its exit is clean */
    }
}

```

Figure 14 Task Function Structure

For Task Creation, the following is the xTaskCreate() definition.

```

xTaskCreate(aTaskFunction, /* The task function name.
                           Pointer to the function that
                           implements the task. */
            "Task 1",      /* Task name for the task. This
                           example use "Task 1"
                           This is to facilitate
                           debugging only. */
            1024,          /* Stack depth (size) -
                           small microcontrollers will
                           use much less stack
                           than this. e.g. 1024, 2048 */
            *pvParameters,/* This example does not use
                           the task parameter. Put NULL
                           if there is no parameters
                           for the Task Function */
            1,              /* This task will run at
                           priority 1. You can assign
                           other numbers for difference
                           priorities */
            NULL );        /* This is used to reference the
                           task. NULL means that it does
                           not use the task handle */

```

Figure 15 xTaskCreate() Function

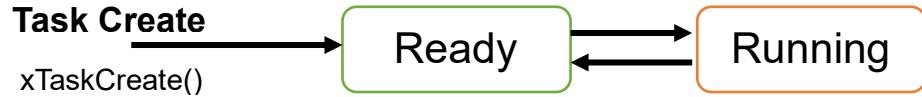


Figure 16 Task Creation, Ready, and Running

```

1 #include <stdio.h>
2 #include "freertos/FreeRTOS.h"
3 #include "freertos/task.h"
4
5 void taskName1(void *p) {
6     for(;;) {
7
8     }
9     vTaskDelete(NULL);
10 }
11
12 void taskName2(void *p) {
13     for(;;) {
14
15     }
16     vTaskDelete(NULL);
17 }
18
19
20 void app_main(void)
21 {
22     xTaskCreate(taskName1, "t1", 2048, NULL, 1, NULL);
23     xTaskCreate(taskName2, "t2", 2048, NULL, 1, NULL);
24 }

```

Figure 17 Typical ESP-IDF FreeRTOS Program Structure for Two Tasks

Above program (figure) defines two task function, inside the task function, there is an infinite loop. We follow the document so we use `for(;;)` but we can use `while(1)` for sure. `xTaskCreate` will put the task into FreeRTOS scheduler. So, the program is a [multi-task real-time application](#).

Reading References for FreeRTOS programming on ESP if you are interested.

- <https://github.com/espressif/esp-idf>
- <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/freertos.html>

Part2 To-do:

1. Create a new program that is based on the skeleton code, “three_task_skeleton.c”, that contains three tasks various priority. You must move your “hello_world_main.c” file to answer folder and create a new file called “`main.c`”. Later, you need save (or copy) your

“`main.c`” files in answer folder for submission. Also, take a screenshot of your output, called “`output2.png`” or “`output2.jpg`” and save it in your answer folder.

Note:

- Refer to the “`CMakeList_for_todo2.txt`” in the demo folder, you need to change the `CMakeList.txt` files in main folder accordingly to build the file.
- You may have to remove the entire “build” folder and rebuild the project.

Hints:

- (1) The following code can help you to delay or slow down the task.
“`vTaskDelay(500/portTICK_PERIOD_MS);`”
- (2) You might need to terminate tasks at certain time (counting number).

Output:

```
I (404) main_task: Started on CPU0
I (414) main_task: Calling app_main()
hello task1: count 0
hello task2: count 0
I (414) main_task: Returned from app_main()
hello task3: count 0
hello task1: count 1
hello task2: count 1
hello task3: count 1
hello task1: count 2
hello task2: count 2
hello task3: count 2
```

Submission:

In answer folder the following files will be included:

- `answer.txt`
- `output1.png` or `output1.jpg`
- `Hello_world_main.c`
- `main.c`
- `output2.png` or `output2.jpg`