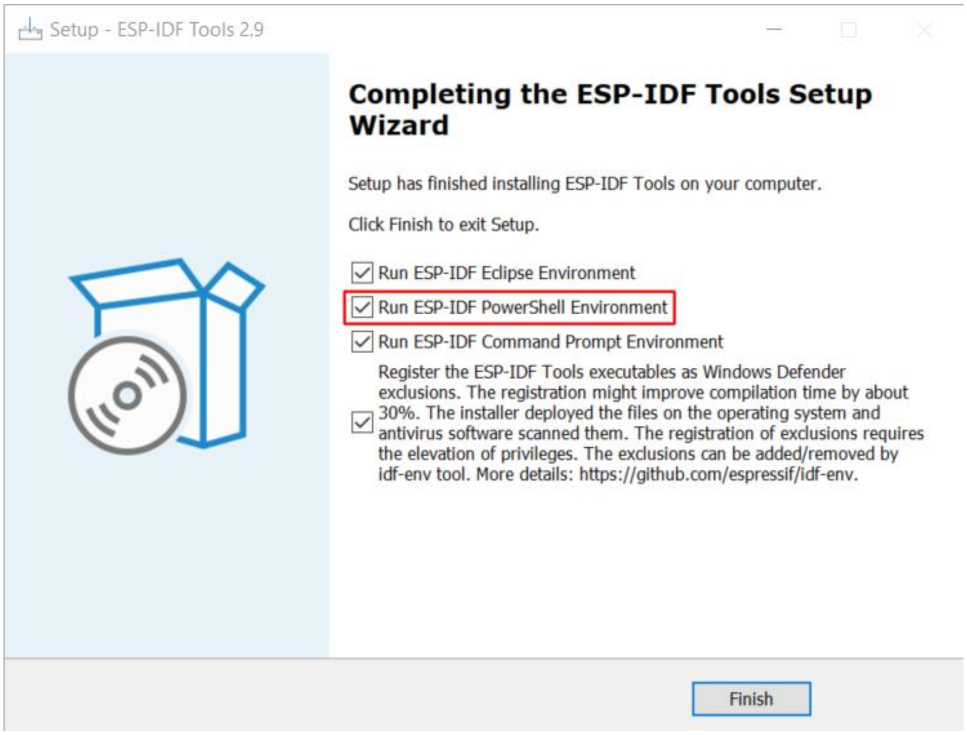
Creating a project on ESP-IDF with Visual Studio Code on Windows OS Tutorial:

These are the steps I followed to get set up the environment correctly and to create and run a project in the ESP-IDF Extension of VS Code:

1. Download and install VS Code for Windows from: https://code.visualstudio.com/
2. Download the ESP-IDF Tools Installer (Universal Online Installer 2.24) for installing the ESP-IDF's prerequisites (Toolchain, Build tools, ESP-IDF) from: https://dl.espressif.com/dl/esp-idf/?idf=4.4
3. Install the ESP IDF Tool Installer.
4. At the end of the installation process, you can check out the option Run ESP-IDF PowerShell Environment or Run ESP-IDF Command Prompt (cmd.exe).



Completing the ESP-IDF Tools Setup Wizard

Setup has finished installing ESP-IDF Tools on your computer.

Click Finish to exit Setup.

☑ Run ESP-IDF Eclipse Environment

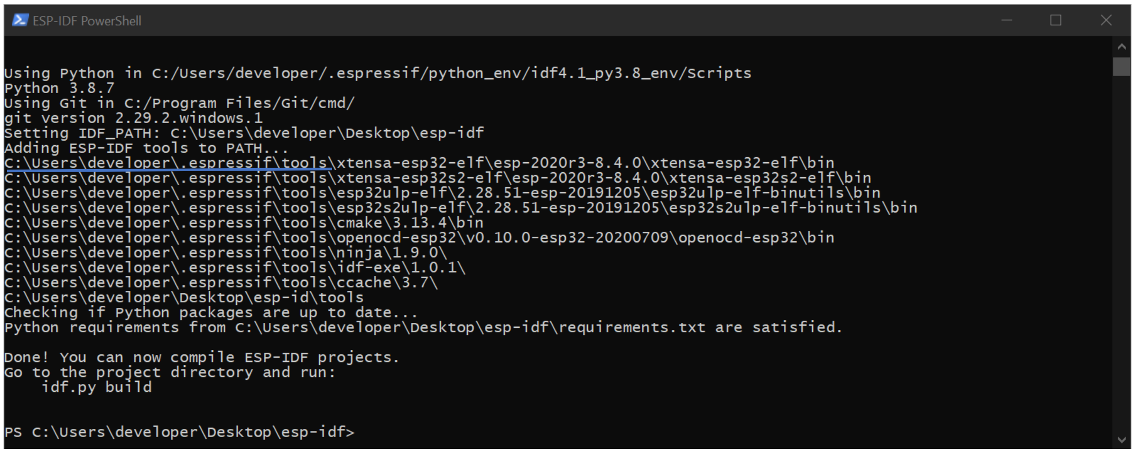
☑ Run ESP-IDF PowerShell Environment

☑ Run ESP-IDF Command Prompt Environment

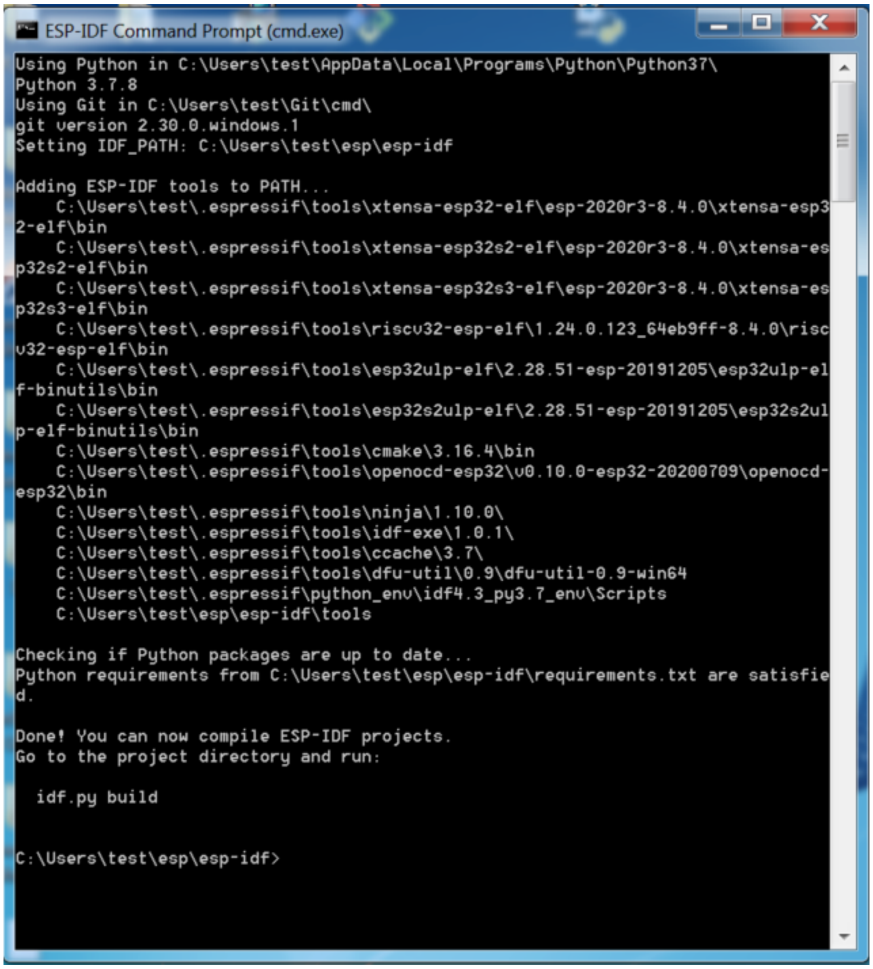
☑ Register the ESP-IDF Tools executables as Windows Defender exclusions. The registration might improve compilation time by about 30%. The installer deployed the files on the operating system and antivirus software scanned them. The registration of exclusions requires the elevation of privileges. The exclusions can be added/removed by idf-env tool. More details: https://github.com/espressif/idf-env.

5. The installer launches ESP-IDF environment in selected prompt.

6. The following two shell prompt windows appear and install the required tools:



```
Using Python in C:/Users/developer/.espressif/python_env/idf4.1_py3.8_env/Scripts
Python 3.8.7
Using Git in C:/Program Files/Git/cmd/
git version 2.29.2.windows.1
Setting IDF_PATH: C:\Users\developer\Desktop\esp-idf
Adding ESP-IDF tools to PATH...
C:\Users\developer\.espressif\tools\xtensa-esp32-elf\esp-2020r3-8.4.0\xtensa-esp32-elf\bin
C:\Users\developer\.espressif\tools\xtensa-esp32s2-elf\esp-2020r3-8.4.0\xtensa-esp32s2-elf\bin
C:\Users\developer\.espressif\tools\esp32ulp-elf\2.28.51-esp-20191205\esp32ulp-elf-binutils\bin
C:\Users\developer\.espressif\tools\esp32s2ulp-elf\2.28.51-esp-20191205\esp32s2ulp-elf-binutils\bin
C:\Users\developer\.espressif\tools\cmake\3.13.4\bin
C:\Users\developer\.espressif\tools\openocd-esp32\v0.10.0-esp32-20200709\openocd-esp32\bin
C:\Users\developer\.espressif\tools\ninja\1.9.0\
C:\Users\developer\.espressif\tools\idf-exe\1.0.1\
C:\Users\developer\.espressif\tools\ccache\3.7\
C:\Users\developer\Desktop\esp-id\tools
Checking if Python packages are up to date...
Python requirements from C:\Users\developer\Desktop\esp-idf\requirements.txt are satisfied.

Done! You can now compile ESP-IDF projects.
Go to the project directory and run:
    idf.py build


PS C:\Users\developer\Desktop\esp-idf>
```



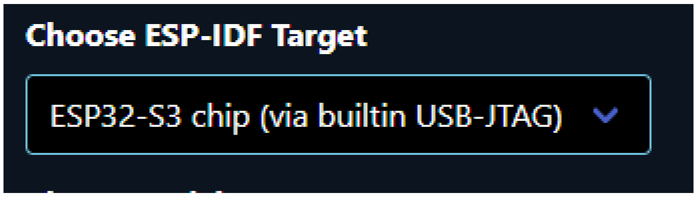
```
Using Python in C:\Users\test\AppData\Local\Programs\Python\Python37\
Python 3.7.8
Using Git in C:\Users\test\Git\cmd\
git version 2.30.0.windows.1
Setting IDF_PATH: C:\Users\test\esp\esp-idf

Adding ESP-IDF tools to PATH...
    C:\Users\test\.espressif\tools\xtensa-esp32-elf\esp-2020r3-8.4.0\xtensa-esp3
2-elf\bin
    C:\Users\test\.espressif\tools\xtensa-esp32s2-elf\esp-2020r3-8.4.0\xtensa-es
p32s2-elf\bin
    C:\Users\test\.espressif\tools\xtensa-esp32s3-elf\esp-2020r3-8.4.0\xtensa-es
p32s3-elf\bin
    C:\Users\test\.espressif\tools\riscv32-esp-elf\1.24.0.123_64eb9ff-8.4.0\risc
v32-esp-elf\bin
    C:\Users\test\.espressif\tools\esp32ulp-elf\2.28.51-esp-20191205\esp32ulp-el
f-binutils\bin
    C:\Users\test\.espressif\tools\esp32s2ulp-elf\2.28.51-esp-20191205\esp32s2ul
p-elf-binutils\bin
    C:\Users\test\.espressif\tools\cmake\3.16.4\bin
    C:\Users\test\.espressif\tools\openocd-esp32\v0.10.0-esp32-20200709\openocd-
esp32\bin
    C:\Users\test\.espressif\tools\ninja\1.10.0\
    C:\Users\test\.espressif\tools\idf-exe\1.0.1\
    C:\Users\test\.espressif\tools\ccache\3.7\
    C:\Users\test\.espressif\tools\dfu-util\0.9\dfu-util-0.9-win64
    C:\Users\test\.espressif\python_env\idf4.3_py3.7_env\Scripts
    C:\Users\test\esp\esp-idf\tools

Checking if Python packages are up to date...
Python requirements from C:\Users\test\esp\esp-idf\requirements.txt are satisfie
d.

Done! You can now compile ESP-IDF projects.
Go to the project directory and run:

  idf.py build


C:\Users\test\esp\esp-idf>
```

7. Let the prompts complete the installations. Note the path where the ESP IDF(IDF_PATH) got installed (e.g. C:\Users\user_name\.espressif). Also note down the path where the ESP IDF Tools (IDF_TOOLS_PATH) got installed (e.g. C:\Users\user_name\.espressif\tools).

8. Open Visual Studio Code in your Windows and follow Step 3 to Step 14 in the tutorial in this link: https://github.com/espressif/vscode-esp-idf-extension/blob/HEAD/docs/tutorial/install.md

Note: In Step 10 of this tutorial, use the IDF_PATH and IDF_TOOLS_PATH you noted down in Step 7 of this Instructions Document. The next steps depend upon what type of ESP32 you are using. For example, ESP32-S3, ESP32-S2 etc.

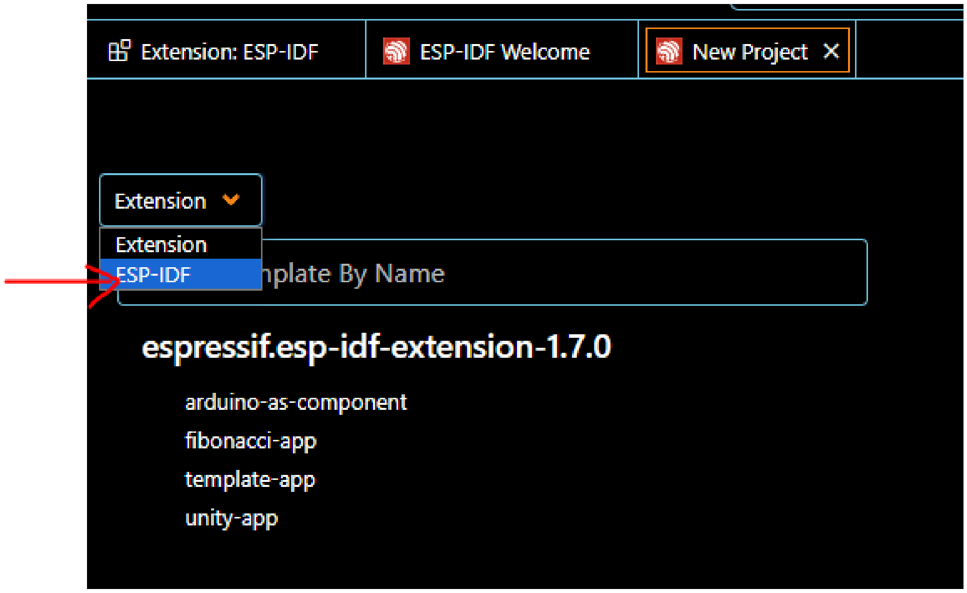
9. Connect your ESP32 to one of the USB-A ports of your laptop/PC. Go to the Device Manager of your windows to figure out which COM Port the ESP got connected to. (e.g. COM4, COM5 etc.).
10. Follow the steps in the following tutorial to create a new project: https://github.com/espressif/vscode-esp-idf-extension/blob/6e9aed55db3d4f6dbd763d4f46e34330f9b62d2c/docs/tutorial/new_project_wizard.md
11. Choose the "Espressif board" and "Serial Port" in the project manager according to the ESP device you are using. For ESP32-S3 the ESP_IDF Target will be:



Choose ESP-IDF Target

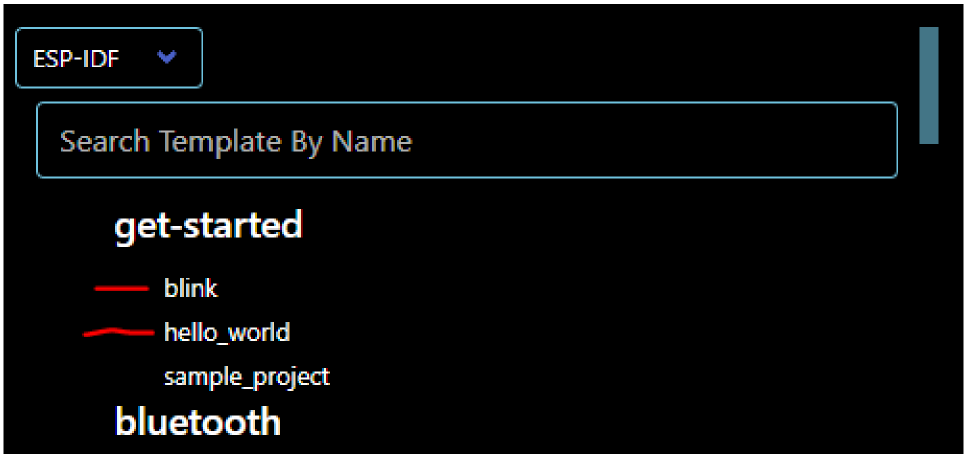
ESP32-S3 chip (via builtin USB-JTAG)

12. Now press the "Choose Template" button from the project window.
13. Choose the Extension as ESP-IDF from the drop-down menu:

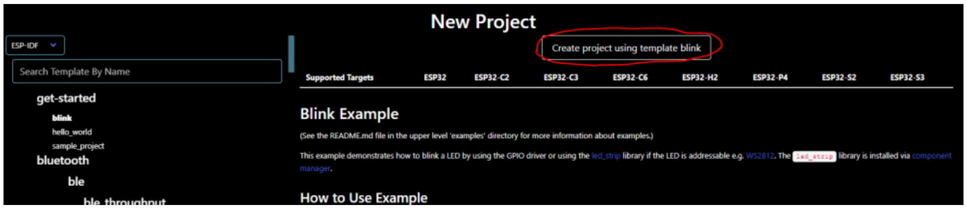


14. The information on the rest of the fields can be found in the tutorial given in Step 10 above. If you are new, it is recommended that you choose the ESP-IDF blink or

hello_world projects:



15. Now press the “Create project using template” option:



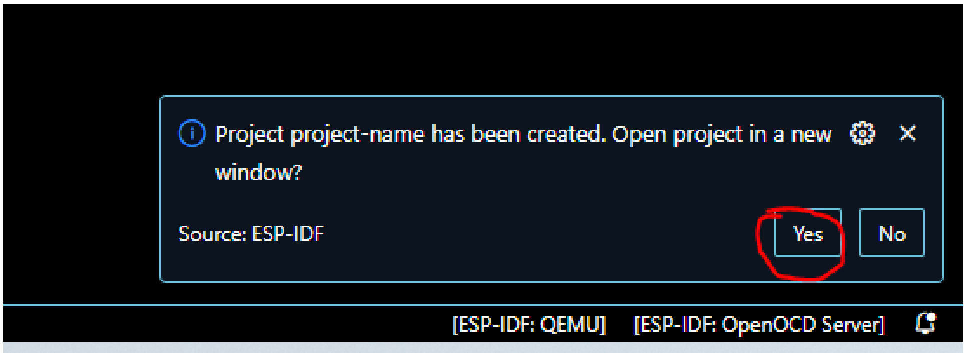
Blink Example

(See the README.md file in the upper level 'examples' directory for more information about examples.)

This example demonstrates how to blink a LED by using the GPIO driver or using the led_strip library if the LED is addressable e.g. WS2812. The led_strip library is installed via component manager.

How to Use Example

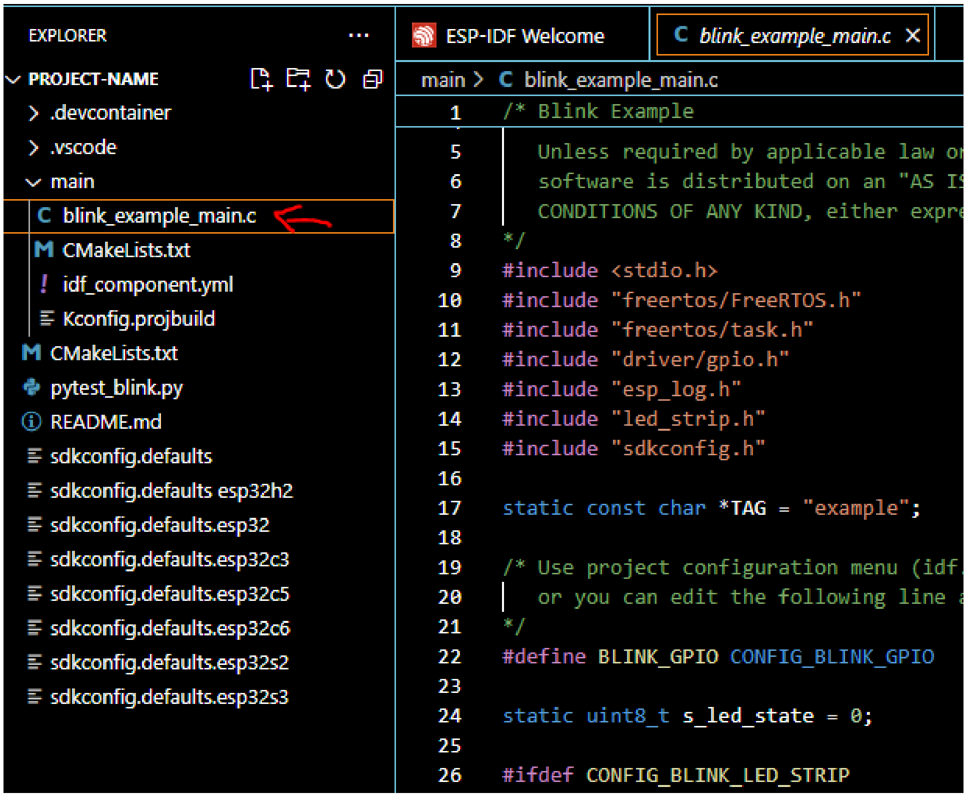
16. A pop-up appears at the bottom-right of the screen, click Yes on it:



Project project-name has been created. Open project in a new window?

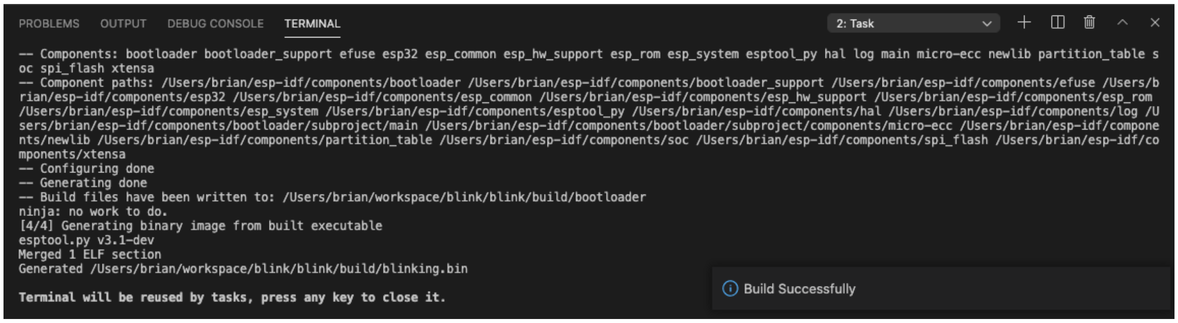
Source: ESP-IDF

17. Navigate to the main file in the project directory and open the .c file of the project:



```
/* Blink Example

   Unless required by applicable law or
   software is distributed on an "AS IS
   CONDITIONS OF ANY KIND, either expre
*/
#include <stdio.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "driver/gpio.h"
#include "esp_log.h"
#include "led_strip.h"
#include "sdkconfig.h"

static const char *TAG = "example";

/* Use project configuration menu (idf.
   or you can edit the following line a
*/
#define BLINK_GPIO CONFIG_BLINK_GPIO

static uint8_t s_led_state = 0;

#ifdef CONFIG_BLINK_LED_STRIP
```

18. Now to build the project, use the ESP-IDF using the Build your Project command (CTRL E B keyboard shortcut). The user will see a new terminal being launched with the build output and a notification bar with a Building Project message until it is done, then a Build done message when finished.



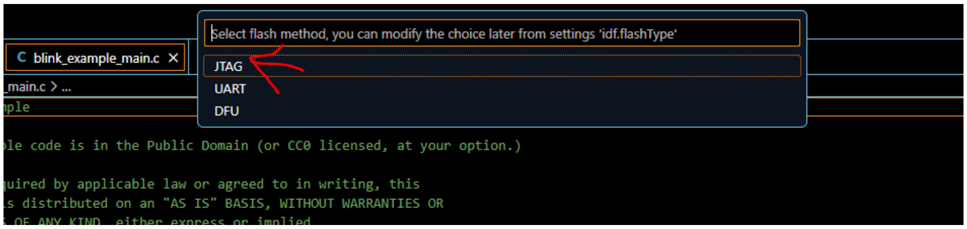
```
-- Components: bootloader bootloader_support efuse esp32 esp_common esp_hw_support esp_rom esp_system esptool_py hal log main micro-ecc newlib partition_table s
oc spi_flash xtensa
-- Component paths: /Users/brian/esp-idf/components/bootloader /Users/brian/esp-idf/components/bootloader_support /Users/brian/esp-idf/components/efuse /Users/b
rian/esp-idf/components/esp32 /Users/brian/esp-idf/components/esp_common /Users/brian/esp-idf/components/esp_hw_support /Users/brian/esp-idf/components/esp_rom
/Users/brian/esp-idf/components/esp_system /Users/brian/esp-idf/components/esptool_py /Users/brian/esp-idf/components/hal /Users/brian/esp-idf/components/log /U
sers/brian/esp-idf/components/bootloader/subproject/main /Users/brian/esp-idf/components/bootloader/subproject/components/micro-ecc /Users/brian/esp-idf/compone
nts/newlib /Users/brian/esp-idf/components/partition_table /Users/brian/esp-idf/components/soc /Users/brian/esp-idf/components/spi_flash /Users/brian/esp-idf/co
mponents/xtensa
-- Configuring done
-- Generating done
-- Build files have been written to: /Users/brian/workspace/blink/blink/build/bootloader
ninja: no work to do.
[4/4] Generating binary image from built executable
esptool.py v3.1-dev
Merged 1 ELF section
Generated /Users/brian/workspace/blink/blink/build/blinking.bin

Terminal will be reused by tasks, press any key to close it.
```

Build Successfully

19. Before flashing the project, the user needs to specify the serial port of the device with the **ESP-IDF: Select Port to Use** command (CTRL E P keyboard shortcut). You can choose between UART/JTAG flashing mode and then a list of serial ports will be shown for the user to select.

20. Now to flash the project, use the **ESP-IDF: Flash your Project** command (CTRL E F keyboard shortcut). Choose UART or JTAG flash mode (Configure JTAG Flashing) and then flashing will start in the previously selected serial port. The user can also use the

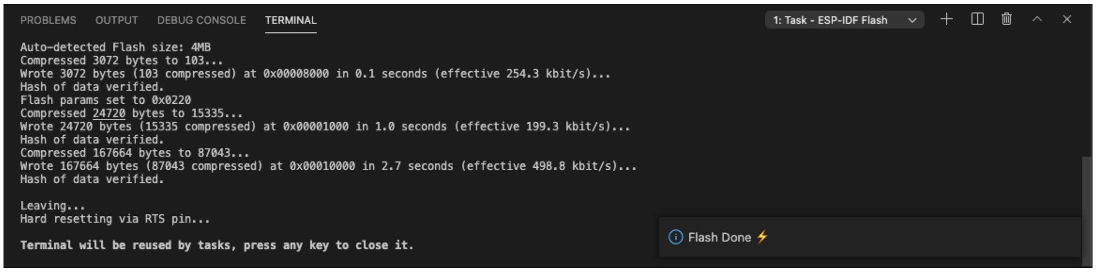
ESP-IDF: Flash (UART) your Project or **ESP-IDF: Flash (with JTAG)** directly.



Select flash method, you can modify the choice later from settings 'idf.flashType'

```
ple
ple code is in the Public Domain (or CC0 licensed, at your option.)

uired by applicable law or agreed to in writing, this
s distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR
 OF ANY KIND, either express or implied
```

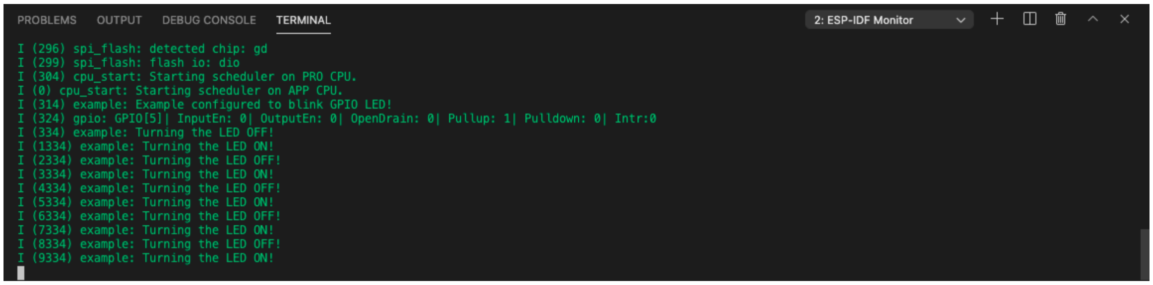
21. The user will see a new terminal being launched with the flash output and a notification bar with Flashing Project message until it is done then a Flash done message when finished.



```
Auto-detected Flash size: 4MB
Compressed 3072 bytes to 103...
Wrote 3072 bytes (103 compressed) at 0x00008000 in 0.1 seconds (effective 254.3 kbit/s)...
Hash of data verified.
Flash params set to 0x0220
Compressed 24720 bytes to 15335...
Wrote 24720 bytes (15335 compressed) at 0x00001000 in 1.0 seconds (effective 199.3 kbit/s)...
Hash of data verified.
Compressed 167664 bytes to 87043...
Wrote 167664 bytes (87043 compressed) at 0x00010000 in 2.7 seconds (effective 498.8 kbit/s)...
Hash of data verified.

Leaving...
Hard resetting via RTS pin...

Terminal will be reused by tasks, press any key to close it.
```

Flash Done ⚡

22. Now to start monitoring your device, use the **ESP-IDF: Monitor your Device** command (CTRL E M keyboard shortcut). The user will see a new terminal being launched with the idf.py monitor output.



```
I (296) spi_flash: detected chip: gd
I (299) spi_flash: flash io: dio
I (304) cpu_start: Starting scheduler on PRO CPU.
I (0) cpu_start: Starting scheduler on APP CPU.
I (314) example: Example configured to blink GPIO LED!
I (324) gpio: GPIO[5]| InputEn: 0| OutputEn: 0| OpenDrain: 0| Pullup: 1| Pulldown: 0| Intr:0
I (334) example: Turning the LED OFF!
I (1334) example: Turning the LED ON!
I (2334) example: Turning the LED OFF!
I (3334) example: Turning the LED ON!
I (4334) example: Turning the LED OFF!
I (5334) example: Turning the LED ON!
I (6334) example: Turning the LED OFF!
I (7334) example: Turning the LED ON!
I (8334) example: Turning the LED OFF!
I (9334) example: Turning the LED ON!
```

23. To stop the terminal with the idf.py monitor output use (CTRL])keyboard shortcut).

The last few steps were copied from the source: https://github.com/espressif/vscode-esp-idf-extension/blob/HEAD/docs/tutorial/basic_use.md but modified for simplicity.