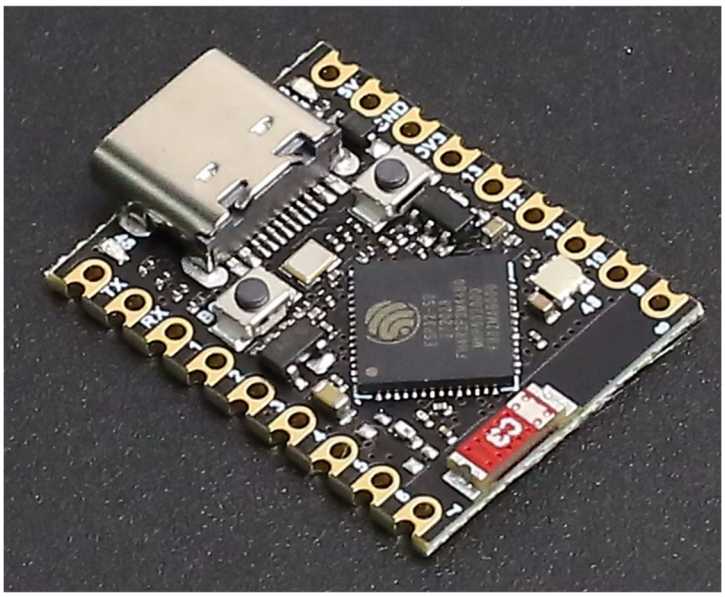
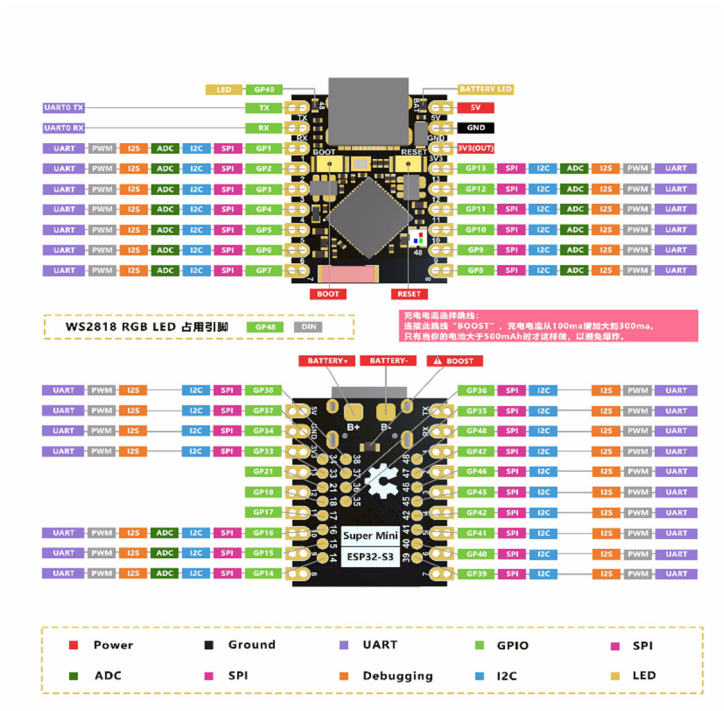
Debug ESP32S3 using ESP-IDF on VSCode

Target hardware

1. ESP32-S3 Super Mini Black Gold
2. Clone of a ZERO board
3. Has 18 pins, boot and reset buttons and a USB-C connector
4. MCU is ESP32-S3 FH4R2
5. Datasheet https://www.espressif.com/sites/default/files/documentation/esp32-s3_datasheet_en.pdf
6. CPU is Xtensa® dual-core 32-bit LX7 microprocessor
7. Clock speed: up to 240 MHz
8. Flash is 4Mb, PSRAM 2Mb and SRAM 512 Kb
9. Assume USB JTAG/debug unit (interface 0) set as COM3
10. Assume USB JTAG/debug unit (interface 2) configured as a Universal Serial Bus Device called USB JTAG/Serial debug unit





USB Ports

There are three types of ESP32xx modules with different number and types of USB ports.

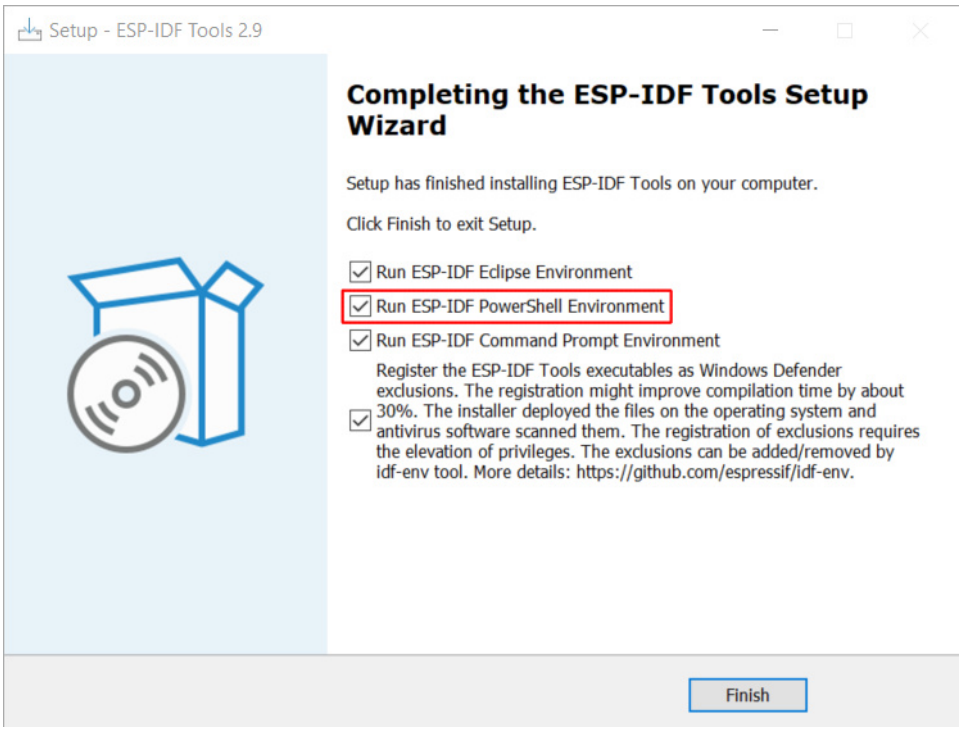
1. ESP32 has one standard COM port used for serial upload and printing.
2. ESP32-Sx mini has one USB port, which presents as:
    a. USB native
        i. USB JTAG/serial debug unit (interface 0) with driver libwdi 1.0.0.0 reported by device manager and usbser 1.0.0.0 by Zadig. Windows will allocate a COM port number e.g. COM4.
        ii. USB/JTAG serial debug unit (interface 2) with driver libwdi 6.1.7600.16386reported by device manager and WinUSB 6.1.7600.16386 by Zadig. Windows will allocate as a Universal Serial Bus device called USB JTAG/serial debug unit. If this device appears as a COM port use Zadig to change the driver to WinUSB.
3. ESP32-Sx Dev kit has two USB ports called COM and USB, which present as:
    a. COM
        i. One standard COM port used for serial upload and printing e.g. COM3.
    b. USB native
        i. USB JTAG/serial debug unit (interface 0) with driver libwdi 1.0.0.0 reported by device manager and usbser 1.0.0.0 by Zadig. Windows will allocate a COM port number e.g. COM4.
        ii. USB/JTAG serial debug unit (interface 2) with driver libwdi 6.1.7600.16386reported by device manager and WinUSB 6.1.7600.16386 by Zadig. Windows will allocate as a Universal Serial Bus device called USB JTAG/serial debug unit. If this device appears as a COM port use Zadig to change the driver to WinUSB.

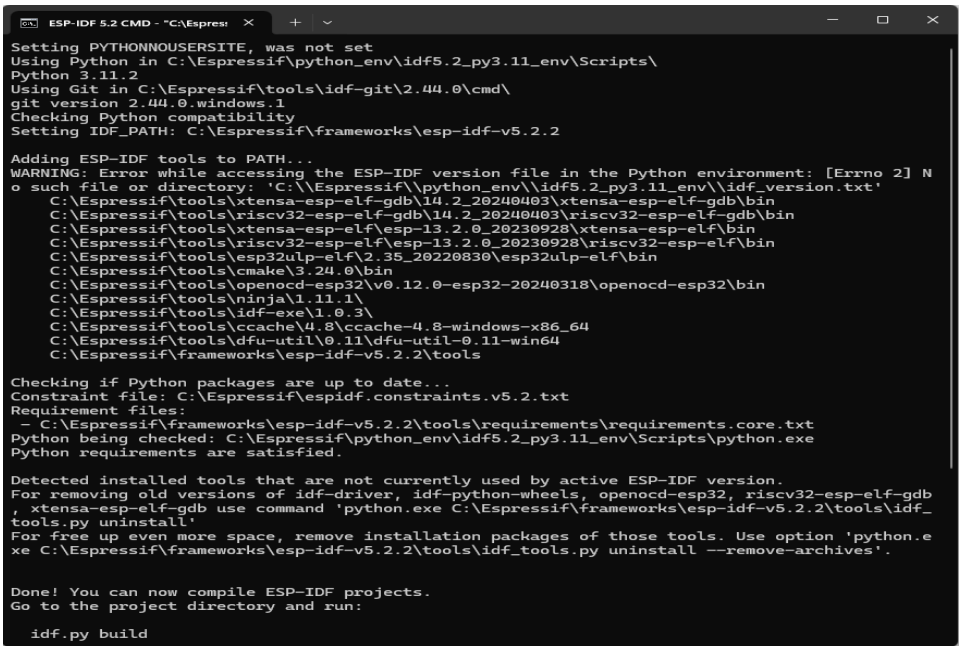
Configuring the USB ports

1. Download and run the latest version of Zadig from https://zadig.akeo.ie/
2. Click Options and select list all devices
3. USB JTAG/Serial debug unit (Interface 0) – check driver is a version of usbser, if not install USB Serial, which will appear in device manager as a COM port in the Ports (COM & LPT) section.
4. USB JTAG/Serial debug unit (Interface 2) – change driver to WinUSB or libusbK, which will appear as a USB Serial Bus device called USB JTAG/serial debug unit or USB JTAG/serial debug unit (Interface 2) in device manager under the section libusbK USB devices. For more info about libusbK devices see: https://libusbk.sourceforge.net/UsbK3/index.html

Install ESP-IDF

1. Download the offline tools installer from https://dl.espressif.com/dl/esp-idf/?idf=4.4
    a. Either a specific version, or use the universal online installer and choose the version
2. During the installation two special shortcuts (CMD & PowerShell) will be added to the desktop
    a. Both shortcuts setup the Python environment ready for ESP-IDF commands
    b. I prefer PowerShell as can cd to a new path in one step





3. Updating ESP-IDF tools and adding ESP-IDF tools to PATH using an export script
    a. Open the PowerShell shortcut and run install.ps1 and then export.ps1
    b. Open the Command shortcut and run install.bat and then export.bat
    c. Look at the output and run any other suggestions for missing items

ESP-IDF on VSCode configuration

1. The black line at the top of the editor pane can be turned on and off by opening the command palette (Menu > View > Command palette > Toggle Editor Sticky Scroll.

IDF_PATH

IDF_PATH is an environment variable that points to the active ESP-IDF folder and is used during operations of the ESP-IDF extension in VSCode and when using a CLI such as the PowerShell and CMD shortcuts added when installing ESP-IDF tools. The following may be useful:

1. The ESP-IDF extension on VSCode appears to set the IDF_PATH when required e.g.
    a. PS E:\Users\Steven\Documents\GitHub\ESP-IDF\test2> & set IDF_PATH='C:/Espressif/frameworks/esp-idf-v5.2.2/'
2. The IDF_PATH can be set from an ESP-IDF terminal within the ESP-IDF extension using:
    a. Set IDF_PATH='C:/Espressif/frameworks/esp-idf-v5.2.2/'
3. The IDF_PATH can be set from an ESP-IDF PowerShell or CMD window using:
    a. Set IDF_PATH='C:/Espressif/frameworks/esp-idf-v5.2.2/'
4. The IDF_PATH can be permanently set by:
    a. Create a text file called export_idf_path.sh within C:\msys32\etc\profile.d containing the line export IDF_PATH="C:/msys32/home/user-name/esp/esp-idf"
5. The IDF_PATH can be permanently set by:
    a. Use Windows search to find Edit Environment Variables and add a user, or system, variable called IDF_PATH with the contents C:\Espressif\frameworks\esp-idf-v5.2.2 note the back slashes.

To view the IDF_PATH environment variable use:

1. echo $Env:IDF_PATH in a ESP-IDF PowerShell windows, but not a CMD window!
2. echo $Env:IDF_PATH from an ESP-IDF terminal within the ESP-IDF extension in VSCode
3. echo %IDF_PATH% from an ESP-IDF CMD or normal Windows CMD window
4. echo $IDF_PATH from an MSYS Bash terminal

For more info see: https://docs.espressif.com/projects/esp-idf/en/v3.3.5/get-started/add-idf_path-to-profile.html and https://docs.espressif.com/projects/esp-idf/en/release-v3.3/get-started-cmake/add-idf_path-to-profile.html

Install ESP-IDF extension on VSCode

1. Start VSCode and go to extensions
2. Search for and install the ESP-IDF extension
3. Select a version of installed ESP-IDF to use e.g. 5.2.2
4. From the commands section select Configure ESP-IDF version and choose EXPRESS
5. Choose Find ESP-IDF on My System as it has already been installed
6. IDF_PATH should look like C:/Espressif/frameworks/esp-idf-v5.2.2/
7. IDF_TOOLS_PATH should look like c:\Espressif\tools
8. Click install button and wait for completion

Creating a new project from an example

1. Click the ESP-IDF: EXPLORER icon in the primary side bar at the left
2. In COMMANDS click Show Examples and choose the IDF version
3. Scroll down to e.g. peripherals > lcd > spi_lcd_touch
4. Click the blue button Create project using example spi_lcd_touch
5. Save project in e.g. E:\Users\Steven\Documents\GitHub\ESP-IDF
6. Ignore warning "e:\Users\Steven\Documents\GitHub\ESP-IDF\spi_lcd_touch /build/compile_commands.json" could not be parsed. 'includePath' from c_cpp_properties.json in folder spi_lcd_touch will be used instead as of course the build folder does not yet exist as the project has not been built!
7. Do Menu > File > Close Folder
8. Edit the name of the project folder to e.g. GC9A01_project and reopen using menu > file > open folder
9. Set the COM port to e.g. COM3

10. Set the target to e.g. esp32s3 and ESP32-S3 chip via builtin USB-JTAG
    a. At this point any Managed Component folders are added
11. Open Menuconfig idf.py menuconfig
12. Search for FLASH and change flash size from 2MB to 4MB
13. Change any other settings as needed e.g. search for LCD and change LCD controller model to e.g. GC9A01
14. Build the project – 963 steps!
15. Select flash method UART or JTAG and flash the target

Creating a minimal new project

1. Click the ESP-IDF: EXPLORER icon in the primary side bar at the left
2. In COMMANDS click Show Examples and choose the IDF version
3. Click sample_project as it always works
4. Click the blue button Create project using example sample_project
5. Save project in e.g. E:\Users\Steven\Documents\GitHub\ESP-IDF
6. Ignore warning "e:\Users\Steven\Documents\GitHub\ESP-IDF\sample_project/build/compile_commands.json" could not be parsed. 'includePath' from c_cpp_properties.json in folder 'SAMPLE_PROJECT' will be used instead as of course the build folder does not yet exist as the project has not been built!
7. Do Menu > File > Close Folder
8. Edit the name of the project folder to e.g. new_project and reopen using menu > file > open folder
9. Set the COM port to e.g. COM3
10. Set the target to e.g. esp32s3 and ESP32-S3 chip via builtin USB-JTAG
    a. At this point any Managed Component folders are added
11. Open Menuconfig idf.py menuconfig
12. Search for FLASH and change flash size from 2MB to 4MB
13. Change any other settings as needed
14. Build the project – 963 steps!
15. Select flash method UART or JTAG and flash the target

Libraries as Components

1. The official source for libraries is the ESP Component Registry at: https://components.espressif.com/
    a. Searchable by keywords.
    b. Can be filtered by target MCU.
    c. GitHub components: https://github.com/espressif/esp-idf/tree/master/components
    d. GitHub examples: https://github.com/espressif/esp-idf/tree/master/examples
    e. The ESP Component Registry is also available from a link in the Welcome to Espressif IDF extension tab in ESP-IDF on VSCode.
    f. The ESP-IDF Component Manager and its use is described here: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-guides/tools/idf-component-manager.html
2. An alternative ESP-IDF Components library for LCD and other hardware is at: https://github.com/UncleRus/esp-idf-lib/tree/master and the searchable list is at: https://esp-idf-lib.readthedocs.io/en/latest/index.html

Add Components (Libraries) to your project

Note: There appears to be something wrong with the component espressif/led_strip 2.5.5 in that the Python command does not work and produces many error messages, but downloading and adding the component archive does work

1. There are two methods to add a component to a project
    a. One involves running a Python command in an ESP-IDF terminal e.g. idf.py add-dependency "lvgl/lvgl^9.2.0" which adds a Component Manager Manifest File in the main folder called idf_component.yml

i. Then run idf.py reconfigure and/or idf.py update-dependencies in an ESP-IDF terminal, which adds the component into a new folder called managed_components
ii. Next build the project before adding any includes

b. If the first method does not work create a folder called components, in the projects folder, and the download and unzip the component archive into that new folder from https://components.espressif.com
i. Then run idf.py reconfigure and/or idf.py update-dependencies in an ESP-IDF terminal
ii. Next build the project before adding any includes

1. Expand the components folder in ESP-IDF Explorer, to see the component name added to the component folder; this does not apply to the managed _components folder
2. Add any includes to main.c
3. Build the project again

Flashing problems

1. Flash size on target is different to the size used to upload via JTAG. The solution is open SDK Configuration Editor (menuconfig) and search for "flash" then edit the flash size to match the target, for example:
   a. ESP32-S3 FH4R2 Mini = 4Mb

Ordering Code	SiP Flash	SiP PSRAM
ESP32-S3	—	—
ESP32-S3FN8	8 MB (Quad SPI)	—
ESP32-S3R2	—	2 MB (Quad SPI)
ESP32-S3R8	—	8 MB (Octal SPI)
ESP32-S3R8V	—	8 MB (Octal SPI)
ESP32-S3FH4R2	4 MB (Quad SPI)	2 MB (Quad SPI)

   b. ESP32-S3 Dev Kit N8R2 = 8Mb

Ordering Code[2]	Flash[3, 4]	PSRAM[5]
ESP32-S3-WROOM-1-N4	4 MB (Quad SPI)	-
ESP32-S3-WROOM-1-N8	8 MB (Quad SPI)	-
ESP32-S3-WROOM-1-N16	16 MB (Quad SPI)	-
ESP32-S3-WROOM-1-H4	4 MB (Quad SPI)	-
ESP32-S3-WROOM-1-N4R2	4 MB (Quad SPI)	2 MB (Quad SPI)
ESP32-S3-WROOM-1-N8R2	8 MB (Quad SPI)	2 MB (Quad SPI)
ESP32-S3-WROOM-1-N16R2	16 MB (Quad SPI)	2 MB (Quad SPI)
ESP32-S3-WROOM-1-N4R8	4 MB (Quad SPI)	8 MB (Octal SPI)
ESP32-S3-WROOM-1-N8R8	8 MB (Quad SPI)	8 MB (Octal SPI)
ESP32-S3-WROOM-1-N16R8	16 MB (Quad SPI)	8 MB (Octal SPI)
ESP32-S3-WROOM-1-N16R16V[8]	16 MB (Quad SPI)	16 MB (Octal SPI)

   c. ESP32-S3 Dev Kit N16R8 = 16Mb

If flashing via JTAG fails, change the flash method to UART and check working by building and flashing, then change the flash method back to JTAG and again check working by building and flashing.

Viewing serial output

1. The blink example prints Turning the LED ON! and Turning the LED OFF!
2. To view open a serial terminal e.g. SmarTTY and connect to COM3 at 115200 baud
3. Or just use the Build, Flash & Monitor icon to open a terminal in the ESP-IDF extension

Debugging

1. Click the Run & Debug icon in the left pane, a triangle arrow and a beetle.
    a. If you don't open Run & Debug and start debugging from elsewhere you may see nothing happening for a while and think it's not working; with Run & Debug open you can see the startup of debugging.
2. Click Menu >Run > Start debugging, or press F5 or click the green hollow arrow at the top near Eclipse CDT GDB adapter.
3. Debug stops at the start of void app_main(void) and code execution can be single stepped, or continued, from there.

Software breakpoints

```
// Insert a breakpoint

  __asm__("break 0,0"); // causes execution to stop and can not be stepped on further
```

Logging

ESP-IDF provides different log levels, each corresponding to a different severity, which are output on a COM port and can be viewed in a serial terminal, like the one built into VSCode:

ESP_LOGE: Error messages

ESP_LOGW: Warning messages

ESP_LOGI: Informational messages

ESP_LOGD: Debug messages

ESP_LOGV: Verbose messages

Example code:

```
#include "esp_log.h"

static const char *TAG = "example";

void app_main() {

  int value = 42;

  ESP_LOGI(TAG, "Hello, this is an informational log. Value = %d", value);

}
```

Output would be:

I (5744331) example: Hello, this is an informational log 42

Status bar

The status bar at the bottom of ESP-IDF on VSCode has the following features:

1. Select ESP-IDF version to use
2. COM port to use for uploads and output from the ESP32xx MCU
3. Target MCU
4. Select, or view, current project folder
5. SDK Configuration editor (menuconfig)
6. Full clean of project
7. Build project
8. Choose flash method e.g. JTAG, UART or DFU (Device Firmware Update)
    a. DFU uses the native USB interface on e.g. GPIO19 & 20

b. The ESP32xx chip needs to be in bootloader mode before it can be detected as a DFU device and flash. This can be achieved by pulling GPIO0 down (e.g., pressing the BOOT button), pulling RESET down for a moment, and releasing GPIO0.
c. https://blog.espressif.com/dfu-using-the-native-usb-on-esp32-s2-for-flashing-the-firmware-b2c4af3335f1
d. A complicated method with no obvious benefits.

9. Monitor device - opens a terminal using the COM port selected in point 2.
10. Debug - starts debugging and stops at the line void app_main(void)
11. Build, flash & monitor – does exactly that!
12. Open ESP-IDF terminal – for entering command e.g.
    a. python –version
    b. idf.py –version
    c. idf.py menuconfig
13. Execute custom task – does not seem to do anything?
14. Problems – shows any reported
15. Port forwarding – see: https://code.visualstudio.com/docs/editor/port-forwarding
16. Select and start debug configuration – choose from available debuggers.
17. PlatformIO Home – only because PIO extension is also installed.
18. Select current build variant.
19. Change the active kit.
20. Other items to the right are either not used or for PlatformIO

Run Python commands

See point 12 above i.e. the only terminal where commands like idf.py build may be executed is in the ESP-IDF terminal, which is started from the button on the status bar.

Someone on the Espressif forum suggested installing python3 -m pip install virtualenv but this does not seem to affect anything.

Example projects

Are located in the ESP-IDF folder e.g. C:\ESP-IDF\esp-idf-v5.3\examples and can be viewed for project creation by clicking the ESP-IDF Explorer icon in the left pane beneath PlatformIO, the clicking “Show examples” in the COMMANDS section at the top.

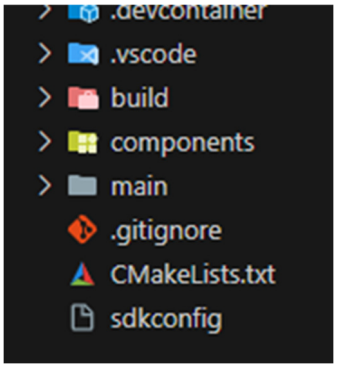
Arduino as a component

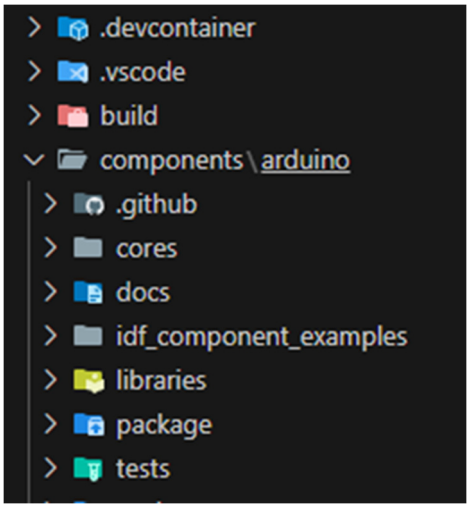
NOTE: This may not be working on newer version of ESP-IDF?

To use Arduino libraries and code structures and commands the Arduino component must be installed.

Go to command palette (Ctrl+Shift+P), type “add arduino” to select “ESP-IDF Add Arduino ESP32 as ESP-IDF Component” command and click on it.

This will create a “components” folder inside your program folder and download the arduino component. If for some reason the arduino component isn’t copied to the folder then go to https://github.com/espressif/arduino-esp32 hit “Code” then “Download zip”, unpack and copy all content of arduino-esp32-master folder into the components folder, which will then magically appear as “components\arduino” when it is expanded to show the contents; it seems to read the contents of the folder and adjust the name presented accordingly. The rename is maintained even after the folder is de-expanded!





https://www.youtube.com/watch?v=hHzGX-K6lmo

https://www.youtube.com/watch?v=7wOpKfeLd7w

https://simplediycircuits.wordpress.com/2023/06/25/migrate-arduino-esp32-project-to-esp-idf-in-visual-studio-code-with-user-libraries-to-part-1/#:~:text=Still%20you%20don't%20have,and%20download%20the%20arduino%20component

arduino-esp32 is at: https://github.com/espressif/arduino-esp32 but it is odd that when downloaded from GitHub via http the size is about 50Mb but if using Git clone it is over 2Gb!

How to use Arduino as an ESP-IDF component is here: https://docs.espressif.com/projects/arduino-esp32/en/latest/esp-idf_component.html but I have yet to get it working.

Note: Git clone downloads can be slow and unreliable, just persevere, or if you can download the zip from GitHub via HTTP, unzip and place in the components/arduino folder. Git clone cannot be restarted, you'll need to rm -rf folder-name where folder name is the name of the folder you are trying to clone, and then restart the clone from the beginning.

Keyboard commands

1. Ctrl Shift P = open command palette
2. Ctrl Shift F = search
3. Shift Alt F = format document

Useful code bits

1. esp_restart();
   a. Stops all running tasks
   b. Resets all peripherals
   c. Reboots the CPU

Problems and solutions

1. Build fails often with Ninja error = try CLEAN and also re-select the target e.g. esp32s3
2. Some esp32sx modules do not reset after upload = press RESET button on module
3. JTAG upload fails = change upload to UART, upload then change back to JTAG

Web resources

1. Get started with ESP-IDF v5.3.1: https://docs.espressif.com/projects/esp-idf/en/v5.3.1/esp32/get-started/index.html
2. Install ESP-IDF on VSCode: https://github.com/espressif/vscode-esp-idf-extension/blob/master/docs/tutorial/install.md
3. Manual installation of ESP-IDF tools: https://docs.espressif.com/projects/esp-idf/en/v5.3.1/esp32s3/get-started/windows-setup.html#

4. ESP-IDF downloads page: https://dl.espressif.com/dl/esp-idf/?idf=5.3.1
5. Install ESP-IDF extension on VSCode: https://github.com/espressif/vscode-esp-idf-extension/blob/master/docs/tutorial/install.md
6. Use ESP-IDF extension on VSCode: https://github.com/espressif/vscode-esp-idf-extension/blob/master/docs/tutorial/basic_use.md
7. Configure ESP32-S3 built-in JTAG Interface: https://docs.espressif.com/projects/esp-idf/en/v5.0.7/esp32s3/api-guides/jtag-debugging/configure-builtin-jtag.html
8. Start a project: https://docs.espressif.com/projects/esp-idf/en/v5.3.1/esp32s3/get-started/windows-start-project.html
9. IDF Frontend idf.py: https://docs.espressif.com/projects/esp-idf/en/v5.3.1/esp32s3/api-guides/tools/idf-py.html
10. https://docs.espressif.com/projects/esp-idf/en/stable/esp32s3/get-started/establish-serial-connection.html
11. https://docs.espressif.com/projects/esp-idf/en/stable/esp32s3/api-guides/jtag-debugging/index.html
12. https://github.com/espressif/vscode-esp-idf-extension/blob/master/docs/C_CPP_CONFIGURATION.md
13. https://docs.espressif.com/projects/esp-idf/en/v3.3.5/api-guides/jtag-debugging/setup-openocd-windows.html
14. https://docs.espressif.com/projects/esp-idf/en/v3.3.5/api-guides/jtag-debugging/index.html#jtag-debugging-configuring-esp32-target
15. https://github.com/espressif/idf-env/releases/tag/v1.2.31
16. https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-guides/tools/idf-component-manager.html