Create, Build and Debug ESP32xx MCUs using ESP-IDF on VSCode

Overview

After trying to use Arduino IDE, PlatformIO and Eclipse ESP-IDF, which either do not support debugging, were very complex to set-up, problematic or unreliable I finally arrived at ESP-IDF on Visual Studio Code and thought I had gone to heaven! Both VSCode and the ESP-IDF extension are free, easy to install and set-up, fully featured and rock-solid reliable, especially for debugging.

Software

1. Download and install VSCode from: https://code.visualstudio.com/download
2. In VSCode Extensions search for and install ESP-IDF

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.
2.

USB Ports

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

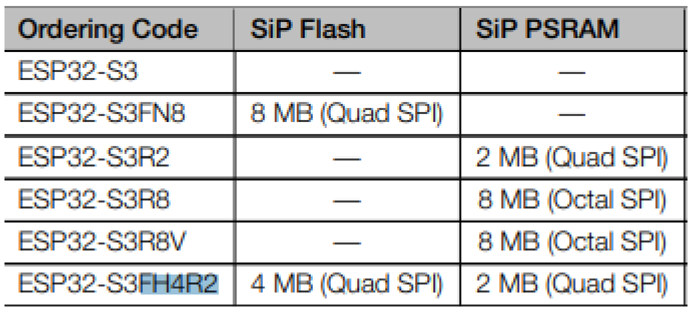
1. ESP32 has one standard COM port used for serial upload and printing.
2. ESP32-Sx mini has one USB port, which presents as:
    a. USB native
        i. USB JTAG/serial debug unit (interface 0) with driver libwdi 1.0.0.0 reported by device manager and usbser 1.0.0.0 by Zadig. Windows will allocate 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.

Flashing

1. The older ESP32 devices can be flashed via the serial COM port, to use JTAG requires connecting a serial to JTAG adapter such as an ESP-PROG.
2. More modern single USB port devices can be flashed via the COM port, or via the JTAG port, selected from the Flash Method button on the status bar. Whichever is used the COM port is available for serial printing or LOG outputs when not being used for flashing.
3. Modern two USB port devices can be flashed via either COM port or via JTAG. I think it is preferable to use the COM port (e.g. COM4) associated with the USB native socket for uploads, which leaves the standard COM port (e.g. COM3) for serial print output.

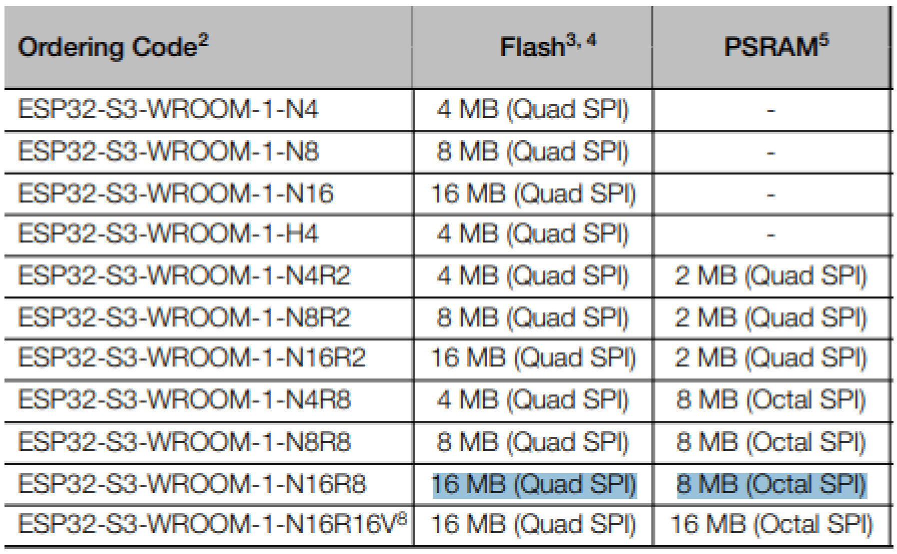
Flashing problems

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
    c. ESP32-S3 Dev Kit N16R8 = 16Mb



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)

2.

Keyboard commands

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

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.

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:

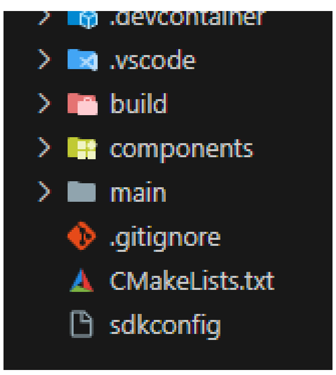
```
I (5744331) example: Hello, this is an informational log 42
```

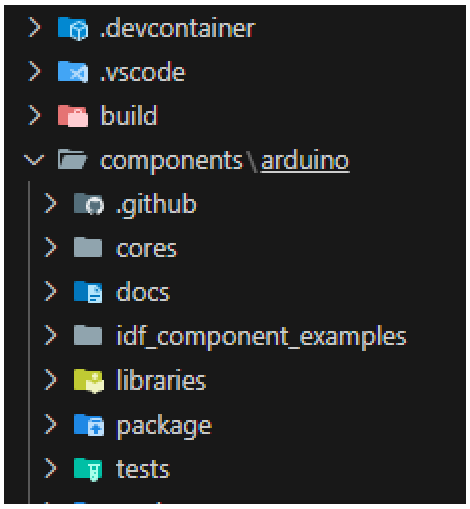
Arduino as a component

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://youtu.be/hHzGX-K6lmo basic set-up

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

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.

Useful code bits

1. esp_restart();
   a. Stops all running tasks
   b. Resets all peripherals
   c. Reboots the CPU
2.