

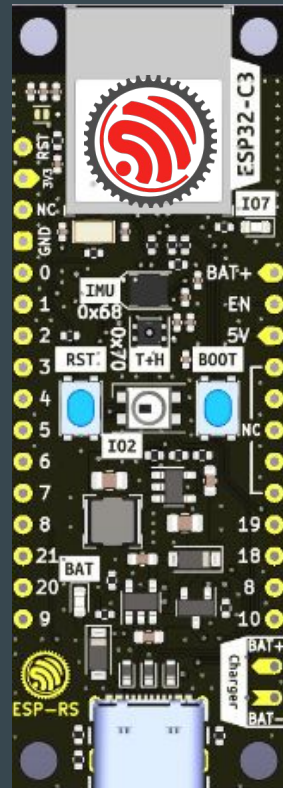
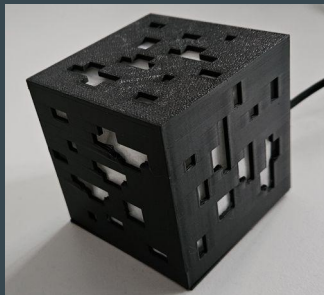
ESP32 Ecosystem

Embedded Development with
Rust, Swift, Python
TypeScript and C

2025-11-20

FEKT VUT

Juraj Michálek
Espressif Systems



Espressif team & friends

Juraj Michálek

- ESP32 Ecosystem Introduction

Petr Gad'orek

- ESP-NOW in Rust

Petr Kubica - <https://helceletka.cz/robotarna/>

- ESP32 in TypeScript & JavaScript

Jan Beran

- ESP32-C6 Workshop

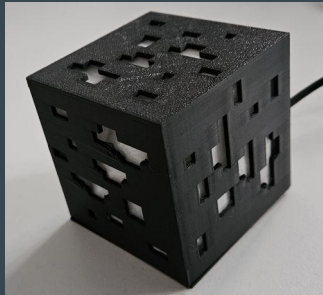


Embedded Projects

IoTcraft

Digital Twin in Voxel World

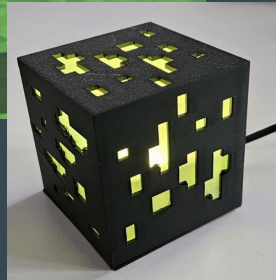
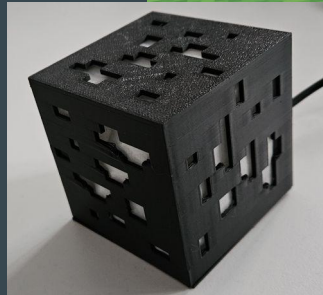
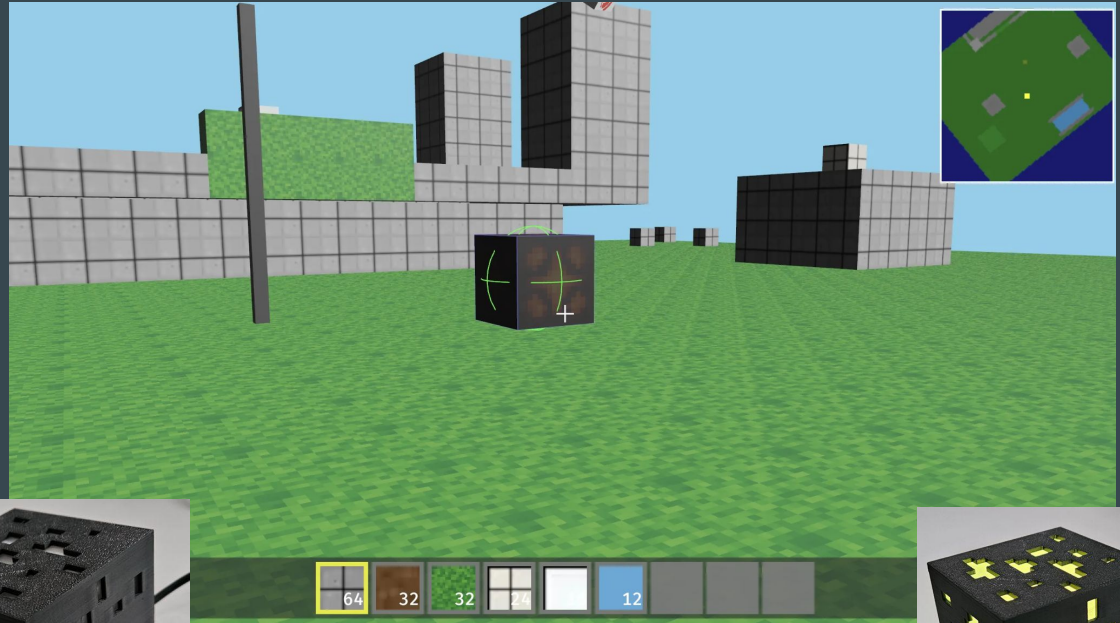
<https://github.com/georgik/iotcraft>



IoTcraft

Digital Twin in Voxel World

<https://github.com/georgik/iotcraft>



ESP-Rainbow



<https://www.crowdsupply.com/atomic14/esp32-rainbow>

Retro ESP32

<https://github.com/retro-esp32/RetroESP32>



OpenTyrian



<https://github.com/georgik/OpenTyrian> - using SDL3

esp32-quake + LibreQauake

<https://github.com/espressif/esp32-quake>

<https://github.com/lavenderdotpet/LibreQauake>

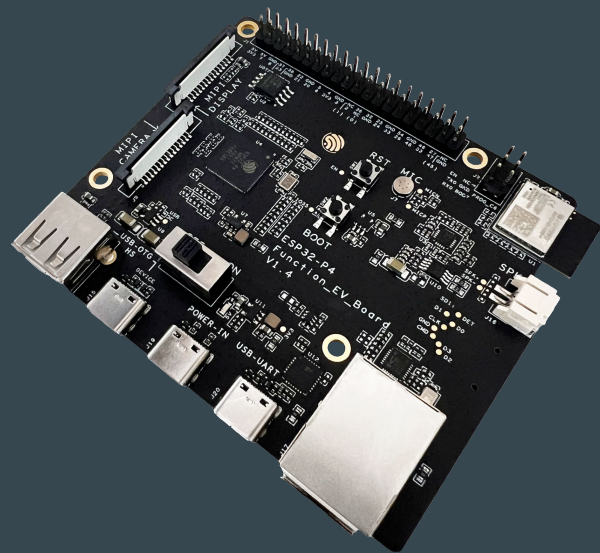


Old games ported to ESP32-P4

OpenTyrian - <https://github.com/georgik/OpenTyrian/>

Doom / FreeDoom - <https://github.com/espressif/esp32-doom>

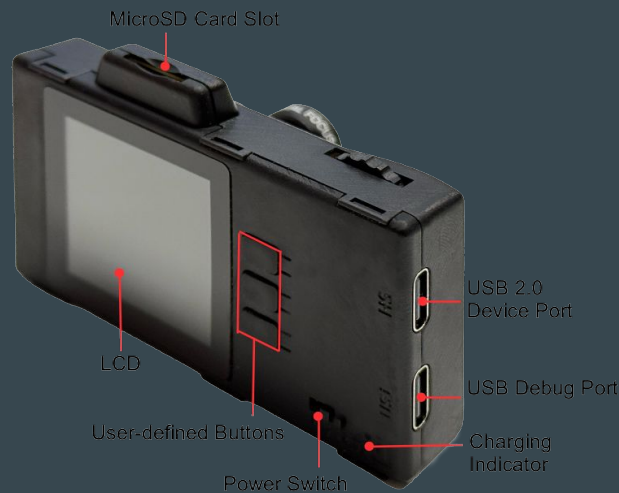
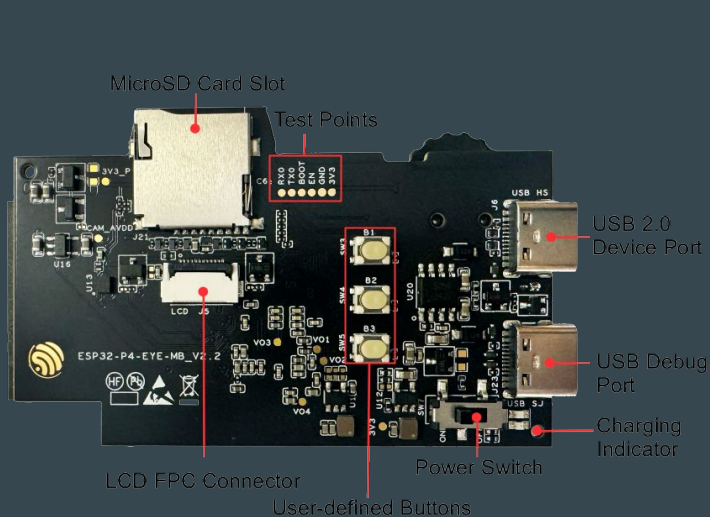
Quake 1 / LibreQuake - <https://github.com/espressif/esp32-quake>



ESP32-P4-EYE

https://docs.espressif.com/projects/esp-dev-kits/en/latest/esp32p4/esp32-p4-eye/user_guide.html

<https://github.com/espressif/esp-dev-kits/tree/master/examples/esp32-p4-eye>



EchoEar

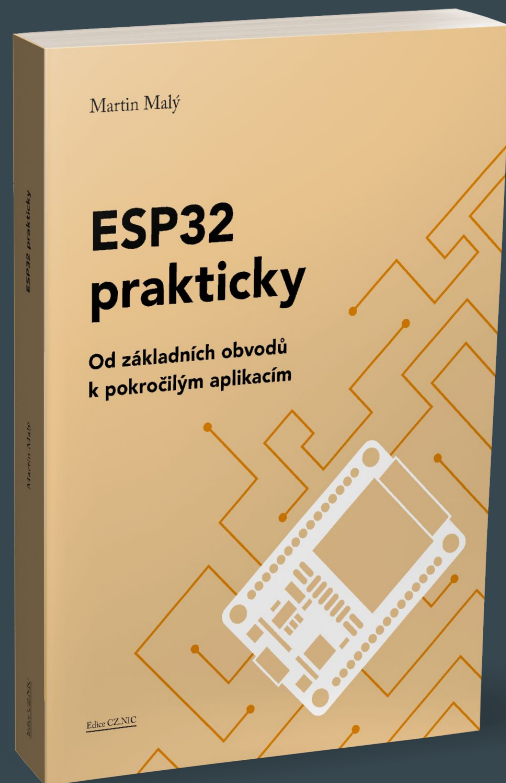


<https://docs.espressif.com/projects/esp-dev-kits/en/latest/esp32s3/echoear/index.html>

Book recommendation: ESP32 prakticky

<https://espx.cz>

<https://github.com/espx-cz>



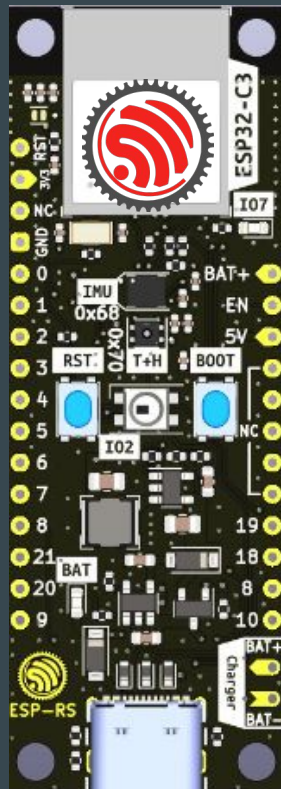
Open Hardware - esp-rust-board

KiCad templates

<https://github.com/esp-rs/esp-rust-board>

ESP32-C3-DevKit-RUST-1

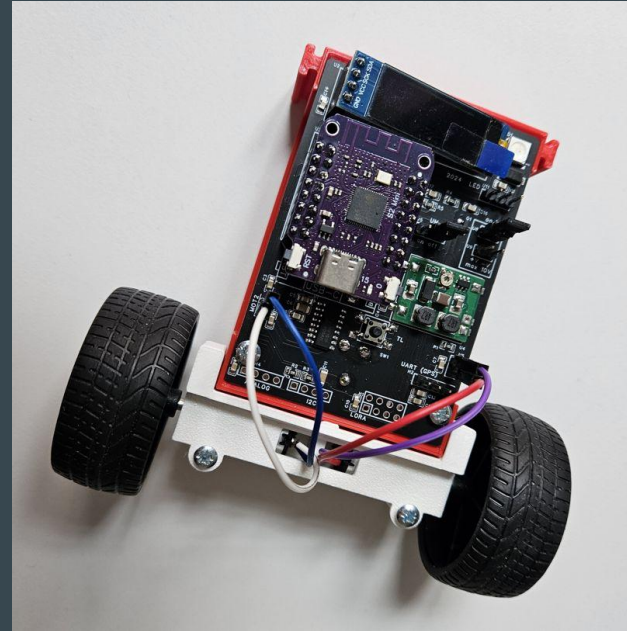
<https://www.espressif.com/en/products/devkits>



Racing Car in MicroPython

ESP32-S2 - WiFi controller car

Developed by Technical Cybernetics UNIZA

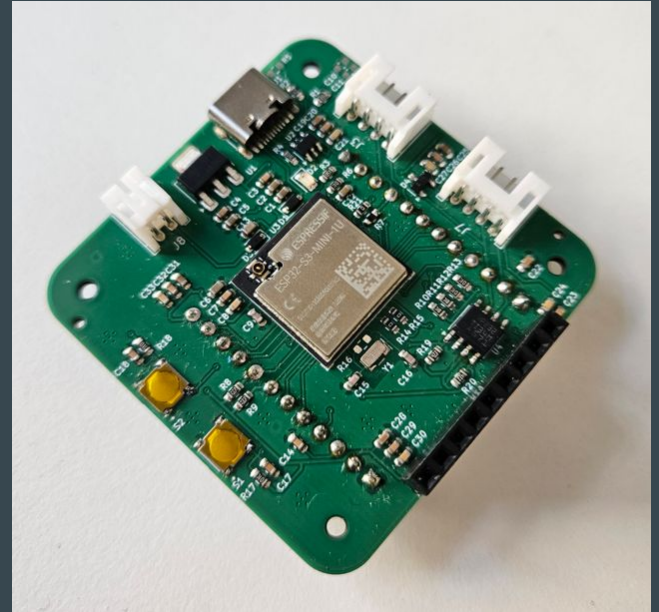


FRI STACK

M5Stack compatible Open Hardware stack

Developed by Technical Cybernetics UNIZA

- https://github.com/andsim04/FRI_STACK_HARDWARE



RISC-V vs Xtensa

RISC-V

- newer
- ESP32-C, ESP32-H, ESP32-P
- upstream Rust compiler is sufficient

Xtensa

- older
- ESP32, ESP32-S2, ESP32-S3
- requires custom installation
- [espup install](#)



<https://www.espressif.com/en/products/longevity-commitment>

ESP Product Selector + Product Comparison

<https://products.espressif.com/>



ESP32-S3

ESP32-S3 is a low-power MCU-based SoC that supports 2.4 GHz Wi-Fi and Bluetooth® Low Energy (Bluetooth LE).

ESP32-S3 has a complete Wi-Fi subsystem and a Bluetooth LE subsystem. State-of-the-art power and RF performance. S3 provides a rich set of peripheral interfaces, and supports ultra-low-power applications. Different security features allow the device to meet stringent security requirements.

Features:

- Core: Xtensa® single-dual 32-bit LX7 CPU, frequency up to 240MHz
- Memories:

Block Diagram:



Product Brief

Docs & Certs

DevKits

List: 203 items

IC/Module

Development Board

Comparison

Export

☐	Index	Name	MPN	Marketing Status	Type	Wi-Fi
☐	1	ESP32-S3	ESP32-S3	Mass Production	SoC	IEEE 802.11 b/g/n; 2.4 ...
☐	2	ESP32-S3	ESP32-S3R2	Mass Production	SoC	IEEE 802.11 b/g/n; 2.4 ...
☐	3	ESP32-S3	ESP32-S3R8	Mass Production	SoC	IEEE 802.11 b/g/n; 2.4 ...
☐	4	ESP32-S3	ESP32-S3R...	Mass Production	SoC	IEEE 802.11 b/g/n; 2.4 ...
☐	5	ESP32-S3	ESP32-S3F...	Mass Production	SoC	IEEE 802.11 b/g/n; 2.4 ...

Product Selector

Product Comparison

Product Portfolio

Sales Questions

Technical Inquiries

Hide Same

Show Same

	ESP32-S3	ESP32-P4NRW16
Series	ESP32-S3	ESP32-P4
CPU	Xtensa® dual-core 32-bit LX7	32-bit RISC-V single-core processor
Freq. (MHz)	240	400
Package (mm)	QFN56 (7*7)	QFN10*10
Dimensions (mm)	7*7	10*10
Temp. (°C)	-40 °C ~ 105 °C	-40 °C ~ 85 °C
Status	Mass Production	Sample
ECO	standard version	
Support IDF	v0.x	

OSes and integration



Rust no_std - <https://github.com/esp-rs/esp-hal> - official support

Rust std - <https://github.com/esp-rs/esp-idf-hal> - community support

Ariel OS - <https://ariel-os.github.io/ariel-os/dev/docs/book/>



Zephyr - <https://zephyrproject.org/>

NuttX - <https://nuttx.apache.org/>



SVD files: <https://github.com/espressif/svd>

IDE for development

Supported by Espressif:

- VS Code with Espressif Extension - <https://developer.espressif.com/tags/vscode/>



Supported by JetBrains:

- CLion - <https://developer.espressif.com/blog/clion/>
- RustRover - <https://www.jetbrains.com/rust/>
- free license for students and open source projects



Rust



Slint

Quick prototyping

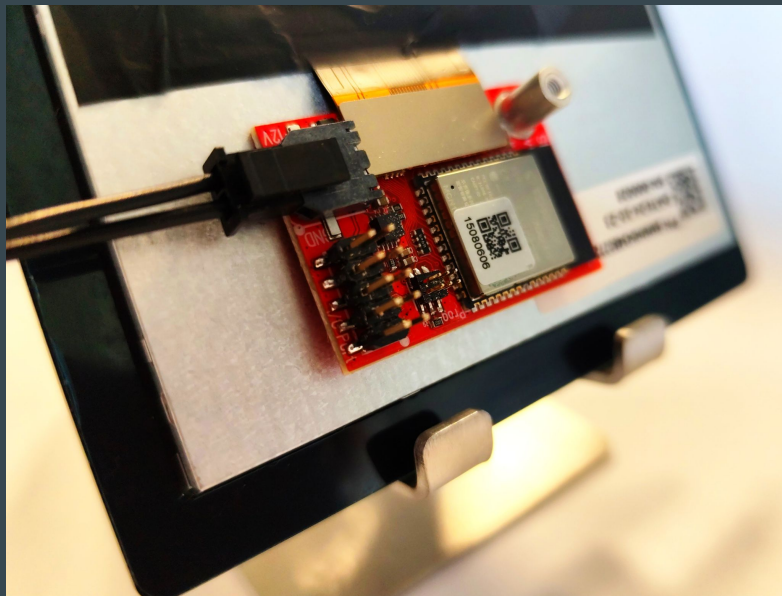
Free and commercial version



Slint Workshop

<https://github.com/WilstonOreo/slint-esp-workshop/>

[ESoPe board](#) + [RGB Schukat Smartwin display-concept](#)



Will be published at Developer Portal: <https://github.com/espressif/developer-portal/pull/495>

We welcome feedback.

Quick start with AI at espressif.com

 **ESPRESSIF**

 **Espressif Docs AI**

How to create Hello world for ESP32-C6 in Rust no_std

To create a "Hello world" application for the ESP32-C6 in Rust using the `no_std` (bare-metal) approach, you should use the `esp-hal` crate, which provides hardware abstraction for Espressif chips in a `no_std` environment. The process involves generating a new project, configuring it for the ESP32-C6, and writing your application logic.

1. Generate a New Project

Espressif provides a project generation tool called `esp-generate` to bootstrap embedded Rust projects. To create a new project for ESP32-C6:

```
cargo install esp-generate
esp-generate --chip esp32c6 hello_world
```

This will scaffold a new Rust project set up for the ESP32-C6 in a `no_std` environment `esp-hal` 1.0.0 beta announcement.

2. Write the Application

A typical `no_std` "Hello world" for embedded is to blink an LED since there is no standard output. However, if you want to print to the serial console, you need to set up UART. The knowledge sources provide a blinky example for ESP32-C3,

Ask a follow-up

Deep thinking

Powered by  kapa.ai

Protected by hCaptcha

esp-generate --chip esp32c6

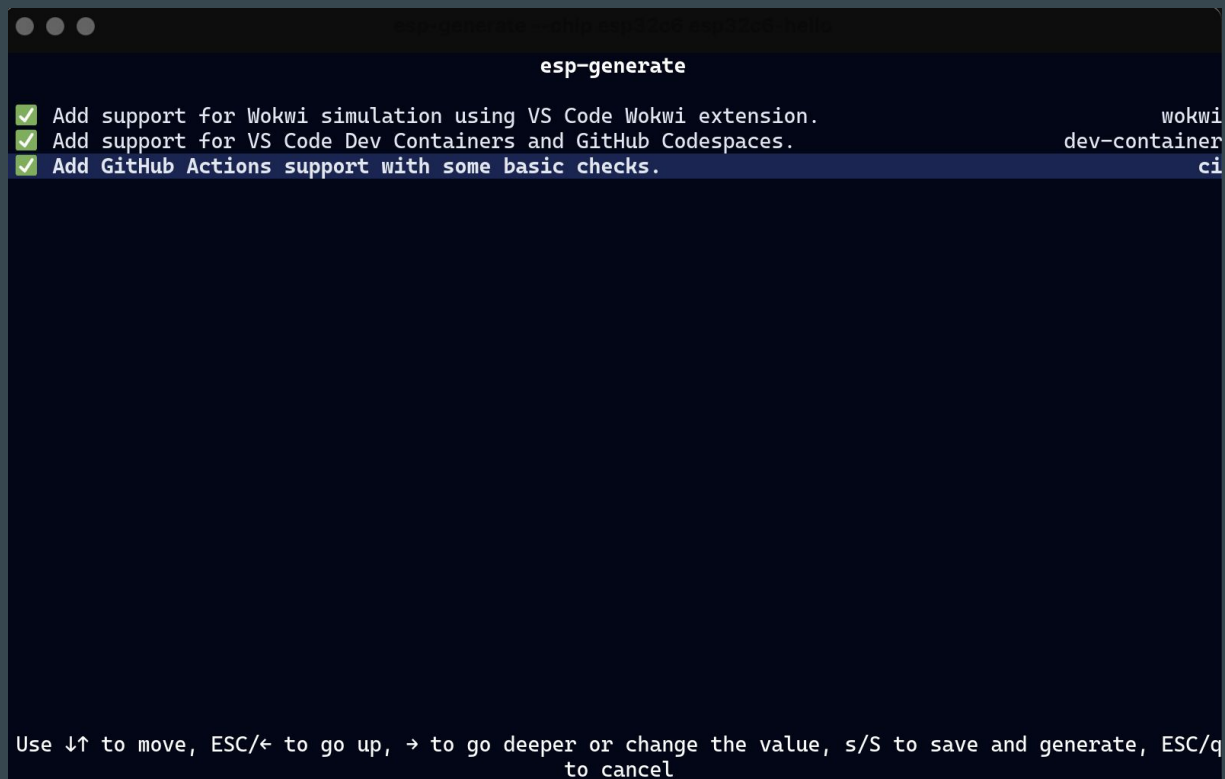
```
esp-generate --chip esp32c6 esp32c6-hello

esp-generate

[✓] Enable unstable HAL features.                                     unstable-hal
[✓] Enable allocations via the esp-alloc crate.                      alloc
[✓] Enable Wi-Fi via the esp-wifi crate.                             wifi
    Enable BLE via the esp-wifi crate (bleps).                      ble-bleps
    Enable BLE via the esp-wifi crate (embassy-trouble).           ble-trouble
[✓] Add embassy framework support.                                   embassy
[✓] Enable stack smashing protection.                               stack-smashing-protection
    Use probe-rs to flash and monitor instead of esptool.          probe-rs
▶ Flashing, logging and debugging (probe-rs)
▶ Flashing, logging and debugging (esptool)
▶ Options
▶ Optional editor integration

This configuration enables unstable esp-hal features. These come with no stability guarantees, and
could be changed or removed at any time. Required by `wifi` and `embassy`.
Use ↓↑ to move, ESC/← to go up, → to go deeper or change the value, s/S to save and generate, ESC/q
to cancel
```

esp-generate - more options



Build, flash, monitor

cargo run --release

shorter: cargo r -r

```
Build: Sep 19 2022
rst:0x1 (POWERON),boot:0xc (SPI_FAST_FLASH_BOOT)
SPIWP:0xee
mode:DIO, clock div:2
load:0x4086c410,len:0xd48
load:0x4086e610,len:0x2d68
load:0x40875720,len:0x1800
entry 0x4086c410
I (23) boot: ESP-IDF v5.1-beta1-378-gea5e0ff298-dirt 2nd stage bootloader
I (24) boot: compile time Jun 7 2023 08:02:08
I (25) boot: chip revision: v0.1
I (29) boot.esp32c6: SPI Speed      : 40MHz
I (33) boot.esp32c6: SPI Mode      : DIO
I (38) boot.esp32c6: SPI Flash Size : 4MB
I (43) boot: Enabling RNG early entropy source ...
I (49) boot: Partition Table:
I (52) boot: ## Label                Usage            Type ST Offset   Length
I (59) boot:  0 nvs                  WiFi data        01 02 00009000 00006000
I (67) boot:  1 phy_init             RF data          01 01 0000f000 00001000
I (74) boot:  2 factory              factory app      00 00 00010000 003f0000
I (82) boot: End of partition table
I (86) esp_image: segment 0: paddr=00010020 vaddr=42000020 size=0cc64h ( 52324) map
I (105) esp_image: segment 1: paddr=0001cc8c vaddr=40800000 size=00014h (   20) load
I (106) esp_image: segment 2: paddr=0001cca8 vaddr=4200cca8 size=4a5cch (304588) map
I (173) esp_image: segment 3: paddr=0006727c vaddr=40800014 size=0e324h ( 58148) load
I (187) esp_image: segment 4: paddr=000755a8 vaddr=40826f48 size=001e0h (   480) load
I (191) boot: Loaded app from partition at offset 0x10000
I (191) boot: Disabling RNG early entropy source ...
```

Simulation: Update wokwi.toml from debug to release

```
[wokwi]
```

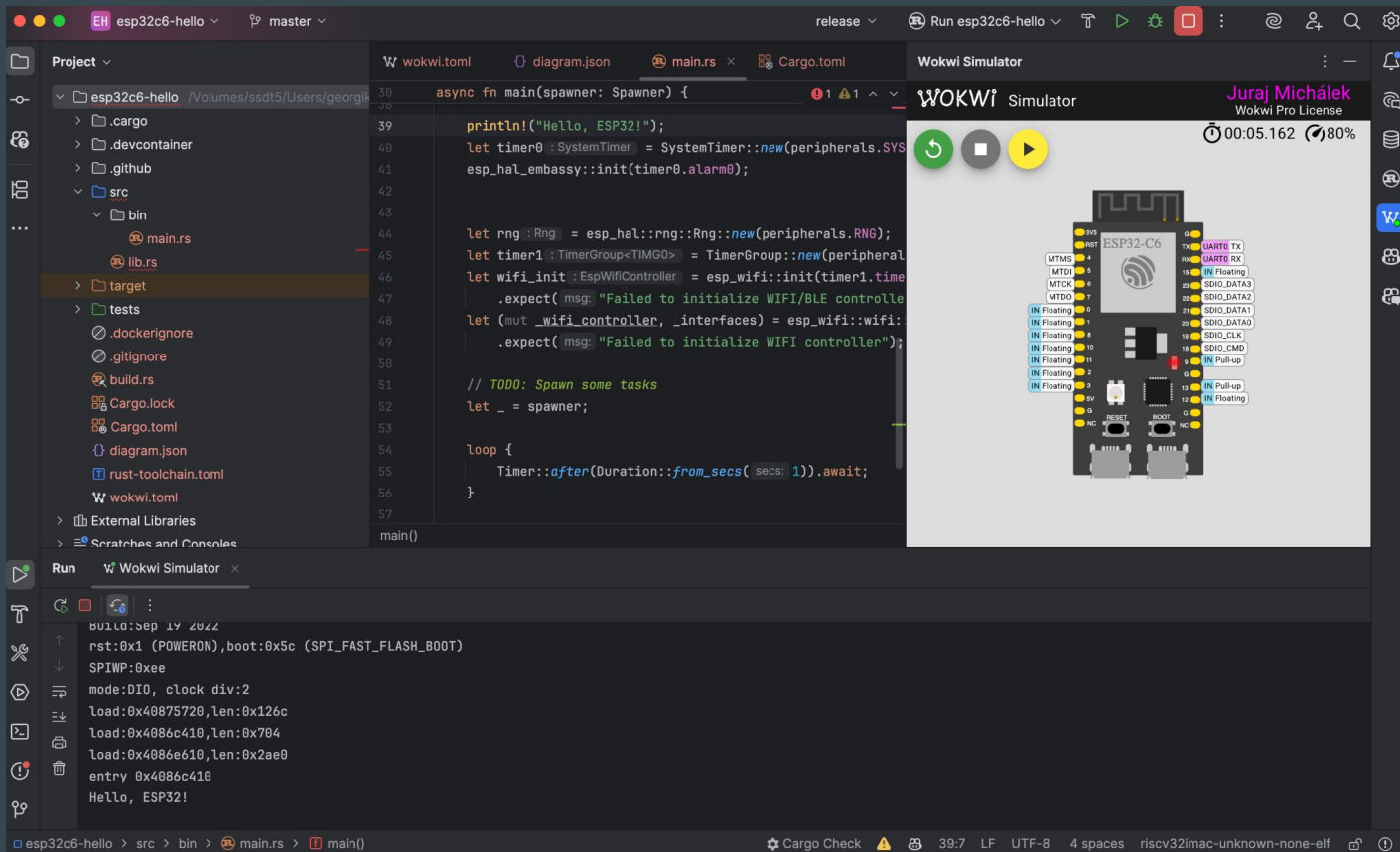
```
version = 1
```

```
gdbServerPort = 3333
```

```
#elf = "target/riscv32imac-unknown-none-elf/release/esp32c6-hello"
```

```
firmware = "target/riscv32imac-unknown-none-elf/release/esp32c6-hello"
```

CLion with Rust plugin or RustRover



<https://developer.espressif.com>



ESPRESSIF

Developer Portal

[Blog](#)

[Workshops](#)

[Events](#)

[Products](#) ▾

[Quick Links](#) ▾



esp-hal 1.0.0 release announcement

30 October 2025 · 3 mins · 

Esp32

Rust

Xtensa

RISC-V

Announcement

AUTHOR

Scott Mabin

► Table of Contents

Announcing esp-hal 1.0, the first Rust SDK for embedded devices.

In February this year, we announced the first `esp-hal 1.0 beta` release. Since then we've been hard at work, polishing and preparing for the full release. Today, the Rust team at Espressif is excited to announce the official `1.0.0` release for `esp-hal`, the *first* vendor-backed Rust SDK!

Open Source - <https://github.com/esp-rs>



esp-rs

Libraries, crates and examples for using Rust on Espressif SoC's

README.md

Rust on Espressif microcontrollers

This organization is home to several projects enabling the use of the [Rust programming language](#) on various SoCs and modules produced by [Espressif Systems](#).

If you are just getting started writing Rust for ESP devices, please first read [The Rust on ESP book](#).

For a curated list of resources for development including tools and projects, see [Awesome ESP Rust](#).

Hardware Abstraction Layer

We offer two choices for Hardware Abstraction Layers:

Repository	Description	Support status
esp-rs/esp-hal	Without support for the Rust standard library (<code>no_std</code>)	Support status Official
esp-rs/esp-idf-hal	With support for the Rust standard library (<code>std</code>)	Support status Community

Rust no_std + esp-hal	
ROM functions	Registers
HW	

<https://github.com/esp-rs/esp-hal>

Official Espressif + Community Support

Requires Rust compiler

Bleeding edge MCUs and drivers might be missing

C wrapper app	Rust std + esp-idf-hal	
ESP-IDF		FreeRTOS
ROM functions	Registers	
HW		

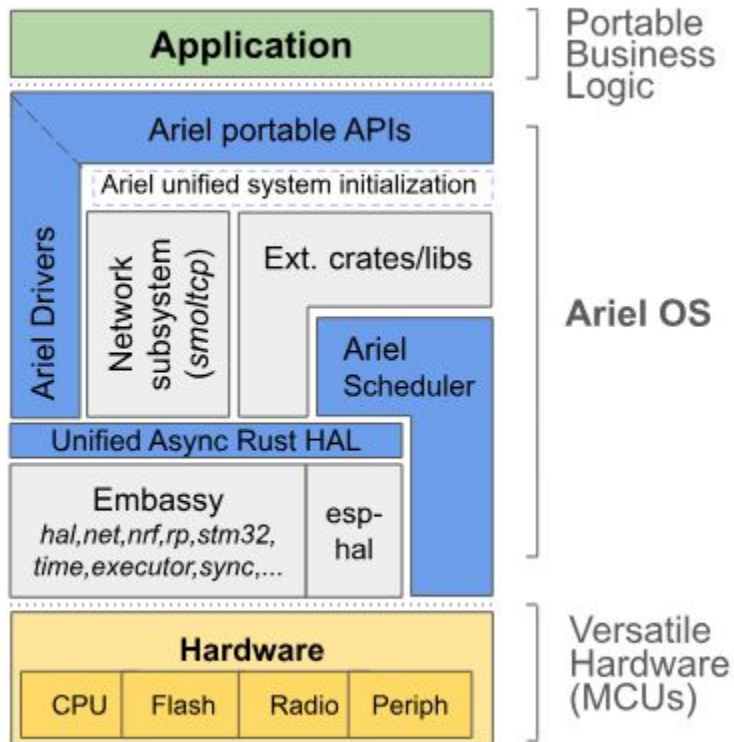
<https://github.com/esp-rs/esp-idf-hal>

Community support

Requires Rust compiler + ESP-IDF

Bleeding edge MCUs and drivers from ESP-IDF

Ariel OS

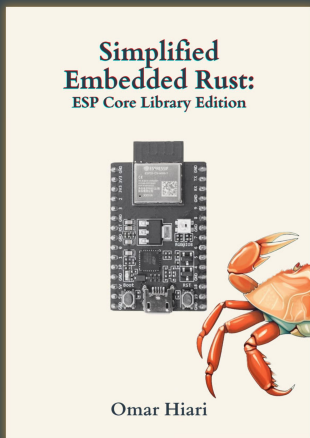


<https://ariel-os.github.io/ariel-os/dev/docs/book/>

Training

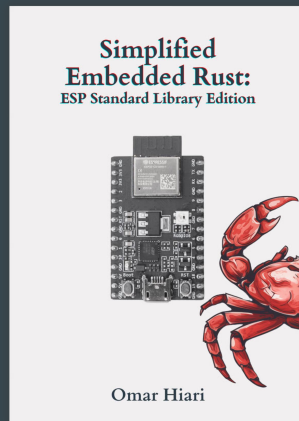
Rust no_std

- https://docs.espressif.com/projects/rust/no_std-training/
- https://github.com/esp-rs/no_std-training
- <https://www.theembeddedrustacean.com/c/ser-no-std>



Rust std

- <https://docs.esp-rs.org/std-training/>
- <https://github.com/esp-rs/std-training>
 - Created in cooperation with [Ferrous Systems](#)
- <https://www.theembeddedrustacean.com/c/ser-std>



rs-matter



license **Apache2** CI **passing** crates.io **v0.1.0** m join matrix 43 users

What is it exactly?

A pure-Rust, `no_std`, no-alloc, async-first, extensible and safe implementation of the [Matter protocol](#).

Scales from bare-metal MCUs with 1MB flash and 256KB RAM to ARM embedded Linux and bigger iron!

Rather than a shrink-wrapped solution, it is first and foremost - a **toolkit**. Users are free to consume all of the APIs, including the provided system clusters, or only pick up bits and pieces. As in:

- ... re-using the transport layer and Secure Channel, but implementing their own Data Model;
- ... custom Exchange responders;
- ... custom mDNS provider;
- ... custom IP network implementation and BLE GATT device implementation;
- ... flexible polling of the `rs-matter` futures as e.g. separate tasks in their async executor of choice;
- ... or just using the shrink-wrapped [rs-matter-stack](#) arrangement and its down-stream crates;
- ... and so on.

<https://github.com/project-chip/rs-matter>

Bevy ECS (Entity Component System)

One of biggest open source Rust projects

- <https://bevyengine.org/>

Rust no_std + Bevy ECS

- <https://developer.espressif.com/blog/2025/04/bevy-ecs-on-esp32-with-rust-no-std/>
- <https://github.com/georgik/esp32-conways-game-of-life-rs>



Rust Bare Metal - Embedded Graphics



Source: <https://github.com/georgik/esp32-spooky-maze-game>

Idea: sharing business logic in Rust between
multiple targets

Game is using Bevy ECS no_std.



<https://github.com/espressif/esp-box>

Swift

ESP32 at Do iOS 2024 - Amsterdam



<https://do-ios.com/>

<https://youtu.be/HFnAvDZtW4A?si=NmlpZ53aoVsfAziW>

Swift Embedded Examples

Install via Swiftly: <https://www.swift.org/swiftly/documentation/swiftly/getting-started>

```
swiftly install && hash -r
```

```
idf.py set-target esp32-c6
```

```
idf.py build flash monitor
```

```
Quit monitor: CTRL+]
```

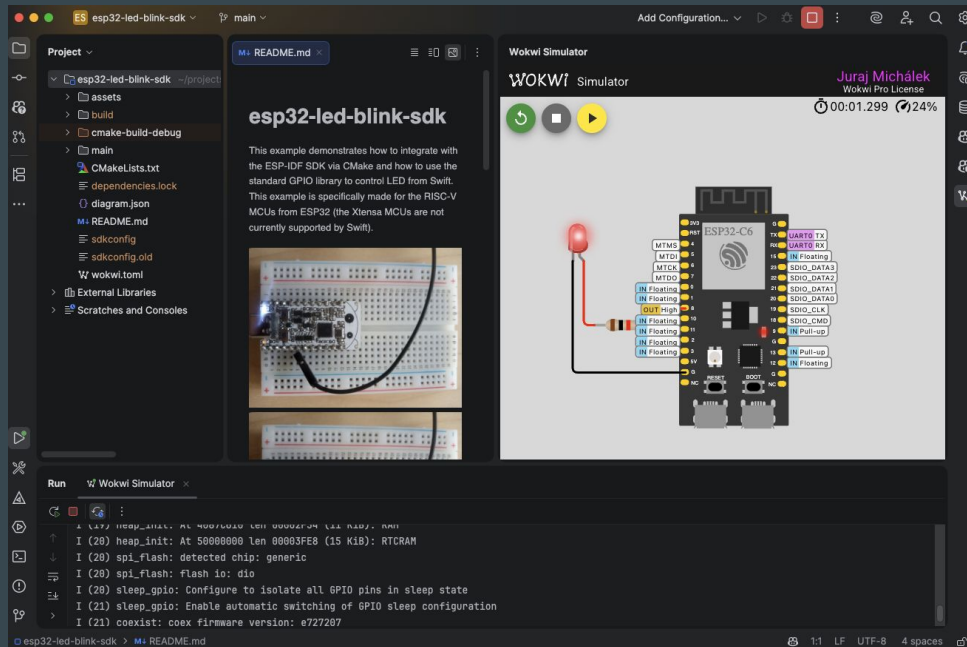
<https://github.com/swiftlang/swift-embedded-examples>

Swift Embedded Examples - esp32-led-blink-sdk

CLion + Wokwi plugin

<https://developer.espressif.com/blog/clion/>

<https://plugins.jetbrains.com/plugin/23826-wokwi-simulator>



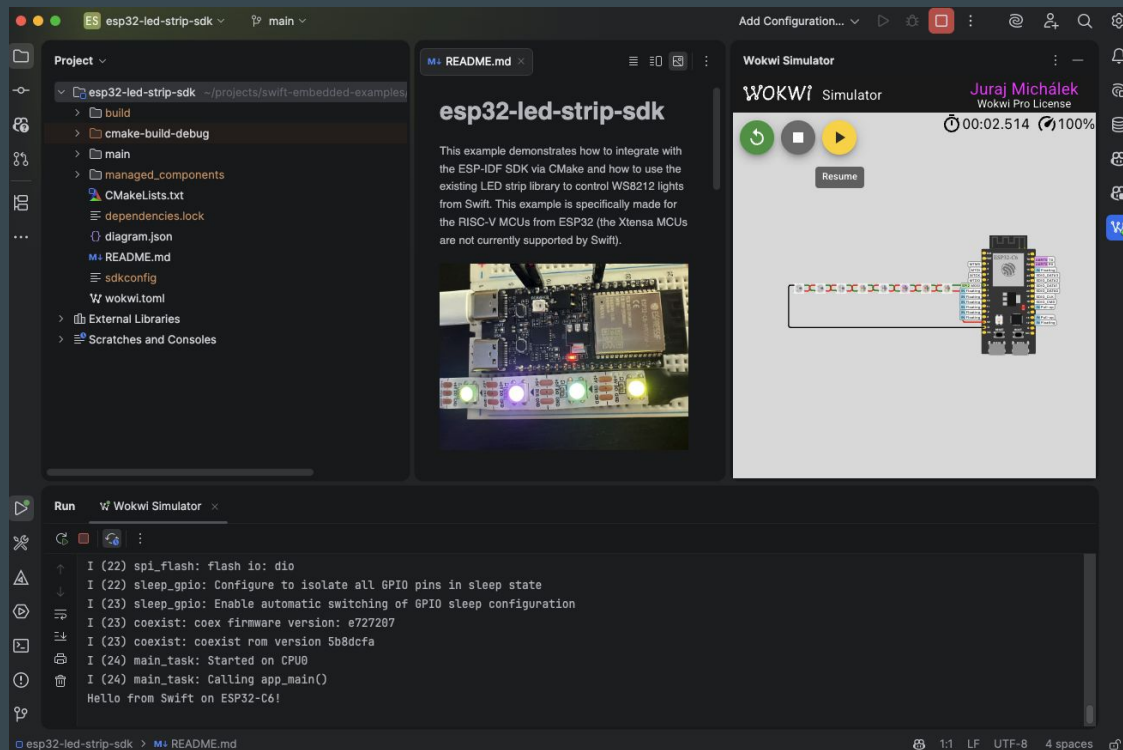
Swift Embedded Examples - esp32-led-strip-sdk

Architecture of ESP-IDF project:

Components

- main
- managed_components
- components (local)

main/idf_component.yml



Supported MCUs

All modern RISC-V: ESP32-C*, ESP32-H*, ESP32-P* family

- easy use of nightly upstream toolchain

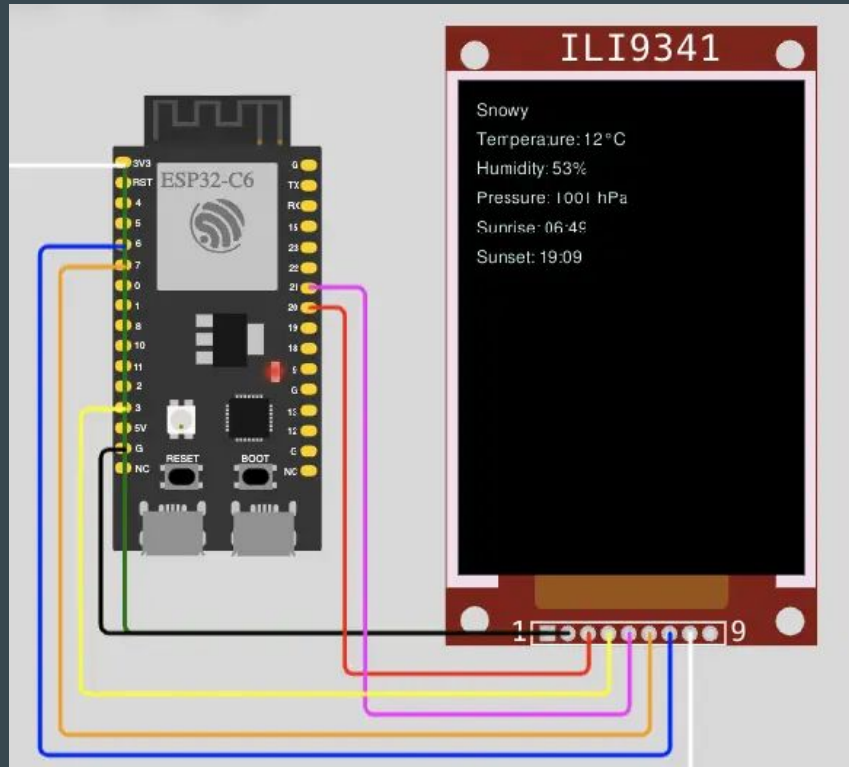
More complex setup: ESP32, ESP32-S* family

- not supported
- would require custom build of Swift toolchain
- these MCUs are older

ESP32 Weather Display

Display connected via SPI

Board Support Package - using Generic



<https://github.com/georgik/esp32-swift-weather-display>

Online [simulation with Wokwi](#)

Smooth integration with C libraries - SDL3

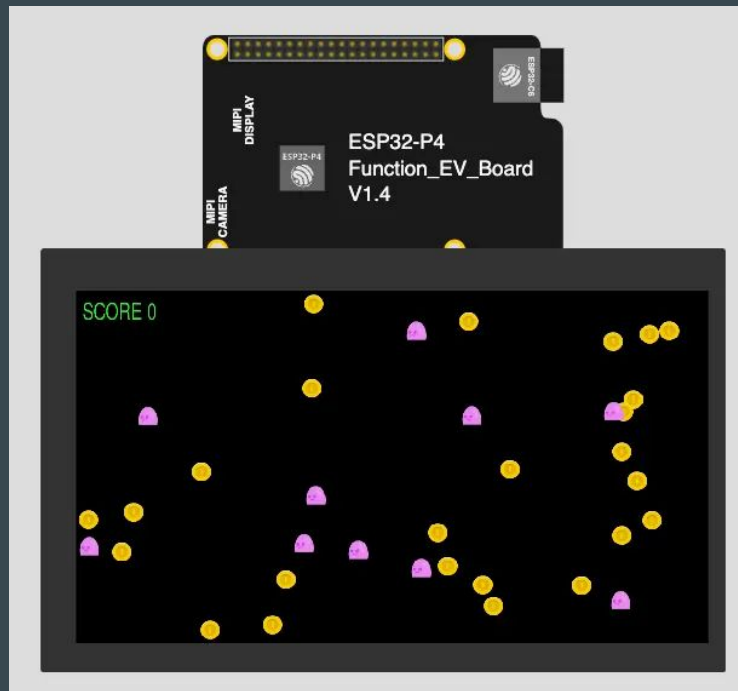
Example with high performance ESP32-P4

RISC-V IMACF

<https://github.com/georgik/esp32-sdl3-swift-example>

Online [simulation with Wokwi](#)

[ESP-IDF SDL3 Component](#)



Embedded Swift - Bare Metal

CMSIS System View Description (CMSIS-SVD)

- <https://github.com/espressif/svd>

Patched version used in Rust no_std

- <https://github.com/esp-rs/esp-pacs/>
- <https://github.com/apple/swift-mmio>

SVD2Swift \

--input ../esp-pacs/esp32c6/svd/esp32c6.svd \

--output Sources/Registers \

--access-level public \

--peripherals GPIO UART0 SPI0

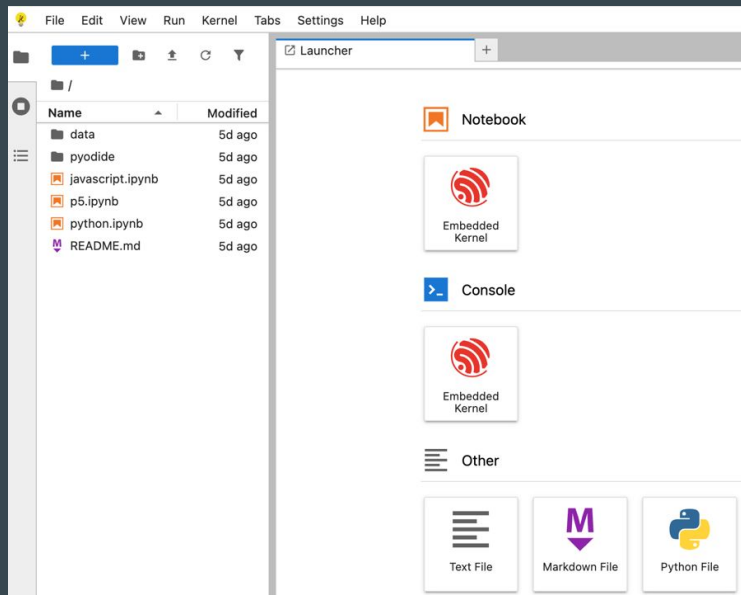
MicroPython

Flow3r

<https://flow3r.garden/>



Jupyter Lite - MicroPython - ESP32

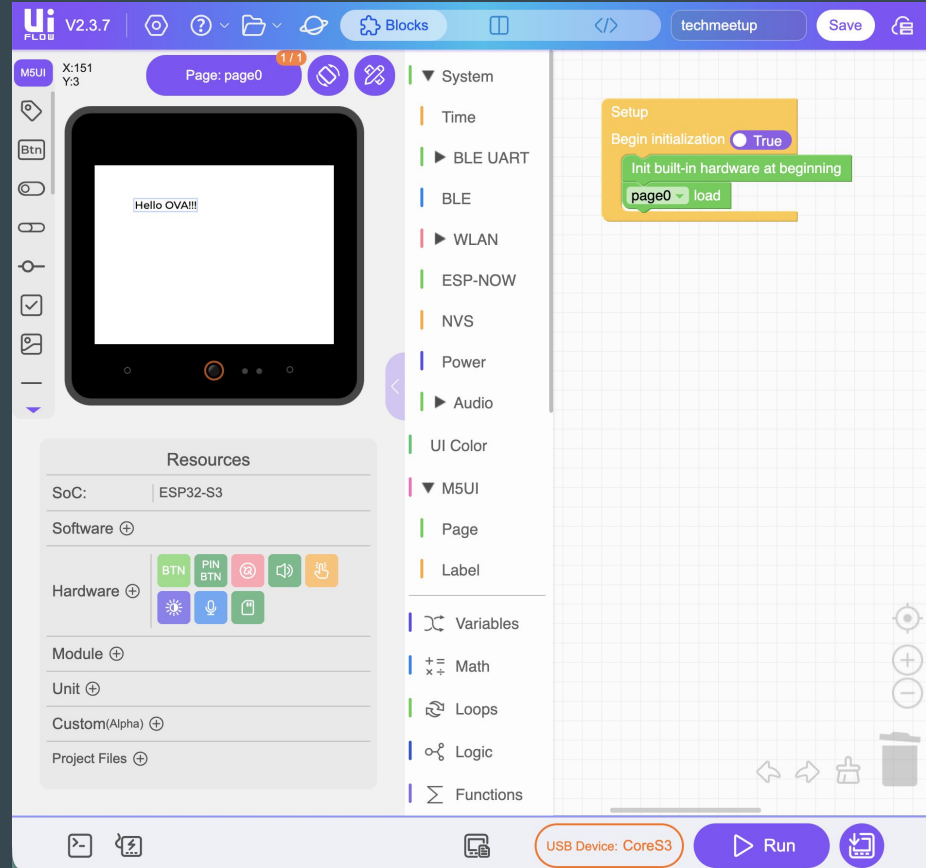


<https://espressif.github.io/jupyter-lite-micropython/lab/index.html>

M5 Stack UIFlow2

<https://uiflow2.m5stack.com/>

Blockly + MicroPython

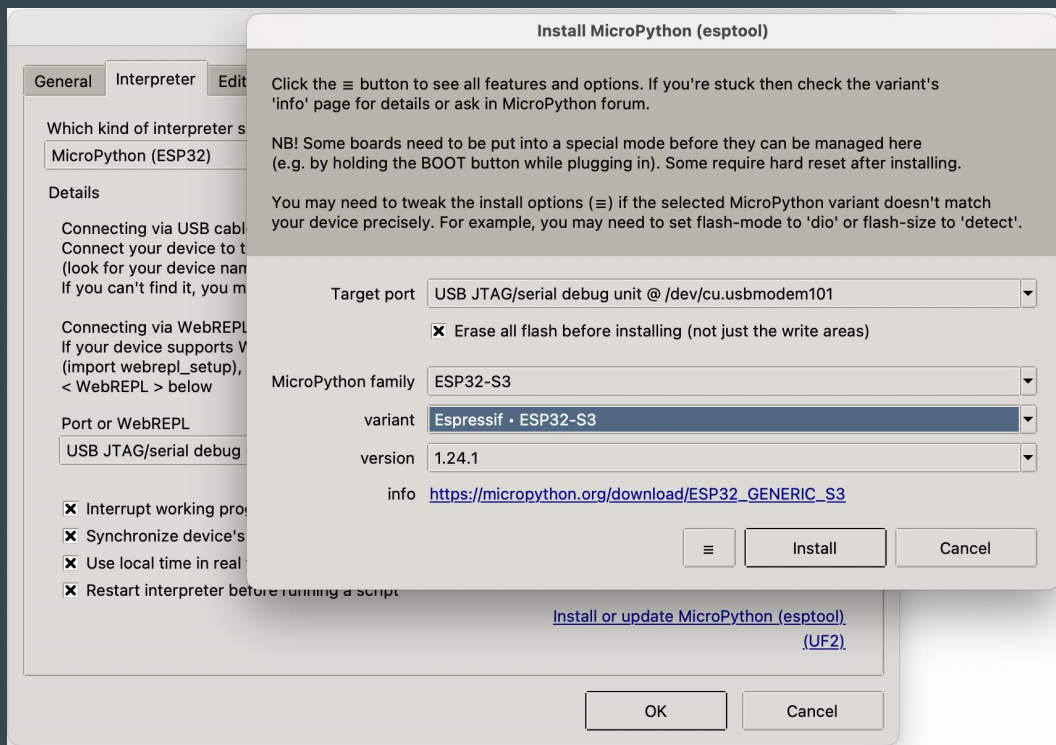


Thonny

Python IDE for beginners

<https://thonny.org/>

Flash MicroPython directly from IDE



TypeScript - Jaculus

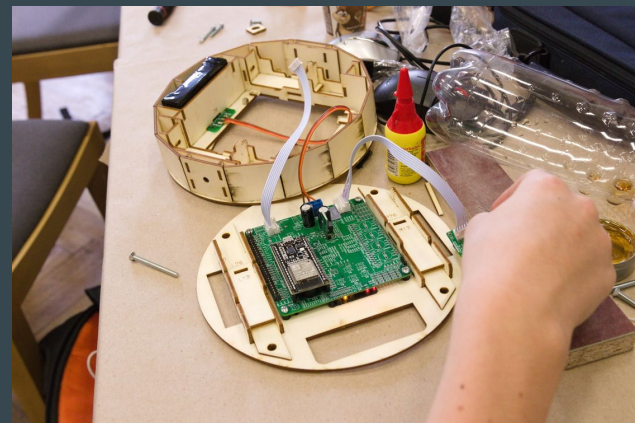
Jaculus

Robotics Brno

<https://jaculus.org/>

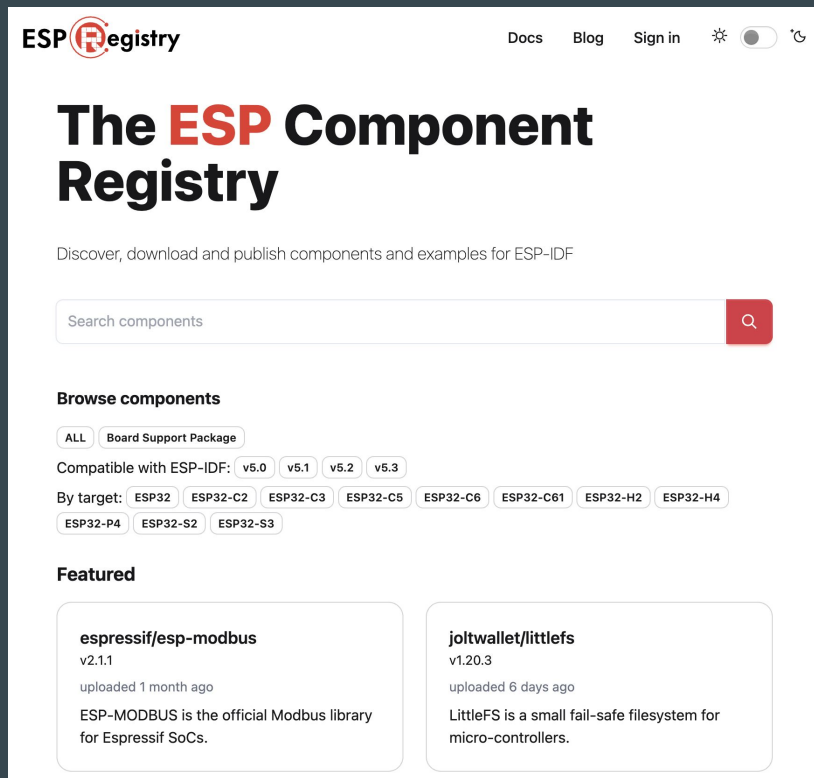
<https://installer.jaculus.org/>

<https://f.kubaandrysek.cz/Jaculus-Wokwi/>



Components

- a.k.a. - libraries
- <https://components.espressif.com/>



The screenshot shows the ESP Component Registry website. At the top, there's a navigation bar with the ESP Registry logo, links for Docs, Blog, and Sign in, and a search icon. The main heading is "The ESP Component Registry". Below it, a subtitle reads "Discover, download and publish components and examples for ESP-IDF". A search bar is present. The "Browse components" section includes filters for "ALL", "Board Support Package", "Compatible with ESP-IDF" (v5.0, v5.1, v5.2, v5.3), and "By target" (ESP32, ESP32-C2, ESP32-C3, ESP32-C5, ESP32-C6, ESP32-C61, ESP32-H2, ESP32-H4, ESP32-P4, ESP32-S2, ESP32-S3). The "Featured" section displays two components: "espressif/esp-modbus" v2.1.1, uploaded 1 month ago, described as the official Modbus library for Espressif SoCs; and "joltwallet/littlefs" v1.20.3, uploaded 6 days ago, described as a small fail-safe filesystem for micro-controllers.

ESP Registry

Docs Blog Sign in

The ESP Component Registry

Discover, download and publish components and examples for ESP-IDF

Search components

Browse components

ALL Board Support Package

Compatible with ESP-IDF: v5.0 v5.1 v5.2 v5.3

By target: ESP32 ESP32-C2 ESP32-C3 ESP32-C5 ESP32-C6 ESP32-C61 ESP32-H2 ESP32-H4 ESP32-P4 ESP32-S2 ESP32-S3

Featured

espressif/esp-modbus
v2.1.1
uploaded 1 month ago

ESP-MODBUS is the official Modbus library for Espressif SoCs.

joltwallet/littlefs
v1.20.3
uploaded 6 days ago

LittleFS is a small fail-safe filesystem for micro-controllers.

Lua as ESP-IDF component

<https://github.com/georgik/esp-idf-component-lua>

```
luaL_openlibs(L);  
  
// Set the Lua module search path to include the assets directory  
if (luaL_dostring(L, "package.path = package.path .. ';./?.lua;/assets/?.lua'")) {  
    ESP_LOGE(TAG, "Failed to set package.path: %s", lua_tostring(L, -1));  
    lua_pop(L, 1); // Remove error message from the stack  
}
```

Smooth integration with C libraries - SDL3

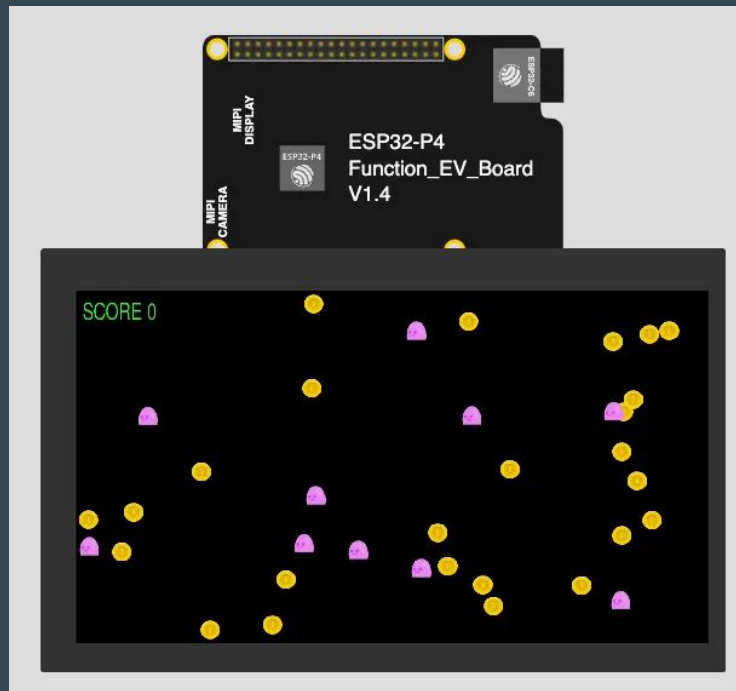
Example with high performance ESP32-P4

RISC-V IMACF

<https://github.com/georgik/esp32-sdl3-swift-example>

Online [simulation with Wokwi](#)

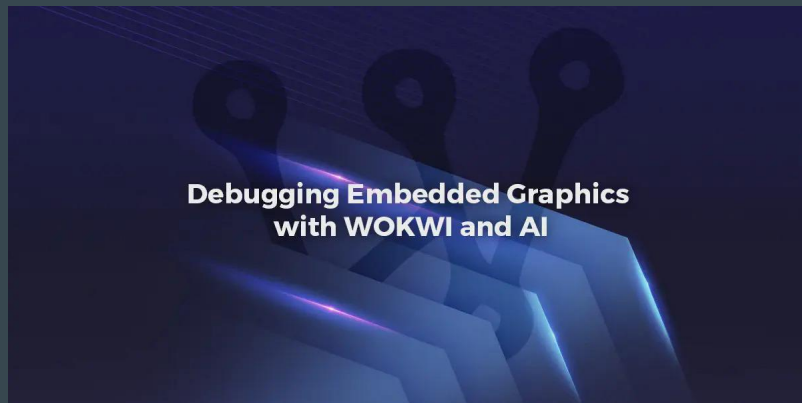
[ESP-IDF SDL3 Component](#)



Raylib

Upstream C library used in ESP-IDF project using git submodule wrapper technique

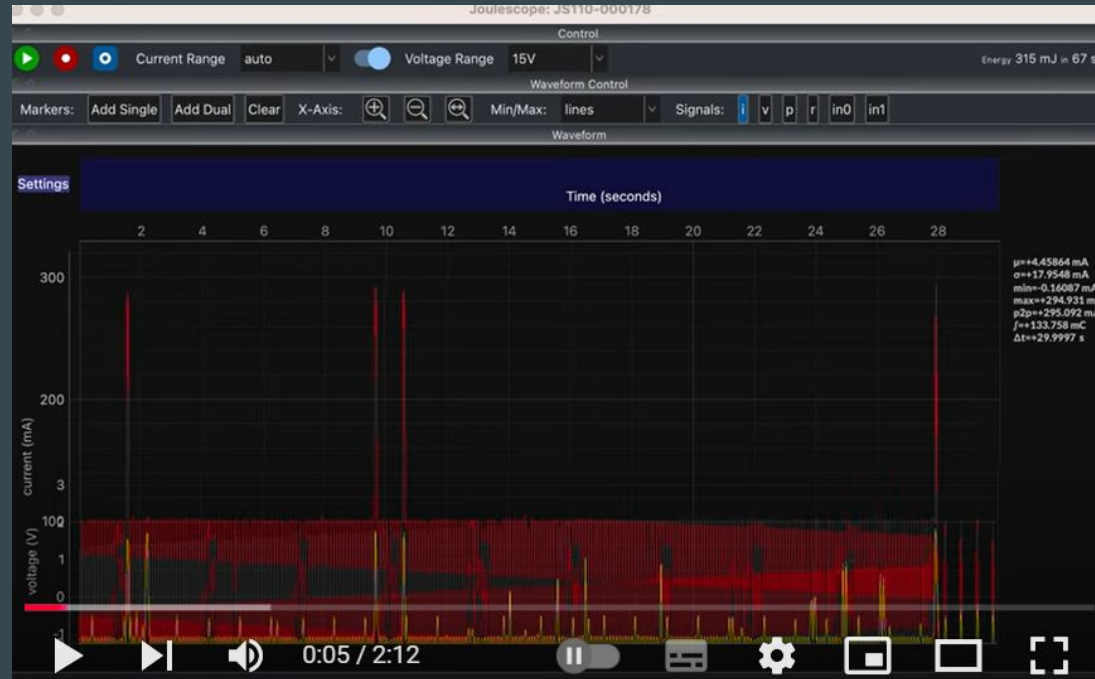
<https://components.espressif.com/components?q=raylib>



<https://developer.espressif.com/blog/2025/10/debugging-embedded-graphics-with-wokwi-and-ai/>

<https://github.com/georgik/esp-idf-component-raylib>

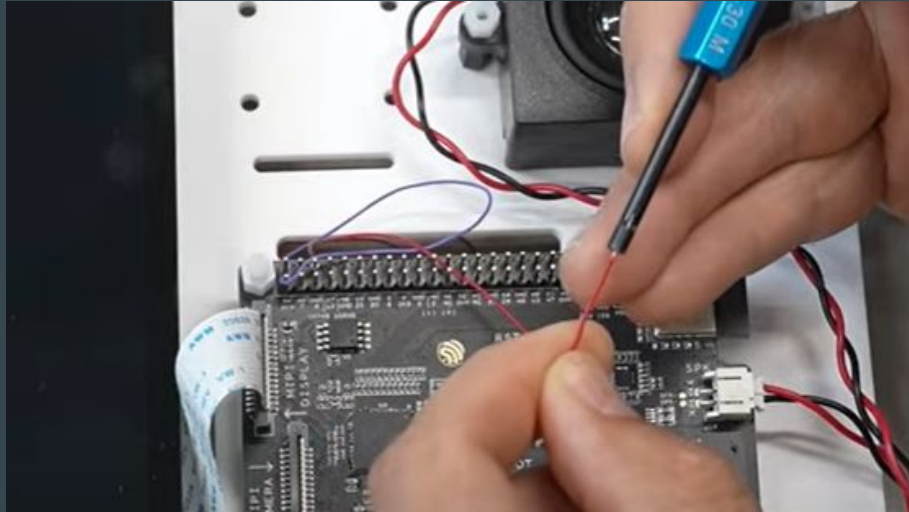
ESP32-C6 TWT



<https://www.youtube.com/watch?v=FA1jqZLig4s>

Tips for Makers

Wire Wrap Tool



Espressif DevCon 2024 - Tips and Tricks

How to “unbrick” ESP32

Many people incorrectly thinks that ESP32 is bricked, in many cases the custom firmware just does not receive UART or USB signals.

Solution: hold BOOT button and press RESET to switch to Download Boot mode

After flashing, just press RESET to return to normal Boot mode.

WiFi Power Saving Modes

ESP32-C6 by default tries to optimize energy usage by default

```
esp_wifi_set_ps(WIFI_PS_MIN_MODEM)
```

```
esp_wifi_set_ps(WIFI_PS_MAX_MODEM)
```

<https://docs.espressif.com/projects/esp-idf/en/latest/esp32c6/api-guides/wifi.html#esp32-c6-wi-fi-power-saving-mode>

Developer Portal

developer.espressif.com

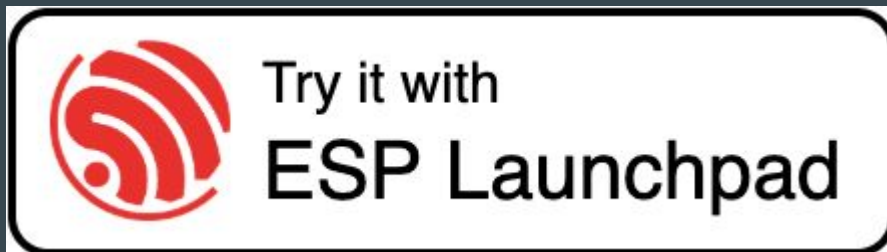
GitHub: <https://github.com/espressif/developer-portal/>

The screenshot displays the Espressif Developer Portal website. At the top, the navigation bar includes the Espressif logo, the text "Developer Portal", and links for "Blog", "Workshops", "Events", "Quick Links", and a search icon. The main content area is titled "Rust" and features a grid of article cards. The first card, "Rust no_std & Bevy ECS", includes an image of an ESP32 board and lists tags: Embedded Systems, ESP32, ESP32-S3, ESP32-C3, Rust, Bevy, No_std, ECS, and WASM. The second card, "esp-hal 1.0.0 beta announcement", features the Espressif logo and tags: ESP32, Rust, Xtensa, RISC-V, and Announcement. The third card, "Simplified Embedded Rust: A Comprehensive Guide to Embedded Rust Development", shows two book covers and tags: Rust, Embedded Systems, ESP32, ESP32-C3, Espressif, Wokwi, Book, and Review. The fourth card, "Rust + Embedded: A Development Power Duo", includes the Espressif logo and tags: Rust, Embedded Systems, ESP32, and Rust Programming Language. The fifth card, "Rust on Espressif chips — 18-10-2021", also features the Espressif logo and tags: ESP32, Rust, and Xtensa.

ESP Launchpad

Flash your firmware from web

<https://github.com/espressif/esp-launchpad>



Espressif Developer Conference 2022-2024 - recording



<https://www.youtube.com/@EspressifSystems>

<https://devcon.espressif.com/>

Embedded World 2026

Meet us in Nuremberg, Germany



embeddedworld

Exhibition&Conference

SPECIAL EDITION

140 Pages

Special edition guest-edited by

ESPRESSIF

Prototyping With Espressif Chips

Try it with ESP Launchpad

ADF, IDF, and Other SDKs

Insights from Espressif Engineers

Automation With Rainmaker and Matter

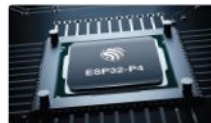
Trying Out the ESP32-S3-BOX-3

ESP32 and ChatGPT

In this issue

- > An Open-Source Speech Recognition Server
- > Power Duct: Rust + Embedded
- > Facial Recognition With ESP32-S3-EYE
- > Walkie-Talkie with ESP-NOW
- > Another Elektor Christmas Tree Project

and much more!



Unleashing the ESP32-P4
The Next Era of Innovative
Microcontrollers

p. 59



Acoustic Fingerprinting
Song Recognition With ESP32

p. 80



A Vision for the AIoT
Interview with Espressif CEO
Teo Swee-Ann

p. 35



9 770932 543006

Elektor Mag

Special Guest Edition

<https://www.elektor.com/products/elektor-special-espessif-guest-edition-2023-pdf-en>

Espressif in Brno

Vlněna Office Park

Espressif Systems (Czech) s.r.o.

Přízova 3, 602 00 Brno

Czechia, Europe

