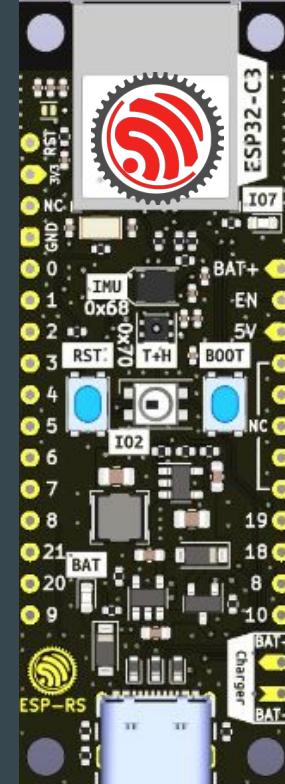




# Rust on ESP32

2025-11-03  
42 Hackathon Challenge

Juraj Michálek  
Espressif Systems



<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](#) [RISCV](#) [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](#).

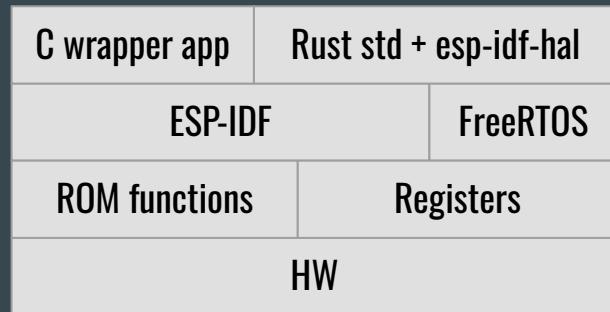
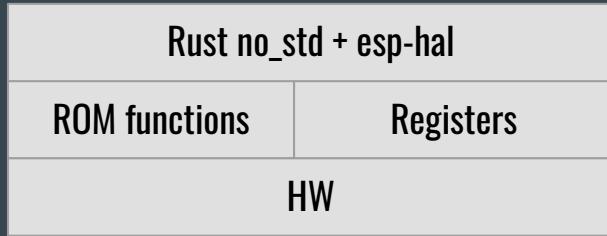
If you are just getting started writing Rust for ESP devices, please first read [The Rust on ESP book](#).

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
<a href="#">esp-rs/esp-hal</a>	<i>Without support for the Rust standard library ( no_std )</i>	Support status <span>Official</span>
<a href="#">esp-rs/esp-idf-hal</a>	<i>With support for the Rust standard library ( std )</i>	Support status <span>Community</span>



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

Official Espressif + Community Support

Requires Rust compiler

Bleeding edge MCUs and drivers might be missing

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

Community support

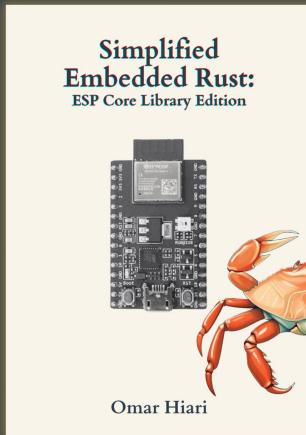
Requires Rust compiler + ESP-IDF

Bleeding edge MCUs and drivers from ESP-IDF

# Training

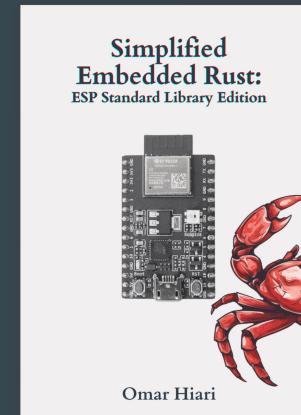
## Rust no\_std

- [https://docs.espressif.com/projects/rust/no\\_std-training/](https://docs.espressif.com/projects/rust/no_std-training/)
- [https://github.com/esp-rs/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>



# 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/>

The screenshot shows the product selector interface for the ESP32-S3. On the left, there's a circular icon with a Wi-Fi signal and the text "ESP32-S3". Below it is a sidebar with links: "Product Brief" (highlighted), "Docs & Certs", and "DevKits". A message states: "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." A "Features" section lists "Core: Xtensa® single-dual 32-bit LX7 CPU, frequency up to 240MHz" and "Memories:". A "Block Diagram" is shown as a complex circuit board layout. At the bottom, there's a table with columns: Index, Name, MPN, Marketing Status, Type, and Wi-Fi. The first five rows are listed, with row 4 being highlighted. Buttons for "Comparison" and "Export" are at the bottom right.

The screenshot shows the product comparison interface. At the top, tabs for "Product Selector" and "Product Comparison" are visible, with "Product Comparison" selected. Below are buttons for "Product Portfolio", "Sales Questions", and "Technical Inquiries". A "Hide Same" and "Show Same" button is also present. The main area compares two products: "ESP32-S3" and "ESP32-P4NRW16". The "Overview" section contains the following data:

	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)	QFN10x10
Dimensions (mm)	7*7	10x10
Temp. (°C)	-40 °C ~ 105 °C	-40 °C ~ 85 °C
Status	Mass Production	Sample
ECO	standard version	
Support IDF	v0.x	

# Quick start with AI at [espressif.com](https://espressif.com)

The screenshot shows a web-based AI interface for Espressif Docs. The main content area displays a guide titled "How to create Hello world for ESP32-C6 in Rust no\_std". The guide explains that 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 espflash.  probe-rs
▶ Flashing, logging and debugging (probe-rs)
▶ Flashing, logging and debugging (espflash)
▶ 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

```
● ● ●                                     Cargo run --release
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-dirty 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=42000000 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=4a5ch (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

The screenshot shows the CLion IDE interface with a Rust project for an esp32c6-hello board. The project structure includes .cargo, devcontainer, .github, src, bin, target, tests, .dockerignore, .gitignore, build.rs, Cargo.lock, Cargo.toml, diagram.json, rust-toolchain.toml, wokwi.toml, External Libraries, and Scratches and Consoles.

The main.rs file contains the following code:

```
async fn main(spawner: Spawner) {
    println!("Hello, ESP32!");
    let timer0: SystemTimer = SystemTimer::new(peripherals.SYSTIMER);
    esp_hal_embassy::init(timer0.alarm0);

    let rng: Rng = esp_hal::rng::Rng::new(peripherals.RNG);
    let timer1: TimerGroup<TIM0> = TimerGroup::new(peripheral);
    let wifi_init: EspWifiController = esp_wifi::init(timer1.time0)
        .expect(msg: "Failed to initialize WIFI/BLE controller");
    let (mut wifi_controller, _interfaces) = esp_wifi::wifi()
        .expect(msg: "Failed to initialize WIFI controller");

    // TODO: Spawn some tasks
    let _ = spawner;

    loop {
        Timer::after(Duration::from_secs(1)).await;
    }
}
```

The Wokwi Simulator window displays the ESP32-C6 board with various pins labeled: TX, RX, MTS, MTDI, MTCK, MTDQ, SDIO\_DATA3, SDIO\_DATA2, SDIO\_DATA1, SDIO\_DATA0, SDIO\_CLK, SDIO\_CMD, NC, GND, RESET, BOOT, and NC. The terminal window shows the build log and the output "Hello, ESP32!".

# Open Source - <https://github.com/espressif>

[!\[\]\(f98fae3e814798c08d139a4c30c9ab3c\_img.jpg\) Overview](#)   [!\[\]\(315a30994a53bdc7c0b3b5d60bd68a8f\_img.jpg\) Repositories 294](#)   [!\[\]\(181a24b40354f1ba57a7fb3f53b75d85\_img.jpg\) Projects 5](#)   [!\[\]\(abdaa36b47fa525142932abbc2512e5b\_img.jpg\) Packages](#)   [!\[\]\(3c06aedd4ec46a7f04c955a640190357\_img.jpg\) Teams 15](#)   [!\[\]\(c6b316602b034bd96f293b8d58b50879\_img.jpg\) People 34](#)   [!\[\]\(fe43f63f9d39ec17f3a86c57307cfd93\_img.jpg\) Sponsoring 2](#)

---



## Espressif Systems

[Verified](#)   [Sponsor](#)

 6.3k followers    Shanghai, China    <http://espressif.com>

---

[README.md](#)

## Welcome to Espressif's site on GitHub

---

Espressif supports a large variety of open-source projects, including SDKs, components, libraries, solutions, and tools, which aim to help developers bring their projects to life.

All of Espressif's official software, relating to the various series of ESP SoCs including ESP32 and ESP8266, are available on this GitHub site. To check out all the series of SoCs from Espressif, please visit our [ESP Product Selector](#).

Below you can find a selection of Espressif's open-source projects. Our full repository list can be found [here](#).

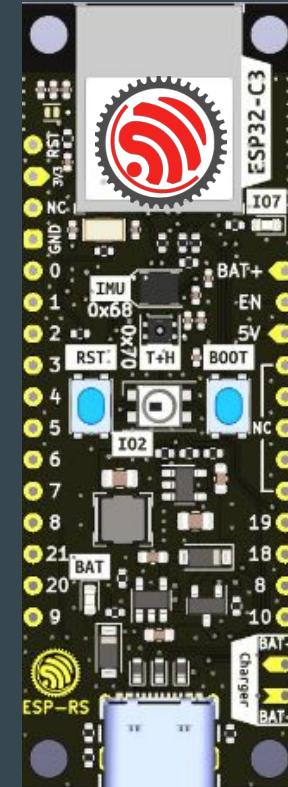
# Open Hardware - esp-rust-board

KiCad templates

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

ESP32-C3-DevKit-RUST-1

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





Slint

Quick prototyping

Free and commercial version



# Example: List of WiFi networks

Rust std

Rust no\_std

Rust no\_std + Embassy

<https://github.com/georgik/esp32-list-wifi-networks/>

# 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-spooky-maze-game>



# 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>

# 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](#).

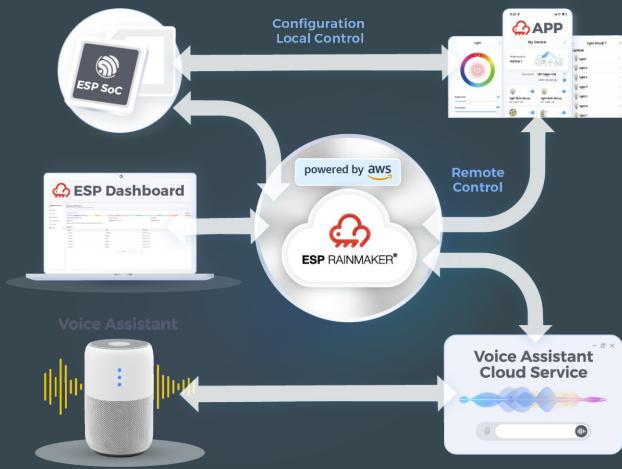
Scales from bare-metal MCUs with 1MB flash and 256KB RAM to ARM embedded Linux and bigger iron!

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>

# rainmaker-rs



## Rust Implementation of ESP Rainmaker

A cross-platform implementation of ESP Rainmaker for ESP32 products and Linux using Rust.

- ESP RainMaker is end-to-end IoT development platform which enables development of IoT applications which can be controlled remotely.
- However, the C based ESP RainMaker SDK(which can be found [here](#)) only supports execution on Espressif's ESP32 SOCs.
- This crate tries to implement similar functionalities for Linux platform along with ESP32(which can further be extended to other microcontrollers).

### What is working

- WiFi Provisioning\*: Providing WiFi for a new device. No need to hardcode WiFi credentials!
- Remote Control: Controlling Devices connected to a node over internet using phone application.
- User Node Association: Associating a specific node to a user account for control.
- Device sharing: Share access to a device with multiple members.
- Control using Home Assistants: RainMaker devices can be added to and controlled using Amazon Alexa / Google Home. More details [here](#)

\* Currently only supported on ESP32

<https://github.com/rainmaker-rs/rainmaker>

# 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/>



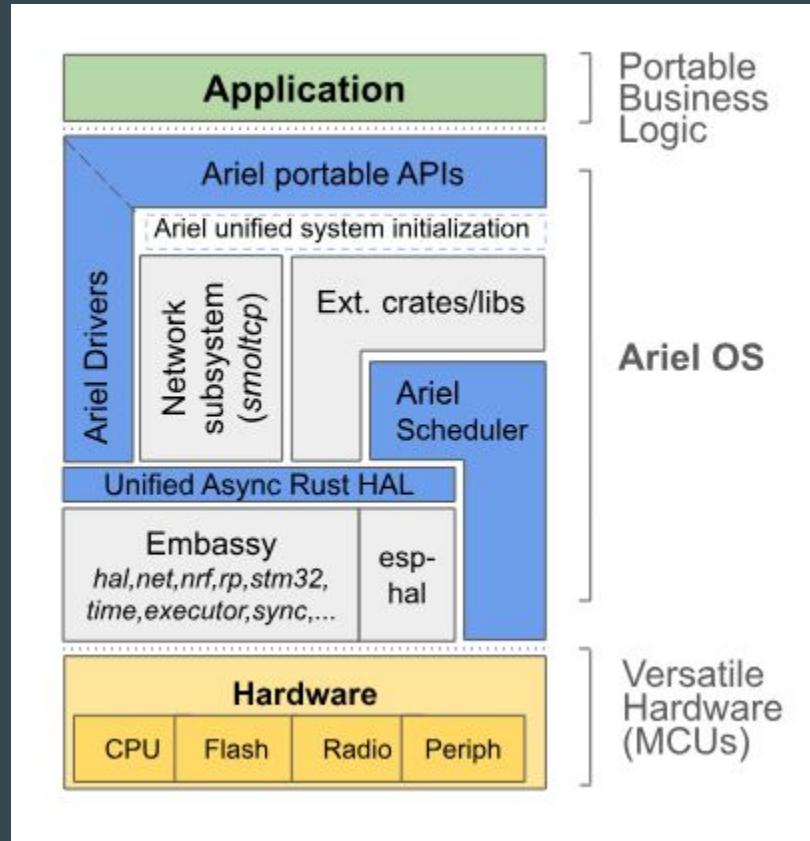
Zephyr - <https://zephyrproject.org/>

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



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

# Ariel OS



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

# IDE for Rust 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/>



# Developer Portal

[developer.espressif.com](https://developer.espressif.com)

The screenshot shows the Espressif Developer Portal's video library section for the 'Rust' category. The interface has a dark header with the Espressif logo and navigation links for Blog, Workshops, Events, Quick Links, and a search bar. Below the header, there are several video thumbnail cards arranged in a grid:

- Rust**: A thumbnail featuring a microcontroller board with a colorful display showing a UI.
- Bevy Entity Component System on ESP32 with Rust no\_std**: An announcement video from April 2025.
- esp-hal 1.0.0 beta announcement**: A video from February 2025.
- Simplified Embedded Rust: A Comprehensive Guide to Embedded Rust Development**: A guide video from June 2024.
- Rust + Embedded: A Development Power Duo**: A video from April 2023.
- Rust on Espressif chips — 18-10-2021**: A video from October 2021.

Each card includes a title, a brief description, a publish date, a duration, and a list of tags at the bottom. The cards are styled with rounded corners and shadows.

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

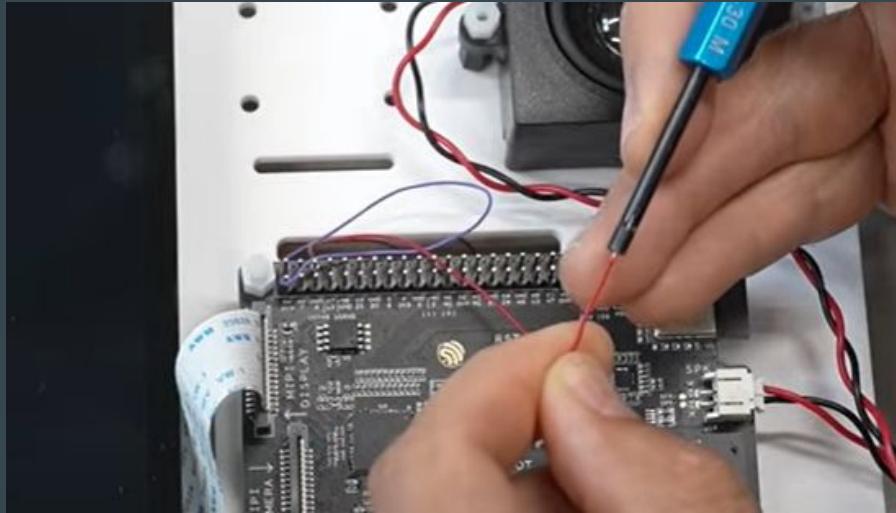
# How to “unbrick” ESP32

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 boot mode**

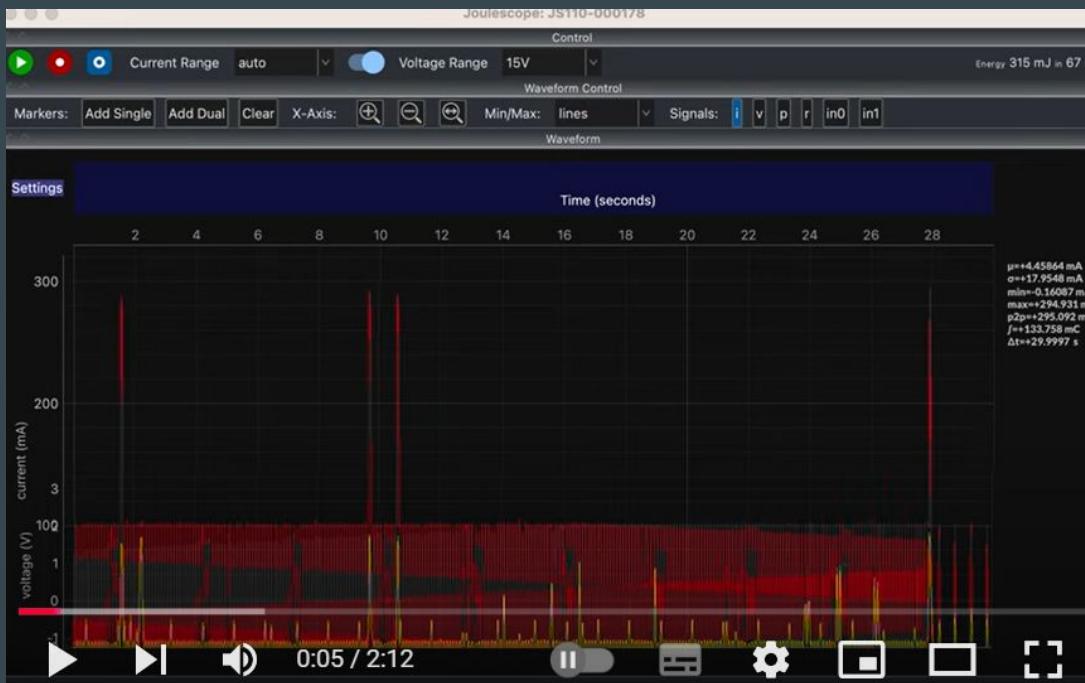
After flashing, just press RESET to return to normal mode.

# Wire Wrap Tool



Espressif DevCon 2024 - Tips and Tricks

# ESP32-C6 TWT



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

# Espressif Developer Conference 2022-2024 - recording



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

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

# Embedded World 2026

Meet us in Nuremberg, Germany



Special edition  
guest-edited by



ESPRESSIF



Insights from  
Espressif  
Engineers

Automation  
With Rainmaker  
and Matter

Trying Out the  
ESP32-S3-BOX-3



Unleashing the ESP32-P4  
The Next Era of Innovative  
Microcontrollers



p. 59  
Acoustic Fingerprinting  
Song Recognition With ESP32



p. 60  
A Vision for the IoT  
Interview with Espressif CEO  
Teo Swee-Ann

140  
Pages

Prototyping With Espressif Chips



ESP32 and ChatGPT

In this issue

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

and much more!

# Elektor Mag

## Special Guest Edition

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

01

9 7093256006

# Espressif in Brno

Vlněna Office Park

Espressif Systems (Czech) s.r.o.

Přízova 3, 602 00 Brno

Czechia, Europe

