



Prototyping and Product Development with Rust on ESP32

2025-07-16

Rust on Embedded - Elektor Academy Pro Conference

Juraj Michálek
Espressif Systems



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

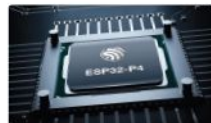
Trying Out the ESP32-S3-BOX-3

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>

Topics for today

Open source - GH org espressif & esp-rs

Bootstrapping Rust no_std

Comparison of Rust std vs. no_std

Slint UI, Embedded Graphics, Bevy ECS

rs-matter, rainmaker-rs

Integration of Rust and OSes including MicroPython

Tips and tricks

Espressif Systems

Who We Are



As a public, multinational, fabless semiconductor company (Espressif Systems, SSE: 688018.SH), we develop cutting-edge, low-power wireless communication chipsets. Our AIoT solutions are green, versatile, and cost-effective. We have a closed-loop development cycle for core technologies, including Wi-Fi & Bluetooth LE & IEEE 802.15.4 protocols, RF, RISC-V MCUs, AI algorithms, operating systems, toolchains, AIoT frameworks, and Cloud services.



1 Billion+
Global IoT SoC
Shipments



Global Leader
In Wi-Fi MCU Market



200+ IPs
In AIoT Technology



10,000+
Satisfied Customers
Worldwide



2 Million+
Active Ecosystem
Followers and
Developers



Supports:

- Open Source
- Open Hardware

Open Source

[Overview](#) [Repositories 294](#) [Projects 5](#) [Packages](#) [Teams 15](#) [People 34](#) [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 Source - 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

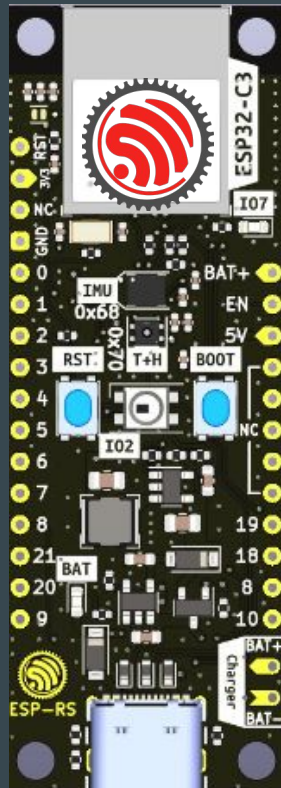
Open Hardware - esp-rust-board

KiCad templates

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

ESP32-C3-DevKit-RUST-1

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



Quick start with AI

 **ESPRESSIF**

Hardware SDKs Cloud Solutions Support Ecosystem Company Contact

Q English ^ Subscribe

ESP32-C5

Now in Mass Production

2.4 and 5 GHz dual-band Wi-Fi, Bluetooth 5 (LE) and 802.15.4 connectivity

Learn More

Espressif DevCon25

Build the Future of Connectivity Together

Espressif Docs AI

How to create Hello world with Rust on ESP32-C6

To create a "Hello world" application with Rust on the ESP32-C6, follow these steps:

1. Install Rust Toolchain and Set Up for ESP32-C6

- Install the Rust toolchain with the `rust-src` component (nightly is required for `std` applications):

```
rustup toolchain install nightly --component rust-src
```
- Add the appropriate target for ESP32-C6 (for `no_std` use):

```
rustup target add riscv32imac-unknown-none-elf
```

For `std` applications, use the `riscv32imac-esp-idf` target, but nothing needs to be installed via `rustup` for this Tier 3 target. Refer to the [Rust on ESP Book](#) for more details.

2. Generate a New Project

- Install `cargo-generate` :

```
cargo install cargo-gener
```
- Use the ESP-IDF template to generate a new project:

Ask a follow-up

powered by kapa.ai

protected by hCaptcha

esp-generate --chip esp32c6

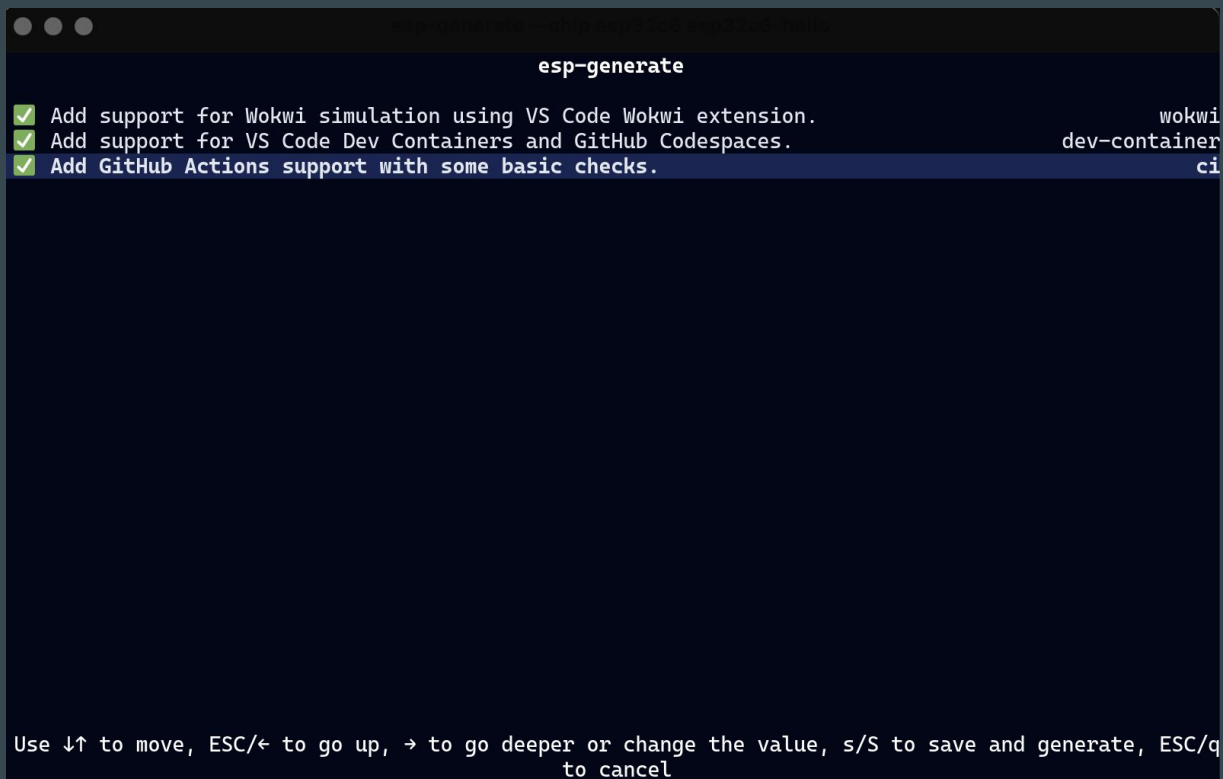
```
esp-generate --chip esp32c6 esp32c6-hello

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"
```

wokwi-cli

<https://docs.wokwi.com/wokwi-ci/cli-installation>

- `curl -L https://wokwi.com/ci/install.sh | sh`
- `iwr https://wokwi.com/ci/install.ps1 -useb | iex`

Run simulation: `wokwi-cli`

Setup inside existing project: `wokwi-cli init`

CI suppor: <https://docs.wokwi.com/wokwi-ci/getting-started>

CLion with Rust plugin or RustRover

The screenshot displays the CLion IDE interface with a Rust project named 'esp32c6-hello'. The main.rs file is open, showing the following code:

```
30 async fn main(spawner: Spawner) {
31
32
33
34
35
36
37
38
39     println!("Hello, ESP32!");
40     let timer0 : SystemTimer = SystemTimer::new(peripherals.SYS_TIMER);
41     esp_hal_embassy::init(timer0.alarm0);
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

The Wokwi Simulator window shows a virtual ESP32-C6 board with various pins connected. The bottom status bar shows the current file path and encoding.

Build output:

```
Build: Sep 19 2022
rst:0x1 (POWERON),boot:0x5c (SPI_FAST_FLASH_BOOT)
SPIWP:0xee
mode:DI0, clock div:2
load:0x40875720, len:0x126c
load:0x4086c410, len:0x704
load:0x4086e610, len:0x2ae0
entry 0x4086c410
Hello, ESP32!
```

esp32c6-hello > src > bin > main.rs > main()

Auto-reload on build

The screenshot displays an IDE environment for developing and simulating an ESP32-C6 project. The main editor shows the `main.rs` file with the following Rust code:

```
30 async fn main(spawner: Spawner) {
31     let config: Config = esp_hal::Config::default().with_cpu_clock(
32         CpuClock::MHz(240));
33     let peripherals: Peripherals = esp_hal::init(config);
34
35     esp_alloc::heap_allocator!(size: 64 * 1024);
36
37     println!("Hello, Elektor!");
38
39     let timer0: SystemTimer = SystemTimer::new(peripherals.SYSTEM_TIMER);
40     esp_hal_embassy::init(timer0.alarm0);
41
42     let rng: Rng = esp_hal::rng::Rng::new(peripherals.RNG);
43     let timer1: TimerGroup<TIMGO> = TimerGroup::new(peripherals.TIMGO, rng);
44     let wifi_init: EspWifiController = esp_wifi::init(timer1.timer0);
45     .expect(msg: "Failed to initialize WIFI/BLE controller");
46     let (mut _wifi_controller, _interfaces) = esp_wifi::wifi_init(
47         _wifi_controller, _interfaces);
48     .expect(msg: "Failed to initialize WIFI controller");
49
50     // TODO: Spawn some tasks
51     let _ = spawner;
52 }
```

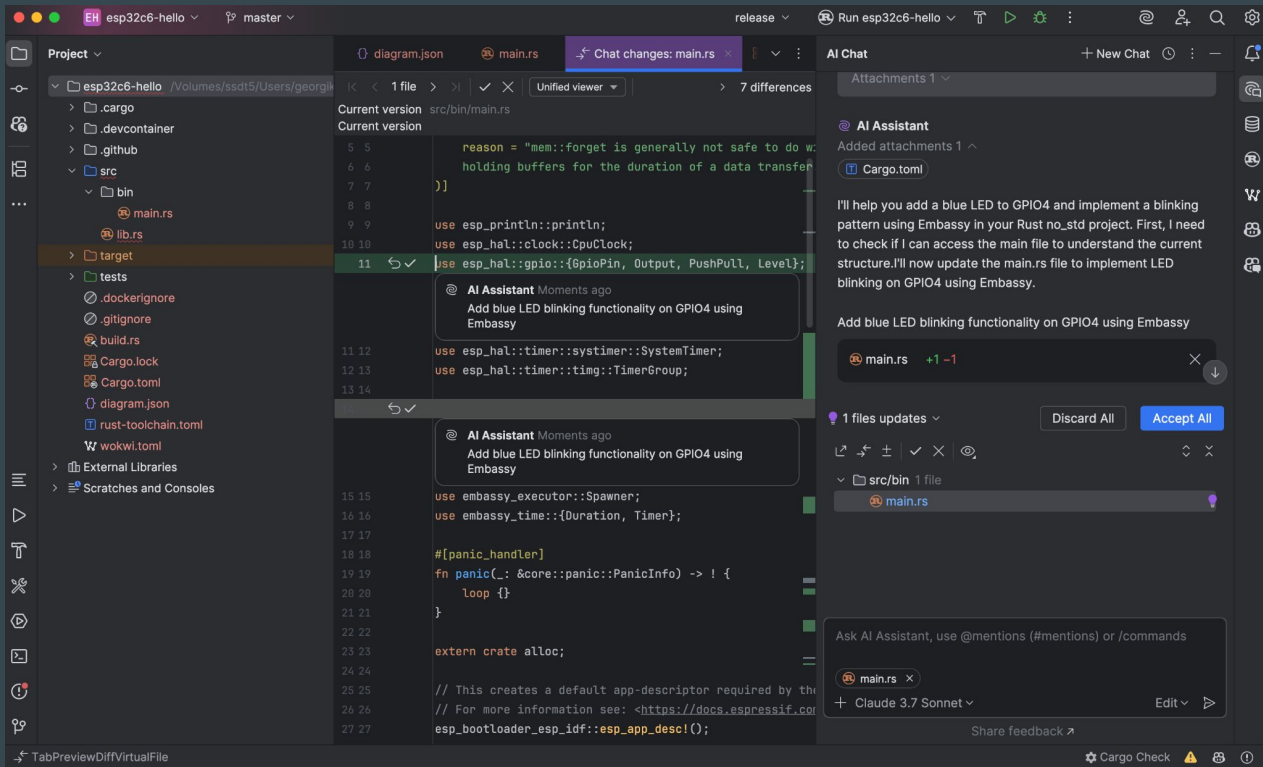
The Wokwi Simulator window on the right shows a virtual ESP32-C6 board. The top right of the simulator displays the user name "Juraj Michálek" and the license "Wokwi Pro License". The simulation time is 00:57.655 and the CPU usage is 99%.

The terminal window at the bottom shows the output of the build process:

```
Build: Sep 17 2022
rst:0x1 (POWERON),boot:0x5c (SPI_FAST_FLASH_BOOT)
SPIWP:0xee
mode:DIO, clock div:2
load:0x40875720,len:0x126c
load:0x4086c410,len:0x704
load:0x4086e610,len:0x2ae0
entry 0x4086c410
Hello, Elektor!
```

The status bar at the bottom indicates the current file is `main.rs` in the `src/bin` directory, and the function being executed is `main()`. It also shows the Cargo Check status, file encoding (UTF-8), and other IDE settings.

Extending with AI



Caveat

Older models tends to use older esp-hal.

Requires iteration and prompt correction.

esp32c6-hello master

Project

esp32c6-hello

.cargo

.devcontainer

.github

src

bin

main.rs

lib.rs

target

tests

.dockerignore

.gitignore

build.rs

Cargo.lock

Cargo.toml

diagram.json

rust-toolchain.toml

wokwi.toml

External Libraries

Scratches and Consoles

Run Wokwi Simulator

LED ON

LED OFF

LED ON

LED ON

LED OFF

LED ON

LED OFF

LED ON

LED OFF

LED ON

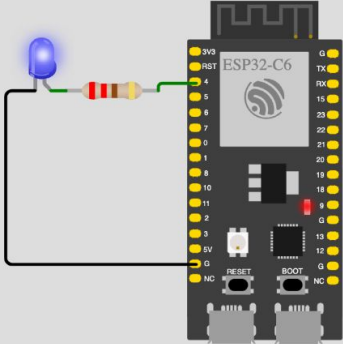
wokwi.toml diagram.json main.rs Cargo.toml

```
4      "parts": [  
19        {  
24          "attrs": { "value": "220" },  
25        },  
26      ],  
27      "connections": [  
28        [ "esp:TX", "$serialMonitor:RX", "", [] ],  
29        [ "esp:RX", "$serialMonitor:TX", "", [] ],  
30        [ "led1:C", "esp:GND.1", "black", [ "h-18.8", "v129.7" ],  
31        [ "led1:A", "r1:1", "green", [ "v0" ] ],  
32        [ "r1:2", "esp:4", "green", [ "v0" ] ]  
33      ],  
34      "serialMonitor": {  
35        "display": "terminal",  
36        "convertEol": true  
37      }  
38    }  
39  ]
```

Wokwi Simulator

Juraj Michálek
Wokwi Pro License

00:14.722 99%



esp32c6-hello diagram.json

Cargo Check 34:5 LF UTF-8 4 spaces* No JSON schema

Deep dive

**There are two ways
how to write code for
MCUs.**

Rust no_std + esp-hal	
ROM functions	Registers
HW	

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

Rust no_std + esp-hal	
ROM functions	Registers
HW	

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

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

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

Rust no_std + esp-hal	
ROM functions	Registers
HW	

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

Official Espressif + Community Support

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

Rust no_std + esp-hal	
ROM functions	Registers
HW	

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

Official Espressif + Community Support

Requires Rust compiler

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

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

Training

Rust no_std

- https://docs.espressif.com/projects/rust/no_std-training/
- https://github.com/esp-rs/no_std-training

Rust std

- <https://docs.esp-rs.org/std-training/>
- <https://github.com/esp-rs/std-training>
- Created in cooperation with [Ferrous Systems](https://ferrous-systems.com)



Ferrous Systems

Rust knowledge. Collected.

Verified

 539 followers  Worldwide  <https://ferrous-systems.com>

Developer Portal

developer.espressif.com

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

 **ESPRESSIF** Developer Portal

Blog Workshops Events Quick Links 

Rust



Rust no_std & Bevy ECS

Bevy Entity Component System on ESP32 with Rust no_std

9 April 2025 · 6 mins

[Embedded Systems](#) [ESP32](#) [ESP32-S3](#) [ESP32-C3](#) [Rust](#) [Bevy](#) [No_std](#) [ECS](#) [WASM](#)



esp-hal 1.0.0 beta announcement

24 February 2025 · 11 mins

[ESP32](#) [Rust](#) [Xtensa](#) [RISC-V](#) [Announcement](#)



Simplified Embedded Rust: A Comprehensive Guide to Embedded Rust Development

7 June 2024 · 3 mins

[Rust](#) [Embedded Systems](#) [ESP32](#) [ESP32-C3](#) [Espressif](#) [Wokwi](#) [Book](#) [Review](#)

 **ESPRESSIF**

Rust + Embedded: A Development Power Duo

19 April 2023 · 11 mins

[Rust](#) [Embedded Systems](#) [ESP32](#) [Rust Programming Language](#)

 **ESPRESSIF**



esp-hal 1.0.0 beta announcement

24 February 2025 · 11 mins · 📄

ESP32 Rust Xtensa RISC-V Announcement

AUTHOR

Scott Mabin

We're excited to announce the **1.0.0-beta.0** release for **esp-hal**, the *first* vendor-backed Rust SDK! This has been a nearly 6 year long process to get to this point, and we now feel we have a solid 1.0 strategy.

Let us take you on the journey of how we got here. If you're just interested in what we're stabilizing today, [click here](#) to jump to it.

Where it All Started

Espressif's Official Support for Rust

Bringing Up the Ecosystem

Focusing on **no_std** Crates

Targeting Stability

<https://developer.espressif.com/blog/2025/02/rust-esp-hal-beta/>

**There are two
architectures
on Espressif MCUs**

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	



Slint

Quick prototyping

Free and commercial version

Rust, C++



Printer MCU demo

Build for no_std:

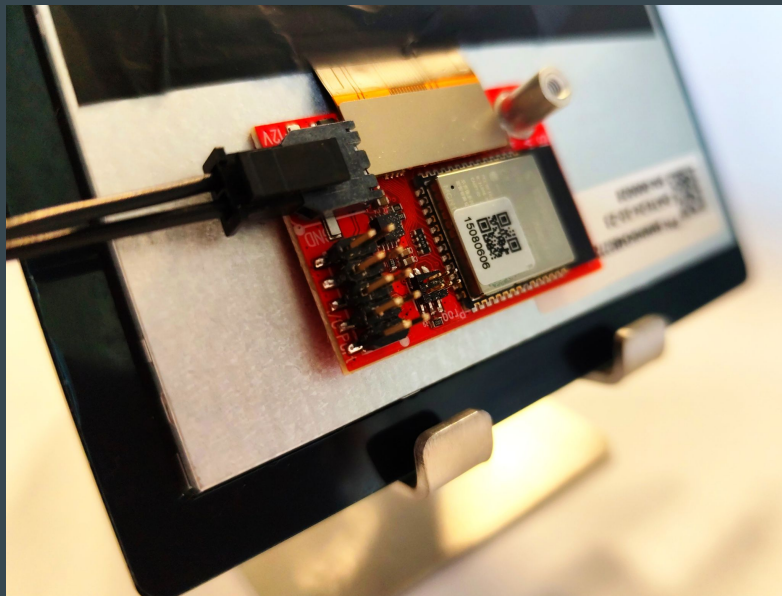
```
CARGO_PROFILE_RELEASE_OPT_LEVEL=s cargo +esp run -p printerdemo_mcu --target  
xtensa-esp32s3-none-elf --no-default-features --features=mcu-board-support/esp32-s3-box-3  
--release --config examples/mcu-board-support/esp32_s3_box_3/cargo-config.toml
```

Monitoring: espflash monitor

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.

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 SOC's.
- 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>

Exposing Rust to Python REPL

<https://github.com/georgik/micropython/commits/experimenta/nmea/>

Rust no_std code - as ESP-IDF component (library)

Bridge code - similar like exposing API from C

Sample code:

- https://github.com/georgik/esp32-conways-game-of-life-python/blob/main/rust-no_std-nmea/nmea.py

NMEA altitude: 61.70

Data:

- **Latitude: 53°21.6801' N**

- **Longitude: -7°29.6628' W**

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/>



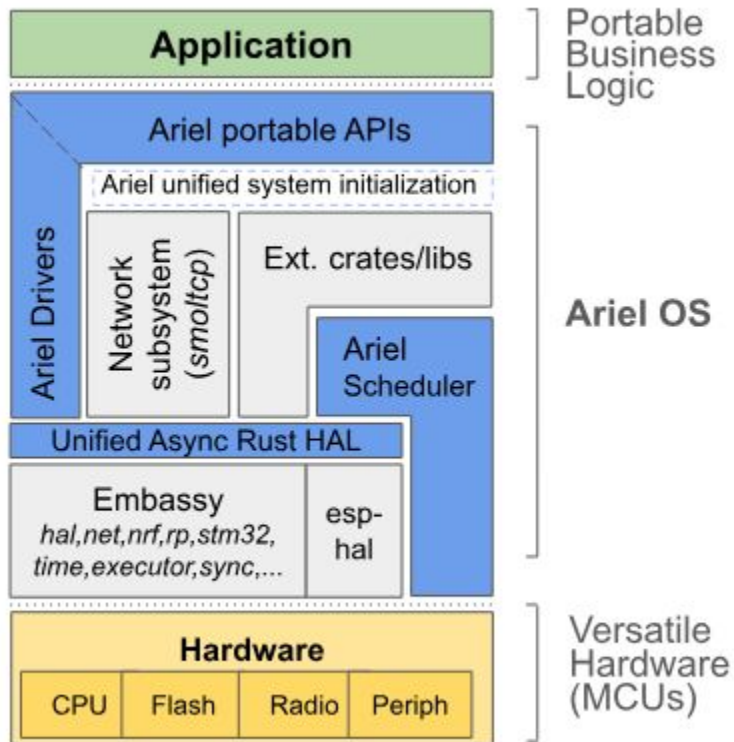
Zephyr - <https://zephyrproject.org/>

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/>



The **ESP** Component Registry

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



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-P4 ESP32-S2 ESP32-S3

Featured

espressif/mdns

v1.4.0

uploaded 2 months ago

mDNS

lvgl/lvgl

v9.2.0

uploaded 1 month ago

LVGL - Light and Versatile Graphics Library

espressif/esp-modbus

v1.0.16

uploaded 1 month ago

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

joltwallet/littlefs

v1.14.8

uploaded 3 months ago

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

espressif/arduino-esp32

v3.0.7

uploaded 23 hours ago

Arduino core for ESP32, ESP32-S and ESP32-C series of SoCs

espressif/openai

v1.0.0

uploaded 5 months ago

OpenAI library compatible with ESP-IDF

wolfssl/wolfssl

v5.7.2

uploaded 3 months ago

wolfSSL Embedded SSL/TLS Library

slint/slint

v1.8.0

uploaded 1 month ago

Slint — declarative GUI toolkit

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

Example: [https://components.espressif.com/components/espressif/i2c bus/](https://components.espressif.com/components/espressif/i2c_bus/)

Tips

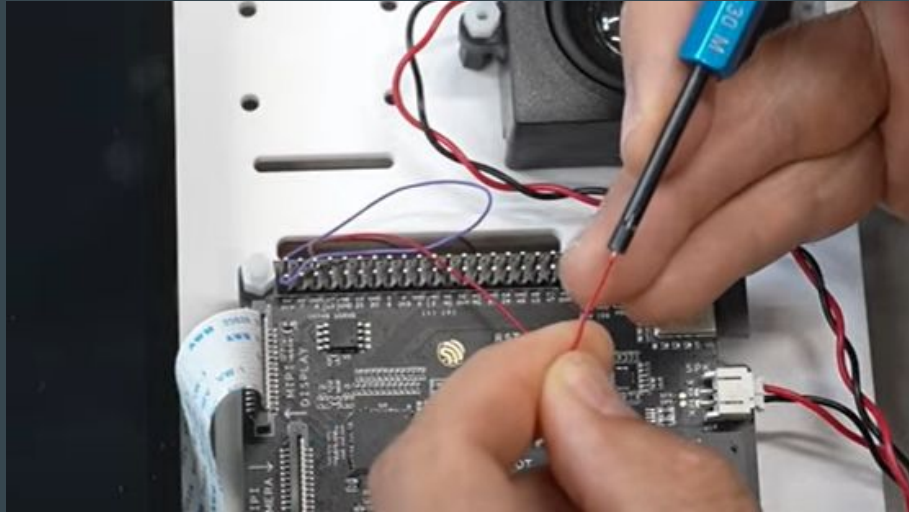
How to “unbrick” ESP32

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

Architecture

Layers:

Application + UI

Graphical library

ESP-BSP (Board Support Package)

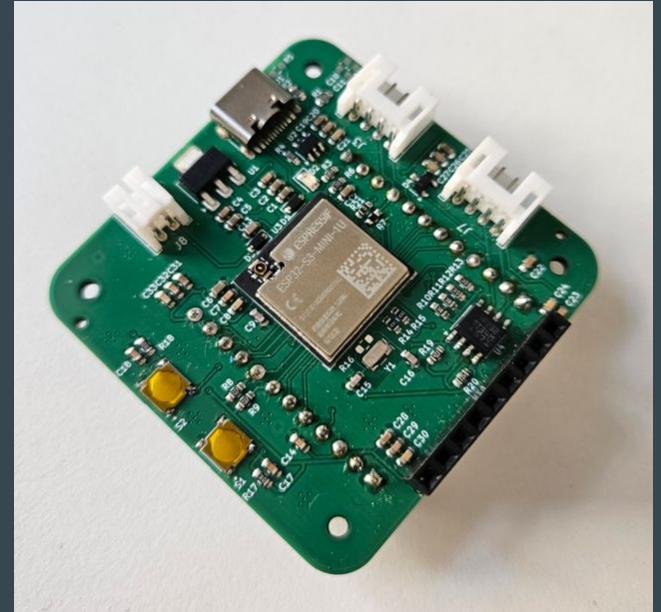
ESP-IDF

FRI STACK

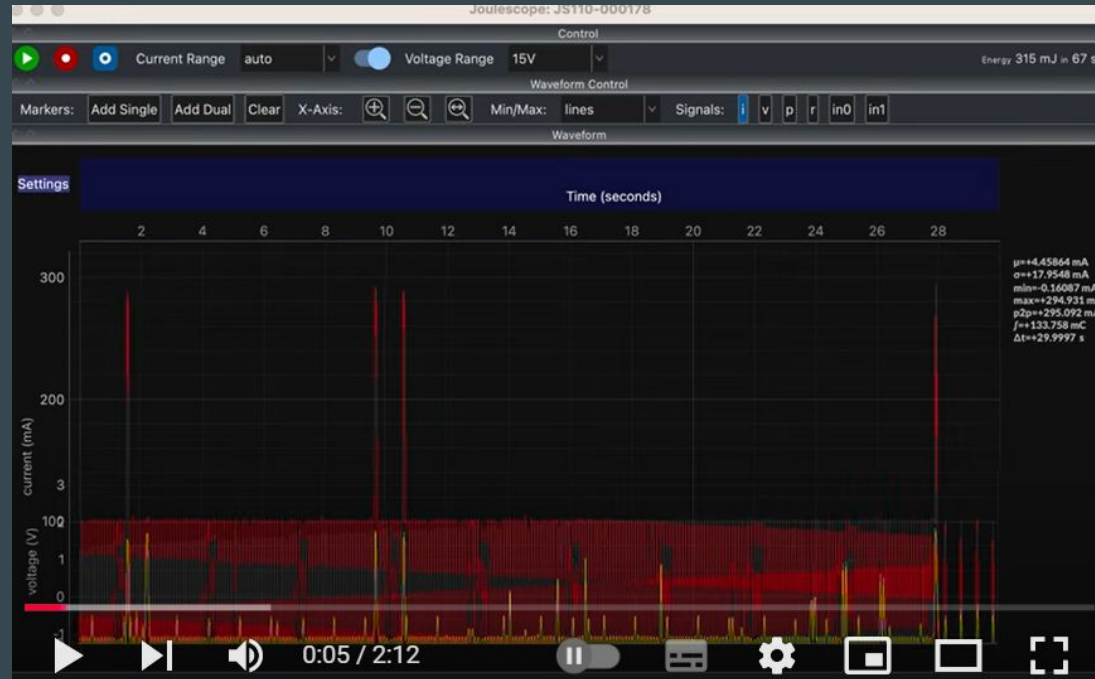
M5Stack compatible Open Hardware stack

Developed by Technical Cybernetics UNIZA

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



ESP32-C6 TWT



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

Demo: Digital Twins - IoTcraft

Example of combining real and virtual world.

ESP32-C6 observing state of virtual lamp and syncing the state to real Neopixel.

- Access Point + DHCP + NAT on M5Stack AtomS3 using ESP-IDF
 - <https://github.com/georgik/esp32-dhcp-server>
- IoTcraft
 - <https://github.com/georgik/iotcraft>

Espressif in Brno

Vlněna Office Park

Espressif Systems (Czech) s.r.o.

Přízova 3, 602 00 Brno

Czechia, Europe



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

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>

