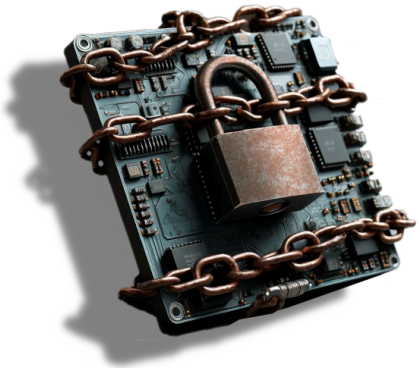


Rust + async ♥ embedded

Embedded Rust mit Embassy

Joël Schulz-Andres <joel@systemscape.de>, Julian Dickert <julian@systemscape.de>

2025-12-01

Agenda

Let's Hack

Part 1: Warum eigentlich Rust?

Memory ≠ Memory

Stack, Heap and Friends

Stack

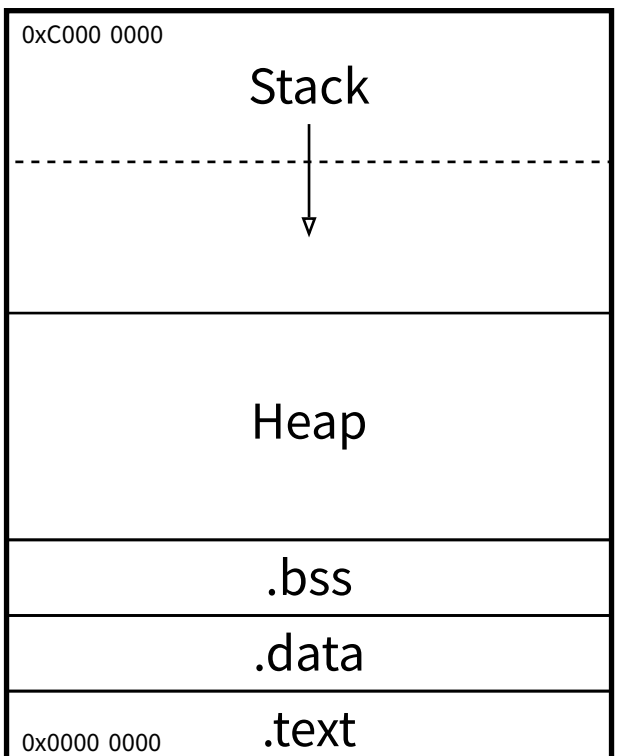
- Wächst in Richtung niedrigerer Speicheradressen
- Lokale Variablen, gespeicherte Register

Heap

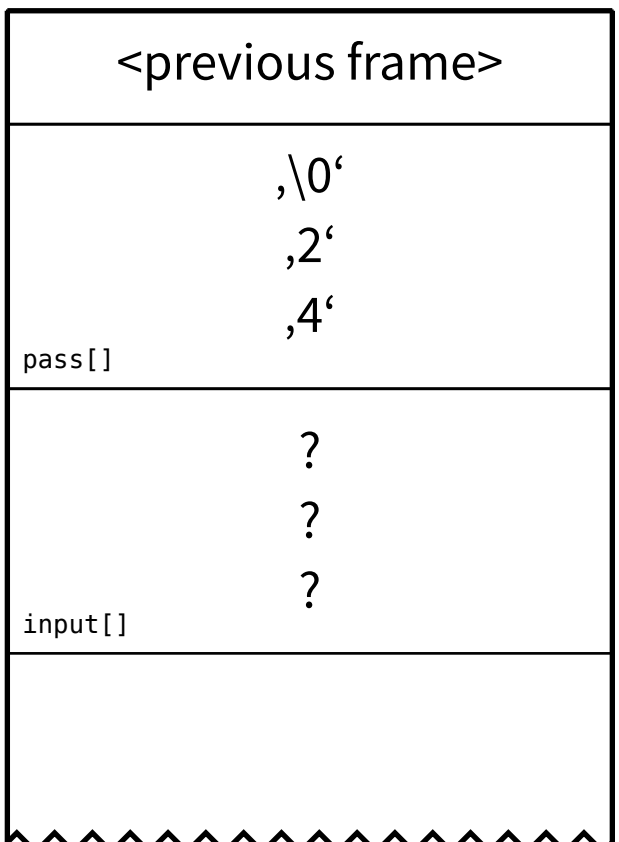
- Größe vorab festgelegt, i.d.R. deutlich größer als Stack
- Dynamische Daten

Friends

Programmcode, statische/uninitialiisierte Daten



Stack

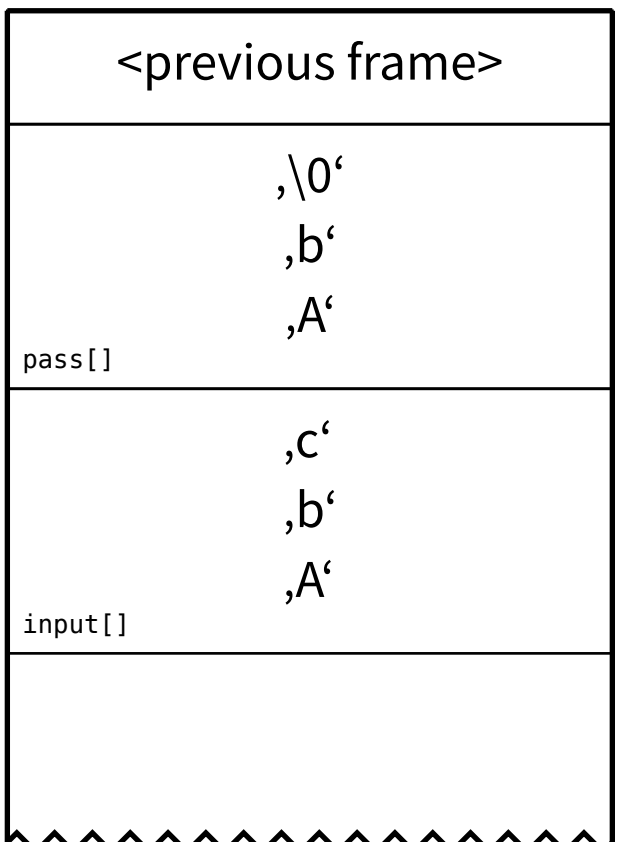



```
1  void authenticate() {
2    char pass[] = "42";
3    char input[3] = {0}; // 2 chars + '\0' = 3
4
5    gets(input);
6
7    if (memcmp(input, pass, strlen(pass)) == 0) {
8      printf("Authenticated!");
9    } else {
10     printf("Wrong password!");
11   }
12 }
```


Dann geben wir Mal ein Passwort ein!

Zum Beispiel: „AbcAb“

Wir haben den Stack zerstört...

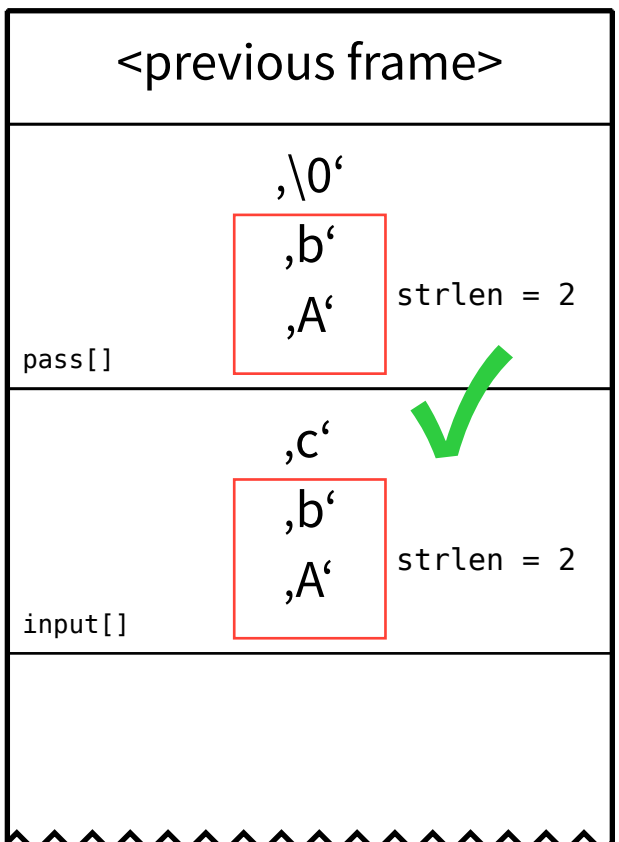


… indem wir ein zu langes Passwort eingegeben haben

```c
char pass[] = "42";
char input[3] = {0}; // 2 chars + '\0' = 3

gets(input);
```


Passwort Überprüfung




```rust
if (memcmp(input, pass, strlen(pass)) == 0) {
  printf("Authenticated!");
} else {
  printf("Wrong password!");
}
```

Authenticated!

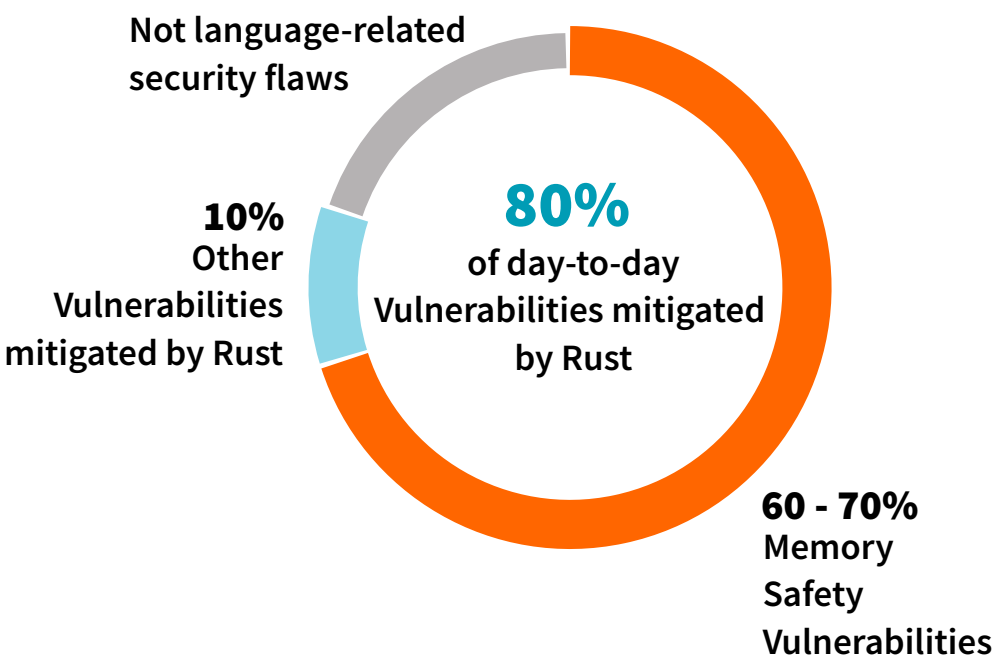
→ C ist sehr alt und, aus Speichersicht, sehr **unsicher**

Einführung

Part 2: Gründe für die Entstehung von Rust

Unsicherheit ist leider Alltag

- **Microsoft:** ca. 70 % der Common Vulnerabilities and Exposures (CVEs) sind Memory-Safety-Vulnerabilities (MSVs), basierend auf CVEs von 2006-2018
- **Google Chromium:** ebenfalls ca. 70% der identifizierten Schwachstellen sind MSVs
- **Mozilla:** 32 von 34 kritischen/schweren Fehlern waren MSVs
- **Google Project Zero:** 67% der Zero-Day-Schwachstellen im Jahr 2021 waren MSVs



Source: https://www.cisa.gov/case-memory-safe-roadmaps

Die Vergangenheit eingebetteter Systeme

Ursache

- C: Werkzeug aus dem letzten Jahrtausend, unkomfortabel
- Erfordert jahrelange Erfahrung für passablen Code → Thema: Fachkräftemangel

Problem

- Unsichere und unzuverlässige Systeme

Source: https://cwe.mitre.org/top25/archive/2023/2023_kev_list.html

Die Vergangenheit eingebetteter Systeme

Ursache

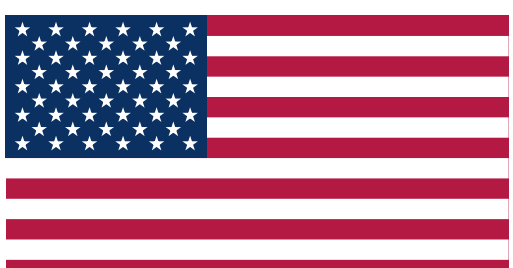
- C: Werkzeug aus dem letzten Jahrtausend, unkomfortabel
- Erfordert jahrelange Erfahrung für passablen Code → Thema: Fachkräftemangel

Problem

- Unsichere und unzuverlässige Systeme

Folge: Gesetzgeber werden aktiv

- EU Cyber Resilience Act
- EU Maschinenverordnung
- Weißes Haus: Drängen auf Nutzung sicherer Programmiersprachen



Source: https://cwe.mitre.org/top25/archive/2023/2023_kev_list.html

"The clear north star for security against memory corruption exploits is the broad usage of Memory Safe Languages [...] Anything less is playing the whack-a-mole of exploit mitigation."

Die Zukunft eingebetteter Systeme

Rust: *„A language empowering everyone to build reliable and efficient software“*

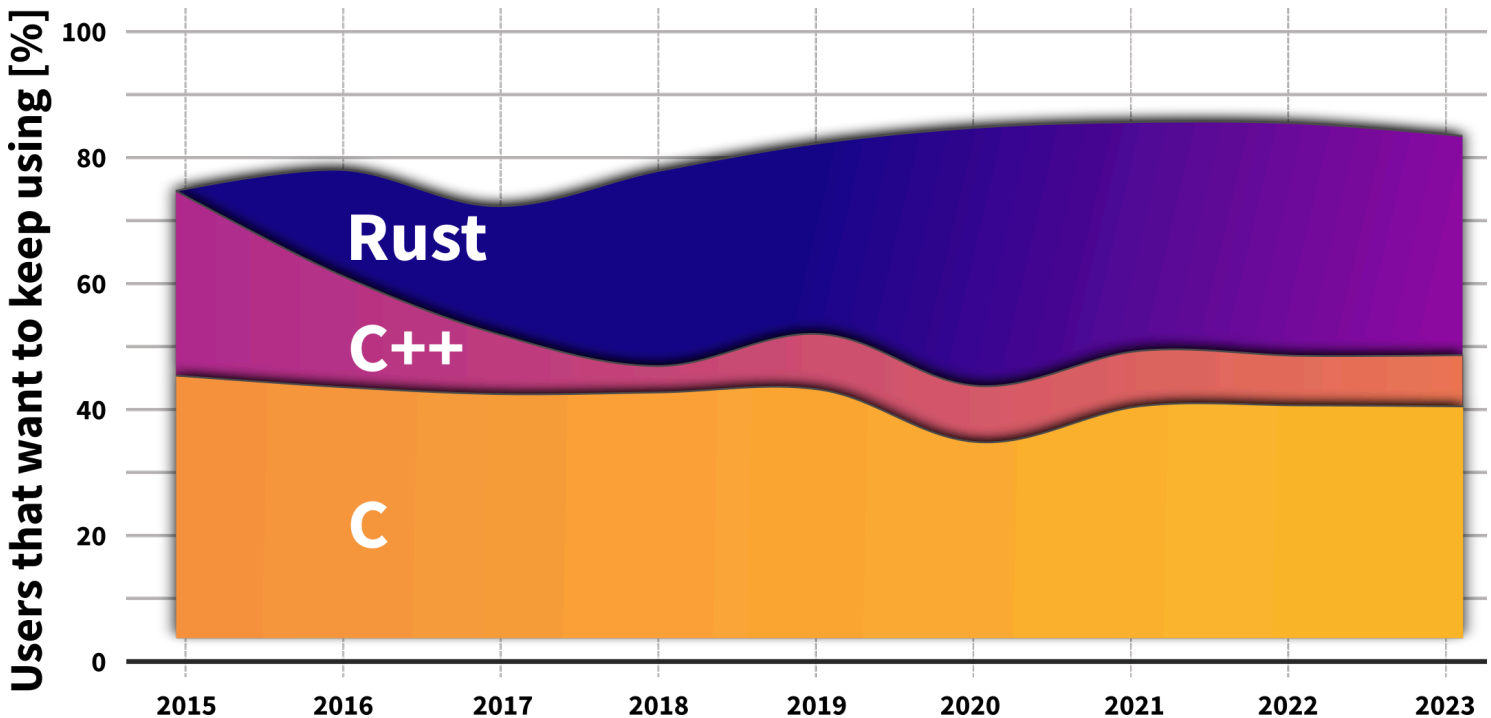
- Moderne Programmiersprache
- Stark typisiert
- Kompiliert, keine Garbage Collection
- Ausgezeichnetes Tooling
- Hilfreich(st)er Compiler
- Speichersicherheit durch Ownership-Modell

Vorteile einer modernen Programmiersprache

Part 3: Was ist Rust?

Rust ist extrem beliebt

Seit 2016 konstant die beliebteste Programmiersprache unter Entwicklern, während über die Hälfte der C und C++ Benutzer angeben, dass sie C(++) lieber nicht weiter nutzen würden.



Source: https://survey.stackoverflow.co/

Wer nutzt es und wofür?

- Etablierte Unternehmen (Auswahl):
  - Google
  - Microsoft
  - Mozilla
  - Dropbox
  - Atlassian
  - Cloudflare

Wer nutzt es und wofür?

- Etablierte Unternehmen (Auswahl):
  - Google
  - Microsoft
  - Mozilla
  - Dropbox
  - Atlassian
  - Cloudflare
- Low-Level:
  - Bootloader
  - Gerätetreiber (z. B. Framework Laptops)
- High-Level:
  - Netzwerkdienste
  - Web Apps

Wer nutzt es und wofür?

- Etablierte Unternehmen (Auswahl):
  - Google
  - Microsoft
  - Mozilla
  - Dropbox
  - Atlassian
  - Cloudflare

- Low-Level:
  - Bootloader
  - Gerätetreiber (z. B. Framework Laptops)
- High-Level:
  - Netzwerkdienste
  - Web Apps

z.B. Cloudflares neuer HTTP Proxy „Pingora“

→ 1 Billion / 10^{12} Requests pro Tag!

Wie sieht Rust aus?

Beispiel Enums und Pattern-Matching

```rust
pub enum StringOrInt { A(String), B(u8) }

fn foo(val: StringOrInt, optional: Option<u8>) {
    match val {
        StringOrInt::A(s) => println!("String: {}", s),
        StringOrInt::B(i) => println!("Int: {}", i),
    }
    match optional {
        Some(i) => println!("Some: {}", i),
        None => println!("None!"),
    }
}

fn main() {
    foo(StringOrInt::A("Hello World".to_string()), Some(42));
    foo(StringOrInt::B(123), None);
}
```

Ensuring Safety

Part 4: Sicherheit und Zuverlässigkeit

Memory Safety

Beim Kompilieren: Ownership & Borrowing

- Jeder Wert hat einen „Owner“ → Bestimmt die „Lifetime“ des Werts im Speicher
- Referenzen: Nur eine *mutable* (`&mut x`) **oder** unendlich viele *immutable* (`&x`) → Keine data races

Zur Laufzeit: Normale Prüfungen

- Array Längenprüfung
- Zugriff auf `None` oder `Err` Wert
- Integer Overflow (im Debug Build)
- …

Beispiel: Durchsetzung von Ownership

```rust
fn main() {
  let x = String::from("Hello World!");
  let y = x;
  println!("x: {}", x);
}
```

```
error[E0382]: borrow of moved value: `x`
 --> src/main.rs:4:23
  |
2 |     let x = String::from("Hello World!");
  |         - move occurs because `x` has type `String`, which does not implement the `Copy` trait
3 |     let y = x;
  |             - value moved here
4 |     println!("x: {}", x);
  |                       ^ value borrowed here after move
  |
  = note: this error originates in the macro `$crate::format_args_nl` which comes from the expansion of the macro `println` (in Nightly builds,
run with -Z macro-backtrace for more info)
help: consider cloning the value if the performance cost is acceptable
  |
3 |     let y = x.clone();
  |              ++++++++

For more information about this error, try `rustc --explain E0382`.
```

Thread Safety: explizit statt implizit

Send

- „Wert kann sicher an anderen Thread gesendet werden“
- Fast alles, was keine Referenz ist

Sync

- „Wert kann sicher zwischen mehreren Threads geteilt werden“ (erfordert `Send`)
  - ISRs werden wie Threads behandelt!
- z. B. `Mutex`, `Atomic`-Typen, Smartpointer

```rust
let rc = std::rc::Rc::new(42); // Reference Counted Pointer
// error[E0277]: `Rc<i32>` cannot be sent between threads safely
std::thread::spawn(move || { println!("rc: {}", rc); });
```

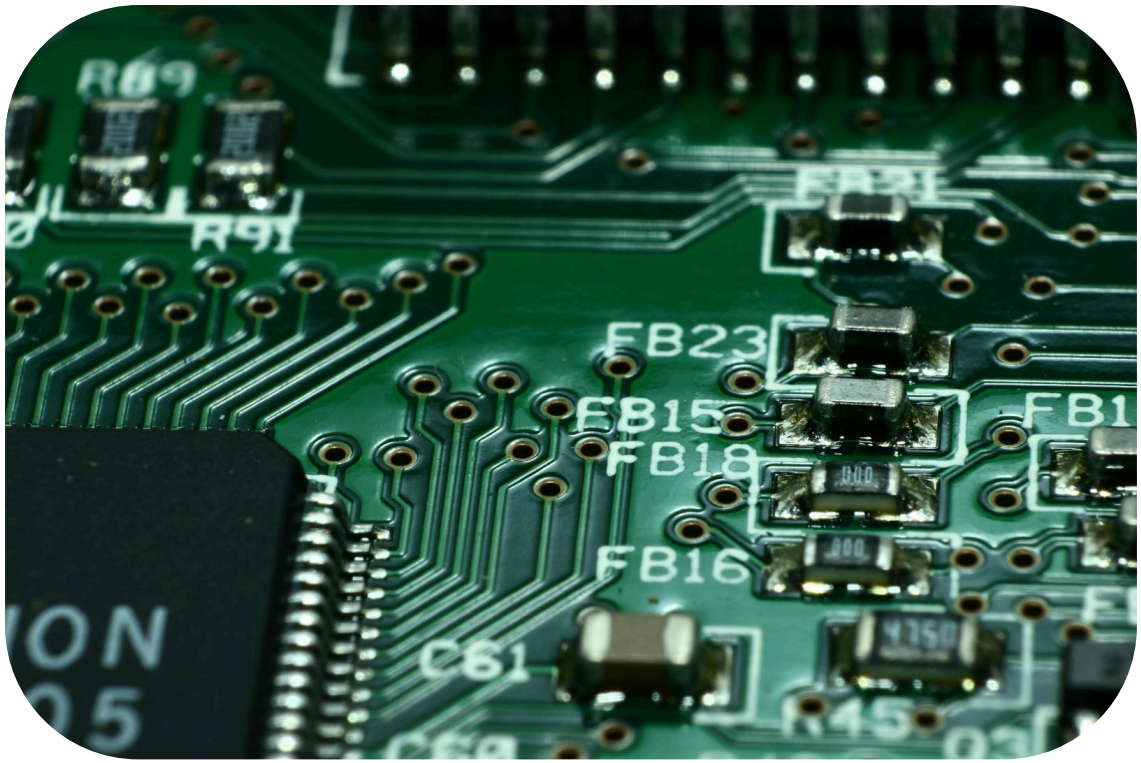
Compiler erzwingt korrekte Typen

Möglichkeit zur Abbildung von Hardware Eigenschaften

```rust
pub trait ClkPin { // Marker trait
    fn enable(&mut self) {
        println!("Enable ClkPin!");
    }
}
struct PA9; // Pin PA9
impl ClkPin for PA9{} // Mark PA9 as clock pin

// This function only works for Clock Pins
fn take_clk_pin(mut pin: impl ClkPin){
    pin.enable(); // Output: "Enable ClkPin!"
}

fn main() {
    take_clk_pin(PA9);
}
```



Rust in Embedded

Performance ohne Kompromisse

Part 5: Performance

Zero-cost Abstractions

Geschickt genutzt, können (meist) ohne weiteren Rechenaufwand viele Fehler vermieden werden:

```
struct SPI1; // No data -> size = 0 bytes -> optimized away by compiler

fn spi1_driver(spi: SPI1) {
    // Freely manipulate any SPI1 registers, nobody else controls SPI1
}

// In practice: obtained from HAL, e.g., `hal.peripherals.SPI1`
spi1_driver(SPI1);

// This won't work. No other driver can be created from the same SPI instance!
spi1_driver(SPI1); // error[E0382]: use of moved value: `spi`
```


Single-core Nebenläufigkeit

```rust
#[task]
async fn blinky(led: PB7) {
    let mut led = Output::new(led, Level::High, Speed::Low);
    loop {
        led.toggle();
        Timer::after_millis(300).await;
    }
}

#[main]
async fn main(spawner: Spawner) {
    let p = embassy_stm32::init(Default::default());
    let mut button = ExtiInput::new(p.PC13, p.EXTI13, Pull::None);
    spawner.spawn(blinky(p.PB7));
    loop {
      button.wait_for_rising_edge().await;
    }
}
```

Möglichkeiten zur Produktivitätsteigerung

Part 6: Produktivität

Compiler und Toolchain Manager: rustc & rustup

- `rustc`: Rust Compiler mit hilfreichen Fehlermeldungen
- `rustup`: verwaltet `rustc` für diverse Toolchains und erlaubt einfaches Cross-kompilieren

Beispiel:

```
$ rustup toolchain install 1.79.0-x86_64-apple-darwin # for macOS
$ rustup target add thumbv7em-none-eabihf # e.g. Cortex-M4F, Cortex-M7F (FPU)
```

```
# "rust-toolchain.toml" in project root defines the toolchain to download and use
[toolchain]
channel = "1.79"
targets = [
    "thumbv7em-none-eabihf",
]
```

Pakete, Formatierung, Dokumentation

- Paketmanager: `cargo <build|run|update|doc|fmt|...>`
  z. B. Projekt kompilieren und starten mit `cargo run`

```
# "Cargo.toml" project file
[dependencies]
cortex-m = { version = "0.7.6", features = ["critical-section-single-core"] }
```

- Einheitliche Codeformatierung mit `rustfmt` bzw. `cargo fmt`
- HTML-Dokumentation aus Docstrings mit `rustdoc` bzw. `cargo doc`:

```
/// GPIO output driver.
pub struct Output {...}
```


Struct embassy_stm32::gpio::Output

source · [-]

```
pub struct Output<'d> { /* private fields */ }
```

[-] GPIO output driver.

Note that pins will **return to their floating state** when `Output` is dropped. If pins should retain their state indefinitely, either keep ownership of the `Output`, or pass it to `core::mem::forget`.

Implementations

[-] `impl<'d> Output<'d>` source

[-] source

```
pub fn new(
    pin: impl Peripheral<P = impl Pin> + 'd,
    initial_output: Level,
    speed: Speed
) -> Self
```

Create GPIO output driver for a Pin with the provided Level and Speed configuration.

[-] `pub fn set_high(&mut self)` source

Set the output as high.

[-] `pub fn set_low(&mut self)` source

Set the output as low.

Tools speziell für Embedded Entwicklung

- Flashing und Debugging: `probe-rs` (v. a. für ARM chips)

```
$ probe-rs run --chip STM32F469NIHx target/thumbv7em-none-eabi/release/myapp
$ cargo run # Alternative with config in .cargo/config.toml
```

- Ressourcenschonendes logging: `defmt` → Keine Übertragung der einzelnen Characters

```
defmt::error!("This is very bad!");
// Output:
0.000000 ERROR  This is very bad!
└─ myapp::____embassy_main_task::{async_fn#0} @ src/bin/myapp.rs:100
```

- `Embassy`: Async Framework mit HALs für STM32, NRF und RP sowie vielen nützlichen crates: https://embassy.dev/

Embedded Rust in freier Wildbahn

Part 7: Praxisbeispiele

Wer benutzt Rust?

- Betriebssyteme:
  - Windows
  - Android
  - Linux → einzige offizielle Kernel Sprache neben C!
- Server: z. B. oxide.computer (komplette Firmware)
- Zugangskontrollsysteme: z. B. akiles.app (Hauptentwickler von Embassy)
- Ladeinfrastruktur: z. B. sksignet.us (Firmware + Embedded UI in Rust)
- Wearables: z. B. hoptech.ca (mind. UI mit Slint in Rust)
- Umweltsensoren: z. B. anyleaf.org (Treiber teilweise in Rust)

Rust lernen

Part 8: Training und weitere Informationen

Ressourcen

Kurse, Bücher, Dokumentation

- Rust book: doc.rust-lang.org/book
- Rust embedded book: docs.rust-embedded.org/book
- Awesome Embedded Rust: github.com/rust-embedded/awesome-embedded-rust
- Systemscape's Introduction to Embedded Rust: yrust.de/intro-vde
- Rust Embedded Matrix Chat: #rust-embedded:matrix.org
- Rust Forum: users.rust-lang.org

Professionelle Unterstützung

Beratung und Training

- Ferrous Systems GmbH (Berlin): Allgemeine Trainings zu (embedded) Rust
- One Variable UG (Berlin): Rapid Rust Prototyping, Verteilte Systeme, Fokus auf Systems Engineering
- Tweede Golf B.V. (Nijmegen, NL): (embedded) Rust Entwicklungsdienstleistungen und Trainings
- Systemscape GmbH (München): Allgemeine Trainings zu (embedded) Rust, Fokus auf Dienstleistungen und Systems Engineering

Rust in a Nutshell

Rust liefert:

- Sicherheit
- Zuverlässigkeit
- Performance
- Tooling

Für die Entwicklung bedeutet das:

- Mehr Effizienz
- Weniger Kopfschmerzen
- Mehr Spaß bei der Entwicklung
- Keine Speicherfehler!

Contact

Systemscape GmbH

+49 (0) 89 2488 021 42

contact@systemscape.de

www.systemscape.de

Disclaimer

The presentation at hand was created by Systemscape.

Systemscape, founded in 2023, is the leading global embedded Rust consultancy. Systemscape advises major international industry and service companies as well as public institutions. Our services cover the entire range of embedded systems consulting from strategic advice to successful implementation. The presentation at hand was particularly created for the benefit of the recipient and is based on certain assumptions and information available at the publishing date. Systemscape does not give an express or implied warranty regarding the correctness and completeness of the information contained in this study. There is no guarantee that the included projections or estimates will be realized. No indication or statement in this study shall be understood as an assured prediction. Information provided by collaborating companies and research institutes has not been verified by Systemscape. The reader should not act on any information provided in this study without receiving specific professional advice. In publishing this presentation, Systemscape reserves the right to make any necessary amendment or substitution and is not obliged to give the recipient access to the additional information. Any other use or disclosure of this study to a third party is strictly prohibited, unless expressly permitted via written consent from Systemscape. The image rights remain with the respective originator at any time. Systemscape shall not be liable for any damages resulting from the use of information contained in the presentation.

www.systemscape.de