CEC Continuing Education Center

DesignNews

Writing Microcontroller Drivers in Rust

DAY 5 : Writing “Hello World”

Sponsored by

DigiKey

BENINGO EMBEDDED GROUP

informa markets

Webinar Logistics

- Turn on your system sound to hear the streaming presentation.
- If you have technical problems, click “Help” or submit a question asking for assistance.
- Participate in ‘Group Chat’ by maximizing the chat widget in your dock.

THE SPEAKER

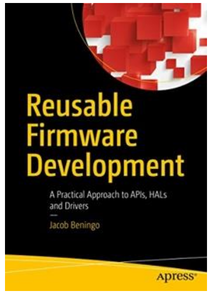
Jacob Beningo

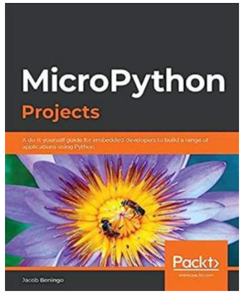
Jacob@beningo.com

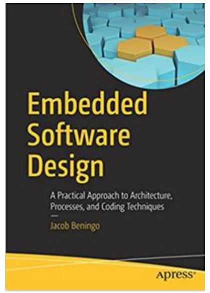
Beningo Embedded Group – CEO / Founder

Focus: Embedded Software Consulting and Training

Help teams deliver higher-quality embedded software faster. We specialize in creating and promoting embedded software excellence in businesses around the world.


Reusable Firmware Development
A Practical Approach to APIs, HALs and Drivers
Jacob Beningo
Apress




Embedded Software Design
A Practical Approach to Architecture, Processes, and Coding Techniques
Jacob Beningo
Apress

Blogs for:

- DesignNews.com
- Embedded.com
- EmbeddedRelated.com
- MLRelated.com

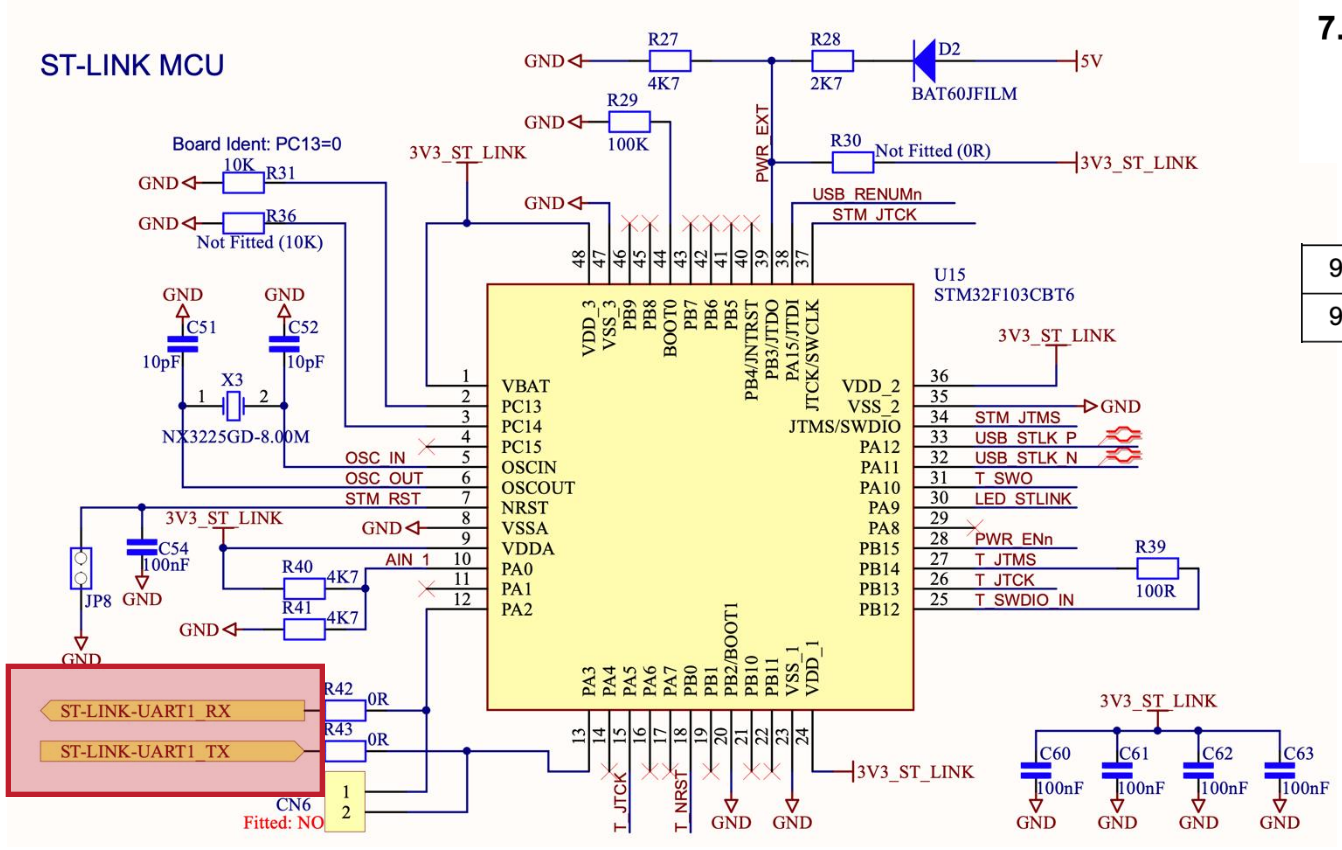
Visit **www.beningo.com** to learn more

01

Writing to the Terminal

Writing to the terminal

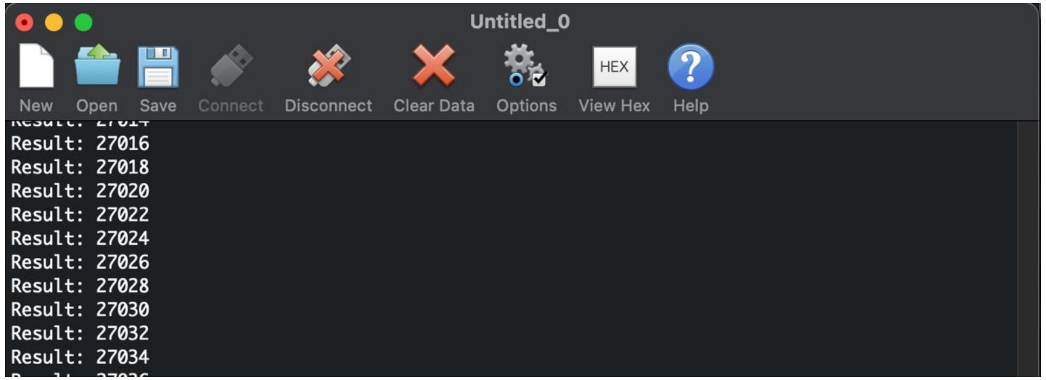
STM32L475 IoT Discovery Board USART1 Terminal



7.10 Virtual COM port

The serial interface USART1 is directly available as a Virtual COM port of the PC connected to the ST-LINK/V2-1 USB connector CN7. The Virtual COM port settings are configured as: 115200 b/s, 8 bits data, no parity, 1 stop bit, no flow control.

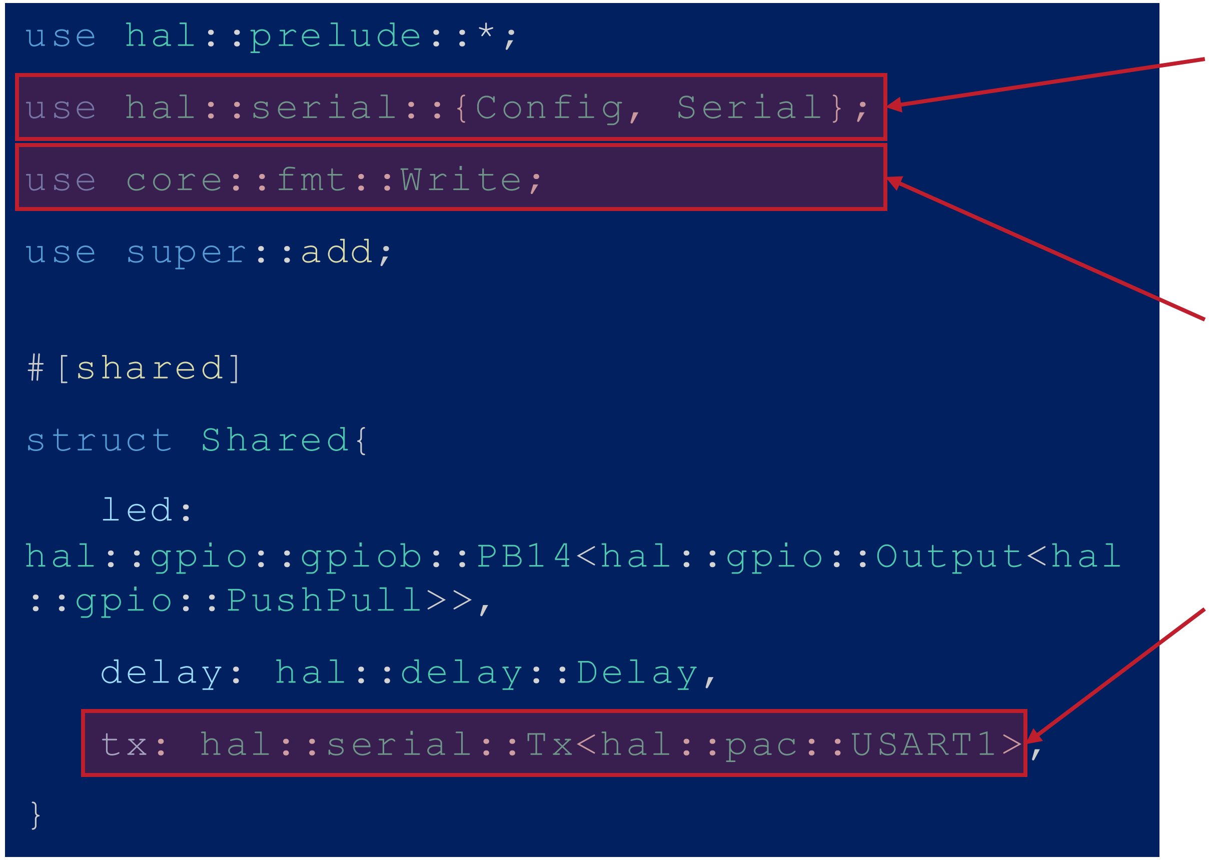
92	PB6	USART1_TX	ST-LINK-UART1_TX
93	PB7	USART1_RX	ST-LINK-UART1_RX



```
Result: 27016
Result: 27018
Result: 27020
Result: 27022
Result: 27024
Result: 27026
Result: 27028
Result: 27030
Result: 27032
Result: 27034
```

Writing to the terminal

Imports and tx defintions



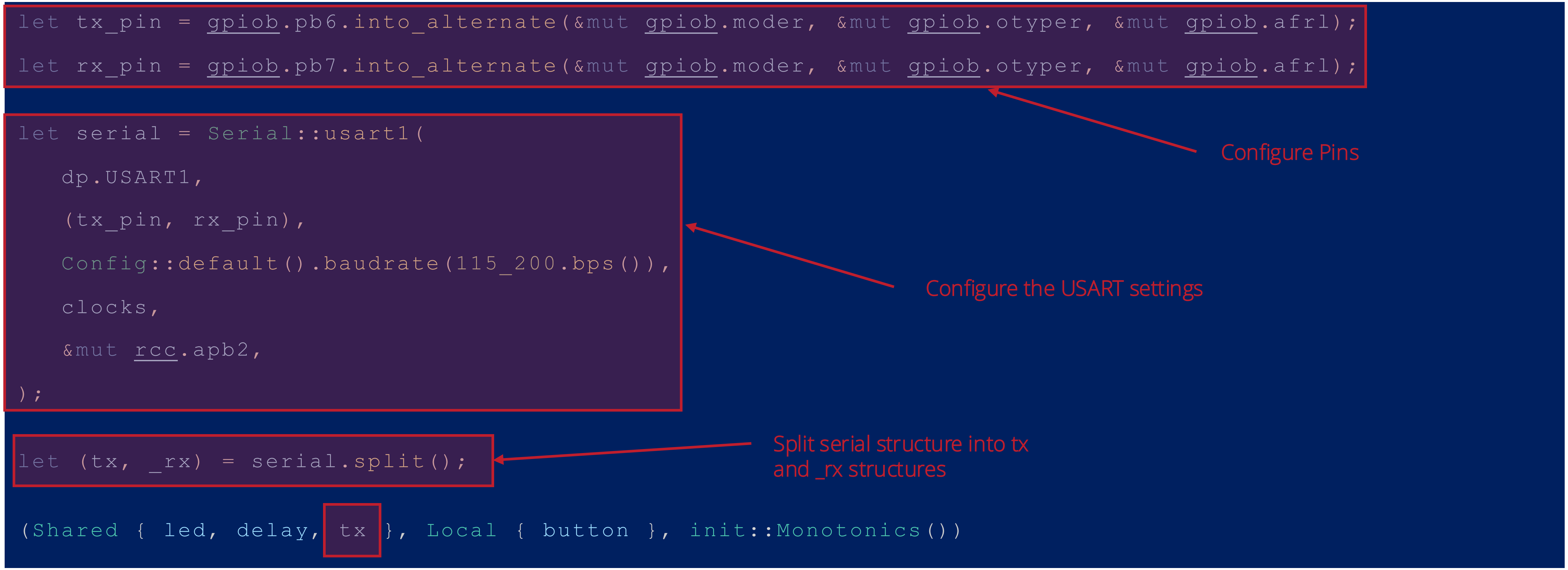
```
use hal::prelude::*;
use hal::serial::{Config, Serial};
use core::fmt::Write;
use super::add;

#[shared]
struct Shared{
   led:
hal::gpio::gpiob::PB14<hal::gpio::Output<hal
::gpio::PushPull>>,
   delay: hal::delay::Delay,
   tx: hal::serial::Tx<hal::pac::USART1>,
}
```

- Import Serial Config and Serial types
  - Configuration structure
  - Serial structure for USART interface
- Import Format
  - Trait for formatting strings
  - Allows formatting strings to USART1
- A field in Shared structure
  - Shared amongst tasks to access the USART
  - USART1 is a specific instance of serial

Writing to the terminal

Pin and Serial Initializations



```
let tx_pin = gpiob.pb6.into_alternate(&mut gpiob.moder, &mut gpiob.otyper, &mut gpiob.afrl);
let rx_pin = gpiob.pb7.into_alternate(&mut gpiob.moder, &mut gpiob.otyper, &mut gpiob.afrl);

let serial = Serial::usart1(
    dp.USART1,
    (tx_pin, rx_pin),
    Config::default().baudrate(115_200.bps()),
    clocks,
    &mut rcc.apb2,
);

let (tx, _rx) = serial.split();

(Shared { led, delay, tx }, Local { button }, init::Monotonics())
```

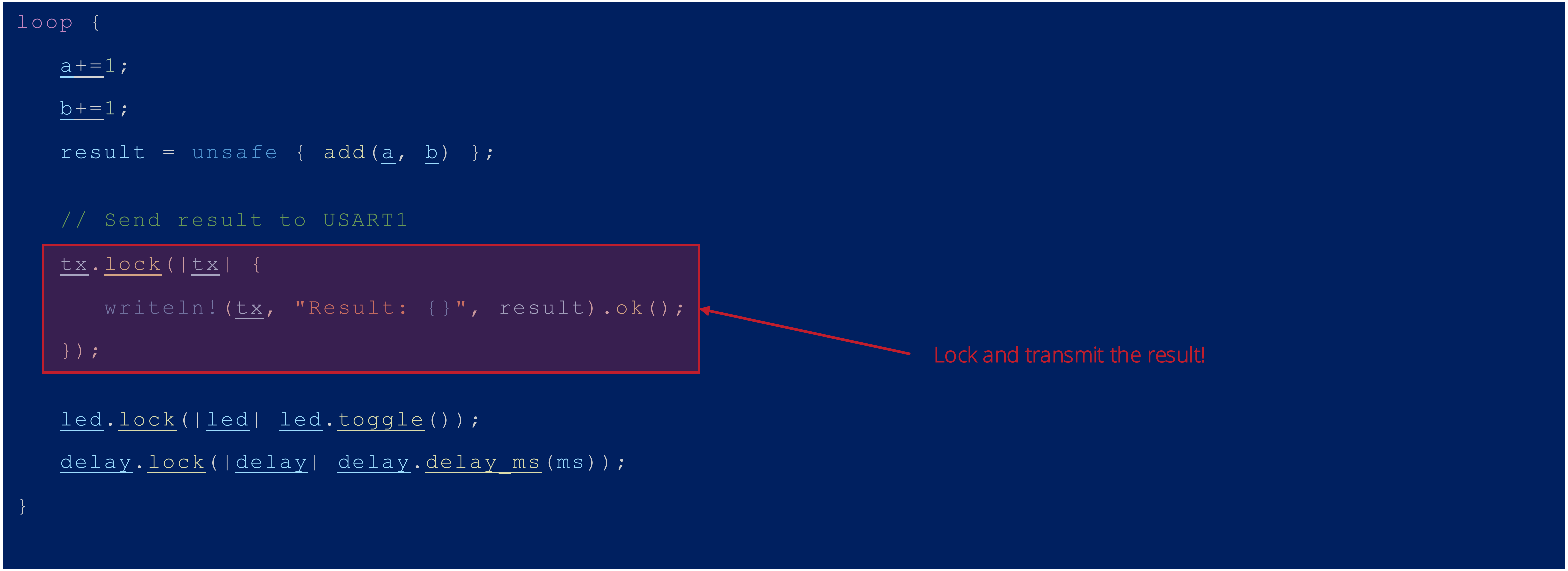
Configure Pins

Configure the USART settings

Split serial structure into tx and _rx structures

Writing to the terminal

Pin and Serial Initializations



```
loop {
    a+=1;
    b+=1;
    result = unsafe { add(a, b) };

    // Send result to USART1
    tx.lock(|tx| {
        writeln!(tx, "Result: {}", result).ok();
    });

    led.lock(|led| led.toggle());
    delay.lock(|delay| delay.delay_ms(ms));
}
```

Lock and transmit the result!

Audience POLL Question

What is the primary purpose of the core::fmt::Write trait in Rust?

A) To implement custom serialization and deserialization logic.
B) To provide methods for formatting and writing text to a string-like buffer.
C) To handle reading and writing files efficiently.
D) To allow writing binary data to standard output.

Review

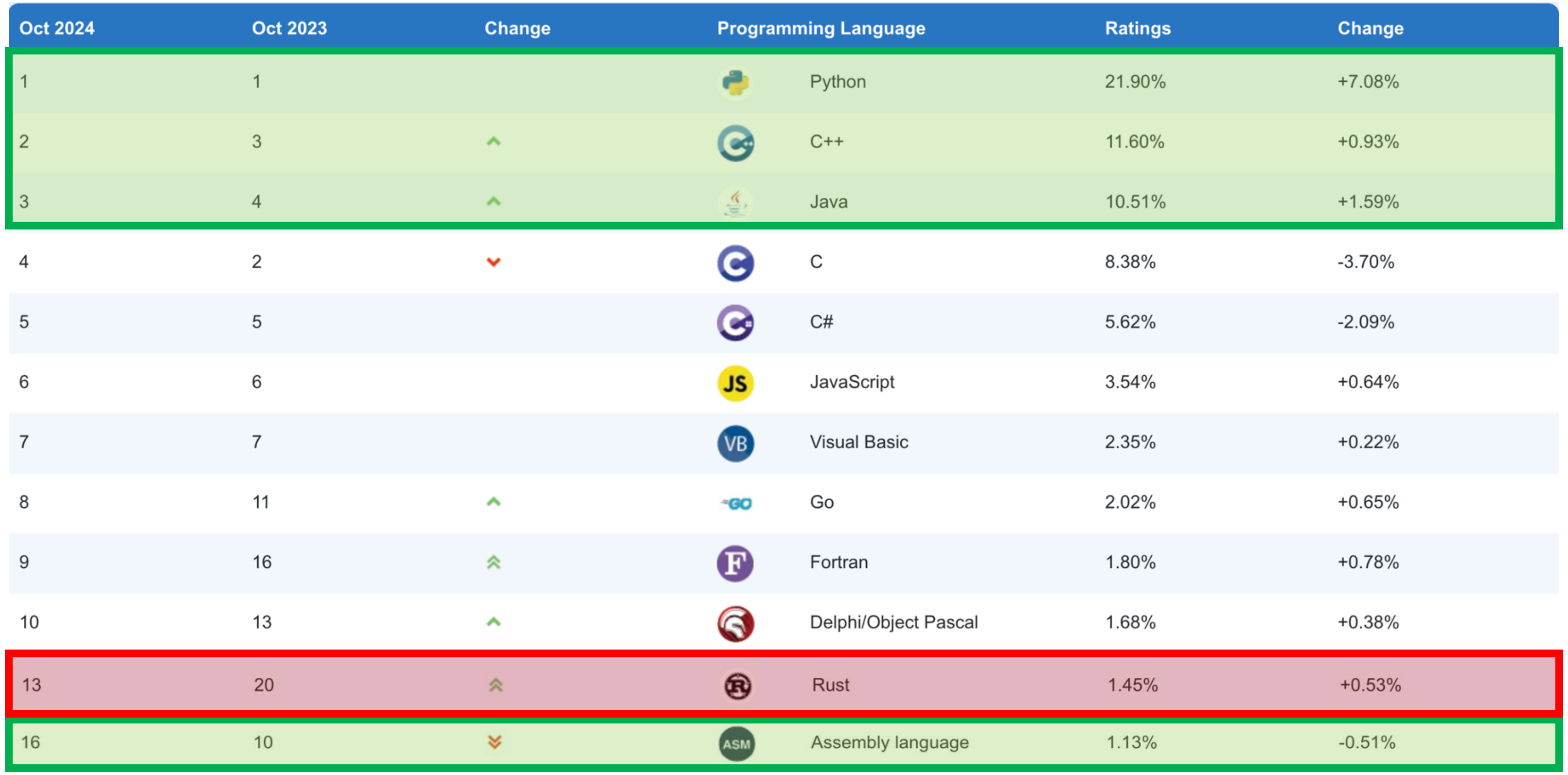
02

Embedded Programming Languages

Embedded Software Languages

Most Popular Embedded

- C (60 - 70%)
- C++ (20% - 25%)
- Python (<5%)
- Assembly
- Other



Oct 2024	Oct 2023	Change	Programming Language	Ratings	Change
1	1		Python	21.90%	+7.08%
2	3	▲	C++	11.60%	+0.93%
3	4	▲	Java	10.51%	+1.59%
4	2	▼	C	8.38%	-3.70%
5	5		C#	5.62%	-2.09%
6	6		JavaScript	3.54%	+0.64%
7	7		Visual Basic	2.35%	+0.22%
8	11	▲	Go	2.02%	+0.65%
9	16	▲▲	Fortran	1.80%	+0.78%
10	13	▲	Delphi/Object Pascal	1.68%	+0.38%
13	20	▲▲	Rust	1.45%	+0.53%
16	10	▼▼	Assembly language	1.13%	-0.51%

Note: 13-14% of Rust Developers are developing bare-metal embedded systems! Source

Rust in Embedded Systems

Why choose Rust for Embedded?

Advantages:

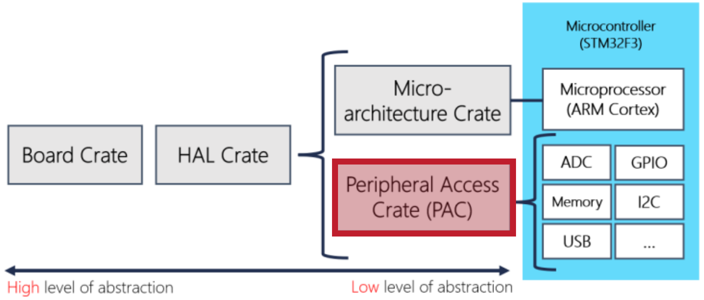
- Memory Safety
- Concurrency Safety
- Zero Overhead Abstractions
- Cross-Platform Development
- Modern Tooling
- Growing ecosystem
- Deterministic resource cleanup
- Compile-time error checking
- Interoperability

Disadvantages:

- Steep learning curve
- Smaller talent pool
- Limited library support for some targets
- Longer compile times
- Evolving language and ecosystem
- Verbose error handling
- Limited support for very low-level dev
- Lack of IDE support similar to C/C++
- Cross compilation complexity

The Peripheral Access Crate (PAC)

Defined

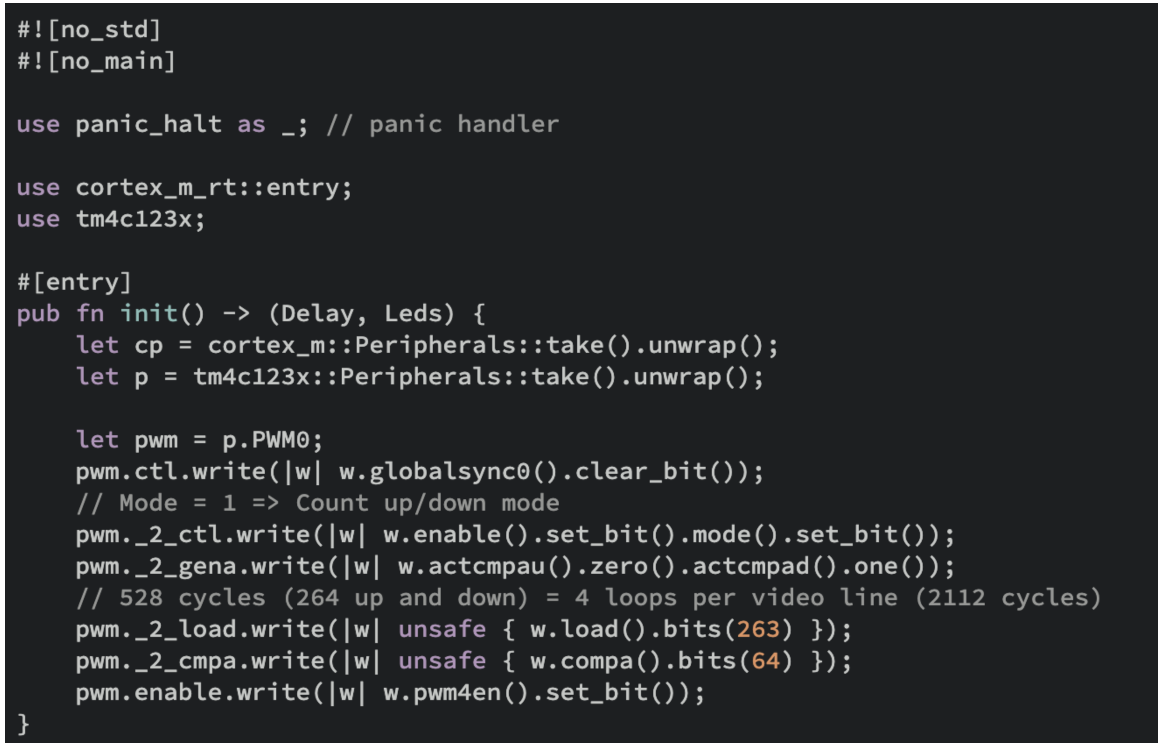

Microcontroller (STM32F3)
Board Crate
HAL Crate
Micro-architecture Crate
Microprocessor (ARM Cortex)
Peripheral Access Crate (PAC)
ADC
GPIO
Memory
I2C
USB
...
High level of abstraction
Low level of abstraction

What is the PAC?

- PAC stands for Peripheral Access Crate.
- Provides direct, low-level access to a microcontroller's peripherals.
- Auto-generated from the microcontroller's SVD (System View Description) files, ensuring accuracy and completeness.

Key Features

- Type Safety: Utilizes Rust's type system to prevent common bugs (e.g., invalid register access).
- Memory Safety: Ensures safe access to peripheral registers, mitigating risks of memory corruption.
- Concurrency Safety: Facilitates safe sharing of peripherals between tasks in concurrent environments.



```rust
#![no_std]
#![no_main]

use panic_halt as _; // panic handler

use cortex_m_rt::entry;
use tm4c123x;

#[entry]
pub fn init() -> (Delay, Leds) {
    let cp = cortex_m::Peripherals::take().unwrap();
    let p = tm4c123x::Peripherals::take().unwrap();

    let pwm = p.PWM0;
    pwm.ctl.write(|w| w.globalsync0().clear_bit());
    // Mode = 1 => Count up/down mode
    pwm._2_ctl.write(|w| w.enable().set_bit().mode().set_bit());
    pwm._2_gena.write(|w| w.actcmpau().zero().actcmpad().one());
    // 528 cycles (264 up and down) = 4 loops per video line (2112 cycles)
    pwm._2_load.write(|w| unsafe { w.load().bits(263) });
    pwm._2_cmpa.write(|w| unsafe { w.compa().bits(64) });
    pwm.enable.write(|w| w.pwm4en().set_bit());
}
```

Audience POLL Question

Are you going to start using Rust for embedded development?

A) Yes, for fun on my own projects

B) Yes, for prototypes at work

C) Yes, for customer deliverables at work.

D) No, not convinced it's the right direction yet.

Next Steps

04

Embedded Rust Docker Container

- https://mailchi.mp/beningo/embedded_rust_docker_container
  - Rust Toolchain
  - Embedded Tools

Beningo Rust Docker Container

Additional Resources

Please consider the resources below:

- Jacob's Blogs
- Jacob's CEC courses
- Embedded Software Academy

- Embedded Bytes Newsletter
  - http://bit.ly/1BAHYXm

www.beningo.com

Consulting

Coaching

Training

CEC Continuing Education Center

DesignNews

Thank You

Sponsored by

DigiKey

BENINGO EMBEDDED GROUP

informa markets