



Evaluation of Rust usage in space applications by developing BSP and RTOS targeting SAMV71

Filip Demski

ADCSS - 2024

Project objectives

Main goal: create a report that analyses how viable is introducing Rust into space domain

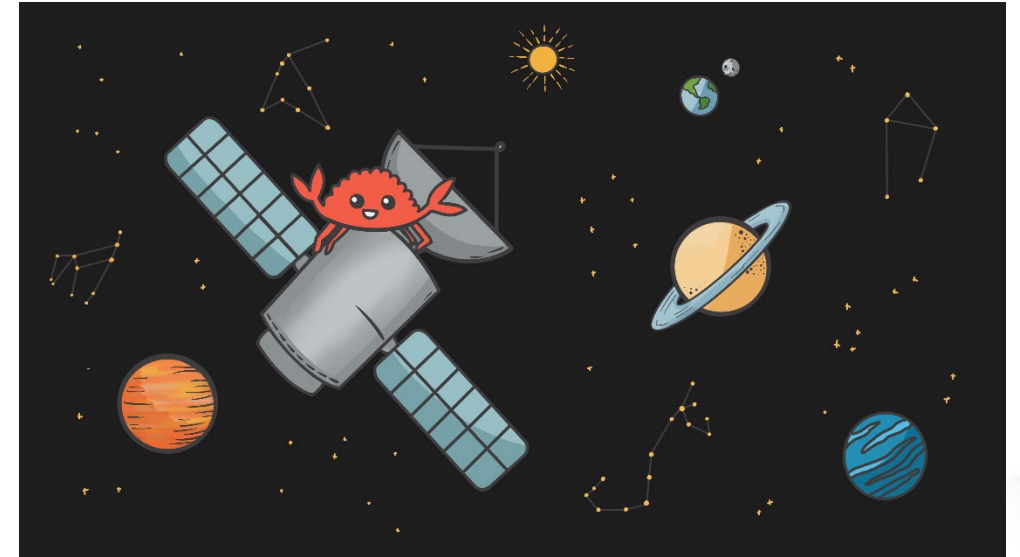
...but for that, we need something practical...

...that fits into tight budget of the project...

- Create a completely functional and usable RTOS, but focus on simplicity, and design that has space applications in mind
- Target SAMV71 devices, create a BSP that implements core peripherals + data acquisition ones for demonstration

Why Rust?

- Memory safety combined with high-performance
- Embedded programming in Rust gained a lot of traction
- Bare metal ARM and RISC-V support at high level
- Interoperability with C and Ada which allows to reuse existing components while introducing Rust
- Growing interest from general embedded and system programming community
- Growing interest from companies like AdaCore or Ferrous Systems in providing qualified Rust toolchain for critical operations
- Fits for common critical software requirements like possibility to disable dynamic allocation



GNAT PRO | FOR RUST



Rust Viability Report

- Examines
 - language features
 - strengths
 - weaknesses
 - the viability of Rust further use in the space applications
- Based on the outputs and conclusions coming from the RTOS and BSP...
- ...as well as on the thought of the developers.
- It's available publicly, send me a message if you are interested!

Strong sides of Rust

- Dedication to memory safety
 - well-designed ownership mechanism enforced during compilation
 - compiler checks helps prevent common but hard to find bugs like data races
- High-performance capabilities
 - language's focus on zero-cost abstractions
 - dynamic allocation is optional, precise but readable control over memory allocation
- Built-in documentation tests and examples
 - Rust toolchain ensures that code examples stay up-to-date

Strong sides of Rust

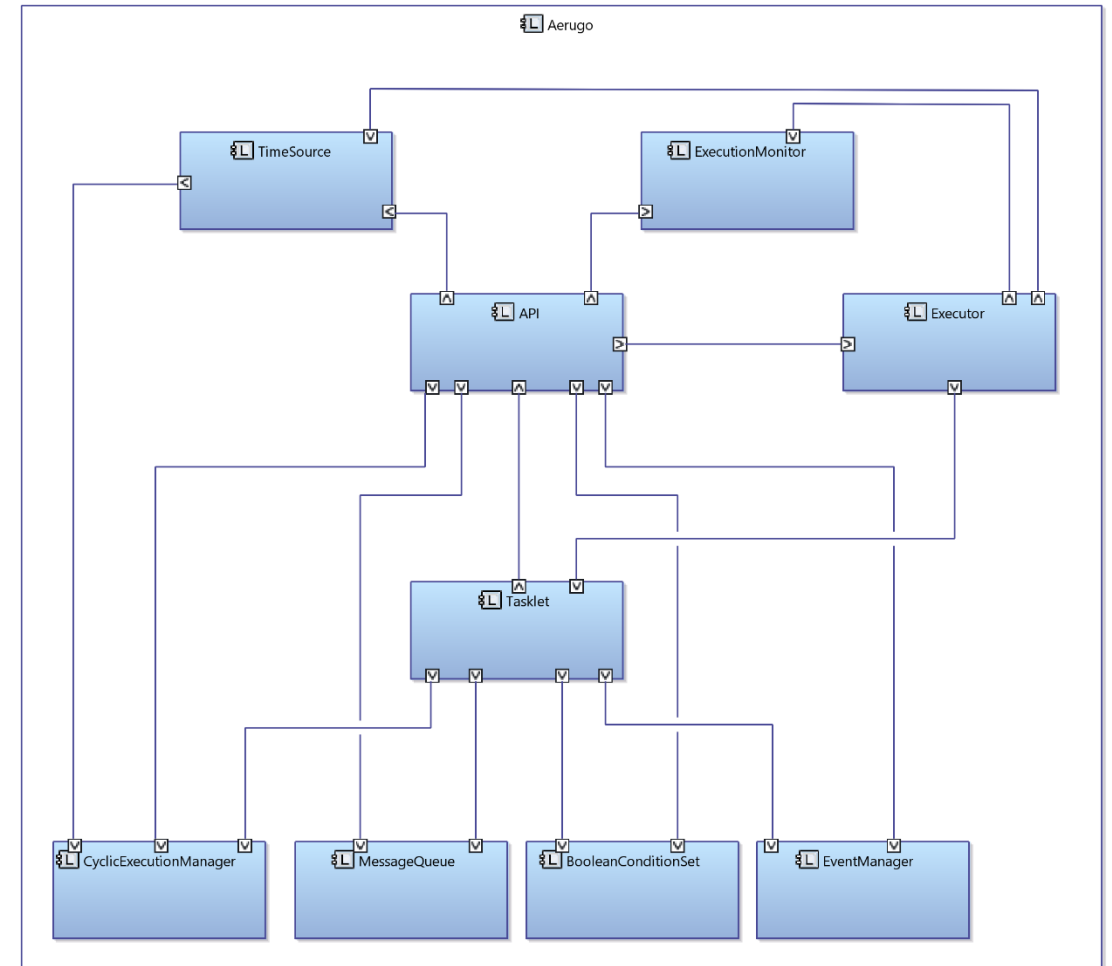
- Active ecosystem and engaged community
 - Rust continues to be the most-admired programming language in Stack Overflow survey
- Absence of legacy burdens, but including interoperability with C and Ada
 - a way to use well-established, well-tested performance-critical libraries without having to rewrite everything from scratch
- Typestate pattern fits driver development, making hardware state management simpler
 - very positive feedback from developer creating the drivers and very positive feedback from another developer using them
- Traits as an alternative to the object-oriented inheritance system
 - useful for embedded development, readable but can be much faster

Weak sides of Rust

- Steep learning curve
 - but long-time benefits, particularly in terms of enhanced code safety, justify the investment
 - rising interest in Rust among developers which could mean more people already having the knowledge wanting and looking for jobs in this language
- Difficulties in changing approach when coming from C and similar languages
- Build times
- Tools and libraries aren't as mature
 - but that is changing year after year, especially embedded which gained a lot of traction recently
- Support for different hardware targets
- Availability and stability of language features

Aerugo – lightweight RTOS in Rust

- Designed with simplicity in mind
- Inspired by FreeRTOS
- Influenced by purely functional programming paradigm and architecture of transputers
- Implemented in form of a round-robin executor
- Aimed at easier potential ECSS qualification

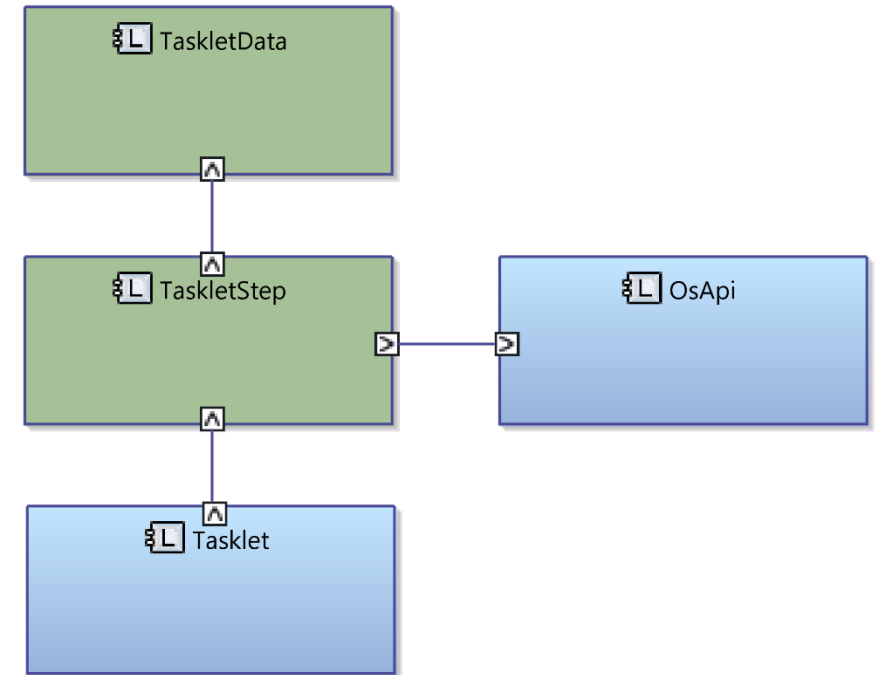


Tasklets inside

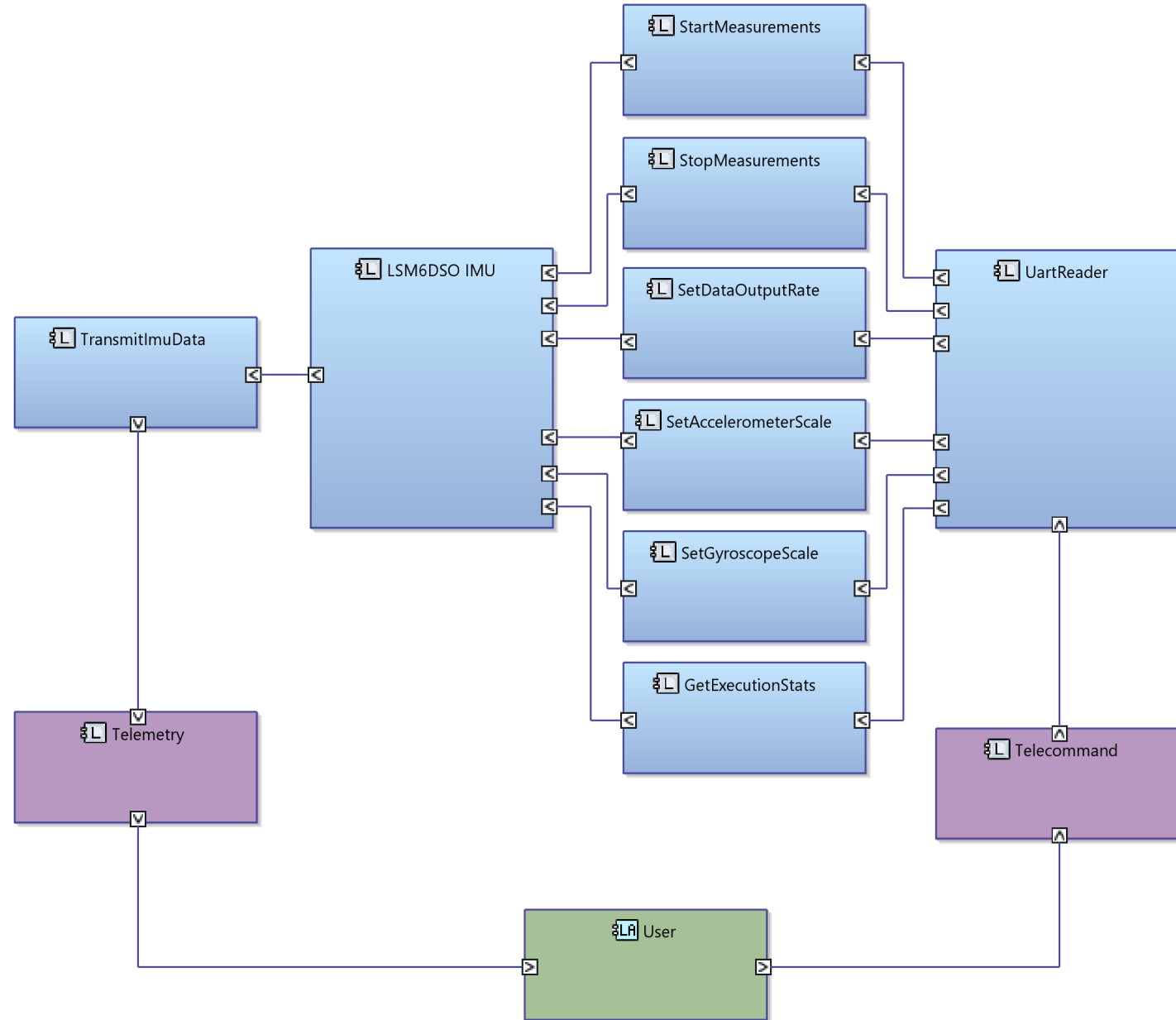
Tasklets instead of traditional tasks based on threads

Tasklets are fine-grained units of computation, that execute a processing step in a finite amount of time.

- Share stack
- Avoid context switches
- Predictable concurrency patterns
- Scheduled for execution once all the data they require is available
- Cannot contain blocking operations waiting on products of other tasklets
- Cannot contain infinite loops



Data must flow



Data pipes

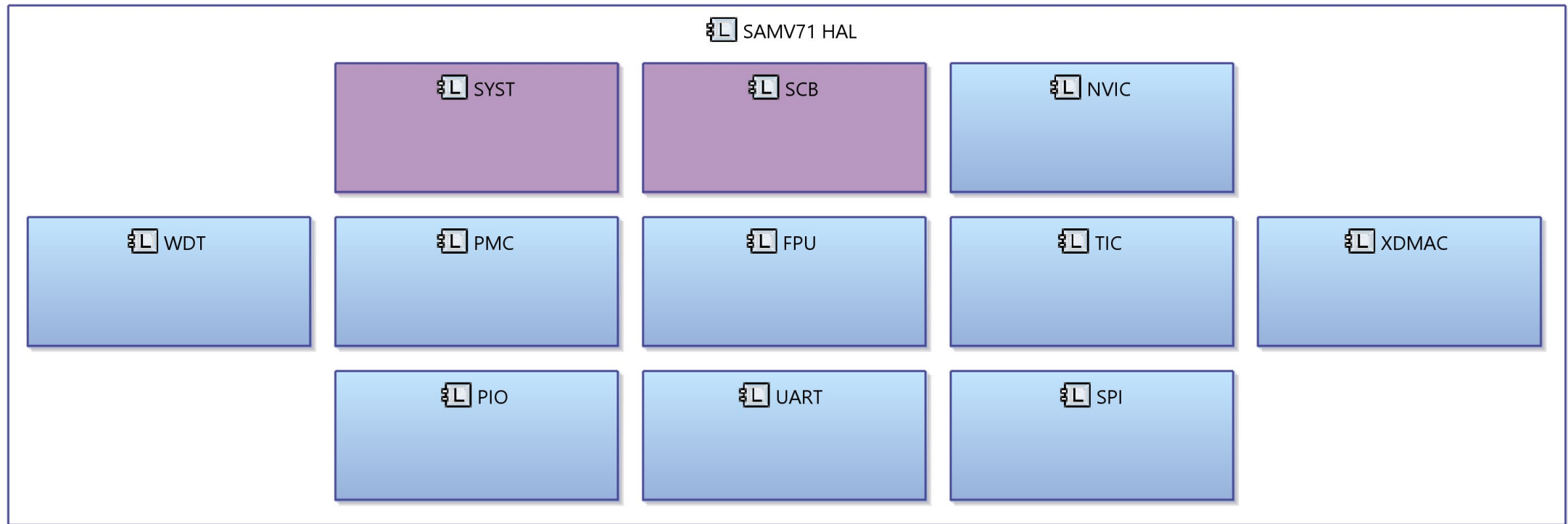
- Message queue
 - Primary way to send data between tasklets
- Event
 - Immediate – basically a *void* message
 - Delayed – replaces the functionality that is traditionally served by timers
 - Decrease coupling between tasklets
 - Cancellable
- Boolean condition
 - Represents some user-defined binary state of the system (readiness, availability, activation, etc.)
 - Can be used as a ‘on-off switch’ of tasklet

SAMV71 BSP

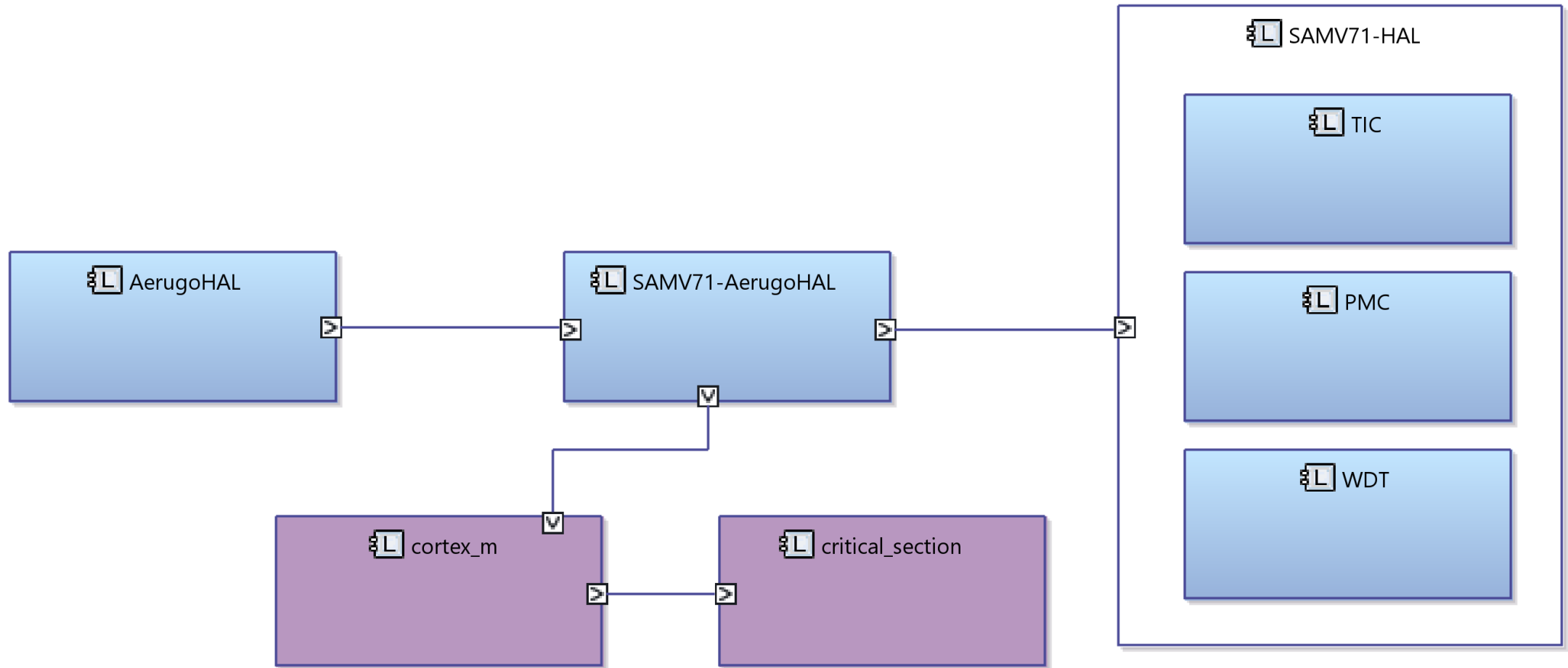
Designed in line with the standards in the embedded Rust community

PAC – Peripheral Access Crate

HAL – Hardware Abstraction Layer



RTOS <=> hardware integration



Integration with new platform is as easy as 1-2-3

How to: Run a `Hello, World!` demo on Aerugo

1. Grab BSP for your desired platform
2. Implement AerugoHAL trait (it's just 3 functions!)
3. Compile and start

```
/// System HAL trait.
pub trait AerugoHal {
    /// Type for system HAL error.
    type Error;

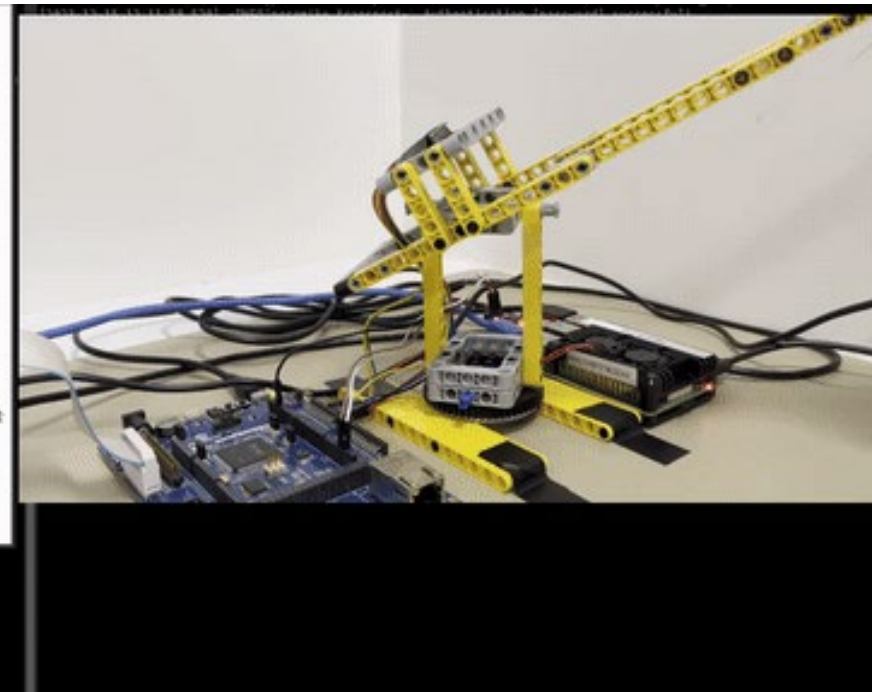
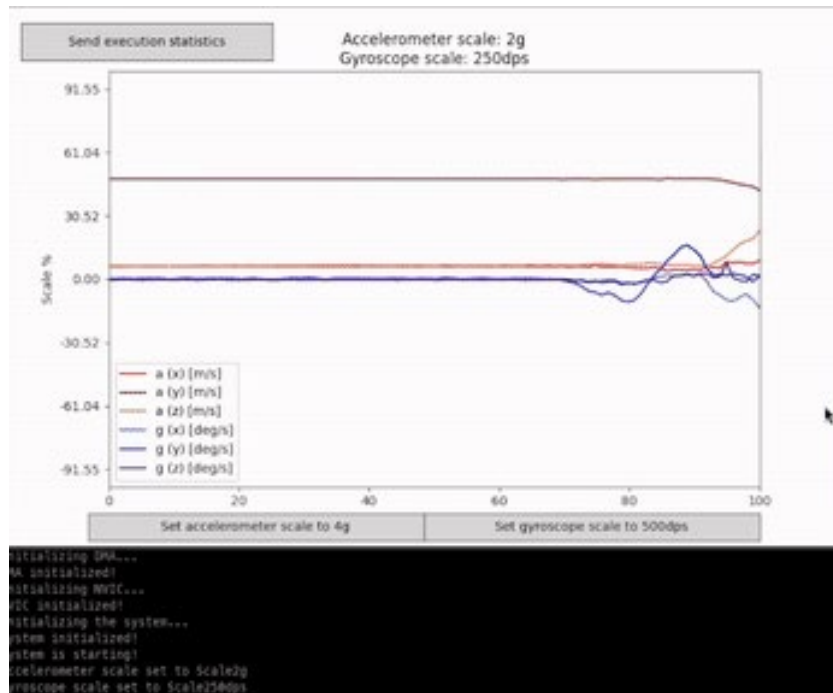
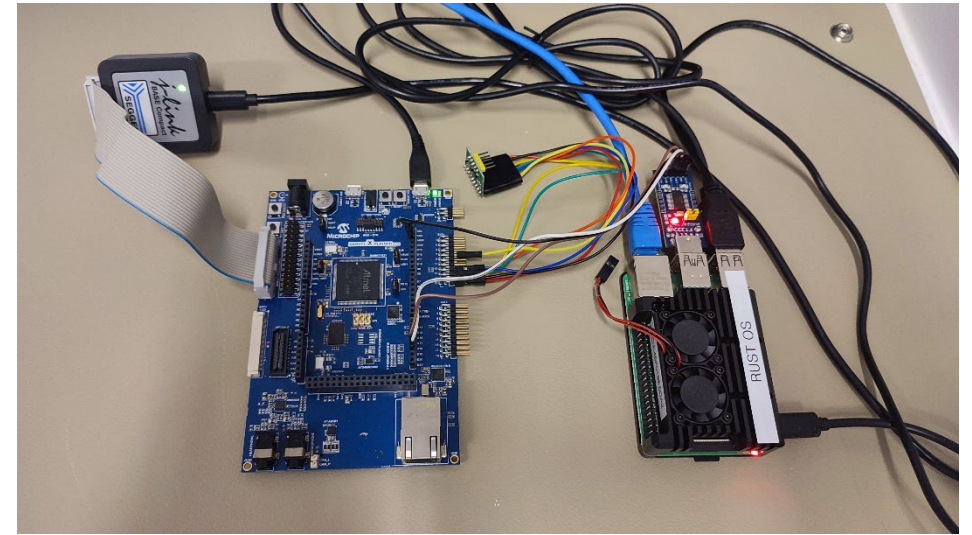
    /// Configure system hardware.
    ///
    /// Implementation should initialize and configure all core system peripherals.
    ///
    /// # Parameters
    /// * `config` - System hardware configuration.
    fn configureHardware(config: SystemHardwareConfig) -> Result<(), Self::Error>;

    /// Gets current system time timestamp.
    fn getSystemTime() -> Instant;

    /// Feeds the system watchdog.
    fn feedWatchdog();
}
```

Aerugo in practice


- running on SAMV71Q21 ARM Cortex-M MCU
- accelerometer-gyroscope instrument, containing a LSM6DSO sensor connected via SPI
- UART C&C TC/TM interface to the host computer
- data sent between SAMV71 board and host computer is packetized using CCSDS encapsulation



Everything is available on the open-source license

n7space / aerugo 276 ★

Safety-critical applications oriented
Real-Time Operating System written in Rust



n7space

<https://github.com/n7space/aerugo>

Way forward

- Asynchronous executor for multithreading systems
- Further development of SAMV71 BSP or creation of BSP for different MCUs
- Tooling to increase the usability of the system
- Aerugo qualification according to ECSS standard

We are actively looking for practical applications and cooperation in Aerugo next steps of development.

Thank you for your attention



Filip Demski
fdemski@n7space.com

Wojciech Olech
wojciech.olech@n7space.com