N7 SPACE

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

Agenda

- Project objectives summary
- Project achievements summary
- Demo
- Conclusions and lessons learned

Project objectives summary

“The proposed activity is to evaluate the usage of Rust programming language in space applications, by providing an RTOS targeting ARM Cortex-M7 SAMV71 microcontroller, a BSP (Board Support Package) and a Demonstration Application.”

Tasks:

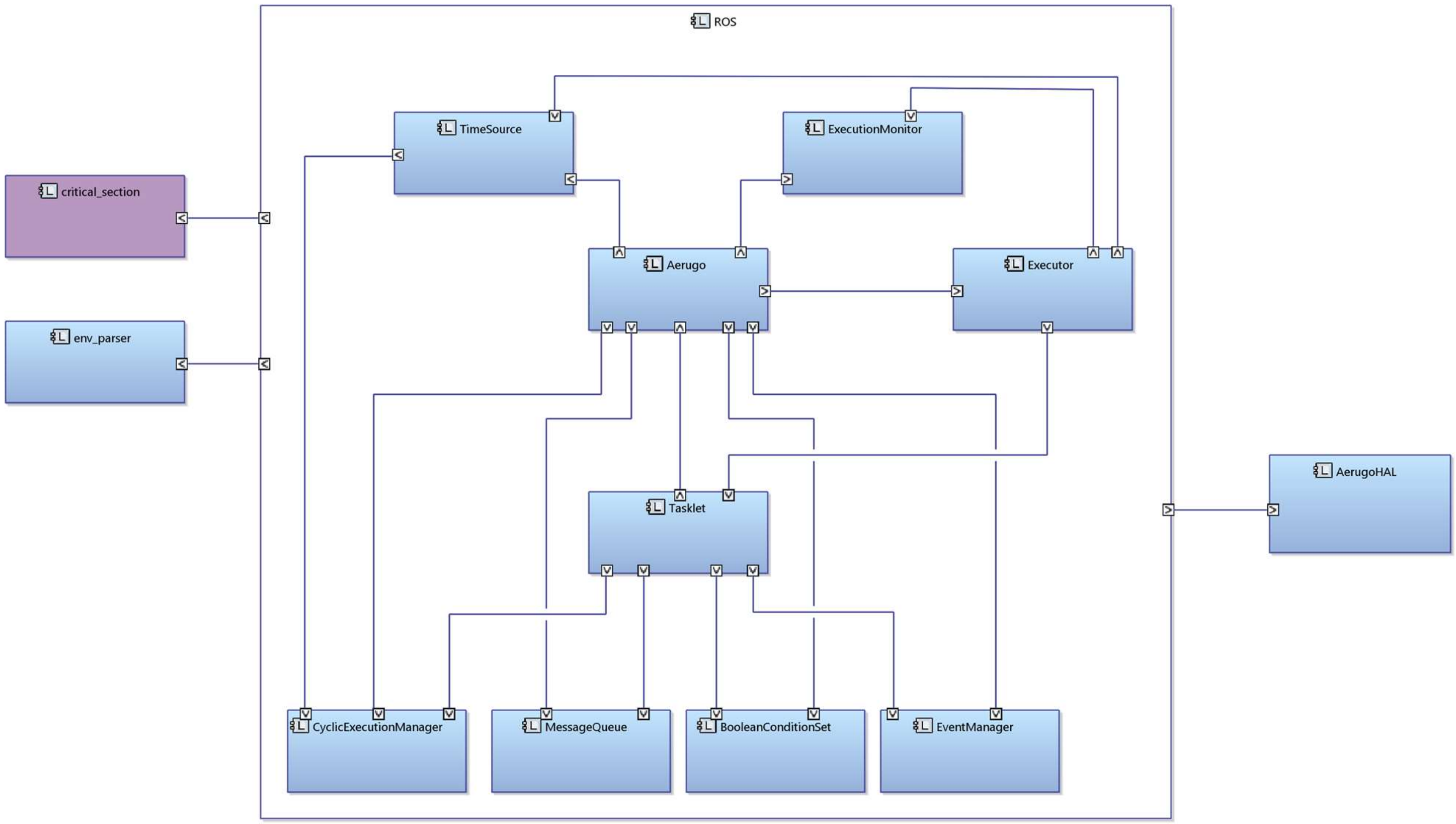
- Aerugo RTOS development
- SAMV71 BSP development
- Demonstration application development
- Creation of Rust Viability Report

PROJECT ACHIEVEMENTS SUMMARY

2024-06-04

RTOS - Aerugo

- Designed with simplicity in mind
  - Inspired by FreeRTOS
  - Influenced by purely functional programming paradigm and architecture of transputers
  - Easier ECSS qualification
- Implemented in the form of an executor
  - 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

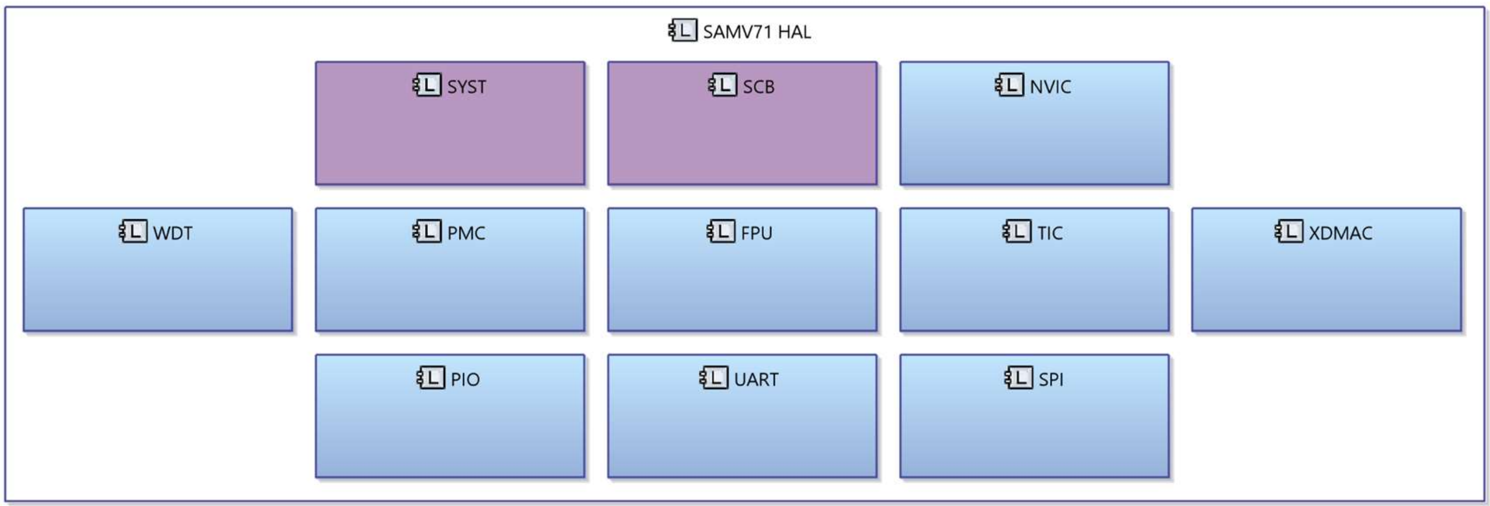

ROS
TimeSource
ExecutionMonitor
critical_section
Aerugo
Executor
env_parser
Tasklet
AerugoHAL
CyclicExecutionManager
MessageQueue
BooleanConditionSet
EventManager

```rust
pub struct TaskUartReaderContext {
    pub data_output_rate_queue: MessageQueueHandle<OutputDataRate, 2>,
    pub accelerometer_scale_queue: MessageQueueHandle<AccelerometerScale, 2>,
    pub gyroscope_scale_queue: MessageQueueHandle<GyroscopeScale, 2>,
}

pub fn task_uart_reader(
    buffer: TelecommandBuffer,
    context: &mut TaskUartReaderContext,
    api: &'static dyn RuntimeApi,
) {
    let header = match CCSDSPrimaryHeader::try_from(&buffer[0..=5].try_into().unwrap()) {
        Ok(header) => header,
        Err(reason) => {
            Telemetry::new_invalid_telecommand_error(InvalidTelecommandError::InvalidCCSDSHeader)
                .write_ccsds_packet(unsafe { UART_WRITER_STORAGE.as_mut().unwrap() });
            logln!(
                "Could not parse CCSDS primary header of received telecommand ({:?}): {:02X?}",
                reason,
                buffer
            );
            return;
        }
    };
```

SAMV71 BSP

- Designed in line with the standards in the embedded Rust community
  - PAC – Peripheral Access Crate
  - HAL – Hardware Abstraction Layer



AerugoHAL

- Interface for integration between Aerugo and HAL

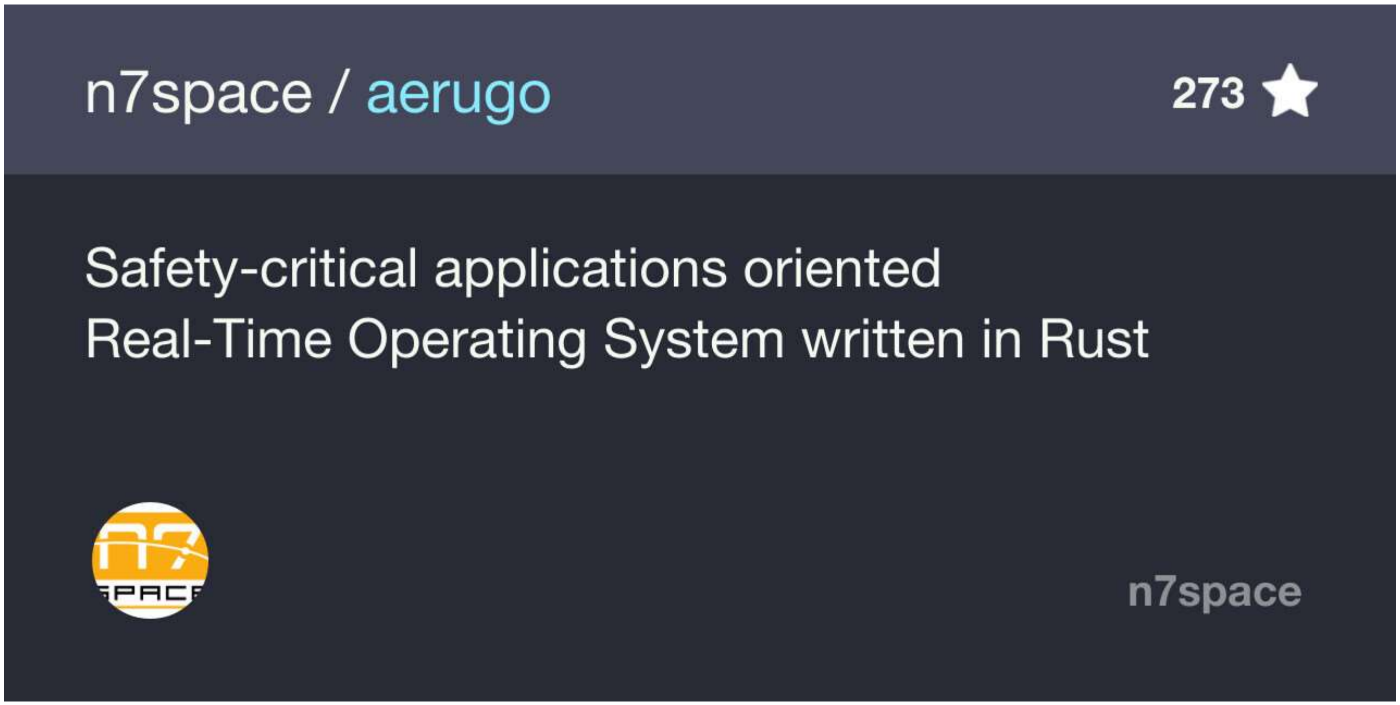
```
/// System HAL trait.
pub trait AerugoHal {
    /// Type for system HAL error.
    type Error;

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

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

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

Everything is available on the open-source license

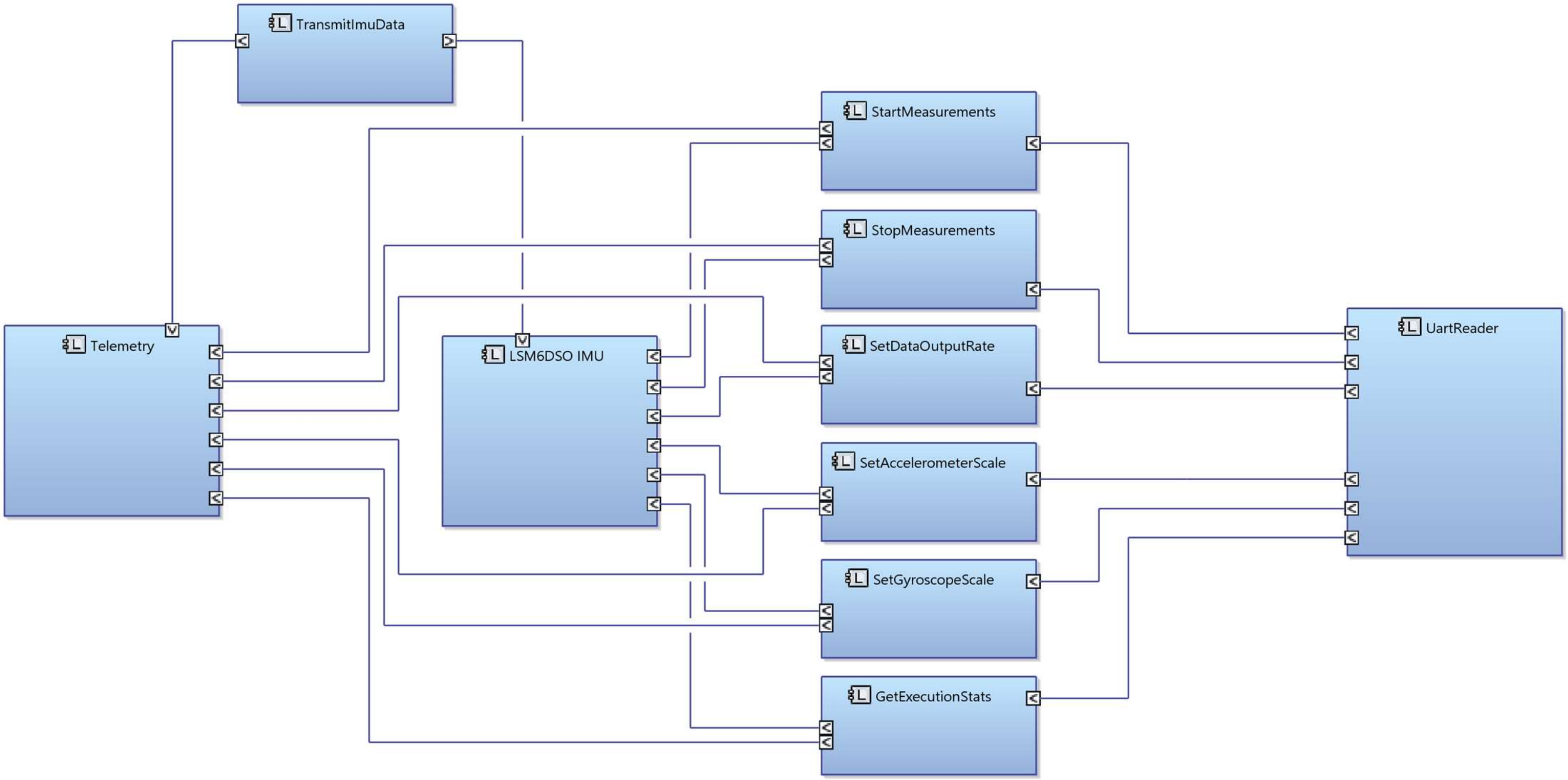


https://github.com/n7space/aerugo

DEMO

Demonstration application

- SAMV71Q21 ARM Cortex-M MCU
- LSM6DSO accelerometer-gyroscope sensor connected via SPI
- UART C&C TC/TM interface to the host computer


TransmitImuData
StartMeasurements
StopMeasurements
Telemetry
LSM6DSO IMU
SetDataOutputRate
UartReader
SetAccelerometerScale
SetGyroscopeScale
GetExecutionStats

CONCLUSIONS AND LESSONS LEARNED

Rust Viability Report

- Examines the strengths, weaknesses and the viability of Rust further use in the space applications
- Based on the outputs and conclusions coming from the Aerugo RTOS
- As well as on the thought of the developers
- Plans to release it publicly

RESULT?

Rust is very promising.

Strong sides of Rust

- Dedication to memory safety
- High-performance capabilities
- Built-in documentation tests and examples
- Active ecosystem and engaged community
- Absence of legacy burdens, but including interoperability with C
- Typestate pattern fits driver development
- Traits as an alternative to the object-oriented inheritance system

Weak sides of Rust

- Steep learning curve
- Difficulties in changing approach when coming from C
- Build times
- Tools and libraries aren’t as mature
- Support for different hardware targets
- Availability and stability of language features

Way forward

- Implement asynchronous executor using Rust `async` feature
- Further development of SAMV71 HAL
  - MCAN, SDRAMC, GMAC, TWIHS...
- Create SAMRH71 HAL
- Qualification according to ECSS standard to the category B

THANK YOU FOR YOUR ATTENTION

Filip Demski
fdemski@n7space.com