O^{S}xide RTOS

Codi Burley, Dan Wendelken, Dominic Farolino, Doug Flick

Goals

- Explore the capabilities of Rust in an embedded device setting. Particularly, in the context of the NRF51 development kit we used for testing.
- Develop a small and basic real time operating system in Rust. This requires the development of a very simple scheduler.
- Consider the advantages using Rust gives over a similar system built in C. Is it faster? Is it safer?

Intellectual Merits

Rust has not made a big footprint in the embedded community

We aimed to contribute a low-level secure system to the IOT field

Develop a safety-first RTOS kernel with the help of Rust

Broader Impacts

Research in the field of secure Real-Time Operating Systems catalyzes the effort to formally prove software (EAL-verified operating systems)

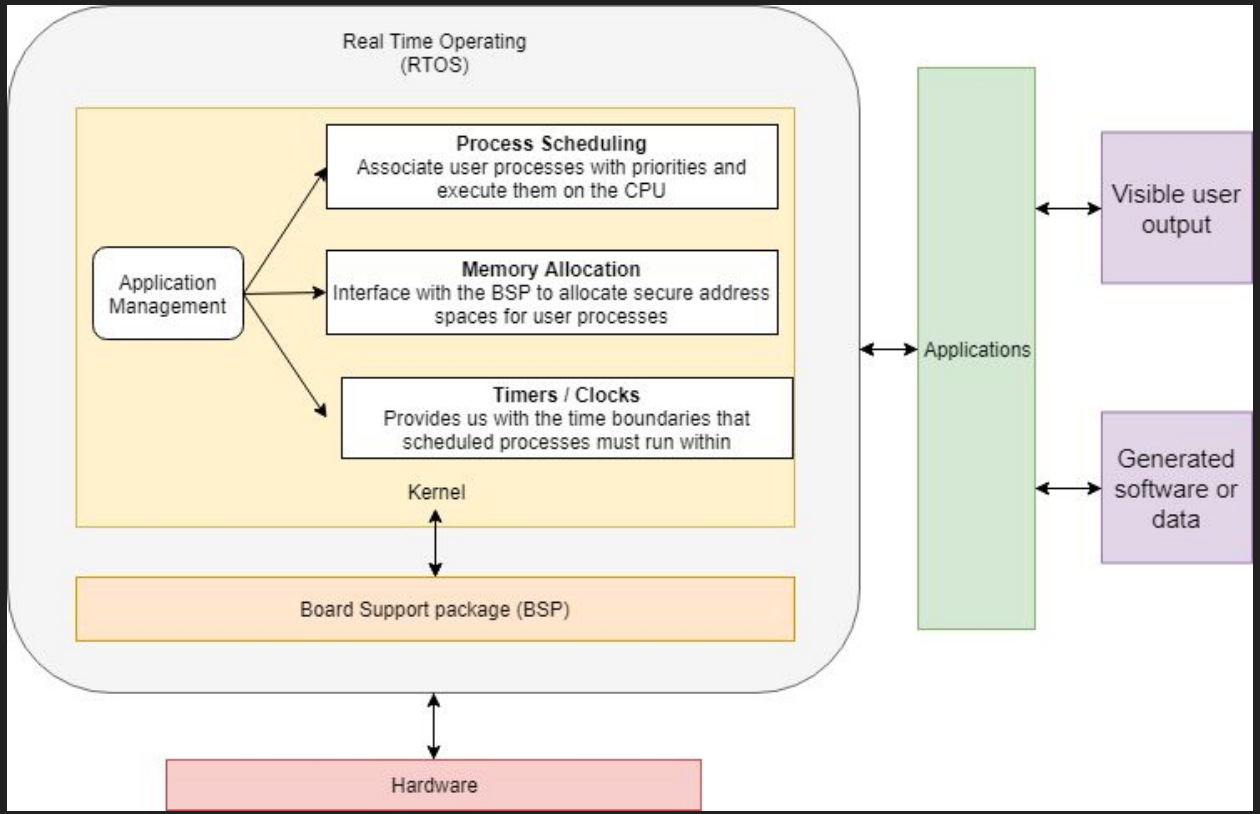
Security in iOT and embedded devices maximizes security of user data

Secure embedded devices guarantee the good nature of technology in medical devices, power plants, and much more

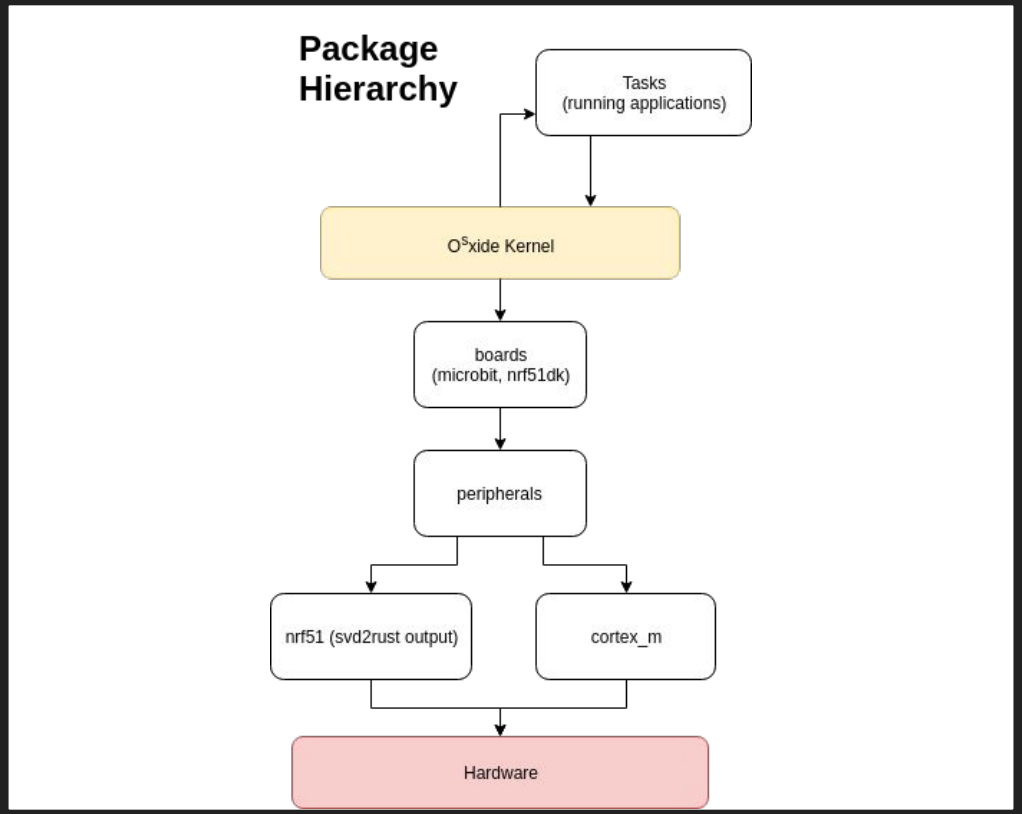
Design specification: Overview

- Multiple layers
  - Hardware Layer
    - Designed to interact directly with the chip and the hardware addresses
    - Implementation creates singletons to hardware such as buttons, leds and timers
  - OS / Kernel Layer
    - Setups up an event driven preemptive scheduler
    - Provides kernel calls that allow interaction with the hardware
  - Application Layer
    - Runs independently from other applications
    - Can be designed to do several tasks according to real time events

Design Specification



Design Specification



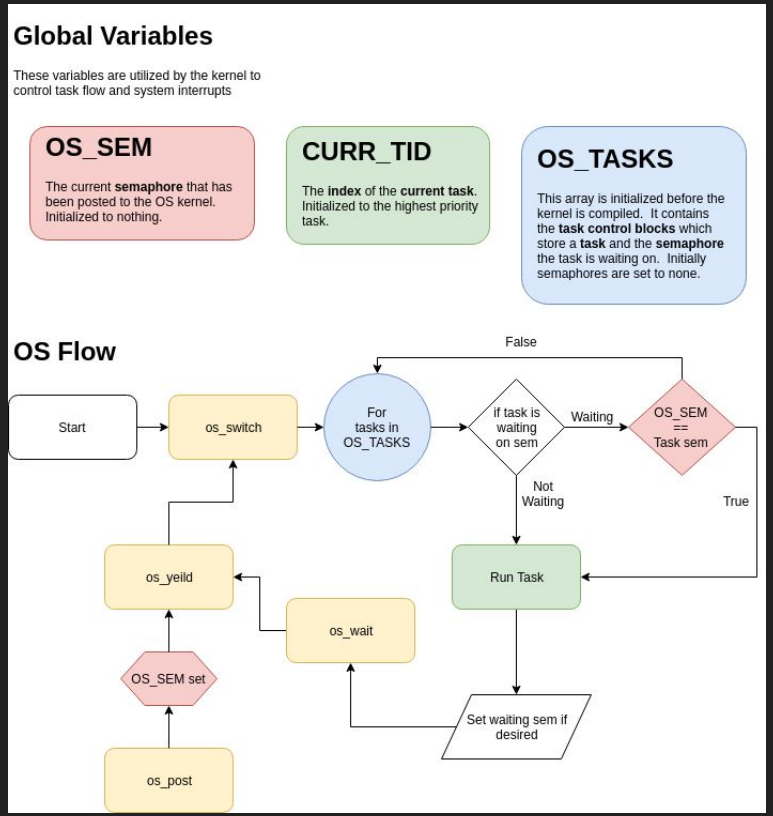
Design Specification: Interrupts

- The interrupts were enabled via a cortex_m package that provides generic access to core peripherals such as the nested vector interrupt controller (NVIC)
- Manufacturer-specific hardware peripherals were accessed using a system view description (SVD), that provides this access via memory addresses
- Interrupts can be fired via periodic events such as timers or randomly such as through GPIO task events

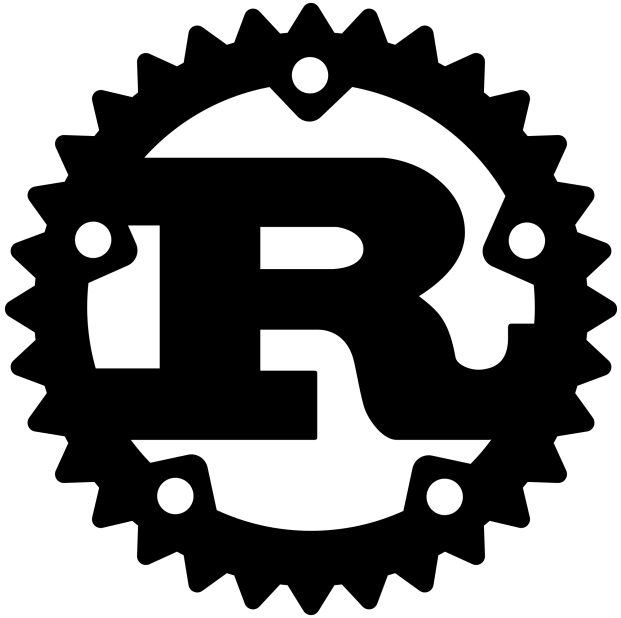
Design Specification: Task Scheduling

- Big part of the operating system to provide execution constraints
- Tasks are layed out linearly in memory
- `os_switch` routine which switches the execution of a task
- Tasks tell the OS what resources it needs/responds to
- `os_switch` responsible
  - Executing tasks whose dependencies fulfilled
  - Posting dependencies to tasks that need them

Kernel Overview



Technologies



NORDIC
SEMICONDUCTOR
Smarter Things

Milestones

Task	Date
Research rust-based RTOS implementations	January 16th
Research RTOS scheduling strategies	January 25th
Software running on the board	February 16th
Basic task scheduling	March 18th
Timer interrupts	March 26th
Interrupt handling integrated with task scheduling	March 27th
Event queue	TODO

Results

- Ability to schedule and run tasks governed by time-based interrupts
- Interface with board IO and use timers to interrupt tasks
- User tasks can interact with OS-posted resources (buttons!)

References

- https://github.com/helena-project/tock