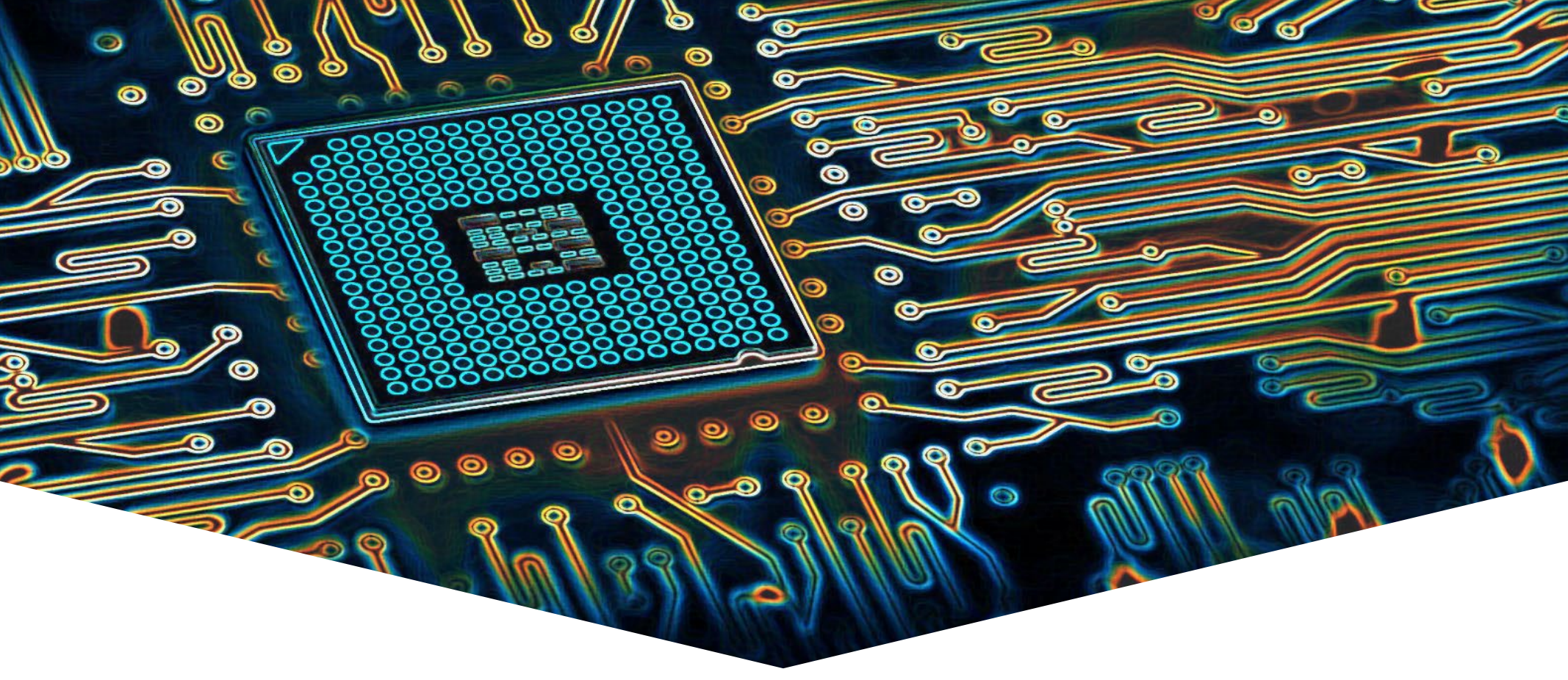


RTS1 Week 8

Leerdoelen Week 8

Leerdoelen week 8.

Je leert:

- Hoe embedded Rust is opgezet voor de STM32.
- Wat er nog mist

Toetsing

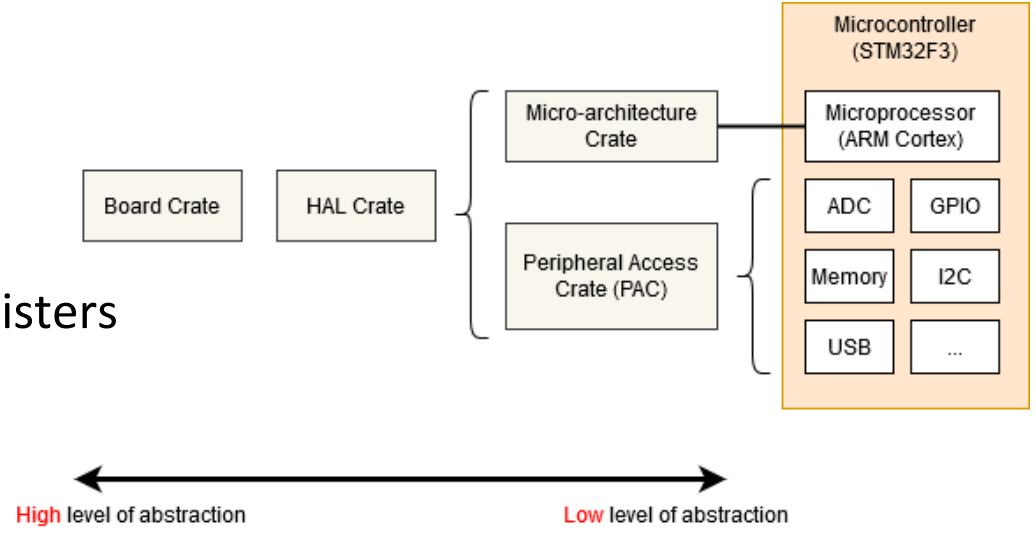
8	C	25 %	… een simpele embedded applicatie of deel van een applicatie in Rust te ontwerpen en te realise-ren.

Toets	Leerdoelen	Weging	Deadline
Verslag 1	1 en 2	25 %	Lesweek 1.2, zondag 23.59 uur
Verslag 2	3 t/m 7	50 %	Lesweek 1.6, zondag 23.59 uur
Verslag 3	8	25 %	Lesweek T1, zondag 23.59 uur

Overzicht embedded Rust

Embedded Rust voor de STM32

- Cortex-m crate
  - In ontwikkeling sinds Sept 2016
  - Voor het benaderen van de cpu registers
- Stm32f4 crate
  - Svd2rust
  - In ontwikkeling sinds April 2018
  - Voor het benaderen van de μC registers
- Stm32f4xx-hal crate
  - In ontwikkelings sinds Okt. 2018
  - Algemene peripheral drivers



Overzicht embedded Rust

Preemptive rust

- [cortex-m-rtic](#) crate
  - Preemptive taken, op hardware (NVIC) gebaseerd, niet echt een OS
- Verschillende RTOS'en zoals FreeRTOS

Cooperatieve scheduler met rust

- async/await
- embassy-rs/embassy: Modern embedded framework, using Rust and async. (github.com)
  - Soort van universele HAL / scheduler voor verschillende fabrikanten. Heeft Async rust ook in een laag gezet.

Features

- **Tasks** as the unit of concurrency [^1]. Tasks can be *event triggered* (fired in response to asynchronous stimuli) or spawned by the application on demand.
- **Message passing** between tasks. Specifically, messages can be passed to software tasks at spawn time.
- **A timer queue** [^2]. Software tasks can be scheduled to run at some time in the future. This feature can be used to implement periodic tasks.
- Support for prioritization of tasks and, thus, **preemptive multitasking**.
- **Efficient and data race free memory sharing** through fine grained *priority based* critical sections [^1].
- **Deadlock free execution** guaranteed at compile time. This is a stronger guarantee than what's provided by the standard `Mutex` abstraction.
- **Minimal scheduling overhead**. The task scheduler has minimal software footprint; the hardware does the bulk of the scheduling.
- **Highly efficient memory usage**: All the tasks share a single call stack and there's no hard dependency on a dynamic memory allocator.
- **All Cortex-M devices are fully supported**.
- This task model is amenable to known WCET (Worst Case Execution Time) analysis and scheduling analysis techniques.

Wat mist er nog voor de STM32

Overview

feature	no_std	std
heap (dynamic memory)	*	✓
collections (Vec, HashMap, etc)	**	✓
stack overflow protection	✗	✓
runs init code before main	✗	✓
libstd available	✗	✓
libcore available	✓	✓
writing firmware, kernel, or bootloader code	✓	✗

* Only if you use the `alloc` crate and use a suitable allocator like alloc-cortex-m.

** Only if you use the `collections` crate and configure a global default allocator.

Wat mist er nog - libstd

Zonder (libstd)OS Geen:

- Std Threads
- Std Message passing
- Std mutex
- Fearless concurrency

Maar… lijkt voor de esp32 wel geïmplementeerd te zijn (boven op esp-idf)
[Using the Standard Library (std) - The Rust on ESP Book (esp-rs.org)]

Embedded Rust Bronnen

[GitHub - rust-embedded/awesome-embedded-rust: Curated list of resources for Embedded and Low-level development in the Rust programming language]

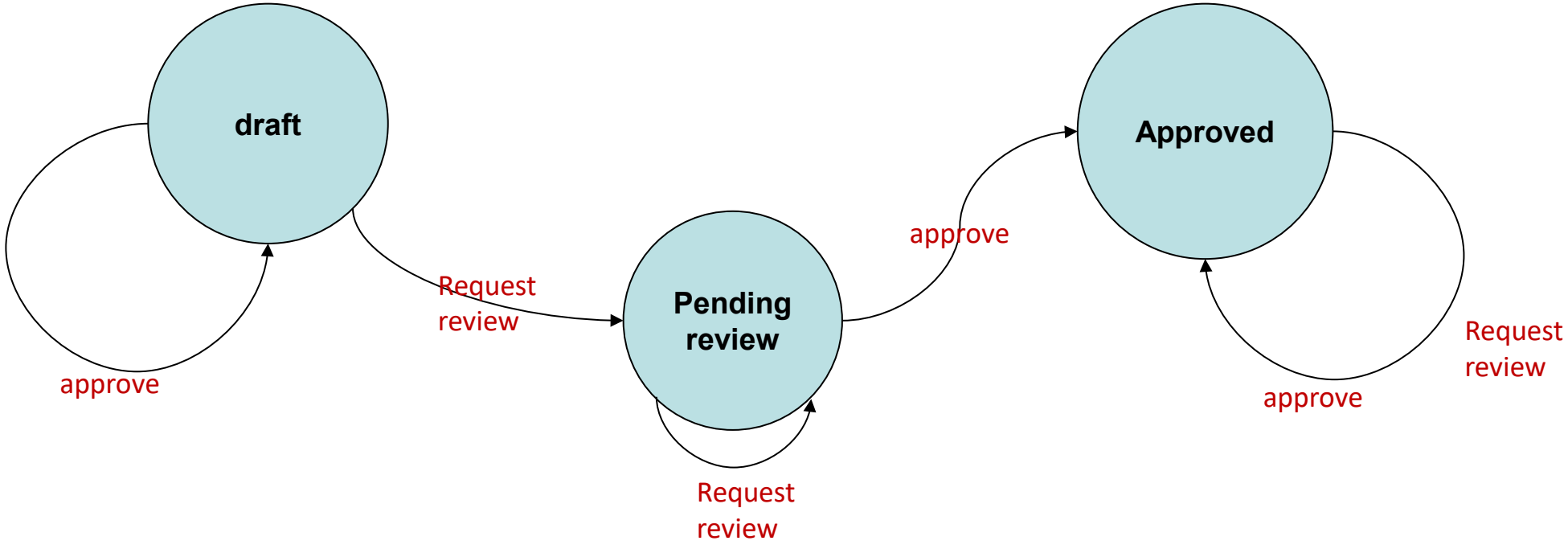
Op deze repo staan links naar vrijwel alle embedded rust gerelateerde dingen.

Opdracht week 8

- Implementeer een statemachine op basis van het “state pattern” ontwerppatroon
  - Op de stm32
  - Door een stoplicht/brug te simuleren
  - Hiermee laat je het gebruik van traits, structs , results e.d. zien en stuur je wat pinnen aan.

Wordt uitgelegd in het hoofdstuk Implementing an Object-Oriented Design Pattern tot paragraaf (maar niet met) trade-offs of the State Pattern

Voorbeeld boek



Wordt uitgelegd in het hoofdstuk Implementing an Object-Oriented Design Pattern tot paragraaf (maar niet met) trade-offs of the State Pattern

Aan de slag!

Aan de slag met [Opdrachten_Week_8.pdf]