Применение Rust для архитектуры RISC-V

План

- Rust и embedded
- RISC-V
- Rust и RISC-V

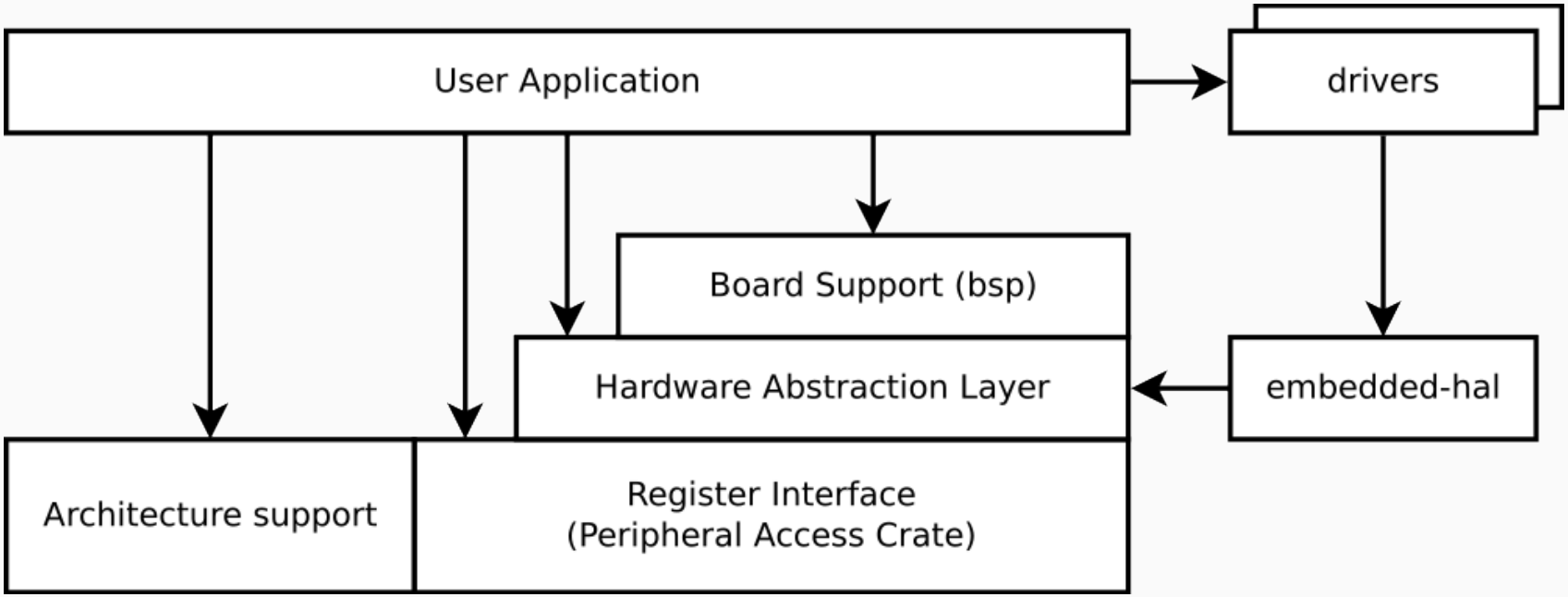
Rust

- Для тех, кто устал от языка C
- Многие распространённые ошибки отлавливаются на этапе компиляции
- Мощная система типов и стандартная библиотека
- Memory-safety без сборщика мусора
- Безопасное взаимодействие между потоками (обработчиками прерываний)
- Производительность на уровне C/C++
- Удобный менеджер пакетов (”крейтов”) и система сборки
- Юнит-тесты и генератор документации из коробки
- Замечательное сообщество и возможность вносить изменения в язык/компилятор

Embedded Rust

- Отлаживать код сложнее, чем для десктопа
  Лучше находить ошибки ещё на этапе компиляции
- Строгая типизация при обращении к регистрам и их полям
- Многие правила MISRA C выполнены ”из коробки”
- Для STM32 порог вхождения ниже, чем в CubeIDE
- Единый HAL
  - переиспользование драйверов внешних устройств
  - хорошая портируемость кода
- Иногда поддержка МК в Rust лучше, чем у вендора

Embedded Rust



https://github.com/rust-embedded/awesome-embedded-rust

”Стили”разработки:

- Embedded Working Group
- Все остальные

Поддержка новых МК

Register Interface: генерируется из SVD файла от вендора

- иногда такого файла нет
- иногда в файле есть ошибки: `svdpatch`

Hardware Abstraction Layer: пишется вручную

- драйверы для внутренних устройств МК, предоставляющие интерфейс `embedded-hal`
- пока что нельзя просто подключить драйвер внутреннего периферийного устройства из другого hal (за исключением USB для STM32), но можно скопипастить

Драйверы внешних устройств: пишутся один раз, работают везде

- даже если ваш МК – это Raspberry Pi под управлением Linux

Инструменты

- Сборка: cargo
- Отладка и запуск: gdb
- openocd/JLink/bmp/...
- IDE: IntelliJ IDEA, Visual Studio Code

```
rustup target add thumbv7m-none-eabi
git clone https://github.com/TeXitoi/blue-pill-quickstart
cd blue-pill-quickstart
cargo run
```

Архитектура RISC-V

- Простой набор инструкций
- Стандартизованные расширения RV32IMAFDC (==RV32GC)
  - I: базовые целочисленные операции
  - M: умножение и деление
  - A: атомарные операции
  - F, D: операции для работы с плавающей точкой
  - C: compressed инструкции: 2 байта вместо 4
- Отсутствие зоопарка режимов работы
  - M: machine mode, доступна вся физическая память
  - S: system mode, для ядер ОС + страничная адресация
  - U: user mode
  - H: hypervisor mode, стандарт ещё не финализирован
- Контроллер прерываний является частью стандарта
- В страничной адресации учтён легаси-опыт x86

Hardware

Чипы:

- HiFive1: FE310-G000, RV32IMAC
- HiFive Unleashed: FU540-C000, RV64IMAFDC, 5 ядер ($999)
- HiFive1 Rev B: FE310-G002, RV32IMAC + User mode ($59)
- LoFive R1: FE310-G002 ($23)
- Kendryte K210: RV64IMAFDC, 2 ядра, 6MB RAM ($8-$20)
- GD32VF103: RV32IMAC ($5-$16)
- VEGABoard: RV32M1 ($43)

FPGA:

- Rocket
- picorv32: RV32I[M][C]
- HeavyX: RV32I
- Syntacore SCR1-SCR7 (спб!)
- lowRISC
- SiFive

Поддержка RISC-V

Языки: C/C++, Rust, Go, Ocaml, ...

RTOS: FreeRTOS, Zephyr, NuttX, seL4, ...

Поддержка в апстриме: gcc, llvm, openocd, gdb, coreboot, U-Boot, ...

Каждый производитель городит свой SDK, в том числе свою версию gcc/openocd

Поддержка RISC-V в Rust

Поддержка таргетов: RV32{I, IMC, IMAC}, RV64{IMAC, GC}
`riscv, riscv-rt`

Поддержка железа: `https://github.com/riscv-rust`

- HiFive1 + RevB: pac+hal+bsp
  `e310x, e310x-hal, hifive1, riscv-rust-quickstart`
- Kendryte K210: pac+hal*
  `k210-pac, k210-hal, k210-example`
- GD32VF103: pac+hal*
  `gd32vf103-pac, gd32vf103-hal, gd32vf103xx-hal`
- picorv32
  `picosoc-example`
  `picorv32, picorv32-rt` https://github.com/ilya-epifanov

* реализовано не всё

Summary

Литература:

- The Rust Programming Language (`rustup doc --book`)
- The embedded Rust book (`rustup doc --embedded-book`)
- The Discovery book (про STM32 F3 Discovery)

GitHub:

- `https://github.com/rust-embedded`
- `https://github.com/riscv-rust`
- `https://github.com/stm32-rs`

Презентация: `https://tinyurl.com/rust-riscv-2019`

Вопросы?

Поддержка новых архитектур

Чего ещё нет?

- AVR (Arduino)
- Xtensa (ESP32)

Что нужно?

- Поддержка архитектуры в апстриме LLVM
- PR в `rust-lang/llvm-project` – принести изменения из апстрима
- PR в `rust-lang/rust` – обновить сабмодуль llvm-project и добавить новые таргеты
- дождаться попадания в nightly (1 день) или stable (2 месяца)

`riscv64gc-unknown-none-elf`: от идеи до попадания в nightly – 15 дней
`riscv32i-unknown-none-elf`: от PR до попадания в nightly – неделя