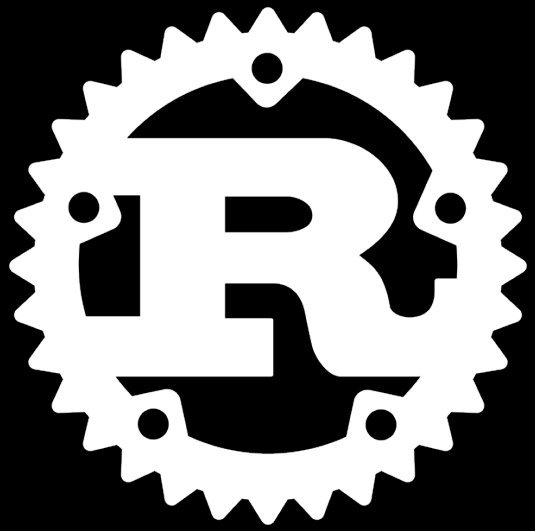
Как Rust не даёт выстрелить в ногу при разработке под микроконтроллеры



...или почему отметка Powered by Rust должна стоять на каждой safety critical системе

О чём поговорим

О чём поговорим

- зачем нужно отказаться от C

О чём поговорим

- зачем нужно отказаться от C
- почему Rust должен стать стандартом

О чём поговорим

- зачем нужно отказаться от C
- почему Rust должен стать стандартом
- safety critical

О чём поговорим

- зачем нужно отказаться от C
- почему Rust должен стать стандартом
- safety critical
- сравним популярные экосистемы

Как дела у ардуинщиков?

Да кто такой этот ваш Rust

Да кто такой этот ваш Rust

- Ownership
- &borrowing
- 'lifetimes

Да кто такой этот ваш Rust

- Ownership
- &borrowing
- 'lifetimes

- zero cost abstractions {}

Да кто такой этот ваш Rust

- Ownership
- &borrowing
- 'lifetimes

- zero cost abstractions {}
- Rust: Structural + Functional

Да кто такой этот ваш Rust

- Ownership
- &borrowing
- 'lifetimes

- zero cost abstractions {}
- Rust: Structural + Functional
- no UB in the *safe mode

*rustc vs me >50:0.5!

Да кто такой этот ваш Rust

- Ownership
- &borrowing
- 'lifetimes

- zero cost abstractions {}
- Rust: Structural + Functional
- no UB in the *safe mode
- cargo ❤

*rustc vs me >50:0.5!

Дайте два!

Дайте два!

- null!

Дайте два!

- null!
- uninitialized data access

Дайте два!

- null!
- uninitialized data access
- integer overflow (debug)

Дайте два!

- null!
- uninitialized data access
- integer overflow (debug)
- integer division by zero

Дайте два!

- null!
- uninitialized data access
- integer overflow (debug)
- integer division by zero
- boolean casts

Дайте два!

- null!
- uninitialized data access
- integer overflow (debug)
- integer division by zero
- boolean casts
- buffer overflow (panics)

Дайте два!

- null!
- uninitialized data access
- integer overflow (debug)
- integer division by zero
- boolean casts
- buffer overflow (panics)
- dangling pointer

Дайте два!

- null!
- uninitialized data access
- integer overflow (debug)
- integer division by zero
- boolean casts
- buffer overflow (panics)
- dangling pointer
- memory leaks (*)

Дайте два!

- null!
- uninitialized data access
- integer overflow (debug)
- integer division by zero
- boolean casts
- buffer overflow (panics)
- dangling pointer
- memory leaks (*)
- double free

Дайте два!

- null!
- uninitialized data access
- integer overflow (debug)
- integer division by zero
- boolean casts
- buffer overflow (panics)
- dangling pointer
- memory leaks (*)
- double free
- use after free

Дайте два!

- null!
- uninitialized data access
- integer overflow (debug)
- integer division by zero
- boolean casts
- buffer overflow (panics)
- dangling pointer
- memory leaks (*)
- double free
- use after free
- race condition

Забудем C как страшный сон?

Забудем C как страшный сон?

- очень большая кодовая база

Забудем C как страшный сон?

- очень большая кодовая база

- проприетарные «железки» и библиотеки

Забудем C как страшный сон?

- очень большая кодовая база
- проприетарные «железки» и библиотеки
- мало специалистов

Забудем C как страшный сон?

- очень большая кодовая база

- проприетарные «железки» и библиотеки

- мало специалистов
- мало компаний знает про Rust

Забудем C как страшный сон?

- очень большая кодовая база

- проприетарные «железки» и библиотеки

- мало специалистов
- мало компаний знает про Rust
- Rust не имеет стандарта (*)

Мы знаем, что существует множество правил техники безопасности, руководств и стандартов, которыми мы руководствуемся, и мы были уверены в превосходной репутации данных машин.

<???>

Мы знаем, что существует множество правил техники безопасности, руководств и стандартов, которыми мы руководствуемся, и мы были уверены в превосходной репутации данных машин.

FDA, разработчик Therac-25

Therac 25

- race condition
- (last win)
- integer division by zero
- boolean casts

Therac 25

- race condition
- (last win)
- integer division by zero
- boolean casts

- эти ошибки в ПО повлекли за собой человеческие смерти

Therac 25

- race condition
- (last win)
- integer division by zero
- boolean casts

- эти ошибки в ПО повлекли за собой человеческие смерти

- люди продолжают гибнуть, но мы можем этого не замечать

Что Rust может изменить?

- большинство правил MISRA-C «встроены» в rustc

Что Rust может изменить?

- большинство правил MISRA-C «встроены» в rustc

- мы можем изолировать unsafe внутри safe абстракций...

Что Rust может изменить?

- большинство правил MISRA-C «встроены» в rustc

- мы можем изолировать unsafe внутри safe абстракций...

... в том числе и работу с микроконтроллером и периферией!

Экосистема

I. SVD (System View Description)

Экосистема

I. SVD (System View Description)

II. unsafe PAC (Peripheral Access Crate)

Пример использования PAC

```
pwm.load.write(|w| unsafe {
    w.load().bits(263)
});

pwm.cmpa.write(|w| unsafe {
    w.compa().bits(64)
});
```

Экосистема

I. SVD (System View Description)

II. unsafe PAC (Peripheral Access Crate)

III. safe HAL (Hardware Abstraction Layer)

Экосистема

I. SVD (System View Description)

II. unsafe PAC (Peripheral Access Crate)

III. safe HAL (Hardware Abstraction Layer)

- использование HAL не создаёт лишний оверхед

Крейт embedded-hal

Крейт embedded-hal

Некоторые трейты:

- InputPin, OutputPin

Крейт embedded-hal

Некоторые трейты:

- InputPin, OutputPin
- Pwm

Крейт embedded-hal

Некоторые трейты:

- InputPin, OutputPin
- Pwm
- DelayMs, DelayUs

Крейт embedded-hal

Некоторые трейты:

- InputPin, OutputPin
- Pwm
- DelayMs, DelayUs
- any protocol: Read, Write
- (uart, spi, i2c)

InputPin, OutputPin

```
fn is_high(&self) -> bool;
fn is_low(&self) -> bool;

fn set_low(&mut self);
fn set_high(&mut self);
```

Pwm

```
fn disable|enable(&mut self, channel:
Self::Channel);

fn get|set_period(&self) ->
Self::Time;

fn get|set_duty(&self, channel:
Self::Channel) -> Self::Duty;

fn get_max_duty(&self) -> Self::Duty;
```

DelayMs, DelayUs

```
fn delay_ms(&mut self, ms: UXX);

fn delay_us(&mut self, us: UXX)
```

Read, Write

- методы различаются от протокола к протоколу

```
fn write(
    &mut self, addr: u8, bytes: &[u8]
) -> Result<(), Self::Error>

fn write(
    &mut self, words: &[W]
) -> Result<(), Self::Error>
```

Реализации embedded-hal

https://github.com/rust-embedded/awesome-embedded-rust#hal-implementation-crates

Проблемы на начало 2020

- отсутсвует стандарт для реализаций HAL

Проблемы на начало 2020

- отсутсвует стандарт для реализаций HAL

- как следствие, каждый сам определяет API конфигурирования периферии

Возможные решения

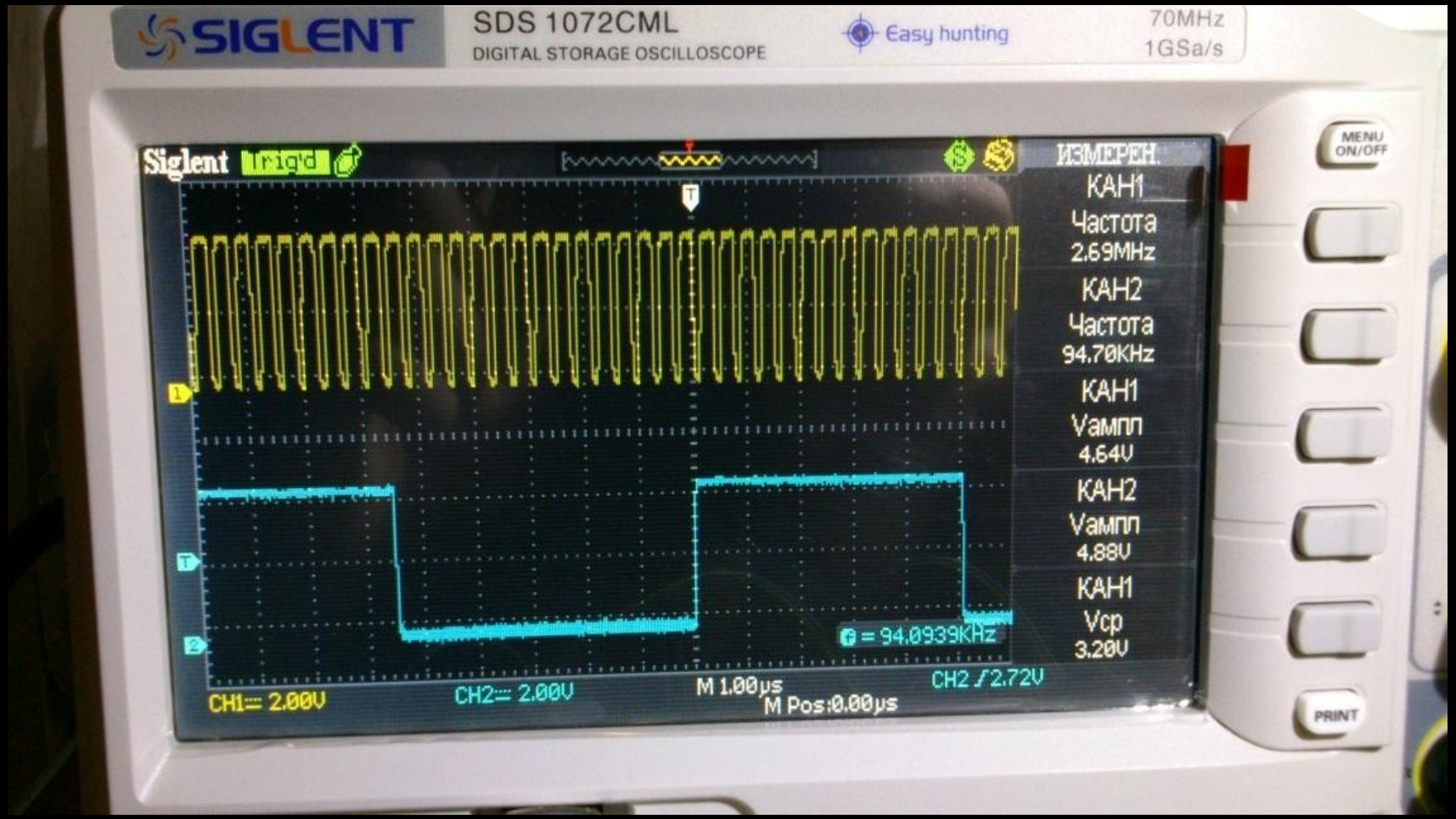
- написать прикладной адаптор, соединяющий разные HAL с одним представлением

Возможные решения

- написать прикладной адаптор, соединяющий разные HAL с одним представлением

- использовать условную компиляцию для разных МК

Привет ардуинщики!



Привет ардуинщики!

```
void pinMode(pin_n, type);
- type: uint8_t
        INPUT(0x0) | OUTPUT(0x1)
```

Привет ардуинщики!

```
void pinMode(pin_n, type);
- type: uint8_t
        INPUT(0x0) | OUTPUT(0x1)

void digitalRead(pin_n, state);
- state: uint8_t
         LOW(0x0) | HIGH(0x1)
```

Runtime проверки digitalWrite

```
if (port == NOT_A_PIN) return;
```

Runtime проверки digitalWrite

```
if (port == NOT_A_PIN) return;

// If the pin that support PWM
output, we need to turn it off before
doing a digital write.

if (timer != NOT_ON_TIMER) {
    turnOffPWM(timer);
}
```

Кот в мешке в digitalWrite

Runtime проверки digitalWrite

```
uint8_t oldSREG = SREG;
cli();

if (val == LOW) {
    *out &= ~bit;
} else {
    *out |= bit;
}

SREG = oldSREG;
```

Итог

- код на C работает быстро, но его написание требует высокой дисциплины

Итог

- код на C работает быстро, но его написание требует высокой дисциплины
- количество ловушек в Arduino сильно меньше, платим производительностью

Итог

- код на C работает быстро, но его написание требует высокой дисциплины
- количество ловушек в Arduino сильно меньше, платим производительностью
- Rust берёт от этих подходов только лучшие черты

Best practice (*)

- не используйте unsafe в бизнес логике прошивки МК

Best practice (*)

- не используйте unsafe в бизнес логике прошивки МК

- инкапсулируйте unsafe внутрь безопасных абстракций

Best practice (*)

- не используйте unsafe в бизнес логике прошивки МК

- инкапсулируйте unsafe внутрь безопасных абстракций

- не доверяйте никому кроме компилятора

«Халява» (с)

https://rust-embedded.github.io/book/
https://t.me/ilyavenner