Part 2

Embedded Rust Workshop

tweede golf

web / security / embedded

Recap

- Rust’s embedded ecosystem
  - HALs, PACs, embedded-hal
- Portable drivers in Rust
  - Traits
  - Generics

Our day

- **Ask questions anytime!**
- Interrupt me when needed
- Help each other out

We'll see how far we get

Our day

- Questions on reading material
- The RTIC runtime
- Exercise 2A: RTIC basics

Bonus material:

- Rust in IoT
- Exercise 2B: Device-host communication

Part 2A

RTIC

tweede golf

web / security / embedded

Sharing Data

- Rust is pedantic about sharing globals
- Locks everywhere!

Global state declaration

```rust
use cortex_m::interrupt::Mutex;
// Flags for events
static BUTTON_1_PRESSED: AtomicBool = AtomicBool::new(false);
static BUTTON_1_RELEASED: AtomicBool = AtomicBool::new(true);
static TIMER0_FIRED: AtomicBool = AtomicBool::new(false);

// Handle to the GPIOTE peripheral. Uninitialized on reset.
// Must be initialized before use
static GPIOTE_HANDLE: Mutex<RefCell<Option<Gpiote>>> = Mutex::new(RefCell::new(None));

// Handle to the TIMER0 peripheral. Uninitialized on reset.
// Must be initialized before use
static TIMER0_HANDLE: Mutex<RefCell<Option<Timer<pac::TIMER0, Periodic>>>> =
    Mutex::new(RefCell::new(None));
```

- Atomics for flags
- `Mutex<RefCell<Option<T>>>` for safe synchronization

Global state initialization

```rust
// Initialize the TIMER0 and GPIOTE handles, passing the initialized
// peripherals.
use cortex_m::interrupt::{free as interrupt_free, CriticalSection};
interrupt_free(|cs: &CriticalSection| {
    // Interrupts are disabled globally in this block
    TIMER0_HANDLE.borrow(cs).replace(Some(timer0));
    GPIOTE_HANDLE.borrow(cs).replace(Some(gpiote));
});
```

- `cortex_m::interrupt::free`needed to mutate data in CS
- Disables **all** interrupts

tweede golf

Global state in ISR

```
#[interrupt]
// TIMER0 interrupt service routine
fn TIMER0() {
    use cortex_m::interrupt::{free as interrupt_free, CriticalSection};
    interrupt_free(|cs: &CriticalSection| {
        if let Some(ref timer0) = TIMER0_HANDLE.borrow(&cs).borrow().deref() {
            // Check whether capture/compare register 0 was reached
            if timer0.event_compare_cc0().read().bits() != 0x00u32 {
                // Raise flag that timer has fired
                TIMER0_FIRED.store(true, Relaxed);
                // Reset cc0, so as to prevent looping forever
                timer0.event_compare_cc0().write(|w| unsafe { w.bits(0) })
            }
        };
    });
}
```

tweede golf

Questions so far?

?

Real-Time Interrupt-driven Concurrency (Née RTFM)

- Divide application into tasks
- Heavily uses interrupts to schedule tasks
- Handles passing global resources

Lock only when pre-emption might cause trouble

RTIC features

- Message passing
- Task scheduling
- Deadlock-free execution
- Works on all cortex-m devices
- (multi-core support)
- Lots of control

RTIC trade offs

Heavy on the macros

- Rust analyzer doesn’t like macros
- Vague compiler errors involving locks
- Compiler still helpful though

Questions so far?

?

RTIC app outline

- App attribute
- Init task
- Idle task
- Hardware tasks
- Software tasks

[Example]

RTIC app attribute

- Point RTIC to PAC
- Monotonic timer for task scheduling
- Procedural macro

Code gets adapted at build!

```
#[rtic::app(
    device=firmware::hal::pac,
    peripherals=true,
    monotonic=rtic::cyccnt::CYCCNT
)]
const APP: () = {
```

RTIC resources

- Shared between tasks
- Some initialized by runtime
- Others lazily initialized in init

```rust
struct Resources {
    gpiote: Gpiote,
    timer0: Timer<TIMER0, Periodic>,
    led1: Pin<Output<PushPull>>,
}
```

RTIC init

- Initialize late resources
- Peripherals in `ctx.device`
- Use PAC and HAL
- Interrupts disabled

```rust
#[init]
fn init(ctx: init::Context) -> init::LateResources {
    let port0 = Parts::new(ctx.device.P0);

    // Init pins
    let led1 = port0.p0_13.into_push_pull_output(Level::High).degrade();
    let btn1 = port0.p0_11.into_pullup_input().degrade();

    // Configure GPIOTE
    let gpiote = Gpiote::new(ctx.device.GPIOTE);
    gpiote
        .channel0()
        .input_pin(&btn1)
        .hi_to_lo()
        .enable_interrupt();

    // Initialize TIMER0
    let mut timer0 = Timer::periodic(ctx.device.TIMER0);
    timer0.enable_interrupt();
    timer0.start(1_000_000u32); // 1000 ticks = 1 ms

    // Return the resources
    init::LateResources {
        led1,
        gpiote,
        timer0,
    }
}
```

RTIC idle task

- Default task
- Sleep, mostly (default)
- Pre-empted by other tasks

```rust
#[idle]
fn idle(_ctx: idle::Context) -> ! {
    loop {
        // Go to sleep, waiting for an interrupt
        cortex_m::asm::wfi();
    }
}
```

RTIC software task

- Capacity
- Priority
- Resources declared
- Message passing
- Task context

Resources are `&mut`!

```rust
#[task(
    capacity = 5,
    priority = 1, // Very low priority
    resources = [led1]
)]
fn set_led1_state(ctx: set_led1_state::Context, enabled: bool) {
    if enabled {
        ctx.resources.led1.set_low().unwrap();
    } else {
        ctx.resources.led1.set_high().unwrap();
    }
}
```

tweede golf

RTIC interrupt declaration

- Used to spawn software tasks
- Need as many as there are software tasks

```
extern "C" {
    // Software interrupt 0 / Event generator unit 0
    fn SWI0_EGU0();
    // Software interrupt 1 / Event generator unit 1
    fn SWI1_EGU1();
    // Software interrupt 2 / Event generator unit 2
    fn SWI2_EGU2();
}
```

RTIC hardware task

- Binds hardware interrupt
- High priority
- Resources
- Spawned SW tasks
- Task context

```rust
#[task(
    binds = TIMER0,
    priority = 99, // Very high priority
    resources = [timer0],
    spawn = [set_led1_state]
)]
fn on_timer0(ctx: on_timer0::Context) {
    let timer0 = ctx.resources.timer0;
    if timer0.event_compare_cc0().read().bits() != 0x00u32 {
        timer0.event_compare_cc0().write(|w| unsafe { w.bits(0) });
        // Try to spawn set_led1_state. If its queue is full, we do nothing.
        let _ = ctx.spawn.set_led1_state(false);
    }
}
```

Questions so far?

?

RTIC task scheduling (init)

- Enable cycle counter in init
- SW Task: ctx.scheduled
- HW Task/init: ctx.start

```
#[init]
fn init(ctx: init::Context) -> init::LateResources {
    // Enable cycle counter
    ctx.core.DWT.enable_cycle_counter();

    // Init peripherals...

    let now = ctx.start;
    // Schedule toggle_led_2 task
    ctx.schedule.toggle_led_2(now, true).unwrap();

    init::LateResources {
        // The resources
    }
}
```

RTIC task scheduling (task)

- Enable cycle counter in init
- SW Task: ctx.scheduled
- HW Task/init: ctx.start

```rust
#[task(
    capacity = 5,
    priority = 2,
    resources = [led2],
    schedule = [toggle_led_2]
)]
fn toggle_led_2(ctx: toggle_led_2::Context, enabled: bool) {
    let led2 = ctx.resources.led2;
    if enabled {
        led2.set_high().unwrap(); // Disable
    } else {
        led2.set_low().unwrap(); // Enable
    }

    // Use ctx.start in HW task and init
    let task_scheduled_at = ctx.scheduled;
    ctx.schedule
        .toggle_led_2(task_scheduled_at + 10_000_000u32.cycles(), !enabled)
        .ok();
}
```

tweede golf

Questions so far?

?

tweede golf

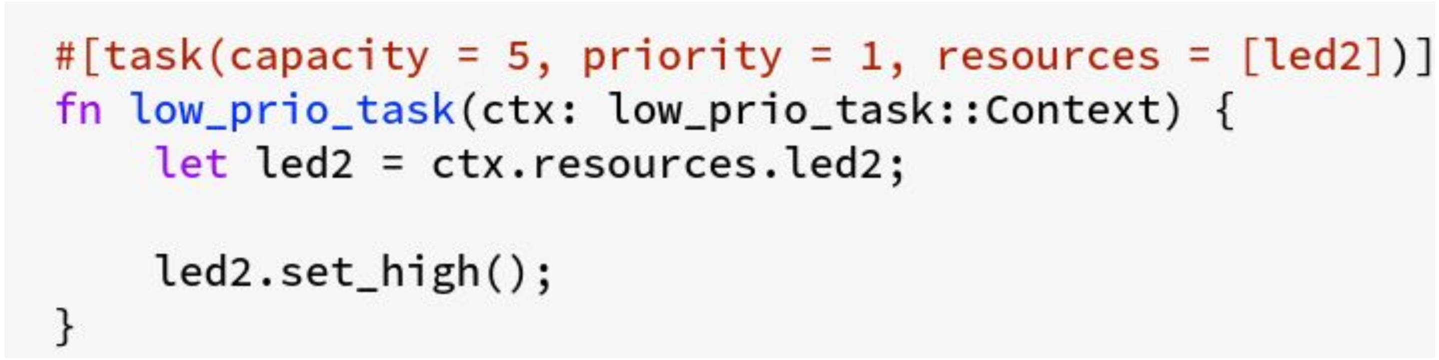
RTIC resource locks

- Task preemption
- Lock needed for lower-prio task
- Temporarily increase task priority
- Only for common resources

Compiler error is pretty bad, beware!

RTIC resource locks

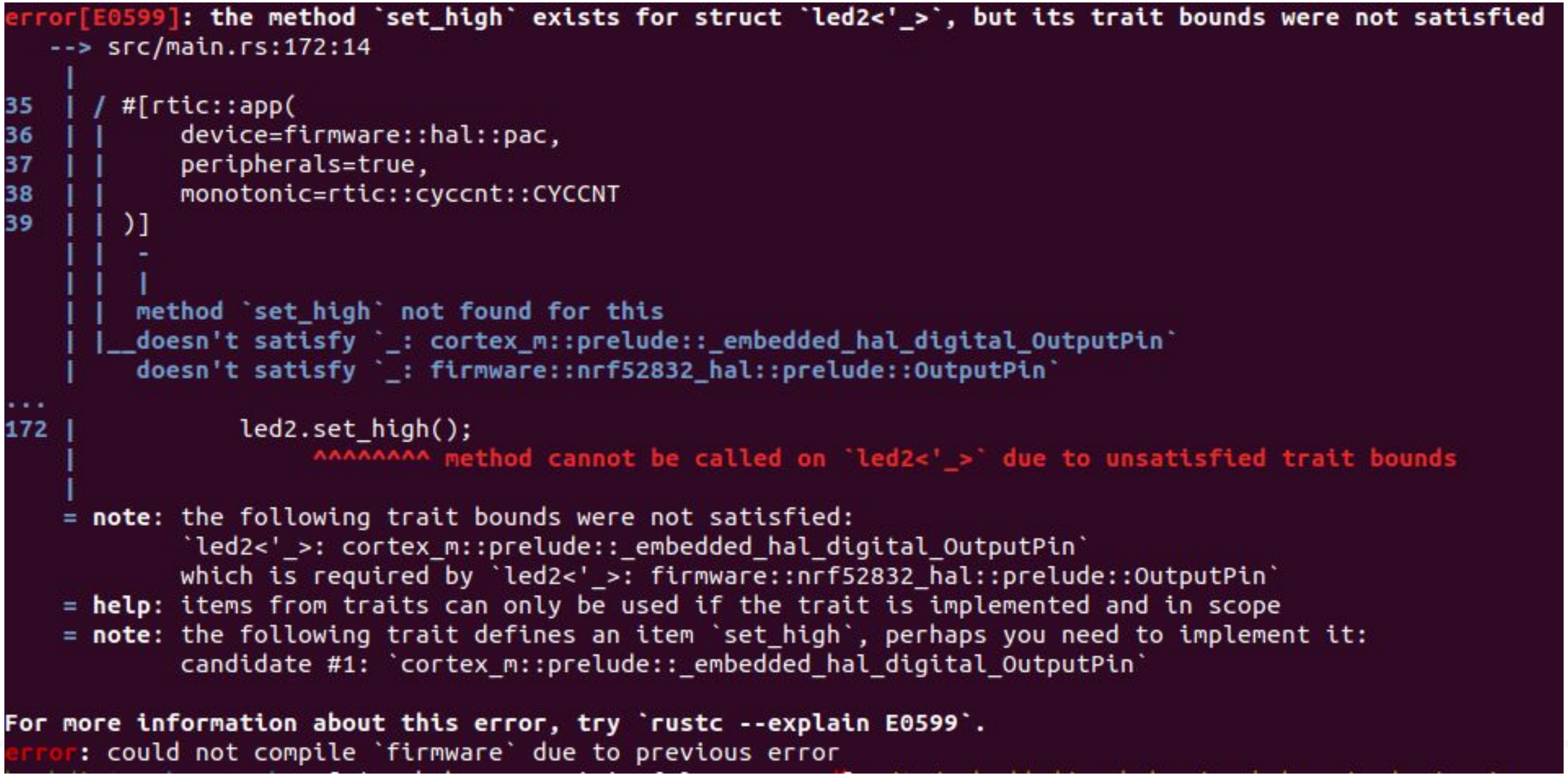
Note: `led2` **is declared by** `toggle_led2` **task, prio 2**



```rust
#[task(capacity = 5, priority = 1, resources = [led2])]
fn low_prio_task(ctx: low_prio_task::Context) {
    let led2 = ctx.resources.led2;

    led2.set_high();
}
```

RTIC resource locks



```
error[E0599]: the method `set_high` exists for struct `led2<'_>`, but its trait bounds were not satisfied
   --> src/main.rs:172:14
    |
35  | / #[rtic::app(
36  | |     device=firmware::hal::pac,
37  | |     peripherals=true,
38  | |     monotonic=rtic::cyccnt::CYCCNT
39  | | )]
    | |  -
    | |  |
    | |  method `set_high` not found for this
    | |__doesn't satisfy `_: cortex_m::prelude::_embedded_hal_digital_OutputPin`
    |    doesn't satisfy `_: firmware::nrf52832_hal::prelude::OutputPin`
...
172 |             led2.set_high();
    |                  ^^^^^^^^ method cannot be called on `led2<'_>` due to unsatisfied trait bounds
    |
    = note: the following trait bounds were not satisfied:
            `led2<'_>: cortex_m::prelude::_embedded_hal_digital_OutputPin`
            which is required by `led2<'_>: firmware::nrf52832_hal::prelude::OutputPin`
    = help: items from traits can only be used if the trait is implemented and in scope
    = note: the following trait defines an item `set_high`, perhaps you need to implement it:
            candidate #1: `cortex_m::prelude::_embedded_hal_digital_OutputPin`

For more information about this error, try `rustc --explain E0599`.
error: could not compile `firmware` due to previous error
```

RTIC resource locks

```rust
#[task(capacity = 5, priority = 1, resources = [led2])]
fn low_prio_task(ctx: low_prio_task::Context) {
    // Locking mutates
    let mut led2 = ctx.resources.led2;

    led2.lock(|led2_lock| {
        led2_lock.set_low().unwrap();
    });
}
```

tweede golf

Questions so far?

?

Exercise 2A

Instructions on **sodaq.workshop.tweede.golf**

Don’t forget to `git pull`!

Exercice 2A round up

- Show your code!
- Any questions?

Part 2B

Rust in IoT

tweede golf

web / security / embedded

Demo: A bigger Project in Rust

- Project structure
- Sharing code
- Serde and postcard
- Command-line application

Questions so far?

?

Exercise 2B

Instructions on **sodaq.workshop.tweede.golf**

Exercice 2B round up

- Show your code!
- Any questions?

AMA

?

tweede golf

tweede golf

web / security / embedded

Castellastraat 26, 6512 EX Nijmegen
info@tweedegolf.com
024 3010 484