Enabling the usage of `embedded-hal-async` based drivers in the Tock kernel

Alexandru Radovici

Motivation

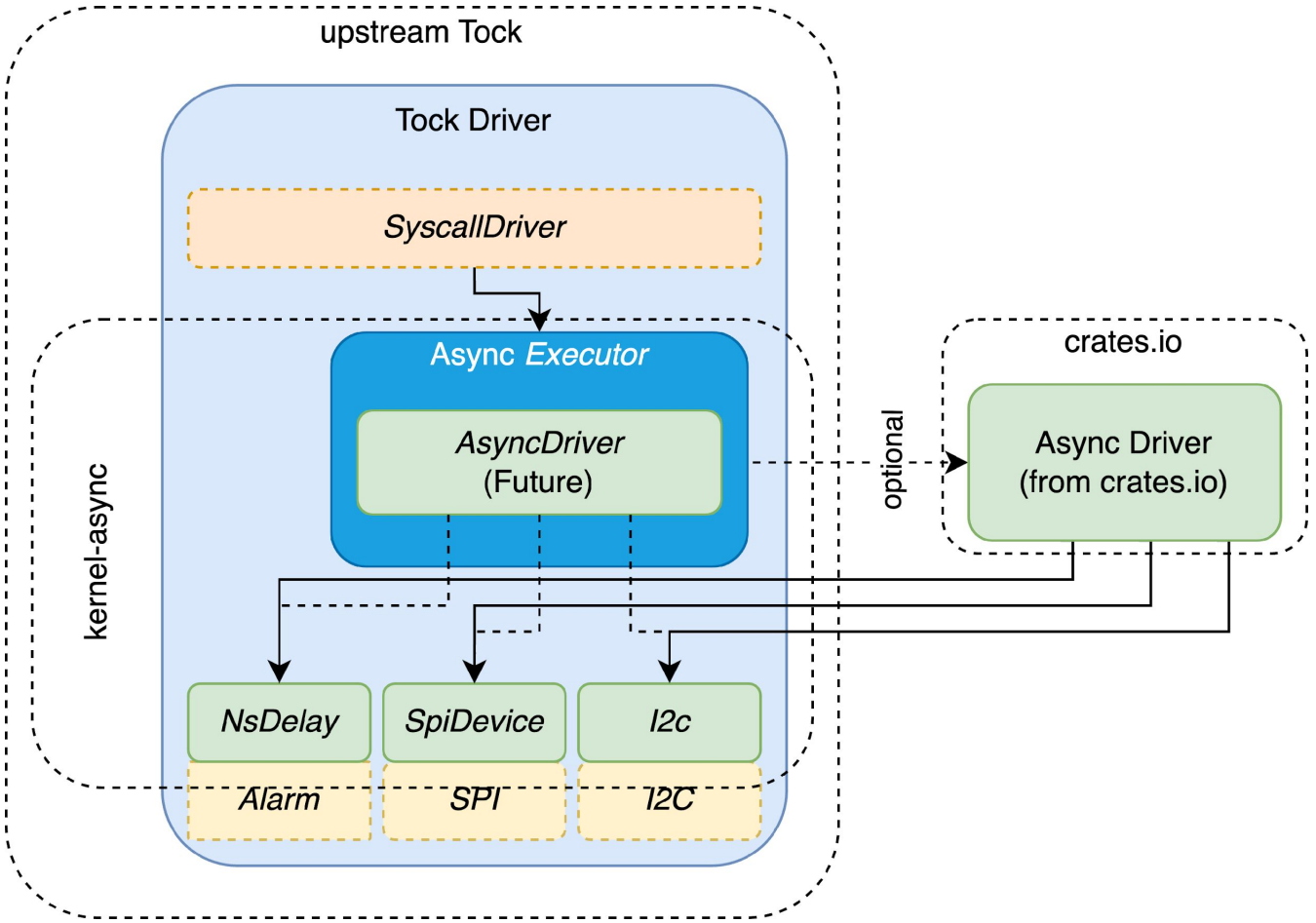
- Tock is *manually* asynchronous
- The Embedded Rust community is used to using `async`/`.await`
- There are **269** crates using the `embedded-hal-async`
  - `embedded-hal` is used by **1343** crates 🤔

Requirements

- Implementation of an executor
- API for drivers to define `async` blocks
- Implementation of the `embedded-hal-async` traits using Tock's infrastructure
- Use only static allocation (no `Box` / `BoxedFuture`)
- No external dependencies (except `embedded-hal-async`)
- Use stable Rust
- No (*unnecessary*) unsafe code

Architecture

- `kernel-async` crate
- Has one external dependency, `embedded-hal-async`
- Provides the implementation of the `embedded-hal-async` traits
  - `DelayNs`



`AsyncDriver` trait

- Implemented by drivers
- Provides access to driver data
- Has to be `'static`
- `run` returns the `async fn` or block

```rust
pub trait AsyncDriver {
    type F: Future + 'static;

    /// The asynchronous part of
    /// the driver
    fn run(&'static self) -> Self::F;

    /// Optional methods that is used by the [`Executor`] to
    /// notify the driver that the execution of the future
    /// ended.
    fn done(&self, _value: <Self::F as IntoFuture>::Output) {}
} trait AsyncDriver
```

`Executor` struct and `Runner` trait

- Holds and executes the `Future`
- Drivers receive a reference the `Executor`
- Circular type due to the generic argument
- Drivers actually receive a **`dyn`** `Runner`

```rust
pub struct Executor<D: AsyncDriver + 'static> {
    future: MapCell<D::F>,
    waker_vtable: &'static RawWakerVTable,
    driver: &'static D,
}
```

```rust
impl<D: AsyncDriver> Runner for Executor<D> {
    fn execute(&'static self) -> Result<(), ErrorCode> {
        if self.future.is_none() {
            self.future.replace(val: self.driver.run());
            self.poll();
            Ok(())
        } else {
            Err(ErrorCode::BUSY)
        }
    }
} impl Runner for Executor<D>
```

Async Hello World

- Prints *Hello*
- Waits for 1s
- Prints *awaited*

```
impl<A: Alarm<'static> + 'static> AsyncDriver for HelloPrintDriver<A> {
    type F = impl Future<Output = ()> + 'static;

    fn run(&'static self) -> Self::F {
        async {
            // you should not be able to run two futures at the same time
            // so this should never panic
            let mut delay_instance = self.delay.get_instance().unwrap();
            // loop {
            debug!("Hello");
            delay_instance.delay_ns(1_000_000_000).await;
            debug!("awaited");
            // }
        }
    }

    fn done(&self, _value: ()) {
        debug!("done");
        self.runner.get().unwrap().execute().unwrap();
    }
}
```

In-driver usage

- Drivers can perform one single action at a time
- This is what most of the Tock drivers do

```
match command_num {
    0 => CommandReturn::success(),
    1 => {
        if let Err(err) = self.runner.get().unwrap().execute() {
            CommandReturn::failure(err)
        } else {
            CommandReturn::success()
        }
    }
    _ => CommandReturn::failure(ErrorCode::NOSUPPORT),
}
```

Instantiation

- This is in the board crate (usually in *main.rs*)

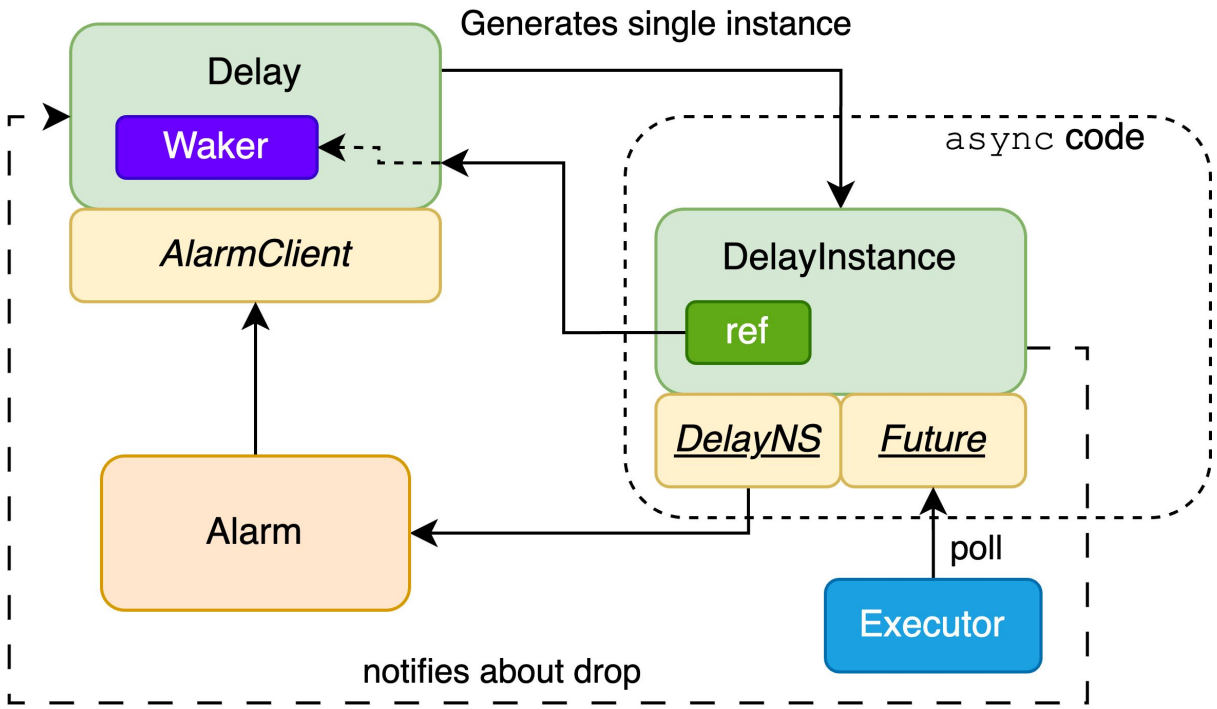
```
let executor: &mut Executor<HelloPrintDriver<…>> = static_init!(
    Executor<
        HelloPrintDriver<
            capsules_core::virtualizers::virtual_alarm::VirtualMuxAlarm<
                'static,
                Pit<'static, RELOAD_1KHZ>,
            >,
        >,
    >,
    Executor::new(hello_print)
);

hello_print.set_runner(Some(executor));
```

Async API

- We want to be compatible with `embedded-hal-async`
- We must provide implementations for at least for
  - DelayNs
  - SPI
  - I2C

Async API

- `embedded-hal-async` traits use **`&mut self`**
- Tock uses **`&self`**
- We need to split the API driver
  - Delay – Tock native driver
  - DelayInstance – provides the `embedded-hal-async` API



Example Implementation

```rust
pub struct HelloPrintDriver<A: Alarm<'static> + 'static> {
    runner: Cell<Option<&'static dyn Runner>>,
    delay: &'static Delay<'static, A>,
}
```

```rust
impl<A: Alarm<'static> + 'static> AsyncDriver for HelloPrintDriver<A> {
    type F = impl Future<Output = ()> + 'static;

    fn run(&'static self) -> Self::F {
        async {
            // you should not be able to run two futures at the same time
            // so this should never panic
            let mut delay_instance = self.delay.get_instance().unwrap();
            // loop {
            debug!("Hello");
            delay_instance.delay_ns(1_000_000_000).await;
            debug!("awaited");
            // }
        }
    }

    fn done(&self, _value: ()) {
        debug!("done");
        self.runner.get().unwrap().execute().unwrap();
    }
}
```

`Delay`

Tock API

`DelayInstance`

`embedded-hal-async` API

Limitations

- The implementation of `AsyncDriver` requires the `impl_trait_in_assoc_type` feature
- We have to name the **`impl`** `Future<Output=()>`
- Drivers perform one single `async` action

```
impl<A: Alarm<'static> + 'static> AsyncDriver for HelloPrintDriver<A> {
    type F = impl Future<Output = ()> + 'static;

    fn run(&'static self) -> Self::F {
```

Future Work

- Try to use the `embassy-rs` example and build it with stable Rust.
- Allow drivers to perform multiple actions
- Understand how to *load* existing drivers from crates.io