Inleiding

Rust op embedded systemen is nog druk in ontwikkeling. Helaas kun je niet de `std` crate gebruiken dus dingen als threads aanmaken is niet mogelijk. Wij zullen onder andere gebruik maken van de STM32F4xx-hal crate om toegang te krijgen tot de hardware. Alle peripherals zijn te gebruiken als singletons. Verder is de hal op zo'n manier opgezet dat de Rust compiler en borrow checker nog steeds nut hebben. Dit betekent ook dat het delen van variabelen tussen een interrupt en de main code erg uitdagend wordt. Om deze reden beperken we ons voor nu tot de delen van embedded Rust die wel goed te gebruiken zijn.

Opdrachten week 8 – Embedded Rust

Je gaat deze week leren hoe je:

- met behulp van embedded Rust GPIO kunt aansturen;
- een toestandsmachine in Rust volgens de State Pattern methode kunt implementeren.

Een toestandsmachine is te implementeren volgens de State Pattern[1] methode. Bij deze (object georiënteerde) methode wordt de toestandsmachine geïmplementeerd in een object waarbij het object zijn gedrag zal aanpassen aan de interne toestand. Het heeft een aantal eigenschappen:

- Het gedrag van elke toestand wordt beschreven in aparte *klassen* (of in Rust `struct`*s*), onafhankelijk van het toestandsmachine-object.
- Alle toestand-`struct`s implementeren dezelfde `trait` zodat het toestandsmachine-*object* (`struct`) onafhankelijk van de huidige toestand, altijd dezelfde methodes kan aanroepen.
- De toestanden zijn zelf verantwoordelijk om de overgang te bepalen naar de volgende toestand.

Het grote voordeel van deze methode is dat het makkelijk uitbreidbaar is en overzichtelijk blijft bij een grote hoeveelheid toestanden, in tegenstelling tot bijvoorbeeld een **`switch`**/**`case`** structuur.

[1] https://en.wikipedia.org/wiki/State_pattern

Deze methode staat ook beschreven in het eerste deel[2] van paragraaf 17.3 van het Rust boek. Onze toestandsmachine zal niet over blog-posts gaan, maar over een stoplicht.

8.1 De docent heeft een basis project opgezet zodat je direct aan de slag kunt met embedded rust. Voer de stappen uit die in de installatiehandleiding zijn te vinden voor week 8. Na afloop van de handleiding ben je in staat rust code te (cross-)compileren en te debuggen. In het quick-start project, onder examples, staan twee voorbeelden die je kunt toepassen in de opdracht:

- In allocator.rs vind je de code en imports die nodig zijn om de alloc crate te kunnen gebruiken en dus de heap. Met een heap-beheerder actief kun je vervolgens `Vec`s, `String`s en smart-pointer typen gaan gebruiken.
- In hello.rs vind je een stukje code om ARM semihosting te kunnen gebruiken en dus tekst te kunnen printen naar de terminal.

Mocht je bij deze stappen vastlopen dan kun je om hulp vragen. Wanneer het werkt, kun je beginnen aan de onderstaande Rust opdracht.

De toestandsmachine zal toegang moeten hebben tot GPIO pinnen. Als je deze via de stm32f4xx-hal crate aanmaakt, dan krijgt iedere pin een eigen uniek type. Dit is natuurlijk niet handig want dan kun je niet een algemene functie of variabele maken om de pinnen door te geven of op te slaan. Het blijkt[3] dat de pinnen worden afgeleid van de embedded-hal crate. De pinnen implementeren diverse `trait`s uit deze crate. De `trait`s zijn te vinden in de embedded-hal documentatie. Hier kun je dus gebruik van maken!

Je kunt totaal voor dit deel 25 punten scoren. Opdracht 8.2, het implementeren van de toestandsmachine, geeft 12 punten. opdracht 8.3 en opdracht 8.4 leveren respectievelijk nog eens 5 en 8 punten. Je kunt er dus voor kiezen om de volle 25 punten, of niet, te halen. Succes! Tip: Loop de uitwerking van paragraaf 17.3 van het Rust boek door en zorg ervoor dat je het begrijpt zodat je een goede basis hebt voor de eindopdracht.

[2] Het tweede deel beschrijft een alternatieve methode. Het nadeel van dit alternatief is dat er niet langer een scheiding is tussen toestandsmachine en de toestanden. Deze alternatieve methode zullen we dus niet toepassen.

[3] Dit was niet te vinden in de documentatie helaas.

8.2 Maak een toestandsmachine volgens de *State Pattern* methode zoals beschreven in het boek. Er zijn drie toestanden: rood, groen en oranje. Tussen de toestanden wordt geschakeld op basis van tijd: respectievelijk 4 s, 3 s en 1 s. (12pt)

8.3 Voeg nu een toestand toe waarbij de blauwe "User" knop aangeeft dat de brug open wilt gaan. Vanuit de rode toestand moet dan naar een andere toestand (brug-open?) worden gesprongen. Wanneer de knop wordt losgelaten springt brug-open naar de rode toestand. In de brug-open-toestand moet de rode led knipperen. Dit knipperen moet elke met een periodetijd van 2 s (2 s aan, 1 s uit). (5pt)

8.4 Een andere sterke eigenschap van Rust is foutafhandeling. Tot nu toe heb je hoogstwaarschijnlijk geen fouten afgehandeld. Maak een eigen fouttype aan met twee mogelijkheden:

- Een fout die aangeeft dat een lamp (output pin) kapot is.
- Een fout die aangeeft dat de brugsensor (input pin) kapot is.

Zorg ervoor dat *alle* fouten naar *boven* worden doorgegeven zodat in de `main` functie de fouten kunnen worden afgehandeld. Maak gebruik van de `hprintln!` macro om in de functie `main` af te drukken welke fout is opgetreden. Spring na de fout uit de toestandsmachine en laat de blauwe led knipperen met een periode van 500 ms om aan te geven dat het stoplicht niet langer functioneert. Pas na een reset van het hele systeem zal de toestandsmachine weer actief worden. Hoe je zelf fouten kunt definiëren en netjes kunt afhandelen staat beschreven in paragrafen 18.4 en 18.5 van het boek Rust by Example. Om de foutafhandeling te testen zou je handmatig een Err(..) terug kunnen geven in een van de toestanden.

Tip: Het is mogelijk om de `Infallible` fout om te zetten naar jouw eigen fouttype zodat je ? kunt toepassen. (8pt)

Verslag week 8

Ter afsluiting van het Rust deel van deze cursus moet een verslag geschreven worden.

Dit verslag bestaat uit een aantal onderdelen:

- een algemene introductie over hoe je de opdracht hebt aangepakt;

- relevante of uitdagende delen van de code met een onderbouwing hoe je die delen in het specifiek hebt opgelost;
- jouw ervaring en mening over het gebruik van Rust in tegenstelling tot C/C++.

Document

Het document mag in het Engels of in het Nederlands geschreven worden. De relevante en aangepaste broncode moet worden bijgevoegd met behulp van kleurcodering. Het document moet beginnen met een titelpagina met de namen van de studenten, studentnummers en de naam van deze cursus.

Inleveren

Het verslag over week 8 wordt ingeleverd als één PDF-document. De bestandsnaam heeft de volgende conventie:
`studentnummer_achternaam_studentnummer_achternaam_rust.pdf`
Bijvoorbeeld: `1012345_Dijkstra_1054321_Hoare_rust.pdf`

Dit bestand moet geüpload worden in de betreffende opdracht in de RTS10 Brightspace cursus. Slechts één van de twee samenwerkende studenten hoeft het bestand te uploaden.