Embedded HAL – Considered Harmful

Philipp Bormuth // awinia.de

Agenda

- Die embedded HAL
- Was ist eigentlich das Problem?
- Und was ist die Lösung?

Die embedded HAL

- Produkt der Rust embedded WG
- Rusts Antwort auf CMSIS
- Definiert einen Zoo von Interfaces für gängige MCU Peripherien
- Benötigt Implementierungen für jedes Target

Was ist eigentlich das Problem?

- Doku weist nur den Weg zur embedded HAL!
- Wenige Implementierungen:
  - STM32
  - Nordic
  - RasPi
  - …und dann wird es schon spannend

Was ist eigentlich das Problem?

- Nicht mit Vendor Tools kompatibel
- Unterkomplex z.B.:
  - Serielle Kommunikation wird auf blockierendes Lesen & Schreiben reduziert

embedded_io::Read

```
pub trait Read: ErrorType
{
   // Required method
   fn read(&mut self, buf: &mut [u8]) -> Result<usize, Self::Error>;
   // Provided method
   fn read_exact( &mut self, buf: &mut [u8] ) -> Result<(), ReadExactError<Self::Error>> { ... }
}
```

embedded_io::Read

- Timeout?
- Character Match?
- Line Idle?
- DMA?
- Break?
- Non-Blocking*?

Vendortools

- Sind oft „semigut“
- Haben dennoch eine Daseinsberechtigung (z.B. CubeMX):
  - Clocktree
  - Peripherals
  - Pins
  - ➔ Visuell, i.d.R. gut verständlich.
  - ➔ Auch für Hardwareleute bedienbar
- Erzeugen aber üblicherweise C Code

Und was ist jetzt die Lösung?

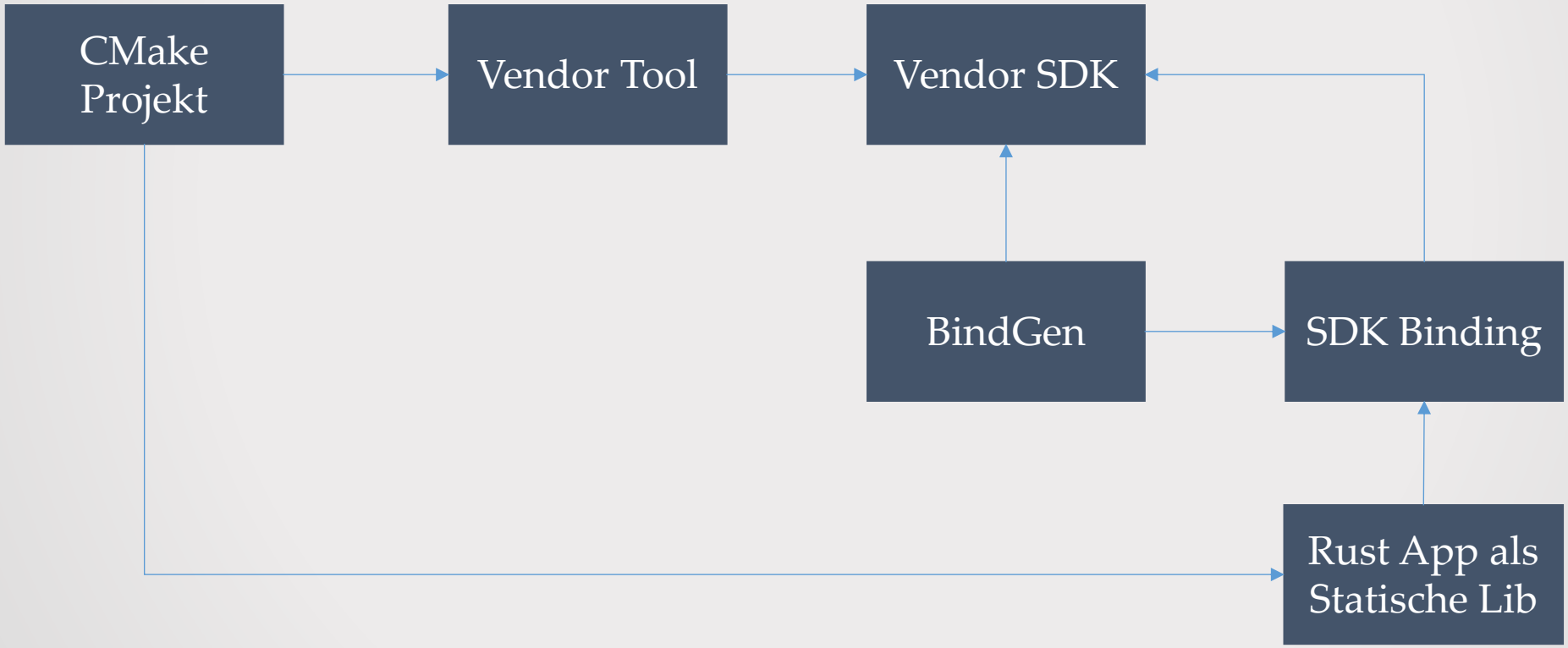
- Grundsätzlich sollte Rust auf nahezu jedem Target, das in den letzten 10 Jahren auf den Markt kam laufen.
- Meist bekommen wir von den Siliconvendors ein SDK, das bereits einen Haufen Funktionalität beinhaltet.
- Rust hat sehr guten C Interop

…warum also nicht das SDK nutzen?

Vendor SDK Nutzung

- CMake
- Corrosion
- GCC-ARM-NONE-EABI
- CubeMX

Struktur


CMake Projekt
Vendor Tool
Vendor SDK
BindGen
SDK Binding
Rust App als Statische Lib

Beispiel

Vorteile

- Funktioniert immer, wenn es C Code gibt
- Keine Abhängigkeit von embedded HAL
- Eigene Abstraktionen schaffen Mehrwert (näher an der Domäne)
- Zusätzliche Libraries via CMake einfach einzubinden
- Wer unbedingt will kann auch noch C++ dazu nehmen
- Überschaubarer Initialaufwand (i.d.R. < 1 Tag)

Nachteile

- Interrupthandling direkt in Rust nicht möglich (muss aus C geforwarded werden)
- Sämtlicher Code ist erstmal unsafe, sichere Abstraktionen müssen geschaffen werden, dadurch mehr Aufwand
- Komplexerer Build durch CMake und Cargo
- 2 Sprachen im Projekt, die sehr unterschiedlich sind

Vielen Dank

www.awinia.de