

Ausarbeitung

Embedded Rust

3. August 2018

Eingereicht von:

Niklas Kühl

inf102861@fh-wedel.de

Betreuer:

Prof. Dr. Ulrich Hoffmann

uh@fh-wedel.de

Inhaltsverzeichnis

1	Einleitung	1
2	Grundlagen	2
2.1	Rust	2
2.2	C vs Rust	2
2.3	Mikrocontroller-Programmierung	3
3	Einrichtung der Entwicklungsumgebung	4
3.1	Systemkonfiguration	4
3.2	Installation des AVR-Rust-Compilers	5
3.2.1	Grundlegendes Vorgehen	5
3.2.2	Übersetzungsvorgang (make)	5
3.2.3	Installationsvorgang (make install)	6
3.2.4	Toolchain konfigurieren (rustup toolchain)	6
3.3	Hello World AVR-Rust	6
3.4	Entwicklungsumgebungen	6
4	Entwicklung	7
5	Ergebnisse	10
6	Fazit	11
7	Notizen	12
7.1	DSL	12
	Literaturverzeichnis	13

Quellcodeverzeichnis

3.1	Kompilierung und Installation des AVR-Rust-Compilers [1]	5
4.1	Test Rust Listing	8
4.2	Und das gleiche mit Makros	8
4.3	Test Rust Listing 2	9

1 Einleitung

2 Grundlagen

2.1 Rust

- Programmiersprache - Sichere Alternative zu C - Dies soll v.a. erreicht werden durch ...

- Makros in Rust - Makros können zur Erweiterung des Sprachumfangs genutzt werden. Problematische Eigenschaften wie sie C-Makros besitzten, werden allerdings vermieden - Makros sind im Compiler integriert Es gibt also keinen Präprozessor wie bei C - Da Rust-Makros nicht wie in C auf Textersetzung basieren, können daraus resultierende Fehler nicht auftreten: - Falsche Ergebnisse durch Operator-vorrang - Macro Hygiene: Innerhalb von Makros definierte Variablen können keine anderen Variablen verstecken, wenn beide den gleichen Bezeichner haben - Makros stellen wenig anforderungen an das, was in ihnen stehen kann - Es können auch Symbole definiert werden - Es lässt sich relativ frei eine eigene Syntax spezifizieren - Parameter = metavariable - Type = fragment specifier - Da Rust Makros zu einem frühen Zeitpunkt der Kompilierung aufgelöst werden, können keine Rust-Typen wie u8 verwendet werden. Stattdessen besitzen die Parameter eines Makros (=metavariable) Typen (=fragment specifier) wie - expr (Beliebiger Rust-Ausdruck, z.B. 2+2, func(x)) - token tree

- Was ist Rust - Vor/Nachteile - Vorteile - Sichere autom. Speicherverwaltung (ohne GC) - Eingebauter Index-Check - Kein Zugriff auf nicht initialisierten Speicher möglich - Nachteile - Für Neulinge erstmal schwer zu verstehen??

- Anwendungsbereiche - Desktop Entwicklung (Firefox) - Systemprogrammierung

2.2 C vs Rust

- Buffer Overflow - Z.B. Es wird Protokoll zeilenweise übertargen -> Zeile ist zu lang und passt nicht in Buffer C: Es passiert irgendwas Rust: In der offiziellen AVR-LibCore wird dann eine Endlosschleife ausgelöst -> ein wenig besser Wenn die libcore angepasst wird, ist auch beliebiges Verhalten möglich, zb. Fehlermeldung oder Reset

- Rust: gewöhnungsbedürftig wg. borrowing-Konzept

2.3 Mikrocontroller-Programmierung

- Standardmäßig in C - Daher auch Fehler von C enthalten - Rust kann hier Vorteile bringen (wo? in welchen Bereichen?) -> Array-Index-Check -> Zugriff auf uninitialisierten Speicher

Evtl die Dreiteilung Analyse-Design-Implementierung??

3 Einrichtung der Entwicklungsumgebung

3.1 Systemkonfiguration

Folgende Programmversionen wurden für die Entwicklung genutzt:

Anwendung	Version	Beschreibung
Betriebssystem	Ubuntu 16.04.4 LTS (64-Bit)	-
rustc (Desktop)	1.26.2	Rust-Compiler für Desktop-Programme
rustup	1.11.0	Rust Update Tool
cargo	1.26.0	Rust-Buildsystem und Paketmanager
xargo	0.3.11	Alternative zu cargo, erlaubt die Verwendung einer benutzerdefinierten Standardbibliothek
rustc (AVR)	1.22.0-dev	Rust-Compiler für AVR-Programme
avr-gcc	4.9.2	AVR-C-Compiler und Linker
avrdude	6.3	AVR-Programmiertool
g++	5.4.0	C++ Compiler

Tabelle 3.1: Verwendete Software

Folgende Hardware kam bei der Entwicklung zum Einsatz:

Hardware	Beschreibung
Mikrocontroller (Entwicklungsbord)	ATmega328P Xplained Mini
Temperatursensor	Dallas 18B20
WiFi-Modul	ESP-8266l

Tabelle 3.2: Verwendete Hardware

3.2 Installation des AVR-Rust-Compilers

3.2.1 Grundlegendes Vorgehen

Um die Entwicklungsumgebung einzurichten, sollte eine normale Rust-Installation vorhanden sein. Zur Installation siehe www.rust-lang.org/en-US/install.html.

Für den Rust-Compiler für AVR-Mikrocontroller gibt es keine vorkompilierten Binärdateien. Daher muss er für das verwendete System neu übersetzt werden. Das grundlegende Vorgehen ist auf der entsprechenden GitHub-Seite beschrieben (github.com/avr-rust/rust). In den folgenden Kapiteln wird auf Eigenheiten und Probleme bei der Installation eingegangen.

```
1  # Grab the avr-rust sources
2  git clone https://github.com/avr-rust/rust.git
3
4  # Create a directory to place built files in
5  mkdir build && cd build
6
7  # Generate Makefile using settings suitable for an experimental
   ↪  compiler
8  ../rust/configure \
9  --enable-debug \
10 --disable-docs \
11 --enable-llvm-assertions \
12 --enable-debug-assertions \
13 --enable-optimize \
14 --prefix=/opt/avr-rust
15
16 # Build the compiler, optionally install it to /opt/avr-rust
17 make
18 make install
19
20 # Register the toolchain with rustup
21 rustup toolchain link avr-toolchain $(realpath $(find . -name
   ↪  'stage1'))
22
23 # Optionally enable the avr toolchain globally
24 rustup default avr-toolchain
```

Listing 3.1: Kompilierung und Installation des AVR-Rust-Compilers [1]

3.2.2 Übersetzungsvorgang (make)

Die ersten Schritte bis zum eigentlichen Übersetzungsvorgang sollten ohne Probleme ausgeführt werden. Der aufwändigste Schritt ist der eigentliche Übersetzungsprozess. Dieser

kann einige Zeit (mehrere Stunden) in Anspruch nehmen und auch mit einem Fehler abbrechen. Im folgenden sind einige Fehler aufgelistet welche beim Übersetzungsvorgang auftraten, sowie mögliche Lösungsansätze:

- Der Übersetzungsprozess bricht mit einem Speicherfehler ab
Wenn ein 32-Bit-Betriebssystem verwendet wird, sollte auf ein 64-Bit-Betriebssystem gewechselt werden. Zum Zeitpunkt dieser Dokumentation war die Übersetzung auf 32-Bit-Systemen nicht möglich. Dies ist ein bekannter Fehler¹.
- Der Übersetzungsprozess wird vom Betriebssystem beendet
Es ist nicht genügend Arbeitsspeicher verfügbar. (In einer virtuellen Maschine mit 4GB RAM schlug die Übersetzung fehl, nach Erhöhung auf 6GB RAM war die Übersetzung erfolgreich)
- Fehler beim Linken des Programms: *cannot find -lffi*
Es müssen die fehlenden Pakete *libffi6* und *libffi-dev* installiert werden.

3.2.3 Installationsvorgang (make install)

Der Befehl `make install` installiert den AVR-Rust-Compiler in das Verzeichnis `/opt/avr-rust`. Dies schlug zunächst fehl, da hierfür Root-Rechte erforderlich sind. Eine Ausführung von `make install` mit Root-Rechten schlug ebenfalls fehl. Daher wurde der Ordner `/opt/avr-rust` manuell mit vollen Berechtigungen angelegt:

```
1 sudo mkdir -m 777 /opt/avr-rust
```

Dadurch konnte die Installation erfolgreich ausgeführt werden konnte. Im Anschluss wurden die Rechte des Ordners dann wieder eingeschränkt.

3.2.4 Toolchain konfigurieren (rustup toolchain)

3.3 Hello World AVR-Rust

3.4 Entwicklungsumgebungen

- Diskussion IDEs - Eclipse mit RustDT - VS Code

¹<https://github.com/rust-lang/rust/issues/39394>

4 Entwicklung

- Erst kleinere Testprogramme - Dann größeres Programm mit Peripherie-Ansteuerung - DSL mit Rust - Method Chaining - Makro DSL - Hello World AVR-Rust - Tests Rust vs C (AVR)

- Test des avr-periphery Programms erfolgte mit einem durch netcat bereitgestellten Server, dadurch konnten die durch den µController gesendeten nachrichten auf einem PC angezeigt werden

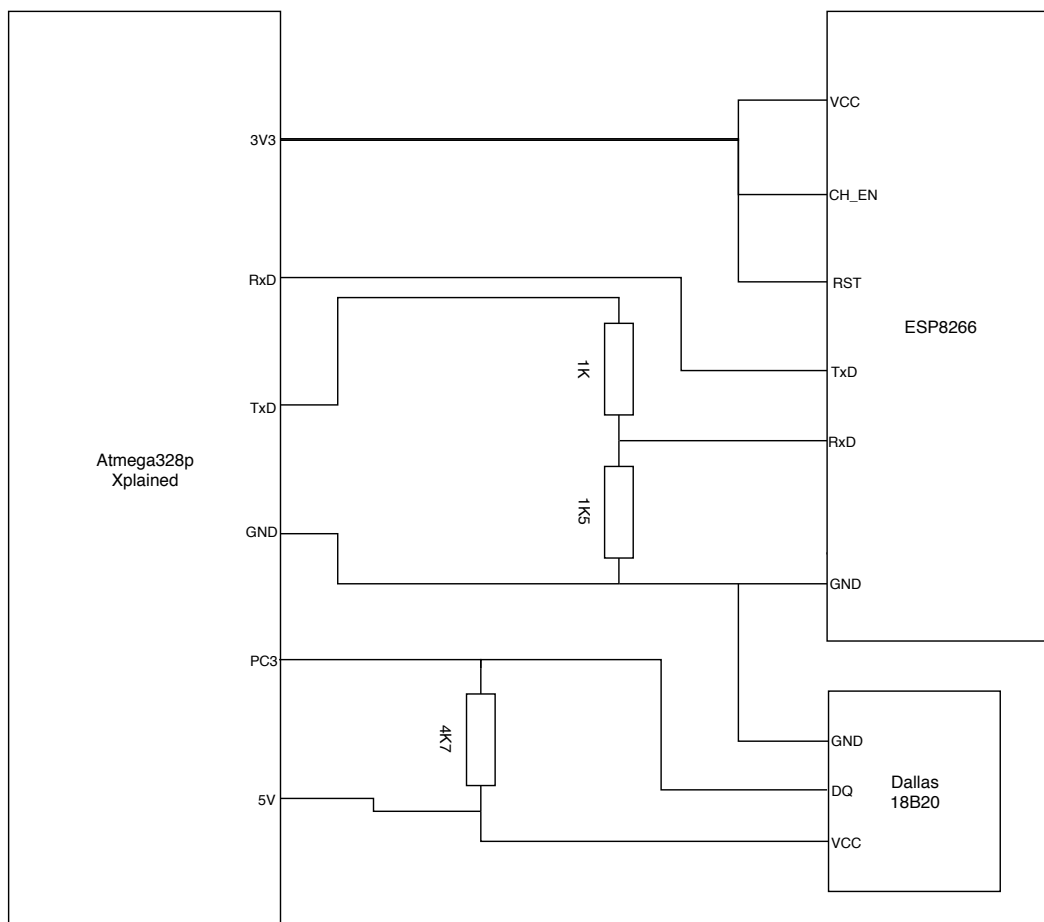


Abbildung 4.1: Schaltplan

```

1  fn open_server_connection() -> Result<TCPConnection, ()> {
2      esp8266::at().send().wait();
3      esp8266::at().ext().reset().send().wait();
4
5      esp8266::at().ext().wifi().mode().set().cw_mode(CWMode::Client)
↪    .send().wait();
6      esp8266::at().ext().multi_connections().set().enabled(true)
↪    .send().wait();
7      esp8266::at().ext().wifi().scan().send().wait();
8
9      let connect_wifi =
↪    esp8266::at().ext().wifi().connection().set().name(WIFI_NAME)
↪    .pw(WIFI_PW).send().wait();
10     if let Ok(_) = connect_wifi {
11         return esp8266::at().ext().tcp().open().set()
12             .tcp_handle(TCPHandle::Multil)
↪    .hostname(HOST).port(PORT).send().wait();
13     }
14     else {
15         return Err(());
16     }
17 }

```

Listing 4.1: Test Rust Listing

```

1  fn open_server_connection() -> Result<TCPConnection, ()> {
2      esp_cmd!(send wait AT)?;
3      esp_cmd!(send wait AT+RST)?;
4
5      esp_cmd!(send wait ATE true)?;
6      esp_cmd!(send wait AT+CWMODE = cw_mode:(CWMode::Client))?;
7      //let mode = esp_cmd!(send get AT+CWMODE?)?;
8      esp_cmd!(send wait AT+CIPMUX = enabled: true)?;
9      esp_cmd!(send wait AT+CWLAP)?;
10
11     esp_cmd!(send wait AT+CWJAP = name:WIFI_NAME, pw:WIFI_PW)?;
12     return esp_cmd!(send wait AT+CIPSTART =
↪    tcp_handle:(TCPHandle::Multil), hostname: HOST, port:PORT);
13 }

```

Listing 4.2: Und das gleiche mit Makros

```

1  #[no_mangle]
2  fn send_temperature() { // Send Temperature to server
3      if let Ok(conn) = open_server_connection() {
4          for _i : u8 in 0.. 10 {
5              conn.send_str(b"Temperature: ").wait();
6              let temp = ds18b20::read_temperature();
7              let temp_str = ds18b20::temperature_to_str(&temp);
8              conn.send_str(&temp_str).wait();
9              conn.send_str(b"\r\n").wait();
10             delay_ms(1000);
11         }
12         close_server_connection(conn);
13     }
14 }

```

Listing 4.3: Test Rust Listing 2

5 Ergebnisse

- Erkenntnisse aus den Tests (Vor/Nachteile zu C) - Probleme bei der Entwicklung

6 Fazit

- Im allgemeinen eignet sich rust zur systmprogrammierung. Durch A, B und C ergeben sich Vorteile gegenüber der Verwendung von C. Dadurch können robustere Programme entwickelt werden.

Die eingesetzte Compiler Version hatte jedoch einige Schwächen

Somit wird vom Einsatz von Rust auf AVR vorerst abgeraten. Wenn der Compiler stabil ist, kann Rust allerdings gut verwendet werden.

7 Notizen

- Sind Maybe-Typen von Rust effizient auf AVR? - Sind ISR's re-entrant bei Rust => Was passiert, wenn bei ISR-Ausführung ein weiterer Interrupt auftritt - Was gibt es für std-libraries für avr rust - unsafe Rust - Was ist daran unsicher - Echte Pointer möglich, die NULL sein dürfen => Spezieller Zeigertyp, die STd-Refs von Rust dürfen auch in unsafe Code nie null sein - Mit diesen Zeigertypen ist auch der Zugriff auf beliebige Speicheradressen möglich - wie kann man die unsafe-sectionen auf ein min reduzieren? => Durch Funktionen und Makros lassen sich unsafe-zugriffe kapseln - z.B. auch erzeugen einer vermeintlich sicheren Referenz auf globales Obj => trotz normaler Referenz nicht thread-safe
- Performance - Test: Fibonacci Berechnung inkl Speicherzugriffe Ergebnis: C etwa um Faktor 1.5 schneller, abhängig vom Optimierungslevel - Code größe 1 C mit opt-size 1500By 2 Rust 2600 3 C mit opt-speed 2800 - RAM Verbrauch C: Bei 120 Calls Stack overflow; $120 \cdot 16 = 1920$ Bytes von 2048 Rust Bei 100 Calls $100 \cdot 16 = 1600$ Bytes - AL-LERDINGS: C COMpiler wurde über Jahre hinweg Optimiert, daher sind diese Zahlen schon ganz gut
- Rust Out of Bounds - panic wird ausgelöst - Nutzer muss panic Funktion definieren, welche festlegt was dann passieren soll - Die Funktion darf nicht zurückkehren; dies wird durch Rust zur Übersetzungszeit festgelegt - ABER: In der avr-libcore wird die benutzer-definierte Funktion nicht aufgerufen; stattdessen wird Endlosschleife ausgelöst - Wenn libcore angepasst wird (WO?) ist bei OOB beliebiges Verhalten wie zB Fehlermeldung oder Neustart möglich
- Vorteile durch Typsystem - Erkennung und Verhinderung von uninitialisierten Variablen - Verwechslung von Variablen werden leichter erkannt (da Zwischen u8 und usize unterschieden wird) - Fehler bei Zuweisung unterschiedlicher Typen - Keine implizite Typumwandlung
- Was fehlt AVR-Rust - Stabiler Compiler - Std-Library - Delay-Funktion - ISR-Definitionen (wie in c die interrupt.h)

7.1 DSL

- Verboten, dass andere Kommandos als die unterstützten zusammengesetzt werden - Fehler werden schon zur Übersetzungszeit erkannt => Wie sehen die Fehlermeldungen aus? (Makro-DSL) Wenn man sich beim Makro verschrieben hat, kommt dann eine aussagekräftige Fehlermeldung??? => Stichwort: Fluent Interface

Literaturverzeichnis

- [1] Github-Seite des AVR-Rust-Projekts. <https://github.com/avr-rust/rust>. Abgerufen am 03.08.2018.
- [2] Rust-Dokumentation. <https://doc.rust-lang.org/>. Abgerufen am 03.08.2018.