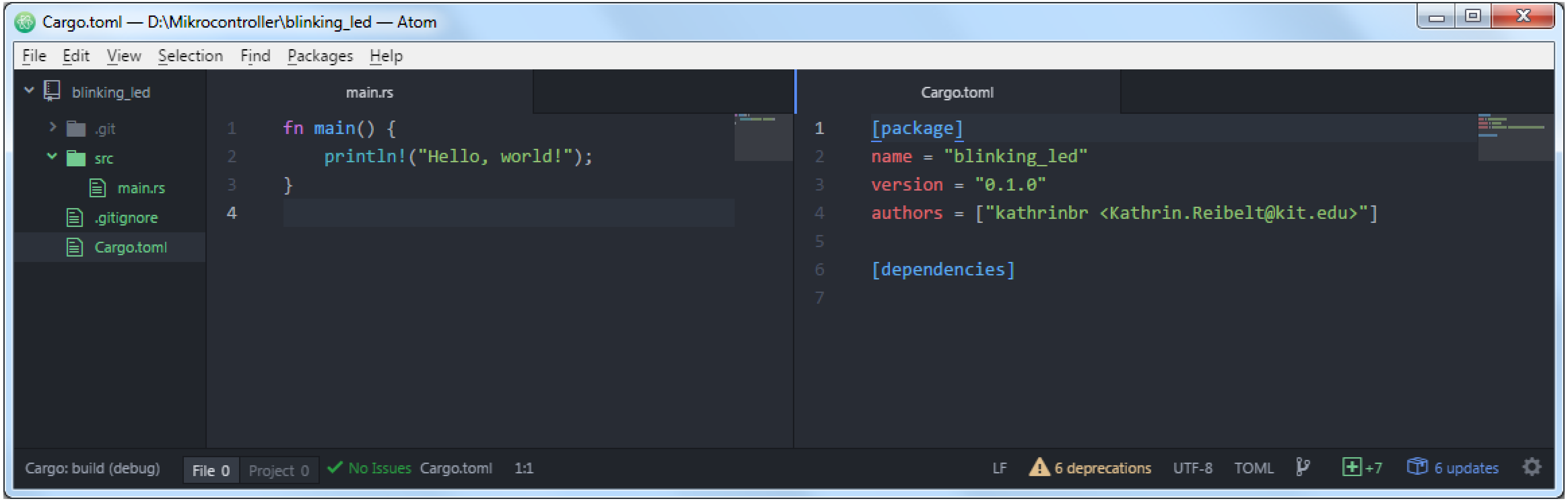
Aufbau eines Projekts für den **Mikrocontroller**

Ansteuerung einer Leuchtdiode

1. Projekt erstellen

In Konsole:

```
xargo new blinking_led --bin
```



```rust
fn main() {
    println!("Hello, world!");
}
```

```toml
[package]
name = "blinking_led"
version = "0.1.0"
authors = ["kathrinbr <Kathrin.Reibelt@kit.edu>"]

[dependencies]
```

2. Abhängigkeiten einbinden

Cargo.toml

```
[package]
…

[dependencies]
cortex-m = "0.1.4 "
r0 = "0.1.0"

[dependencies.stm32f7_discovery]
git = "https://github.com/embed-rs/stm32f7-discovery.git"
version = "0.1.0"
```

3. Compiler Konfigurieren

Cargo.toml

```toml
[package]
…

[dependencies]
…

[profile.release]
lto = true
```

3. Compiler Konfigurieren

• 📁 .cargo └→ config	für welches Target soll gebaut werden
• stm32f7.json	Beschreibung des Targets
• stm32f7.ld	targetspezifische Konfigurationen
• Xargo.toml	xargospezifische Abhängigkeiten

5. Compilerswitches, Bibliotheken

main.rs

```
#![no_std]
#![no_main]

extern crate stm32f7_discovery as stm32f7;

// initialization routines for .data and .bss
extern crate r0;

use stm32f7::{system_clock, board, embedded};
```

6. Speicher reservieren

main.rs

```
…
#[no_mangle]
pub unsafe extern "C" fn reset() -> ! {
    extern "C" {
        static __DATA_LOAD: u32;
        static __DATA_END: u32;
        static mut __DATA_START: u32;
        static mut __BSS_START: u32;
        static mut __BSS_END: u32;
    }
…
}
```

7. Speicher zuweisen

main.rs

```
...

#[no_mangle]
pub unsafe extern "C" fn reset() -> ! {
    extern "C" {
        ...
    }
    let data_load = &__DATA_LOAD;
    let data_start = &mut __DATA_START;
    let data_end = &__DATA_END;
    let bss_start = &mut __BSS_START;
    let bss_end = &__BSS_END;
    ...
}
```

8. Initialisieren, main aufrufen

main.rs

```
#[no_mangle]
pub unsafe extern "C" fn reset() -> ! {
    extern "C" {
        …
    }
    …
    // initializes the .data section
    //(copy the data segment initializers from flash to RAM)
    r0::init_data(data_start, data_end, data_load);
    // zeroes the .bss section
    r0::zero_bss(bss_start, bss_end);

    main(board::hw());
}
```

9. main - function

main.rs

```
…

fn main(hw: board::Hardware) -> ! {

    ...

}
```

10. hw exrahieren

```
fn main(hw: board::Hardware)
```

```
    let board::Hardware { rcc,
                          pwr,
                          flash,
                          gpio_a,
                          gpio_b,
                          gpio_c,
                          gpio_d,
                          gpio_e,
                          gpio_f,
                          gpio_g,
                          gpio_h,
                          gpio_i,
                          gpio_j,
                          gpio_k,
                          .. } = hw;
```

10. Excurs: Dokumentation

- Konsole: `xargo doc --open`
- …\blinking_led\target\stm32f7\doc\stm32f7_discovery \index.html

10. Excurs: Dokumentation

```
extern crate stm32f7_discovery as stm32f7;
use stm32f7::{system_clock, board, embedded};
main(board::hw());
```

- (linkes Menü) Crates:
  stm32f7_discovery
- Crate stm32f7_discovery:
  (oben) pub extern crate embedded_stm32f7 as board;
- Crate embedded_stm32f7:
  (unten) Functions: hw -> Hardware
- Felder, z.B.:

```
pub gpio_a: &'static mut Gpio,
pub gpio_b: &'static mut Gpio,
    …
```

11. Pins in neues struct zusammenfassen

fn main(…)

```
use embedded::interfaces::gpio::{self, Gpio};

let mut gpio = Gpio::new(gpio_a,
                         gpio_b,
                         gpio_c,
                         gpio_d,
                         gpio_e,
                         gpio_f,
                         gpio_g,
                         gpio_h,
                         gpio_i,
                         gpio_j,
                         gpio_k);
```

11. Excurs: Dokumentation

```
let mut gpio = Gpio::new(gpio_a,
                                ...);
```

- embedded_stm32f7::Hardware:

```
pub gpio_a: &'static mut Gpio,
pub gpio_b: &'static mut Gpio,
      ...
```

- Gpio:

  Struct mit 9 Feldern

  kein Konstruktor mit Argumenten

11. Excurs: Dokumentation

```
use embedded::interfaces::gpio::{self, Gpio};
```

- Gpio:

  (oben) embedded

- Crate embedded:

  Modules: interfaces

- Module embedded::interfaces:

  Modules: gpio

- Module embedded::interfaces::gpio:

  Structs: Gpio

- Module embedded::interfaces::gpio::Gpio:

  Methods: u.a. Konstruktor

12. initialisieren

fn main(…)

```
system_clock::init(rcc, pwr, flash);

// enable all gpio ports
    rcc.ahb1enr.update(|r| {
        r.set_gpioaen(true);
        r.set_gpioben(true);
        r.set_gpiocen(true);
        r.set_gpioden(true);
        r.set_gpioeen(true);
        r.set_gpiofen(true);
        r.set_gpiogen(true);
        r.set_gpiohen(true);
        r.set_gpioien(true);
        r.set_gpiojen(true);
        r.set_gpioken(true);
    });
```

13. Led - pin vorbereiten

```
fn main(…)
```

```
// configure led pin as output pin
    let led_pin = (gpio::Port::PortI, gpio::Pin::Pin1);

    let mut led = gpio.to_output(led_pin,
                    gpio::OutputType::PushPull,
                    gpio::OutputSpeed::Low,
                    gpio::Resistor::NoPull)
        .expect("led pin already in use");

// turn led on
    led.set(true);
```

13. Excurs: Dokumentation

- Module embedded::interfaces::gpio::Gpio:

```
fn to_output(&mut self,                 Variants:
             pin: (Port, Pin),          PortA-PortK, Pin1-Pin15
             out_type: OutputType,      PushPull, OpenDrain
             out_speed: OutputSpeed,    Low, Medium, High, VeryHigh
             resistor: Resistor)        NoPull, PullUp, PullDown
             -> Result<OutputPin, Error> Ok(T), Err(E)
                                        fn get(), fn set(bool)
                                        PinAlreadyInUse(Pin)
```

14. Programm

fn main(…)

```
let mut last_led_toggle = system_clock::ticks();
loop {
    let ticks = system_clock::ticks();

    // every 0.5 seconds
    if ticks - last_led_toggle >= 500 {
        // toggle the led
        let led_current = led.get();
        led.set(!led_current);
        last_led_toggle = ticks;
    }
}
```

15. Bauen und Laden

- Konsole: `xargo build` → Finished
- Neue Konsole: `st-util`
  (Win: stlink-1.3.1-win32\bin\st-util.exe)
- Aus stm32f7-discovery:
  - .gdbinit
  - gdb.sh / gdb.bat
- Erste Konsole: `gdb.sh` bzw. `gdb.bat`