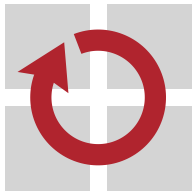
Übungen zu Grundlagen der systemnahen Programmierung in C (GSPiC) im Wintersemester 2017/18

2018-01-09

Bernhard Heinloth

Lehrstuhl für Informatik 4
Friedrich-Alexander-Universität Erlangen-Nürnberg



Lehrstuhl für Verteilte Systeme
und Betriebssysteme

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

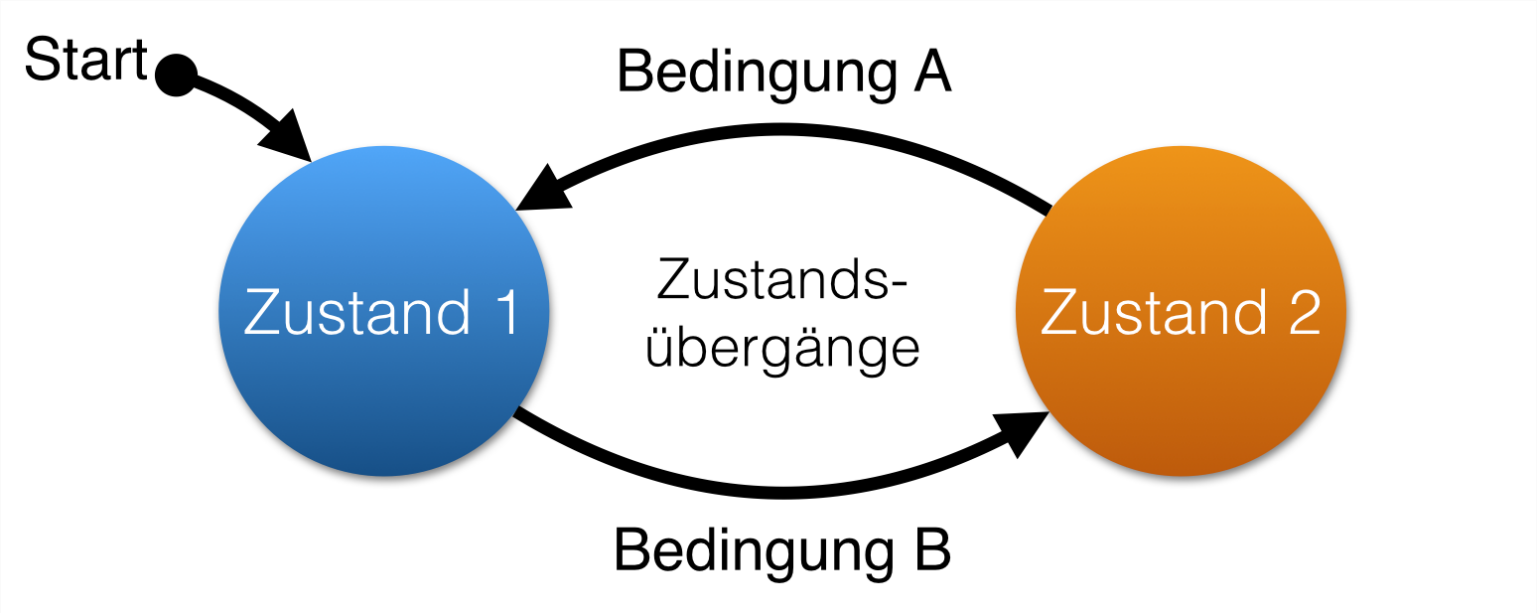
TECHNISCHE FAKULTÄT

Aufgabe: Ampel

Aufgabe: Ampel

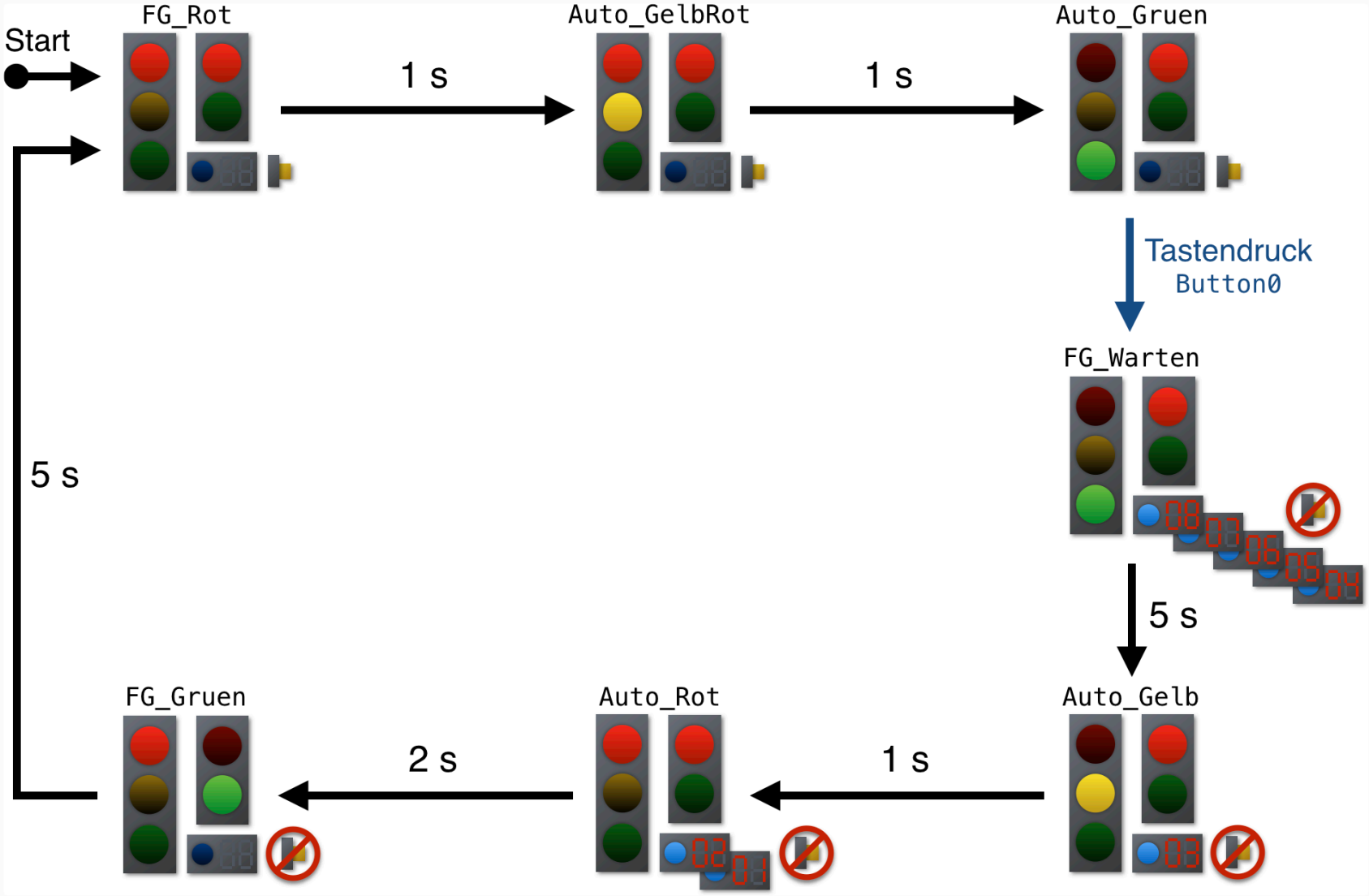
- Implementierung einer (Fußgänger-)Ampel mit Wartezeitanzeige
- Ablauf (exakt) nach Aufgabenbeschreibung (Musterlösung verfügbar)
- Hinweise
  - Tastendrücke und Alarme als Events (kein aktives Warten)
  - In Sleep-Modus wechseln, wenn keine Events zu bearbeiten sind
  - nur eine Stelle zum Warten auf Events (`sleep-Loop`)
  - Deaktiveren des Tasters durch Ignorieren des Events (Änderung der Interrupt-Konfiguration ist nicht notwendig)
  - Abbildung auf Zustandsmaschine sinnvoll
  - Verwendung von `volatile` begründen

Zustandsmaschinen

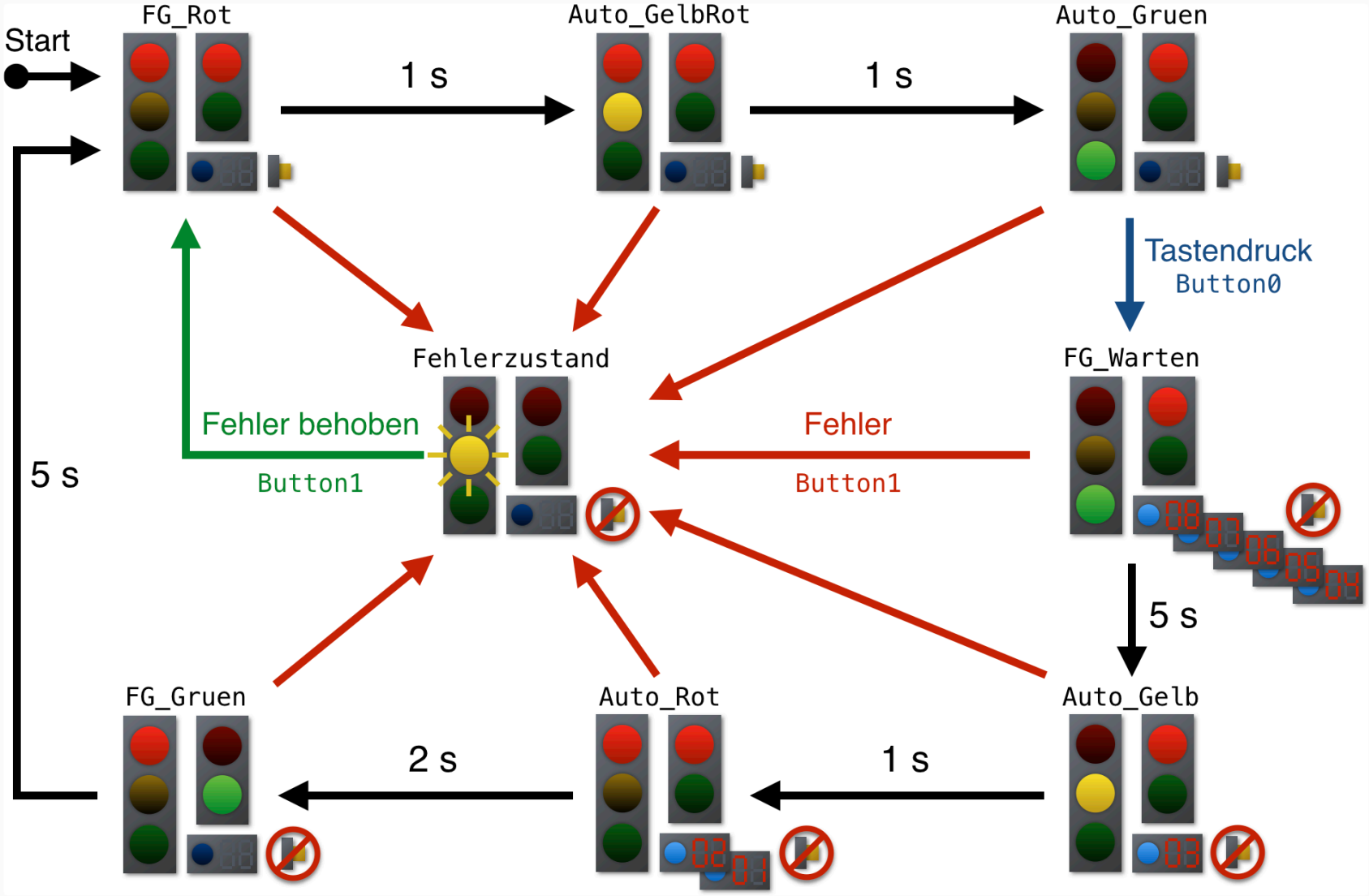


- **Zustände** mit bestimmten Eigenschaften; definierter Initialzustand
- **Zustandswechsel** in Abhängigkeit von definierten Bedingungen

Ampel als Zustandsmaschine



Ampel als Zustandsmaschine



Festlegen von Zuständen: enum-Typen

- Festlegung durch Zahlen ist fehleranfällig
  - Schwer zu merken
  - Wertebereich nur bedingt einschränkbar
- Besser `enum`:

```
01 enum state { STATE_RED, STATE_YELLOW, STATE_GREEN };
02
03 enum state my_state = STATE_RED;
```

- Mit `typedef` noch lesbarer:

```
01 typedef enum { STATE_RED, STATE_YELLOW, STATE_GREEN } state;
02
03 state my_state = STATE_RED;
```

Zustandsabfragen: switch-case-Anweisung

```
01 switch ( my_state ) {
02 case STATE_RED:
03   ...
04   break;
05 case STATE_YELLOW:
06   ...
07   break;
08 case STATE_GREEN:
09   ...
10   break;
11 default:
12   // maybe invalid state
13   ...
14 }
```

- Vermeidung von `if-else`-Kaskaden
- `switch`-Ausdruck muss eine Zahl sein (besser ein `enum`-Typ)
- `break`-Anweisung nicht vergessen!
- Ideal für die Abarbeitung von Systemen mit verschiedenen Zuständen
  ⇒ Implementierung von Zustandsmaschinen

Konfiguration der Timer

- Nachbilden der Funktionalität aus dem `timer` Modul der `libspicboard`
- Regelmäßiger Alarm, um Zustandsübergänge zu realisieren
- Der ATmega328PB bietet 5 verschiedene Timer
- Für diese Aufgabe: Verwendung von `TIMER0` (8-bit Timer)
  - `TIMER0` kann Werte zwischen 0 und 255 annehmen
  - Wird automatisch, abhängig von der CPU Geschwindigkeit, erhöht
  - Kann benutzt werden, um Interrupts beim Erreichen bestimmter Werte oder bei Überläufen auszulösen
  - Für diese Aufgabe: Verwendung des Überlaufinterrupt (OVF)

Konfiguration der Timer

- Geschwindigkeit durch `prescaler` einstellbar (Anzahl der CPU-Takte, bis der Zähler inkrementiert wird)
- Zum Beispiel:
  - 8-bit Timer mit Überlaufinterrupt
  - CPU Frequenz: 1 MHz
  - `prescaler`: 64

  → alle $\frac{64}{1\,\text{MHz}} = 64\,\mu\text{s}$ wird der Zähler erhöht

  → alle $\frac{64 \cdot 256}{1\,\text{MHz}} = 16.4\,\text{ms}$ wird der Überlaufinterrupt ausgelöst

- CPU Geschwindigkeit des ATmega328PB: 16 MHz
  - Wie viele Interrupts müssen auftreten, bis 1 s vergangen ist?
  - Welcher `prescaler` ist am ressourcenschonendsten?

Konfiguration der Timer

- Clock Select (CS) Bits befinden sich beim ATmega328PB im `TC0` Control Register B (`TCCR0B`)
- (De-)aktivieren den `TIMER0` und stellen die Geschwindigkeit ein

CS02	CS01	CS00	Beschreibung
0	0	0	Timer aus
0	0	1	prescaler 1
0	1	0	prescaler 8
0	1	1	prescaler 64
1	0	0	prescaler 256
1	0	1	prescaler 1024
1	1	0	Ext. Takt (fallende Flanke)
1	1	1	Ext. Takt (steigende Flanke)

Konfiguration der Timer

- TIMER0 Overflow Interrupt Enable (TOIE0) Bit befindet sich beim ATmega328PB im TC0 Interrupt Mask Register (TIMSK0)
- (De-)aktivieren des Überlaufinterrupts

```
01 ISR(TIMER0_OVF_vect) {
02   // [...]
03 }
04
05 void init(void) {
06   // Timer mit prescaler 64 aktivieren
07   TCCR0B |= (1 << CS01) | (1 << CS00);
08   TCCR0B &= ~(1 << CS02);
09
10   // Überlaufunterbrechung aktivieren
11   TIMSK0 |= (1 << TOIE0);
12
13   // [...]
14 }
```